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

Alberto Bastos Vargas
Enginyeria en Informàtica

Josep Maria Camps Riba

16 de gener de 2012

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.

1. Resum

El desenvolupament de software viu temps d'evolució permanent i d'un alt nivell d'exigència. En temps on predomina la immediatesa per preparar i accedir la informació, aquest requisit d'eficiència és encara més forta i agressiva que mai. Com a cap visible, el llenguatge de programació Java i la seva especificació J2EE han dominat i conservat una posició de privilegi en els darrers anys, però no s'aconsegueix aquesta condició sense un bon motiu. Aquest motiu, a banda de la constant evolució i la magnitud de la seva comunitat, una forta política de reutilització abanderada per l'existència de marcs de treball que permeten iniciar els projectes amb una base ja disponible.

Disposem d'un programari de base per a qualsevol problema que volem encarar amb J2EE. Per exemple, tenim llibreries ORM per facilitar l'accés a dades, o complexes estructures de codi que implementen el patró MVC sense necessitat de repetir línies i línies de programari. És aquesta darrera possibilitat la que ocuparà els nostres esforços, mirant d'entendre i aprofundir en el món dels framework de presentació.

El resultat de la nostra dedicació serà una síntesi dels principals arguments de les solucions ja existents al mercat en matèria de frameworks J2EE de presentació, així com el disseny i implementació de la nostra pròpia proposta basada en una combinació de idees pròpies i tècniques més atractives observades durant l'estudi esmentat.

Es tanca així el cercle de auto-alimentació que permet a J2EE continuar essent una tecnologia clau: facilitar la vida del desenvolupador, i permetre que aquest facilita la vida dels que arriben a continuació.

2. Índex de continguts

1. Resum ____ 3
2. Índex de continguts ____ 4
3. Índex de figures ____ 9
4. Cos de la memòria ____ 10
   4.1. Introducció ____ 10
      4.1.1. Justificació i context ____ 10
      4.1.2. Objectius ____ 12
      4.1.3. Enfocament i metodologia ____ 13
      4.1.4. Planificació ____ 13
      4.1.5. Productes obtinguts ____ 18
         4.1.5.1. Casos pràctics d'eines existents ____ 18
         4.1.5.2. F Framework ____ 18
         4.1.5.3. Cas pràctic de F Framework ____ 19
      4.1.6. Breu descripció de la resta de capítols ____ 19
         4.1.6.1. Components J2EE ____ 19
         4.1.6.2. El patró FrontController ____ 19
         4.1.6.3. Estudi d'eines existents ____ 19
         4.1.6.4. Desenvolupament d'un framework: FFramework ____ 19
         4.1.6.5. Aplicació de prova ____ 20
   4.2. Components J2EE ____ 21
      4.2.1. Servlets ____ 21
      4.2.2. Filters ____ 21
      4.2.3. Session ____ 21
      4.2.4. Context ____ 22
      4.2.5. Listeners ____ 22
      4.2.6. Java Server Pages ____ 22
      4.2.7. JSP TagLIbs ____ 22
   4.3. El patró FrontController ____ 23
   4.4. Estudi d'eines existents ____ 24
      4.4.1. Spring Web MVC Framework ____ 24
         4.4.1.1. Identificació i història ____ 24
         4.4.1.2. Estil i arquitectura ____ 25
         4.4.1.3. Components i Beans ____ 25
            I. Controller ____ 26
            II. Handler Mapping ____ 27
            III. View Resolver ____ 27
            IV. Handler Exception Resolvers ____ 27
         4.4.1.4. Configuració ____ 28
            I. Convencions per estalviar configuració ____ 29
         4.4.1.5. Característiques fora d'estudi ____ 29
            I. Detecció de configuració regional ____ 29
            II. Suport per a temes ____ 29
            III. Enviament d'arxius ____ 30
            IV. Llibreria JSP TagLib ____ 30

V. Arquitectura de portlets 30
4.4.1.6. Cas pràctic 30
I. Creació del projecte i llibreries necessàries 30
II. Configuració del DispatcherServlet 32
III. Creació del controlador 32
IV. Creació de punts d'accés: llistat de contactes 33
V. Creació de vistes: llista de contactes 33
VI. Resta de punts d'accés 34
VII. Un darrer problema: rebre estructures complexes 35
VIII. Afegir una segona tecnologia de vistes 36
IX. Tractament d'excepcions 36
4.4.1.7. Conclusions 37
4.4.2. Struts 2 Framework 38
4.4.2.1. Identificació i història 38
4.4.2.2. Estil i arquitectura 38
4.4.2.3. Components del framework 39
I. Actions 39
II. Results 40
III. Interceptors 40
IV. TagLib JSP 40
V. ValueStack i OGNL 41
4.4.2.4. Configuració 41
I. Configuració per XML 41
II. Configuració per convenció de noms 42
III. Configuració per anotacions Java 42
4.4.2.5. Validació de dades 42
4.4.2.6. Característiques fora d'estudi 43
I. Struts Tiles 43
4.4.2.7. Cas pràctic 43
I. Creació del projecte i llibreries necessàries 43
II. Configuració del FilterDispatcher 44
III. Creació de les accions de l'aplicació 45
IV. Creació de la classe controlador 46
V. Creació de les vistes 47
VI. Validació de dades 47
VII. Tractament d'excepcions 49
4.4.2.8. Conclusions 49
4.4.3. Java Server Faces 50
4.4.3.1. Identificació i història 50
4.4.3.2. Estil i arquitectura 50
4.4.3.3. Components del framework 51
I. Taglibs 52
II. Components UI 52
III. Managed Beans 52
4.4.3.4. Configuració 53
4.4.3.5. Validació de dades 53
4.4.3.6. Característiques fora d'estudi 53

I. Suport AJAX (RichFaces, etc.) 54
II. Suport per temes 54
4.4.3.7. Cas pràctic 54
I. Creació del projecte 55
II. Creació del Managed Bean 55
III. Creació de les vistes 56
IV. Estructura DataModel 57
V. Atribut “rendered” 58
VI. Validació de dades 58
VII. Tractament d’excepcions 59
4.4.3.8. Conclusions 59
4.4.4. Grails 60
4.4.4.1. Identificació i història 60
4.4.4.2. Groovy 60
4.4.4.3. Estil i arquitectura 61
4.4.4.4. Components del framework 61
I. Classes de domini 61
II. Controller 61
III. View 62
4.4.4.5. Configuració 62
4.4.4.6. Scaffolding 63
4.4.4.7. Característiques fora d’estudi 64
I. Cas pràctic 64
4.4.4.8. Conclusions 65
4.4.5. Conclusions de l’estudi 66
4.5. Desenvolupament d’un framework: FFramework 67
4.5.1. Principis bàsics 67
4.5.1.1. Fàcil configuració 67
4.5.1.2. Innovació: més anotacions, menys herències 67
4.5.1.3. No requerir molts arxius: convivència de dades i accions 68
4.5.1.4. Si vols accedir, pots accedir 68
4.5.1.5. Una vista, molts contenidors 68
4.5.1.6. Potenciar la vista amb automatismes 69
4.5.2. Tecnologies clau 69
4.5.2.1. Anotacions Java 69
4.5.2.2. Java Reflection 70
4.5.2.3. Reflection avançada per conèixer genèrics 70
4.5.2.4. FreeMarker 71
4.5.3. Casos d’ús 72
4.5.3.1. CUS1. Instal·lar framework 72
4.5.3.2. CUS2. Afegir contenidor de dades 72
4.5.3.3. CUS3. Afegir dades d’entrada i sortida 73
4.5.3.4. CUS4. Afegir punts d’entrada o accions 74
4.5.3.5. CUS5. Afegir paràmetres d’entrada a una acció 74
4.5.3.6. CUS6. Afegir i instal·lar conversors globals 75
4.5.3.7. CUS7. Afegir i instal·lar conversors específics 76
4.5.3.8. CUS8. Afegir i instal·lar validadors de dades 77

4.5.3.9. CUS9. Afegir i instal·lar gestors d'excepcions globals 77
4.5.3.10. CUS10. Afegir i instal·lar gestors d'excepcions específics 78
4.5.3.11. CUS11. Afegir directives pròpies 79
4.5.3.12. CUS12. Afegir vista 79
4.5.4. Classes del sistema 80
4.5.4.1. Diagrama de classes 80
4.5.4.2. FServlet 80
4.5.4.3. FInstalledBean 81
4.5.4.4. FInstalledAction 81
4.5.4.5. FInstalledParam 81
4.5.4.6. FInboundConverter 81
4.5.4.7. FInboundValidator 81
4.5.4.8. FExceptionHandler 81
4.5.4.9. FRequestResult 82
4.5.4.10. FDirective 82
4.5.5. Inicialització del sistema 83
4.5.5.1. Diagrama de seqüència 83
4.5.5.2. Descripció de la inicialització 85
4.5.6. Processat de petició 86
4.5.6.1. Diagrama d'activitat 86
4.5.6.2. Descripció del processat 86
4.5.7. Guia d'anotacions 88
4.5.7.1. @FBean 88
4.5.7.2. @FAction 88
4.5.7.3. @FParam 89
4.5.7.4. @FGlobalConverter 89
4.5.7.5. @FGlobalExceptionHandler 89
4.5.7.6. @FConverters 89
4.5.7.7. @FExceptionHandlers 90
4.5.7.8. @FDirective 90
4.5.8. Guia d'interfícies 90
4.5.8.1. FInboundConverter 90
4.5.8.2. FInboundValidator 91
4.5.8.3. FExceptionHandler 91
4.5.9. Manual d'ús 91
4.5.9.1. Instal·lació del framework 91
4.5.9.2. Creació de components satèl·lits 92
4.5.9.3. Creació de les vistes 93
4.5.9.4. Posada en marxa 93
4.5.9.5. Com referir-se a una acció 93
4.5.9.6. Com actualitzar a atributs del controlador 94
4.5.9.7. Accions que terminen amb una redirecció 94
4.5.10. Components per defecte 95
4.5.10.1. Conversors de dades 95
4.5.10.2. Gestor d'excepcions 95
4.5.10.3. Directives 96
I. Directives per a camps de formulari 96

II. Directives per a mostrar errors _____ 98
III. Directiva per a mostrar valors _____ 99
IV. Directiva per iterar col·leccions _____ 99
4.5.11. Diari de desenvolupament _____ 100
4.5.11.1. Escaneig d'anotacions _____ 100
4.5.11.2. Detecció de genèrics _____ 101
4.6. Aplicació de prova _____ 103
4.6.1. L'aplicació: base de dades cinematogràfica _____ 103
4.6.2. Dissenyant el sistema _____ 103
4.6.3. Directiva de bucle i mostrar dades iterades _____ 104
4.6.4. Directiva de llistes desplegables sense atribut de controlador vinculat 104
4.6.5. Reutilització de validadors entre accions de diferents controladors 105
4.6.6. Problemes amb tipus de dada complexos _____ 105
4.6.7. Control d'accés mitjançant validadors _____ 105
4.6.8. Resum de l'experiència _____ 106
4.7. Millores i ampliacions _____ 107
4.7.1. Suport per a llistes de dades complexes _____ 107
4.7.2. Multiidioma _____ 107
4.7.3. Múltiples tecnologies de vista _____ 107
4.7.4. Major flexibilitat dels objectes Java _____ 108
4.7.5. Variacions del patró FrontController _____ 108
4.7.6. Validadors amb paràmetres propis _____ 108
4.7.7. Eina CASE _____ 108
4.7.8. Tractament de recursos no relatius al framework _____ 109
4.8. Conclusions _____ 110
5. Glossari _____ 113
6. Bibliografia _____ 114
7. Annex I. Descàrrega de llibreries externes _____ 117
7.1. SpringExample (cas pràctic amb Spring MVC) _____ 117
7.2. StrutsExample (cas pràctic amb Struts 2) _____ 117
7.3. FacesExample (cas pràctic amb JSF 2.0) _____ 118
7.4. FFramework _____ 118

3. Índex de figures

Planificació detallada. _____ 15
Diagrama de Gantt de la planificació (I) . _____ 16
Diagrama de Gantt de la planificació (II). _____ 17
Diagrama de seqüència del patró FrontController._____ 23
Motor de Servlets executant una arquitectura Front Controller._____ 25
Ubicació i responsabilitats del WebApplicationContext de Spring MVC. _____ 26
Llibreries del cas pràctic amb Spring MVC. _____ 31
Diagrama d'activitat de Struts 2. _____ 39
Llibreries del cas pràctic amb Struts 2. _____ 44
Cicle de vida a Java Server Faces._____ 51
Camí de les dades a Java Server Faces._____ 51
Llibreries del cas pràctic amb Java Server Faces. _____ 55
Diagrama de classes de FFramework. _____ 80
Diagrama de seqüència de inicialització de FFramework (I)._____ 83
Diagrama de seqüència de inicialització de FFramework (II). _____ 84
Diagrama d'activitat del processament d'una petició a FFramework. _____ 86

4. Cos de la memòria

4.1. Introducció

4.1.1. Justificació i context

Els que ens dediquem al desenvolupament de software en els temps actuals ens podem considerar afortunats. De la mateixa manera que la història ens ha deixat èpoques daurades de la pintura, la literatura o el cinema, els darrers anys han suposat l'explosió i continua evolució del desenvolupament de programari.

En un breu període de temps, el que inicialment eren grans ordinadors de la mida d'una habitació i peces de programari codificades en targetes perforades, ha evolucionat fins a complexes arquitectures de la mida d'una ungla i vastos programes amb anys de desenvolupament que qualsevol usuari pot disposar en el seu ordinador personal. Aquestos avenços, entre d'altres, han apropat la informàtica a un públic més global, i com a conseqüència ha augmentat el nombre de persones interessades en aportar el seu propi esforç i desenvolupar el seu propi programari.

Podem considerar que un punt clau en aquesta cursa contra el rellotge ha estat l'explosió dels continguts distribuïts, observant com a punt àlgid d'aquest fenomen la construcció d'una xarxa universal a la que tothom amb un dispositiu adequat i accés telefònic hi pot accedir. Cada cop que Internet aterrava a un nou país, la quantitat de població que vol consultar els seus continguts augmenta fins a xifres inimaginables fa tan sols un grapat de dècades.

Aquest volum de tràfic, juntament al cada cop més gran volum d'informació que la xarxa alberga, ens porta a la necessitat d'un software i hardware cada cop més refinat, preparat i optimitzat per a operacions exigents tan en complexitat com en quantitat. El que l'any 1990 era suficient per servir continguts, roman obsolet tant sols 3 o 4 anys més endavant. Les instal·lacions dissenyades per emmagatzemar informació es veuen obligades a una continua expansió de la qual encara no coneixem fins on pot arribar.

Trobem per tant dues variables en franc augment. Per una banda, una comunitat de gent interessada en programar cada cop més gran i fragmentada. Per un altra, uns requisits del software cada cop més exigents. El món creix a una velocitat vertiginosa, i el software ha d'acompanyar aquest ritme.

El que inicialment va irrompre com una tècnica més sembla ser ja un requisit gairebé indispensable si no es vol perdre el ritme: parlem de la reutilització. Detectar fases comuns i establir un mecanisme que ens permeti no haver de repetir les nostres passes per a cada desenvolupament proporciona una clara avantatge contra el rellotge. Per altra banda, saber que el resultat del nostre esforç no serà flor d'un dia i que podrà ser aprofitat en futures situacions ens obliga a

mantenir una ment oberta i preparada per a moltes variants del mateix repte, i no només aquella concreta que afrontem en un moment determinat.

Dit d'una altra manera, la reutilització ens permet reduir els costos de desenvolupament, i ens permet que aquest desenvolupament sigui molt més flexible i preparat per al gran nombre de diferents reptes que es poden presentar.

Una de les tecnologies que millor ha entès l'escenari que es presenta ha estat Java. Aquest llenguatge gaudeix d'una popularitat tal que la comunitat de desenvolupadors que aporten la seva feina i permeten el seu aprofitament és d'una mida que aconsegueix que es tracti d'un dels llenguatges més emprats a tot el món durant molts d'anys consecutius. Es tracta d'un cercle viciós en el que, gràcies a la seva fama, més i més gent ho acull com a llenguatge per els seus desenvolupaments, i en conseqüència augmenta la disponibilitat d'eines que la propera generació de desenvolupadors pot aprofitar.

Dins del llenguatge Java, l'especificació J2EE augmenta més encara aquesta percepció de constant evolució i millora definint esquemes i eines que s'adapten a infinitud de situacions. D'una sèrie de restriccions imposades per l'especificació, han nascut un bon nombre de llibreries i marcs de treball que permeten reduir la càrrega del desenvolupament eliminant fases o components redundants. Si la gran majoria de projectes J2EE apliquen els mateixos principis, sembla obvi que no cal implementar cada cop una nova estructura de base que compleixi aquestos principis.

Aquest és l'argument amb el que apareixen els frameworks. Conjunts de programari que ofereixen un marc inicial sobre el que desenvolupar una nova aplicació, alliberant als recursos de tasques que ja han estat avaluades i solucionades amb anterioritat. Aspectes de l'arquitectura com la cohesió i el baix acoblament, l'escalabilitat, o la implementació d'un patró de disseny venen ja resolts per l'ús d'un framework i tot l'esforç del desenvolupador pot encarar-se al problema concret que es vol resoldre.

Aquest mateix motiu és la raó de ser d'aquest projecte. L'èxit i evolució de Java i J2EE es basa en el principi de reciprocitat pel qual desenvolupadors que s'han beneficiat de recursos disponibles, desenvolupen els seus propis recursos. Aquest projecte marca el punt d'inflexió en que l'autor passa d'aprofitar feina de tercers a aportar la seva pròpia sortida.

Amb l'esforç de síntesi que suposa estudiar les opcions existents del mercat, i l'esforç que representa idear i implementar una opció de collita pròpia, es pretén compensar en la mesura possible tot allò que el llenguatge, l'estàndard i la comunitat han aportat a l'autor del projecte durant molt de temps.

4.1.2. Objectius

La intenció final d'aquest projecte és la d'aprofundir en el món J2EE en general i dels framework de presentació en concret, tant en lo que es refereix a adquirir coneixements com a fer aportacions pròpies envers aquesta tecnologia.

En una primera fase, estudiarem l'estat actual del món dels marcs de treball de presentació implementats sobre J2EE i farem una mirada més intensiva sobre aquelles propostes que gaudeixen d'una major popularitat i acceptació entre la comunitat de desenvolupadors. Fruit d'aquesta experiència, que combina una part teòrica (estudi de les capacitats de l'eina) i una altra de pràctica (realització de casos pràctics), coneixerem de primera ma quines són aquelles funcionalitats i serveis que sembla indispensable oferir per a ser considerat un bon framework J2EE.

Per altra banda, conèixer en major grau algunes de les eines amb més notorietat ens permetrà detectar els arguments a favor i en contra de cadascuna d'aquestes, donant prioritat a l'enfocament d'usabilitat, és a dir, a quina dificultat comporta pel desenvolupador la seva decisió d'utilitzar un framework o un altre.

Enllestit l'estudi previ i obtingudes unes conclusions, passem a una segona fase on en base a tota aquesta entrada de informació fem un balanç d'allò que hem observat i ho traduïm en la nostra pròpia proposta de framework de presentació J2EE.

La sortida d'aquesta fase serà una primera versió preliminar d'un marc de treball que acollirà aquelles funcionalitats que més atractives o útils ens han semblat de la fase anterior, dissenyades i implementades segons el nostre propi criteri. Aquesta eina de sortida vindrà acompanyada d'una documentació que permeti a un desenvolupador conèixer i emprar la nostra eina de la mateixa que nosaltres hem estudiat les existents.

Com a petit annex en forma de tercera fase, ens posarem en la pell d'un desenvolupador extern que decideix donar una oportunitat al nostre framework com a base per implementar el seu propi projecte. Amb aquesta acció podrem detectar aquells punts que no hem detectat o be no hem sabut definir de la millor forma possible, permetent així el inici d'una hipotètica nova iteració d'evolució i millora de la nostra eina.

En resum, aquest projecte pretén ser una feina de formació, divulgació i aportació de l'àmbit dels frameworks de presentació sota J2EE.

4.1.3. Enfocament i metodologia

Per a la consecució dels objectius establerts ens hem decantat per un itinerari de fites, en les que la feina necessària queda desglossada en petites unitats amb prou entitat per si mateixes i que permetin assolir un avenç tangible del projecte en períodes breus de temps.

Donat que el projecte esdevé la responsabilitat d'una única persona, durant el disseny d'aquestes fites es cerca un enfocament de cadena, en la que cada tasca pot fer servir la sortida produïda per les tasques anteriors i aporta uns resultats que puguin esdevenir necessaris per a tasques posteriors. Es reserva la possibilitat de dissenyar tasques que es poden realitzar en paral·lel (especialment les referides a documentació), tot i que no serà una característica prioritària.

En tot moment s'ha pretès donar al projecte un enfocament eminentment pràctic. Tractar la teoria sempre que sigui necessari, però donar prioritat a un punt de vista més proper a l'experiència final d'un desenvolupador o usuari d'una aplicació. En aquest aspecte és l'objectiu aportar com a resultat de la nostra feina material inèdit, i no només un resum o recopilació de coneixement ja disponible en altres fonts d'informació.

Una situació prou habitual en projectes que impliquen cert desenvolupament com aquest és donar prioritat a la producció posposant les tasques de documentació fins a l'infinit. La conseqüència d'aquesta política és un impacte negatiu en la qualitat d'aquesta documentació, ja que en fer-se amb setmanes o fins i tot mesos de diferència respecte allò que es documenta, el coneixement no es troba tan fresc i expressar-ho sobre el paper no resulta tan fluït com si es realitza immediatament. Per aquest motiu el contingut de les nostres fites inclourà tant la seva vessant de creació com de documentació, procurant sempre mantenir la documentació el més actualitzada possible respecte a la feina realitzada.

Establertes aquestes línies principals, obtenim un jerarquia de fites que detallem en el posterior apartat de planificació del projecte.

4.1.4. Planificació

L'escenari acadèmic en el que s'inclou la realització del projecte suposa un fort condicionant en el moment de la seva planificació, tant en la llargària d'aquest com en certes fites temporals que han de suposar la generació d'una sortida amb prou entitat com per suposar un lliurament.

Un cop obtinguda la llista de tasques en que es composa tot el treball planificat, aquesta s'ajusta amb el calendari proposat i es fa una previsió de l'ordre i temps

requerit per a cada tasca per tal d'afavorir la consecució de certs objectius intermedis coincidint amb les dates establertes.

En quant a l'estudi d'eines existent, es decideix reservar el mateix temps de dedicació per a cadascuna, sense donar prioritat a cap en particular. Aquesta pretensió inicial queda lleugerament modificada en prescindir d'un cas pràctic pel cas de Grails, per lo qual el temps requerit per l'estudi d'aquest darrer és inferior a la resta.

En quant al desenvolupament d'un framework propi, es decideix establir una regla aproximada de 50%-50%, on la meitat del temps pertany a la implementació pròpiament dita i l'altre 50% a tasques anteriors i posteriors com són l'anàlisi i disseny de l'eina, i la fase de proves tant unitàries com de integració.

La generació d'una documentació que acompanya totes les fases del projecte, que porten com a resultat final aquest document de memòria, constitueixen aproximadament un 20% del temps total del projecte. Tot i mostrar-se visualment com una tasca independent que té lloc en el darrer moment, en realitat es tracta d'una feina acumulativa i realitzada en paral·lel a la resta de fites del projecte.

El resultat final és una planificació de 203 hores de dedicació (40 hores per a la memòria del projecte) i una agrupació en fites/lliurament com la que segueix:

PAC1: 5 d'octubre de 2011

- Concepció del projecte: establiment d'objectius, planificació inicial.
- Realització del pla de treball.

PAC2: 10 de novembre de 2011

- Estudi, cas pràctic i conclusions de cadascun dels frameworks existents:
    - Spring MVC
    - Struts
    - Java Server Faces
    - Grails
- Documentació de l'estudi esmentat.
- Redacció de certs apartats introductoris de teoria general (patrons, definicions, etc.).

PAC3: 19 de desembre de 2011

- Anàlisi, disseny, implementació i proves de F Framework.
- Documentació de la proposta de framework.

Lliurament final: 16 de gener de 2012

- Realització i documentació del cas pràctic amb F Framework.

- Redacció de parts comunes de la memòria (introducció, conclusions, etc.).
- Maquetació final de la memòria del projecte.

A continuació es mostra tant una taula detallada amb duració i fites de cada tasca, com el diagrama de Gantt resultant.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesora
1	Concepció del projecte	4 horas	jue 29/09/11	vie 30/09/11	
2	**Estudi d'eines existents**	**32,5 días**	**sáb 01/10/11**	**vie 04/11/11**	**1**
3	**Struts**	**9,5 días**	**sáb 01/10/11**	**mar 11/10/11**	
4	Estudi de l'eina	6 horas	sáb 01/10/11	mar 04/10/11	
5	Configuració de l'entorn	2 horas	mar 04/10/11	mié 05/10/11	4
6	Cas pràctic	9 horas	mié 05/10/11	lun 10/10/11	5
7	Anàlisi i conclusions	2 horas	lun 10/10/11	mar 11/10/11	6
8	**Spring MVC**	**9,5 días**	**mar 11/10/11**	**vie 21/10/11**	**3**
9	Estudi de l'eina	6 horas	mar 11/10/11	vie 14/10/11	
10	Configuració de l'entorn	2 horas	vie 14/10/11	sáb 15/10/11	9
11	Cas pràctic	9 horas	sáb 15/10/11	jue 20/10/11	10
12	Anàlisi i conclusions	2 horas	jue 20/10/11	vie 21/10/11	11
13	**Java Server Faces**	**9,5 días**	**vie 21/10/11**	**lun 31/10/11**	**8**
14	Estudi de l'eina	6 horas	vie 21/10/11	lun 24/10/11	
15	Configuració de l'entorn	2 horas	mar 25/10/11	mar 25/10/11	14
16	Cas pràctic	9 horas	mié 26/10/11	sáb 29/10/11	15
17	Anàlisi i conclusions	2 horas	lun 31/10/11	lun 31/10/11	16
18	**Grails**	**4 días**	**mar 01/11/11**	**vie 04/11/11**	**13**
19	Estudi de l'eina	6 horas	mar 01/11/11	jue 03/11/11	
20	Anàlisi i conclusions	2 horas	vie 04/11/11	vie 04/11/11	19
21	**Finalització d'estudi d'eines**	**0 días**	**vie 04/11/11**	**vie 04/11/11**	**2FF**
22	**Framework propi**	**38 días**	**sáb 05/11/11**	**jue 15/12/11**	**2**
23	Anàlisi de requeriments	14 horas	sáb 05/11/11	sáb 12/11/11	
24	Disseny	14 horas	sáb 12/11/11	sáb 19/11/11	23
25	Implementació	40 horas	sáb 19/11/11	sáb 10/12/11	24
26	Proves	8 horas	lun 12/12/11	jue 15/12/11	25
27	**Finalització framework propi**	**0 días**	**jue 15/12/11**	**jue 15/12/11**	**22FF**
28	Aplicació de proves	14 horas	vie 16/12/11	vie 23/12/11	22
29	Modificacions de l'eina	4 horas	vie 16/12/11	sáb 17/12/11	27
30	**Memòria del projecte**	**40 horas**	**vie 23/12/11**	**vie 13/01/12**	**28**
31	**Finalització del projecte**	**0 días**	**vie 13/01/12**	**vie 13/01/12**	**30FF**

Figura 1. Planificació detallada.

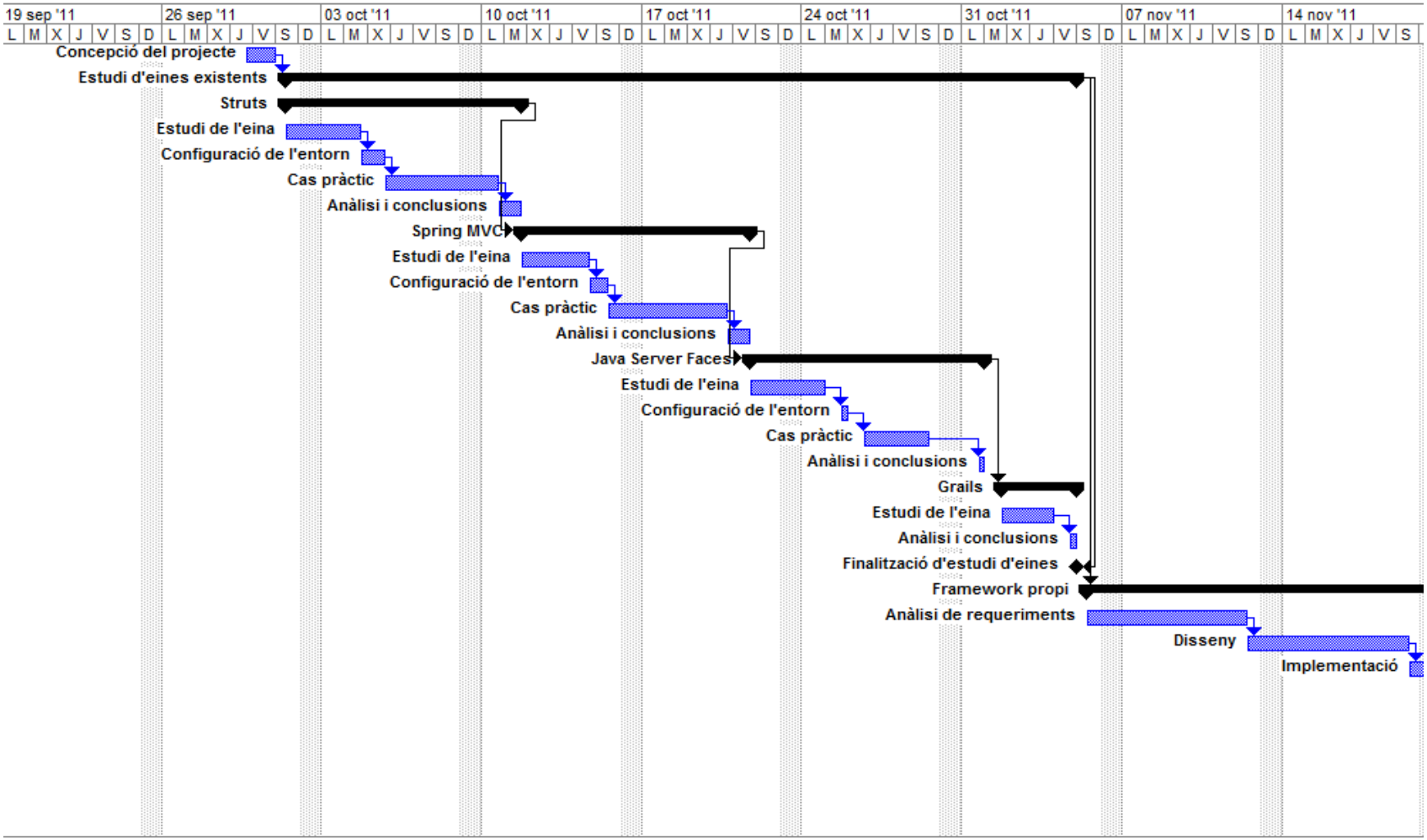

Figura 2. Diagrama de Gantt de la planificació (I) .

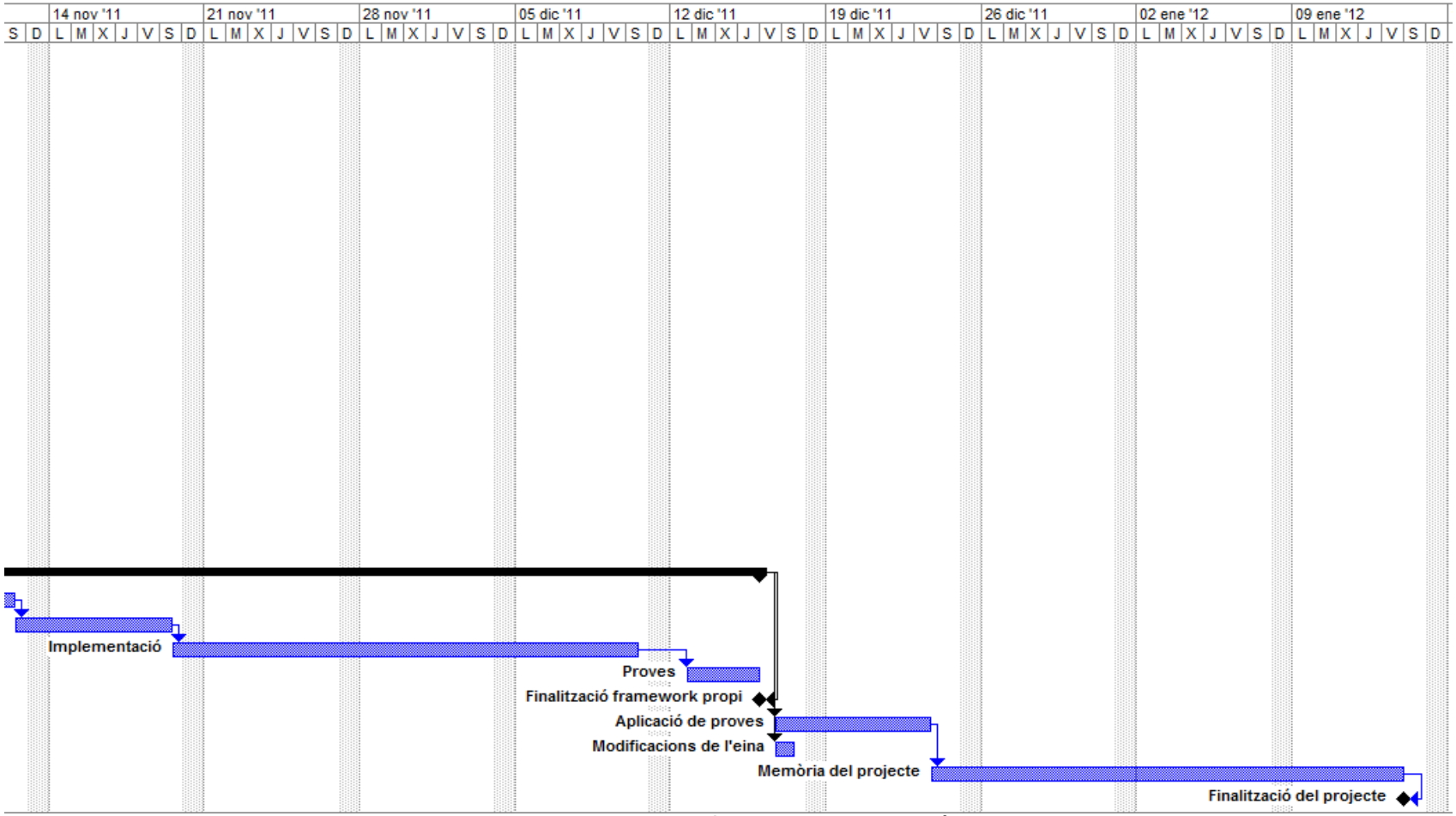

Figura 3. Diagrama de Gantt de la planificació (II).

4.1.5. Productes obtinguts

La realització d'aquest projecte produeix, a banda d'aquest document de memòria, una sèrie de programari fruit de les diferents fases en les que es divideix. Podem agrupar aquestes peces de software en els següents apartats:

4.1.5.1. Casos pràctics d'eines existents

Col·lecció de projectes web J2EE senzills, tots ells mantinguts amb l'IDE NetBeans7, amb petites demostracions pràctiques de cadascun dels frameworks de presentació J2EE existents al mercat, a saber: Spring, Struts y Faces. Grails queda fora en aquest cas per motius que es detallen en el seu apartat.

Tots aquestos projectes resulten en una sèrie de planes web que permeten mantenir els continguts d'una agenda de contactes. Per a cada projecte es disposa d'un directori amb el seu nom preparat per ser importat des del propi IDE.

Donat que es tracta d'una matèria fora de l'àmbit d'aquest projecte, la capa de persistència queda reservada per un conjunt de classes Java que s'inclouen a cada projecte. Aquest programari es troba en el projecte amb nom "DemoAgenda".

4.1.5.2. F Framework

Com a principal aportació del projecte en general i la seva vessant més pràctica en concret, s'ofereix la nostra pròpia proposta de marc de presentació J2EE.

Empaquetat en forma de fitxer comprimit es disposa de tot lo necessari per entendre, utilitzar i fins i tot manipular el nostre framework, a saber:

- Conjunt de llibreries externes necessàries per a la compilació i/o execució del framework (directori /extlib).
- Arxiu .jar amb els binaris del framework, preparats per ser desplegats a un servidor d'aplicacions que compleixi l'especificació J2EE (directori /build).
- Documentació JavaDoc per a tots els paquets i classes que composen el codi font del framework, amb descripcions tant del seu propòsit general com dels mètodes individuals.
- Codi font del framework, conservant l'estructura de paquets original i incloent la definició de classes, interfícies i anotacions (directori /src).

4.1.5.3. Cas pràctic de F Framework

Com a punt final i acompanyant a la darrera etapa del projecte, s'adjunta l'aplicació web desenvolupada per a testejar el nostre framework i fer una detecció darrera d'errors i possibles millores. L'aplicació es troba a un directori "/FMovieDatabase" preparat per ser importat amb NetBeans7.

4.1.6. Breu descripció de la resta de capítols

4.1.6.1. Components J2EE

Breu enumeració dels principals components que defineix l'estàndard J2EE, necessaris com a coneixement de base per abordar l'anàlisi dels frameworks desenvolupats sobre aquesta tecnologia.

4.1.6.2. El patró FrontController

Definició i desenvolupament del patró de disseny més estès per a resoldre el problema de la captura i processament de peticions d'usuari.

4.1.6.3. Estudi d'eines existents

Estudi de quatre dels frameworks J2EE amb més popularitat i acceptació dins del sector del desenvolupament de software. Les opcions escollides són Spring MVC, Struts 2, JavaServer Faces i Grails.

Per a cadascuna, l'estudi es divideix en una fase introductòria que resumeix els principis i la història de cada alternativa, continua amb un resum de l'arquitectura i les diferents característiques que engloba cada framework, segueix amb el desenvolupament d'un petit cas pràctic que permet conèixer en primera persona com és l'experiència de utilitzar l'eina, i es tanca l'anàlisi amb un apartat d'observacions i conclusions obtingudes.

Finalment es presenta un apartat de conclusions globals que pretén obtenir una visió comparativa de les quatre eines estudiades i una primera aproximació a aquelles característiques que es volen heretar per al nostre framework.

4.1.6.4. Desenvolupament d'un framework: FFramework

Documentació relativa a la nova proposta de marc de presentació J2EE implementada pel responsable del projecte.

El capítol s'inicia amb dos apartats introductoris: un amb un resum d'aquelles funcionalitats o principis bàsics que es desitja inculcar en el disseny de l'eina, i

un altre que enumera aquelles tecnologies que esdevenen clau per entendre la implementació.

Tot seguit es passa a documentar la fase d'anàlisi amb la relació de casos d'ús contemplats per al sistema, expressant l'itinerari d'esdeveniments en que es composa cada funcionalitat. En un punt intermedi entre l'anàlisi i el disseny, s'enumeren les classes principals que composen l'eina, amb una breu descripció de la finalitat de cadascuna.

Posteriorment s'inclouen un seguit de diagrames que faciliten la comprensió de certes fases clau del comportament del framework, com són la inicialització i el processat d'una petició.

En termes més tècnics i ja ficats en plena implementació, s'enumeren les anotacions Java que serveixen de definició pels diferents components de l'eina, així com la llista d'interfícies ofertes.

S'inclou un petit manual d'ús que serveix com a guia d'inici ràpid per aquells desenvolupadors que decideixen posar a prova els serveis que ofereix el marc de presentació, seguint un guió cronològic de les passes a seguir.

Per acabar, es fa un resum dels comportaments per defecte que adopta el framework en cas que el desenvolupador no els vulgui sobreescriure, i s'exposen aquelles etapes del desenvolupament del framework que han esdevingut més dificultoses a mode de diari de desenvolupament.

4.1.6.5. Aplicació de prova

S'exposa a mode de diari l'experiència resultant de desenvolupar una petita aplicació de gestió utilitzant FFramework com a programari de base. S'enumeren aquelles parts crítiques del procés indicant per a cada cas quines conclusions s'extrauen i quines limitacions o modificacions suposen per a l'eina.

4.2. Components J2EE

Aquest projecte tracta de l'estudi i desenvolupament de frameworks de presentació sota la tecnologia J2EE. Com a requisit per entendre aquesta tecnologia, sembla convenient fer un breu repàs als principals components estàndard inclosos a l'especificació oficial de Sun indispensables per entendre una eina desenvolupada amb aquesta tecnologia.

4.2.1. Servlets

Podríem considerar els servlets la part central i més important d'una aplicació J2EE. Aquestes peces de codi ens permeten habilitar accessos amb entrada i sortida dinàmica de dades, és a dir, servir continguts que varien en funció de la petició de l'usuari.

Tot i que permeten rebre i emetre continguts amb diversos protocols, el més habitual és fer-ho sota el protocol HTTP. Per aquesta raó, J2EE ofereix la classe abstracta HttpServlet, que defineix mètodes com doGet o doPost que reben l'entrada (HttpServletRequest) i sortida (HttpServletResponse) de la petició HTTP, permetent al desenvolupador llegir informació de la petició i generar una resposta com ara escriure un HTML o fer una redirecció.

4.2.2. Filters

Els filtres permeten establir interceptors de les peticions abans i després que aquestes arribin al component encarregat de processar-les (normalment un Servlet). Donada una ruta sobre la qual es realitza la petició, poden coexistir diversos filtres encadenats per finalitats diverses, com ara fer comprovacions sobre l'estat de la petició o bé aplicar certes transformacions sobre aquesta abans de deixar-la passar pel seu tractament.

Els filtres són especialment útils per tasques que s'han d'executar transversalment per a tota l'aplicació, com ara comprovacions de seguretat i autenticació o sistemes de logging que mantenen un diari amb tota l'activitat del sistema.

4.2.3. Session

Els objectes Session són la representació J2EE de l'àmbit de sessió de l'usuari que realitza la petició. D'aquesta manera s'habilita un espai que sobreviu al limitat cicle de vida de les peticions HTTP i, per tant, ens permet crear aplicacions amb estat.

Aquesta sessió d'usuari es manté al llarg de les peticions per mitjà d'uns identificadors que poden ser propagats, entre d'altres, com a un paràmetre més de la petició o a través de cookies del navegador web.

4.2.4. Context

L'objecte context representa l'àmbit d'aplicació del nostre sistema. Habilita un espai que sobreviu al cicle de vida de les peticions i les sessions d'usuari, i per tant ens permeten disposar de informació que perdura durant tota la existència de la nostra aplicació.

4.2.5. Listeners

Els listeners (de l'anglès "listen", escoltar) ens permeten detectar i actuar en conseqüència davant certs esdeveniments que es produeixen dins del context d'una aplicació J2EE. Distingim entre diferents listeners segons l'objecte que protagonitza l'esdeveniment. Així, per exemple, un HttpSessionListener ens permet fer un tractament especial quan es crea o destrueix una sessió d'usuari, i un ServletContextListener ens permet fer certes tasques d'inicialització a alliberació de recursos quan engeguem per primer cop o destruïm la infraestructura necessària per executar un Servlet.

4.2.6. Java Server Pages

Les JSP són el component proposat per J2EE per a la generació de vistes. Es tracta d'arxius amb una gran similitud amb planes HTML, però amb la possibilitat d'incrustar codi Java a través del que es denomina "scriptlets".

En el moment "d'executar" una JSP, el que fa realment el servidor és una traducció per generar un HttpServlet com els que hem vist anteriorment. La majoria de servidors d'aplicacions ens permeten veure el codi d'aquest servlet generat per entendre millor com s'acaben processant les nostres plantilles amb JSP.

4.2.7. JSP TagLIbs

Les llibreries d'etiquetes (TagLibs) són una ampliació de la potència es les pàgines JSP que apropen encara més els mons de Java i HTML. Permeten definir "etiquetes" amb una sintaxi totalment integrada amb el llenguatge HTML però que provoquen l'execució del seu comportament, implementat en Java, en temps de renderització de la pàgina.

L'ús de JSP TagLibs permet fer encara més intensa la separació entre lògica i presentació, ja que amb elles és possible alliberar totalment les planes de vista de codi de programació.

4.3. El patró FrontController

Els marcs de treball de presentació amb els que treballarem requereixen l'ús de patrons de disseny per tal de donar solució de la forma més senzilla i eficient possible a certs problemes propis d'una aplicació J2EE basada en el paradigma del model vista controlador (MVC).

Entre totes les solucions escollides per cada framework, un denominador comú acostuma a ser seguir l'estratègia del FrontController. Sota aquest principi de disseny s'habilita un punt central per a encaminar les dades, encarregat de rebre les peticions de l'usuari i delegant el seu processat a aquells components que prèviament hagin indicat que són capaços de tractar-les.

El patró FrontController permet als frameworks ser molt poc invasius en quant a la configuració purament J2EE. L'escenari més habitual es que, per configurar un framework, només sigui necessari desplegar en el descriptor de l'aplicació un sol Servlet o Filter (que serà el FrontController). La resta de paràmetres necessaris per instal·lar cada servei queden delegats a la solució que escull la pròpia eina (fitxers de configuració propis, convenció de noms, etc.).

En el cas dels marcs de treball de presentació, el patró FrontController veu encara més ampliada la seva utilitat, ja que no es limita a fer tasques de redirecció de la informació. Aquesta peça de software també s'encarrega de mantenir tot el workflow necessari per processar la petició, invocant en l'ordre corresponent els diferents components que intervenen durant el procés.

La següent figura mostra un diagrama de seqüència típic d'aplicacions que segueixen el patró de FrontController pel tractament de les peticions rebudes:

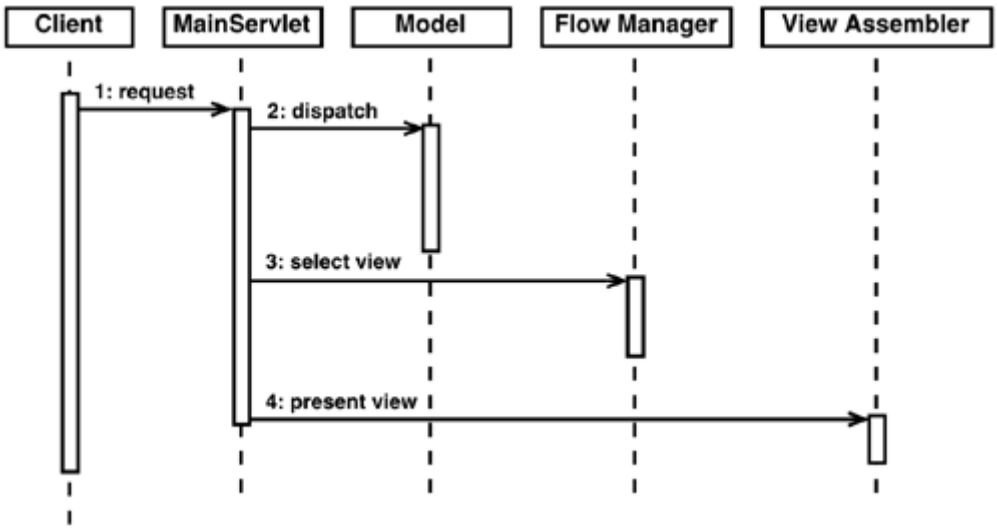

Figura 4. Diagrama de seqüència del patró FrontController.

4.4. Estudi d'eines existents

A continuació s'exposa la sortida de l'estudi previ d'una sèrie de marcs de treball de presentació J2EE ja existents al mercat. La selecció de quins frameworks estudiar és el fruit de combinar certs criteris, essent els principals la seva popularitat i/o acceptació i la inclusió de certes característiques o principis que els fan interessants per a l'extracció d'idees de cara a la nostra pròpia proposta.

L'estudi cerca un equilibri entre els detalls tècnics i la usabilitat de l'eina. Sense oblidar la comprensió de l'arquitectura interna de cada framework, dedicarem igualment els nostres esforços a conèixer quines són les sensacions que cada eina aporta al desenvolupador que decideix treballar amb elles. D'aquesta manera no ens limitarem a conèixer quines alternatives es troben millor construïdes o definides, si no que també ens proposarem saber quines ofereixen una millor experiència al seu usuari final, que en aquest cas és el propi desenvolupador d'aplicacions.

Per aquesta raó, la major parts dels casos no es limitaran a fer una enumeració teòrica de les bondats de cada frameworks, si no que aniran un pas més enllà i simularan la posada en marxa d'una petita aplicació d'exemple amb cadascuna de les eines. Aquesta aplicació consisteix en una agenda de contactes que ens permeti consultar i modificar els seus registres a través d'un navegador web.

4.4.1. Spring Web MVC Framework

4.4.1.1. Identificació i història

Spring és un dels frameworks J2EE més populars i amb major presència i acceptació entre la comunitat de desenvolupadors. La seva història es remunta a 2002 quan el seu autor, Rod Johnson, decideix acompanyar amb casos pràctics el seu llibre Expert One-on-One J2EE Design and Development, l'esperit del qual és agrupar consells i bones pràctiques que millorin l'experiència de desenvolupar aplicacions sota els requisits de J2EE.

Arrel de l'èxit del seu treball, l'eina que l'acompanya comença a créixer gràcies a la cada cop més nombrosa comunitat que el recolza, fins el punt d'esdevenir avui dia una de les eines més consolidades i amb major nombre de serveis oferts dins del seu sector.

La part dedicada a implementar el model MVC neix a posteriori sota el nom de Spring Web MVC Framework com a alternativa a Jakarta Struts, el framework més estès en aquell moment i que els desenvolupadors de Spring MVC consideren una eina amb moltes carències.

Actualment, Spring Framework en general i Spring MVC en concret han patit diverses evolucions en forma de noves versions, essent avui dia la 3.x la branca estable més recent. En el camí ha sofert canvis que de forma més o

menys notòria han provocat certa polèmica als cercles del desenvolupament J2EE, com ara l'omissió de certes bones pràctiques i regles proposades per la pròpia Sun Microsystems (responsable de Java, ara en possessió d'Oracle). Els autors de Spring sempre han defensat que la seva feina neix i evoluciona amb una clara intenció d'alternativa, i això li permet no ajustar-se sempre a allò que la comunitat i altres organismes exigeixen.

4.4.1.2. Estil i arquitectura

Spring MVC és un framework de presentació orientat a petició, és a dir, basat en els esdeveniments que produeix el navegador en forma de HttpServletRequest. Es basa, per tant, en els servlets definits a l'especificació J2EE.

El funcionament gira al voltant d'un únic controlador que exerceix el rol d'intermediari entre vistes i model, amb ajut de petits afegits en forma de regles de mapping i adaptadors on el desenvolupador implementa les seves accions. Parteix, per tant, d'una clàssica arquitectura que segueix el patró de FrontController, on aquesta pedra angular que dirigeix el tràfic d'informació rep el nom de Dispatcher Servlet.

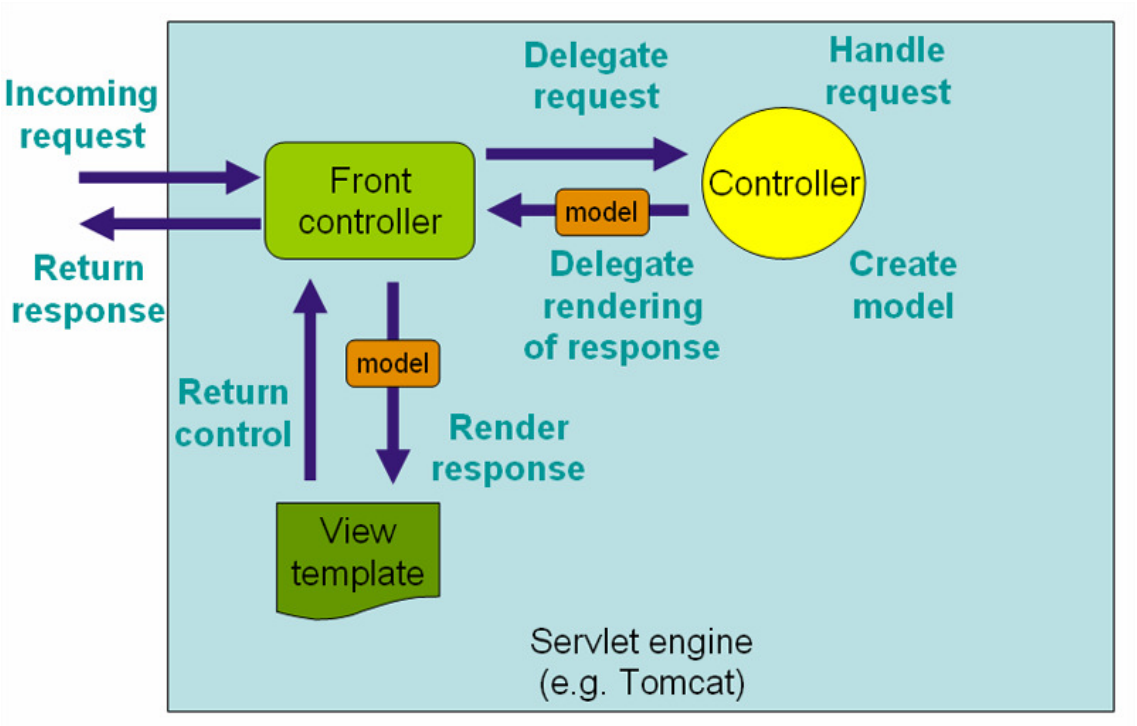

Figura 5. Motor de Servlets executant una arquitectura Front Controller.

4.4.1.3. Components i Beans

Com hem introduït, Spring MVC permet ampliar la seva base amb comportaments propis de cada aplicació gràcies a un sistema de petits afegits de codi que s'activen en fases clau del procés de tractament i resolució de cada

petició de l'usuari. El contenidor on queden englobats tots aquests dispositius de software rep el nom de WebApplicationContext, i la seva concepció transcendeix més enllà del propi framework, ja que ve heretat d'altres capes i àmbits d'actuació de Spring Framework. Val a dir que seguint la seva política podem definir en petits fragments de codi els nostres controladors, gestors de vistes o interceptors que permeten fer un processat anterior i posterior de la informació, entre d'altres exemples. Alguns dels tipus d'afegit més habituals s'enumeren a continuació.

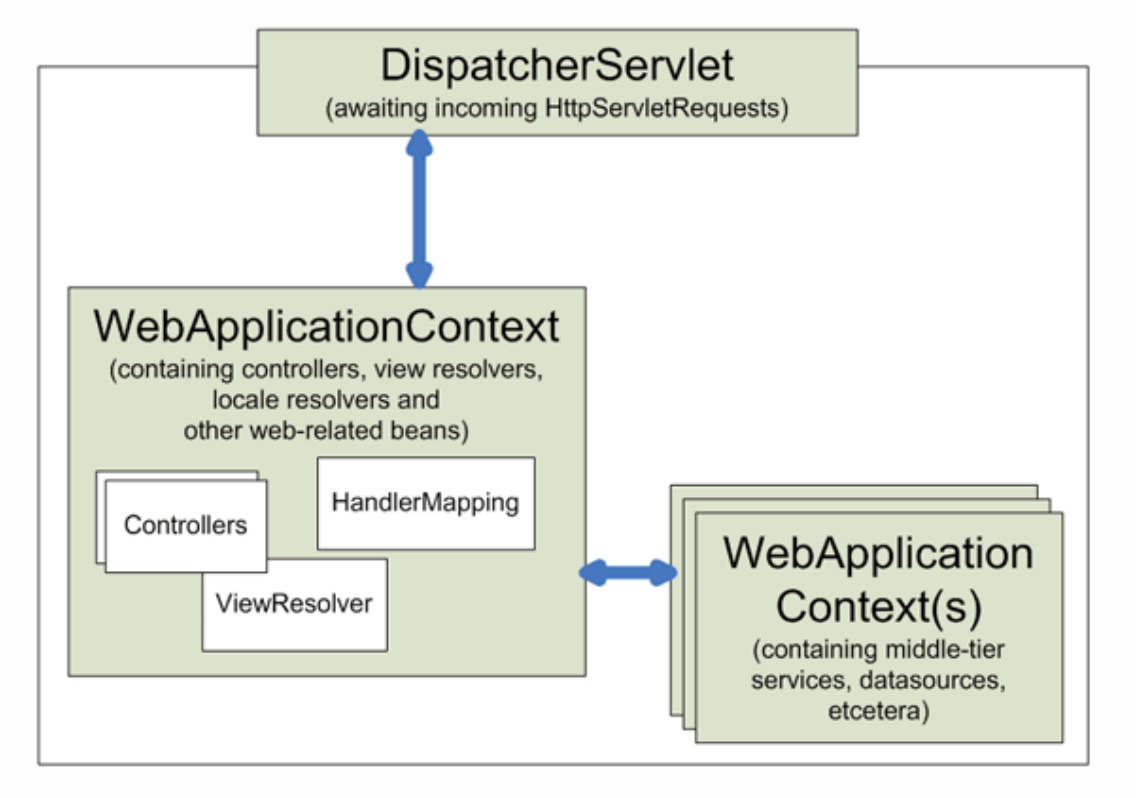

Figura 6. Ubicació i responsabilitats del WebApplicationContext de Spring MVC.

I. Controller

Es tracta d'un dels components més bàsics i alhora més transcendents del sistema. Són aquestes les peces de codi on queden definits i mapejats els diferents punts d'entrada de l'aplicació web, coneguts dins del llenguatge Spring com a RequestMapping.

Cadascun dels RequestMapping pertanyents a un controlador poden tenir la seva pròpia configuració, referint-nos a aspectes com el tipus de petició HTTP que atenen, els seus paràmetres d'entrada i sortida o l'estratègia definida per resoldre la vista encarregada de generar una resposta (delegació, escriptura directa, redirecció, etc.).

La definició i obtenció de paràmetres d'entrada per a cada acció ofereix una àmplia varietat d'opcions, des de tipus bàsics sense cap mena de control, fins a complexos contenidors d'objectes que prèviament han passat per un procés de

validació de dades. L'elecció d'una estratègia o altre resulta totalment transparent al codi de les accions, que es limiten a rebre els valors enviats per l'usuari en forma de paràmetres d'entrada a la signatura del mètode.

La generació de la sortida, per altre banda, manté el mateix esperit de contemplar gairebé qualsevol possibilitat que desitgi el desenvolupador. Una acció pot invocar una vista en base a un simple identificador de text, havent complimentat abans tota una sèrie de dades de resposta gràcies a l'ús de mapes nadius de Java (classe java.util.Map) o be Plain Java Objects tan complexos com el desenvolupador desitgi. O bé pot fer referència una vista en base a una URI del sistema. O fins i tot pot escriure per si mateix el contingut de la vista en resposta a la petició de l'usuari.

II. Handler Mapping

Sota la nomenclatura de HandlerMapping trobem la proposta de Spring MVC per al tractament anterior i posterior de la informació. En aquestes peces de codi es defineixen comportaments a executar en base a diferents criteris de la petició, que poden anar des d'un camp de la capçalera HTTP fins a un valor concret dins d'un paràmetre de la Request.

Els HandlerMapping només "s'activaran" quan la petició compleixi les regles establertes.

III. View Resolver

Els encarregats de "resoldre vistes" fan realitat un dels grans atractius de Spring MVC: el suport per la generació de vistes utilitzant un ampli ventall de tecnologies.

Els ViewResolver s'encarreguen de, donat un identificador de vista, analitzar si aquest coincideix amb una vista implementada amb la tecnologia que ell sap gestionar. D'aquesta manera es permet que dins un mateix sistema puguin conviure diverses estratègies per a la generació de vistes, com clàssiques pàgines JSP, objectes JSON, documents XML o plantilles FreeMarker, entre d'altres.

Com a afegit, el propi Spring MVC ofereix implementacions per defecte per moltes de les tecnologies de generació de vistes més populars. D'aquesta manera, configurar l'estratègia de generació de vistes d'una aplicació esdevé una tasca molt més senzilla, amb l'afegit de poder optar per diferents tecnologies per a parts separades de l'aplicació.

IV. Handler Exception Resolvers

Aquestos components són un tipus especial de gestors que intercepten la informació. Com el seu nom deixa entreveure, en aquest cas permeten decidir

com ha de respondre el sistema quan es provoca una excepció, permetent distingir entre tot tipus d'errors contemplat.

De mateixa manera que succeeix amb els View Resolvers, el propi framework ofereix ja un gestor per defecte sota el nom DefaultHandlerExceptionResolver que tradueix certes excepcions pròpies de Spring per tal que l'usuari rebi un codi d'error HTTP conseqüent amb el que ha succeït.

4.4.1.4. Configuració

Coneguda la filosofia del framework i les parts implicades en una aplicació que ho fa servir, donem pas ara a exposar com cal configurar un sistema sota Spring MVC per tal que les diferents peces encaixin.

Des de la versió 2.5, Spring MVC ofereix dues clares alternatives per a la configuració del sistema: la primera, heretada de versions anteriors, es basa en fitxers de configuració XML seguint els esquemes definits per a especificar els diferents components que conformen el sistema. La segona, més innovadora i seguint una clara tendència iniciada amb la versió 5 de Java, permet la configuració en base a notacions incrustades al codi Java, el que formalment es coneix com Java Annotations.

No hi ha una recomanació oficial ni consens respecte a quin tipus de configuració emprar, si be els casos d'estudi més estesos acostumem a optar per una solució híbrida, conservant els fitxers de configuració XML per aquells aspectes més bàsics de l'estructura de l'aplicació, i aprofitant la potència de la injecció de dependències amb anotacions per aspectes més concrets.

Independentment de l'estratègia adoptada, cal començar per la part més bàsica. Per a indicar que la nostra aplicació utilitza un DispatcherServlet de Spring MVC, instal·lem aquest component a l'arxiu web.xml comú a totes les aplicacions J2EE.

És a les propietats inicials d'aquest servlet "mestre" on podem ja optar per una estratègia o un altre.

En cas de decidir emprar una solució de fitxers XML o be una solució híbrida, el DispatcherServlet cercarà per defecte un fitxer amb el nom <ServletName>-context.xml dins el Classpath, tot i que la ubicació i nom d'aquest document es pot configurar mitjançant la propietat contextConfigLocation.

Si optem per una solució híbrida, una de les primeres entrades d'aquest document XML de configuració propi de Spring MVC pot ser <context:component-scan>, que permet indicar el paquet arrel on iniciar un recorregut de recerca i processament de notacions.

I. Convencions per estalviar configuració

Com a afegit al capítol de configuració, l'equip de Spring MVC ha adoptat una sèrie de comportaments per defecte que permeten al desenvolupador, sempre que treballi segons una convenció de nomenclatura, estalviar gran part de la configuració necessària.

Aquestes convencions inclouen relacions automàtiques entre controladors i URI sota la qual aquestos queden instal·lats, construïda en base al nom de la pròpia classe que implementa el Controller. Per exemple, un controlador anomenat HelloWorldController per defecte escoltarà les peticions fetes a l'adreça /helloworld*.

Els comportaments automàtics també afecten en gran mesura a la generació de vistes, tant en lo que respecta a dades d'entrada de la vista com a la pròpia resolució d'aquesta. Per exemple, podem afegir a la resposta objectes del tipus Company o List<User> i automàticament els seu identificadors a la vista seran "company" i "userList", basats en el nom de la seva classe. En un exemple del segon cas, si un controlador no indica cap vista en el moment de tractar una petició rebuda a la URI /userList.html, el framework assumirà que l'identificador de la vista requerida és "userList".

4.4.1.5. Característiques fora d'estudi

Per una qüestió de recursos i temps disponibles i prioritats establertes, Spring MVC ofereix tota una sèrie de característiques que queden fora de l'abast d'aquest estudi. Fem una breu enumeració d'algunes de les més interessants.

I. Detecció de configuració regional

Suport per configurar dinàmicament el Locale que deu emprar la presentació, amb l'ajut d'un interceptor Locale Resolver que permet resoldre qüestions d'internacionalització, descobrint quin idioma ha de ser emprat segons la informació rebuda de la petició i el client que l'ha generat.

II. Suport per a temes

Complet suport per personalitzar l'estil de la presentació en base a un sistema de skins (temes), agrupant tots els recursos (resources) de cada tema disponible i permetent la commutació entre ells de forma senzilla. En el procés intervenen interceptors com el Theme Resolver que permet decidir en temps d'execució el tema a utilitzar, i una sèrie de directrius per importar el tema actiu a la vista.

III. Enviament d'arxius

Funcionalitat que permet l'enviament de peticions HTTP POST amb arxius adjunts gràcies al Multipart File Resolver, un interceptor que detecta les comandes d'aquest tipus i exerceix de traductor per tal que les dades d'entrada arribin correctament als controladors del sistema.

IV. Llibreria JSP TagLib

Conjunt de tags compatibles amb la generació de vistes JSP que permeten l'obtenció i manipulació de dades sense necessitat d'utilitzar scriptlets Java incrustats en la resposta a generar.

V. Arquitectura de portlets

Per proximitat al tipus de sistema que estudia aquest projecte, queda fora del seu abast la possibilitat d'implementar sota Spring MVC aplicacions basades en l'especificació de Portlets.

4.4.1.6. Cas pràctic

Per a familiaritzar-se amb el desenvolupament d'una aplicació J2EE sobre el framework Spring MVC, simulem la implementació d'una aplicació d'exemple utilitzant aquesta eina. Assumim que ens ve donat el codi que accedeix a la base de dades, així com el disseny de pàgines que s'espera a l'aplicació final.

Tant el codi de suport del que partim com el de la pròpia aplicació del cas pràctic es troba disponible als annexos d'aquesta memòria.

Exposarem el cas pràctic seguint una estratègia cronològica, enumerant els passos a seguir fins obtenir l'aplicació final i indicant a cada pas les decisions preses, així com els descobriments o complicacions sorgides durant cada etapa.

I. Creació del projecte i llibreries necessàries

La primera fase és la posada en marxa d'un projecte J2EE i la inclusió de llibreries necessàries per poder compilar i executar el framework Spring MVC. Per al primer punt utilitzem l'assistent del IDE NetBeans7, a través del menú "New, Project, Java Web Project". Li anomenarem SpringExample.

Per a la inclusió de llibreries, descarreguem el paquet complet de Spring Framework de la seva pàgina web. Ara be, com ja sabem, Spring Framework és una eina que va molt més enllà d'un framework de presentació, per la qual cosa no necessitem els gairebé 30mb de llibreries que inclou el .zip

descarregat. La llista final de llibreries necessàries pel tipus d'aplicació que volem implementar, ja sigui en temps de compilació o d'execució, és la següent:

- org.springframework.core-3.0.6.RELEASE
- org.springframework.asm-3.0.6.RELEASE
- org.springframework.beans-3.0.6.RELEASE
- org.springframework.context-3.0.6.RELEASE
- org.springframework.context.support-3.0.6.RELEASE
- org.springframework.expression-3.0.6.RELEASE
- org.springframework.web-3.0.6.RELEASE
- org.springframework.web.servlet-3.0.6.RELEASE

Addicionalment, es requereix de les següents llibreries de tercers per tal d'evitar errors del tipus ClassNotFoundException quan engeguem la nostra aplicació:

- asm-3.3.1
- cglib-2.2.2
- commons-logging-1.1.1
- FreeMarker

En darrera instància, afegim com a dependència i part integrant del sistema el projecte de NetBeans "DemoAgenda", que inclou les classes necessàries per gestionar contactes de l'agenda d'una suposada base de dades. Les seves classes quedaran empaquetades dins un arxiu .jar en el moment de desplegar la nostra aplicació.

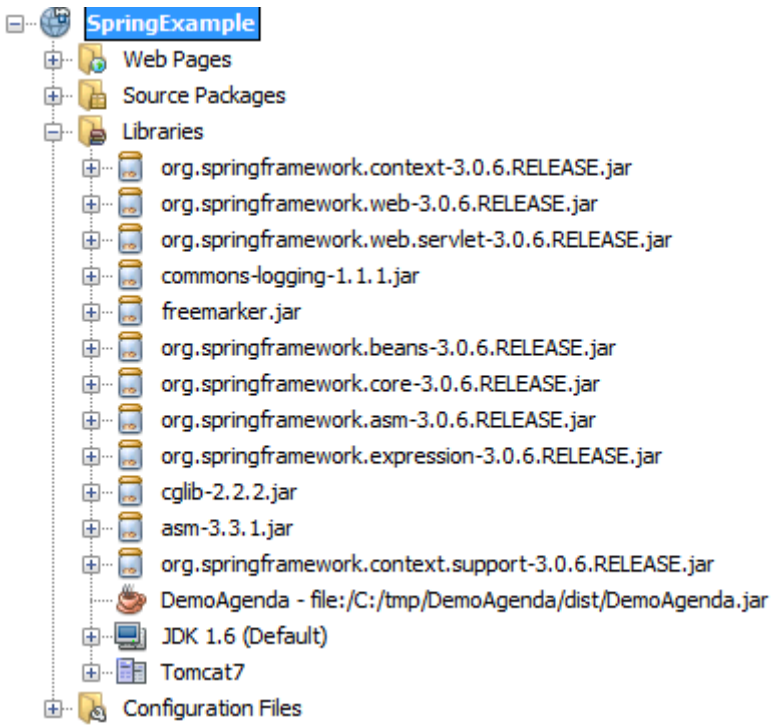

Figura 7. Llibreries del cas pràctic amb Spring MVC.

II. Configuració del DispatcherServlet

El punt de partida d'una aplicació amb Spring MVC és configurar el context base del qual pengen totes les URI que el framework gestionarà. Això és fa mapejant el DispatcherServlet contra la ruta base d'aquestes URI:

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

<servlet-mapping>
    <servlet-name>SpringExample</servlet-name>
    <url-pattern>/app/*</url-pattern>
</servlet-mapping>
```

Com que no indiquem explícitament quin tipus de configuració volem emprar, la llibreria optarà per la seva configuració per defecte i cercarà un fitxer amb nom SpringExample-servlet.xml dins del classpath del projecte. Aquest arxiu és qui inclou la configuració del framework.

Spring MVC permet, des de la seva versió 2.5, configurar el seu controlador a través d'arxius xml "tradicionals", o bé fent ús d'anotacions Java. Per aprofitar la possibilitat de disposar de la configuració molt més lligada a allò que es configura, optem per la segona opció. Necessitem utilitzar el tag context:component-scan indicant el paquet arrel on el framework ha d'iniciar l'escaneig d'anotacions.

```
<context:component-scan base-package="edu.uoc.examples.spring "/>
<context:component-scan base-package="edu.uoc.examples.spring.config
"/>
```

III. Creació del controlador

La base per disposar de punts d'accés que el framework tractarà és crear un controlador. Creem una classe AppController i amb l'anotació @Controller notifiquem a la llibreria que es tracta d'un controlador de peticions. En aquest punt podríem configurar aspectes addicionals com ara la ruta base de totes les accions que contempla el controlador, però optem per definir-ho amb rutes absolutes per a cada mapeig que crearem.

```
@Controller
public class AppController {
    ...
}
```

IV. Creació de punts d'accés: llistat de contactes

La primera plana que implementarem serà la un llistat de contactes, disponible a la ruta "/list" de l'aplicació. Comencem per crear un mètode al nostre controlador amb la següent signatura:

El nom del mètode no segueix cap regla: el podríem anomenar de qualsevol manera. La part important es troba a l'anotació que el precedeix:

```
@RequestMapping(value = "list", method = RequestMethod.GET)
public ModelAndView list() {
    ...
}
```

Amb aquesta anotació indiquem al framework que el mètode executa la lògica necessària per processar les peticions rebudes a la ruta "list". Addicionalment, restringim el seu accés a peticions HTTP de tipus GET.

Ara que tenim un espai on programar la nostra lògica, tot el que hem de fer és renderitzar una vista anomenada "list" a la que li oferim les dades corresponents a la llista completa de contactes. A continuació ens encarregarem de preparar aquesta vista.

```
@RequestMapping(value = "list", method = RequestMethod.GET)
public ModelAndView list() {
    ModelAndView mav = new ModelAndView();
    mav.addObject("entryList", AgendaDAO.getDAO().findAll());
    mav.setViewName("list");
    return mav;
}
```

V. Creació de vistes: llista de contactes

Fins ara, hem creat un punt d'entrada al sistema que carrega la llista de contactes i la delega a una vista anomenada "list". Ara és el moment de crear aquesta vista i fer saber a Spring MVC com processar-la.

Una de les virtuts del framework sobre el que estem treballant és que no ens limita a una única tecnologia per a la creació de vistes. A més de la solució més habitual (pàgines JSP), podem decidir emprar altres tecnologies. Entre d'elles, amb l'esperit d'aprofitar el cas pràctic per conèixer alternatives, escollim la de plantilles amb sintaxi de FreeMarker.

Per indicar al framework que estem preparats per processar vistes amb FreeMarker, és necessari crear una "ViewResolver", que és un objecte que s'encarrega de cercar, a partir d'un identificador de vista, si disposa d'alguna vista que es correspongui amb aquest identificador. Pel cas de FreeMarker, la llibreria ens ofereix la classe FreeMarkerViewResolver a la que li indiquem el prefix i sufix de les plantilles que crearem.

Aquest ViewResolver queda implementat dins d'una classe sense cap mena de requisit. L'únic que Spring MVC necessita és que indiquem amb l'anotació @Bean que el ViewResolver és un component que s'ha d'instal·lar al WebApplicationContext.

```
@Bean
ViewResolver freemarkerViewResolver() {
    FreeMarkerViewResolver vw = new FreeMarkerViewResolver();
    vw.setPrefix("");
    vw.setSuffix(".tmpl");
    return vw;
}
```

Només queda una darrera peculiaritat: pel cas de FreeMarker, cal indicar al fitxer de configuració de la llibreria el directori a partir del qual penjaran les nostres plantilles.

```
<bean id="freemarkerConfig"
class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfi
gurer">
    <property name="templateLoaderPath" value="/WEB-INF/FreeMarker/"/>
</bean>
```

Ara ja estem preparats per crear la nostra vista. Amb el nom "list.tmpl", no és més que un fitxer amb codi HTML al que hem inserit directives de FreeMarker per iterar per la llista de contactes, i mostrar per cadascun les seves dades amb accessos per esborrar i editar registres.

VI. Resta de punts d'accés

Amb la mateixa filosofia, anem implementant la resta d'accessos de l'aplicació. Algunes d'aquestes funcionalitats requereixen de nous elements de Spring MVC. Per exemple, en el cas de l'operació d'esborrat d'una sèrie de contactes, cal configurar el mètode que implementa el punt d'accés per saber interpretar la llista d'identificadors i codificar-ho com un Array que rebem com a paràmetre:

```
@RequestMapping(value = "delete/list", method = RequestMethod.POST)
public String deletelist(
    @RequestParam(value = "id", required = false)
    List<Integer> ids
) throws AgendaException {
    ...
}
```

Com es pot veure, d'això s'encarrega l'anotació @RequestParam. En aquesta ocasió indiquem també que l'acció pot provocar un error a l'agenda (la classe AgendaException pertany al codi que ve proporcionat). Més endavant parlarem de com lidiar amb aquestes excepcions.

Seguint amb estratègies inèdites, quan acabem amb èxit operacions com l'esborrat o l'edició de registres, volem tornar cap a l'acció de llistat de contactes. A més, volem que aquesta llista pugui saber que venim d'una

d'aquestes accions i mostri un missatge en conseqüència. El que fem és redirigir cap a l'acció corresponent a través d'una redirecció, i indicar el nostre origen amb un paràmetre get.

```
return "redirect:../list?callback=delete";
```

Per aquesta raó la signatura dels mètodes pot retornar tipus diferents. En el cas de la llista, generàvem un ModelAndView que inclou informació tant de la vista com de les dades a mostrar. Per aquest darrer cas, només retornem un String amb l'identificador de la vista o, com és el cas, la instrucció d'executar una redirecció cap a una URI relativa.

Encara completant la resta d'accions, ens trobem amb dos punts d'accés que volen que part de la informació viatgi codificada al punt d'accés de la URI. En concret, si accedim a la URI "delete/XXX", aquest XXX representarà l'identificador del contacte de l'agenda que volem esborrar. Aquest parseig que tradueix fragments de la URI amb paràmetres d'entrada s'aconsegueix amb l'anotació @PathVariable:

```
@RequestMapping(value = "delete/{id}")
public String deleteentry(@PathVariable int id) throws AgendaException
{
    ...
}
```

VII. Un darrer problema: rebre estructures complexes

La darrera funcionalitat que volem implementar (crear o modificar contactes), ens porta a una nova situació: volem que a partir del formulari HTML, rebem un objecte del tipus AgendaEntry amb valors dels atributs corresponents als camps d'aquest formulari. La manera d'aconseguir aquest comportament és utilitzar l'anotació @ModelAttribute:

```
@RequestMapping(value = "save", method = RequestMethod.POST)
public String save(@ModelAttribute("entry") AgendaEntry entry)
throws AgendaException {
    ...
}
```

Això ens porta a un nou problema: la data de naixement de cada contacte, indicada a l'atribut "AgendaEntry.birthDate", no es tradueix correctament. Per defecte FreeMarker tradueix els camps de text corresponents a objectes Date seguint un format diferent del "dd/mm/aaaa" que desitgem. Per indicar que desitgem un format diferent a totes les accions implementades pel controlador, fem una instal·lació inicial del nostre propi conversor de dates, configurable gràcies a la classe CustomDateEditor:

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.registerCustomEditor(Date.class
    , new CustomDateEditor(new SimpleDateFormat("dd/MM/yyyy"), true));
}
```

L'anotació @InitBinder indica al framework que el mètode que li segueix es dedica a instal·lar nous components i comportaments que afecten a tot el controlador i, per tant, s'ha de tenir en compte i executar el seu contingut quan s'inicialitza el controlador.

VIII. Afegir una segona tecnologia de vistes

Fins ara totes les vistes implementades ho feien a través de la tecnologia FreeMarker. Spring MVC ens garanteix que dins d'una mateixa aplicació, i fins i tot en un mateix controlador, es pot delegar la presentació a vistes programades amb tecnologies diferents. Per provar-ho, desitgem crear un punt d'accés "home" amb una benvinguda a l'aplicació creada en base a una pàgina JSP.

```
@RequestMapping(value = "home", method = RequestMethod.GET)
public String home() {
    return "home";
}
```

Per indicar al framework que hi ha un segon component ViewResolver amb capacitat per cercar plantilles JSP creem en la mateixa classe de configuració del primer cas un segon mètode amb l'anotació @Bean, en aquest cas instanciant un InternalResourceViewResolver que cerqui vistes amb extensió ".jsp" a la carpeta "WEB-INF/jsps" del projecte. No és necessari més configuració, amb aquestos passos ja disposarem de dos classes ViewResolver encarregades de acceptar la responsabilitat de generar les vistes.

```
@Bean
ViewResolver jspViewResolver() {
    InternalResourceViewResolver vw = new
InternalResourceViewResolver();
    vw.setPrefix("/WEB-INF/jsps/");
    vw.setSuffix(".jsp");
    return vw;
}
```

IX. Tractament d'excepcions

La major part dels mètodes que hem implementat com a mapeig de URIs declaren la possible generació d'una AgendaException. Aquest error es produeix quan es pretén realitzar certes accions il·legals com ara accedir a un contacte amb ID inexistent.

Per indicar a Spring MVC què ha de fer quan una acció provoca una AgendaException, cal configurar una acció que s'executarà com a conseqüència de l'aparició d'aquestos errors. El cos d'aquest mètode segueix la mateixa filosofia que la resta d'accions (retornar un ModelAndView, un String amb el ID de vista, etc.). L'única diferència és que en aquest cas fem ús de l'anotació @ExceptionHandler, i per defecte tenim dos paràmetres d'entrada: la instància de l'excepció, i la request que l'ha provocada:

```
@ExceptionHandler(AgendaException.class)
public ModelAndView handleAgendaException
(AgendaException ex, HttpServletRequest req) {
    ...
}
```

4.4.1.7. Conclusions

Amb aquestes passes hem aconseguit una eina de gestió de contactes que permet llistar, editar, crear i esborrar registres. Durant la seva implementació ens hem familiaritzat amb la manera de procedir amb Spring MVC, especialment en lo referent a la configuració del sistema a través d'anotacions Java. Fem una ullada retrospectiva i identifiquem els següents punts forts i febles:

Lo millor:

- Configuració amb anotacions còmode i intuïtiva, evitant llargs documents XML i una millor ubicació de la informació.
- Possibilitat d'utilitzar generadors de vistes alternatius, i fins i tot fer conviure més d'un simultàniament.

Lo pitjor:

- Certes restriccions que dificulten l'accés a recursos estàtics, com ara imatges o fulls d'estil CSS. La millor solució sembla ser separar totalment les planes de l'aplicació en la seva pròpia ruta, ja que així no interfereix amb la ruta dels recursos estàtics.
- Configuració de validació de formularis d'entrada poc intuïtiva. Després de dedicar cert temps a intentar implementar una validació de les dades, concloem que la documentació no és del tot clara i el sistema no és prou intuïtiu.

4.4.2. Struts 2 Framework

4.4.2.1. Identificació i història

Struts Framework surt a la llum pública per primer cop en maig del 2000 quan el seu autor, Craig McClanahan, dona la seva feina a la Apache Foundation. No és fins l'any 2005 quan esdevé un dels projectes de primer nivell de la fundació, amb el nom en clau de Jakarta Struts.

McClanahan, conegut per la seva feina com a part del grup de consultors que defineixen els components estàndard Servlet i JSP així com la seva arquitectura de contenidor de servlets Catalina (part clau del servidor d'aplicacions Tomcat), va crear un marc de treball que alleugerava al desenvolupador de les tasques pròpies del patró Model Vista Controlador, especialment en lo que correspon al itinerari de les peticions.

Tot i ser un dels frameworks considerats "lleugers" amb més acceptació dins el sector J2EE, la seva segona versió suposa tota una revolució, ja que es tracta d'un desenvolupament pràcticament nou i no d'una modificació de l'entrega anterior. Un esforç que cerca posar-se al nivell dels seus principals competidors, essent Spring MVC aquell amb qui les comparacions són inevitables tot i definir camins molt diferents des de la seva concepció.

4.4.2.2. Estil i arquitectura

Struts és un framework orientat a petició que, com acostuma a ser el cas habitual, segueix el patró de disseny FrontController per habilitar un punt central del sistema encarregat de redirigir tota la informació d'entrada (requests) i sortida (responses).

L'arquitectura de Struts 2 és una de les principals novetats respecte a la seva versió original. En els seus inicis, el patró de controlador principal s'implantava amb la solució òbvia d'un servlet especial anomenat ActionServlet. En canvi, en la seva evolució aquest controlador ha passat a implementar-se amb un altre component estàndard de J2EE, els filtres.

Així doncs, quan treballem amb Struts 2 el que fem és instal·lar un filtre anomenat FilterDispatcher que s'encarregarà de revisar les peticions incloses en una ruta i determinar si el sistema contempla algun controlador (les Action) que sap com respondre a cada petició. La denominació "Action" no varia respecte a la versió original del framework, tot i que ara les classes tenen més responsabilitats que originalment.

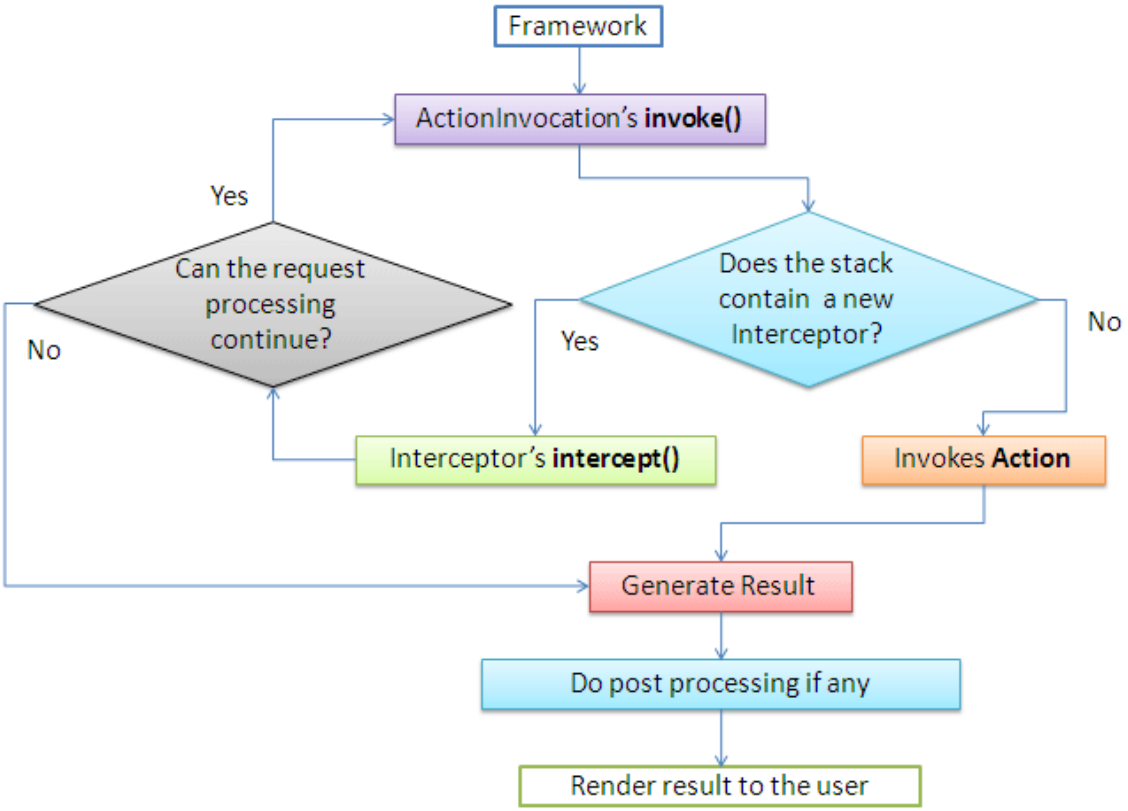

Figura 8. Diagrama d'activitat de Struts 2.

Completen l'arquitectura base del framework els interceptors, que són petits afegits que realitzen tasques no directament relacionades amb la lògica de negoci a partir de la petició i la resposta, com ara comprovacions i transformacions, i els results, que són els components encarregats de rebre (opcionalment) unes dades ja processades i generar una resposta en base a una vista que pot estar implementada per diferents tecnologies, tot i que JSP és la solució més habitual.

4.4.2.3. Components del framework

Com acabem d'introduir, els principals components d'una aplicació que funciona amb Struts són el FilterDispatcher, les Actions, els Interceptors i els Results. El primer és un punt de partida i el desenvolupador només ha de recórrer a ell per a tasques de configuració, així que estudiem més detingudament les altres tres parts del trencaclosques.

I. Actions

Les diferents Actions del sistema composen la major part de la lògica de l'aplicació. És aquí on es reben peticions (un o més tipus de peticions per a cada Action), es processen i es determina quina resposta s'ha de donar determinant quin Result serà l'encarregat de generar la vista. Però a més, és comú que les pròpies Action compleixin funcions de contenidor de dades (cosa

que ja succeïa en Struts 1), o fins i tot sigui l'encarregat final d'executar comportaments que els Interceptors decideixen que s'han de dur a terme.
Podríem dir per tant, que en el cas més habitual serà a les Action on el desenvolupador invertirà una major part del seu temps.

Una de les principals novetats que Struts 2 insereix respecte a les Action és la possibilitat que aquestes classes siguin Plain Old Java Objects, és a dir, objectes sense cap peculiaritat d'herència o la necessitat d'implementar una Interface de la llibreria en concret. Això s'aconsegueix gràcies a la potenciació de les tecnologies de Reflection i Introspection, així com l'ús d'anotacions Java per a marcar certes condicions que compleixen les classes i mètodes que defineixen.

II. Results

Els Results són els objectes responsables de la part de presentació. Identificats amb cadenes de text que emeten les accions, tenen associades plantilles de sortida que s'encarreguen de processar i omplir amb dades dinàmiques. Tot i que el cas més habitual siguin les JavaServer Pages o JSP, Struts ofereix suport per a la generació de vistes amb tecnologies com Velocity Templates o FreeMarker. De fet, la documentació deixa constància de que l'ús de FreeMarker a nivell intern és intensiu, raó per la qual necessitarem incloure aquesta llibreria a la nostra aplicació encara que no generem amb ella les nostres vistes.

III. Interceptors

Els Interceptors són petites unitats de processament que s'encadenen en la configuració del controlador. La seva execució es pot donar tant abans com després del procés central de la petició (el cos de la Action), i acostumem a tenir objectius molt concrets i aïllats entre si. Per exemple, podem tenir un Interceptor encarregat de la validació de les dades d'entrada, així com un altre que s'encarrega de transformar les dades rebudes en format text en instàncies Java que el controlador pugui gestionar. Com es pot deduir, alguns d'aquestos interceptors són gairebé imprescindibles i el seu comportament més habitual ja ve donat per classes per defecte del propi framework, estalviant haver de "inventar la roda".

IV. TagLib JSP

Com hem referit en ocasions anteriors, Struts 2 ofereix la possibilitat d'implementar les vistes amb diferents tecnologies, però en la majoria de casos la solució escollida són les pàgines JSP. En favor d'aquesta decisió juga l'existència d'una potent llibreria d'etiquetes per a l'obtenció i manipulació de dades rebudes per la capa de presentació.

Sota el nom de “Struts 2 UI Tags” trobem tota una sèrie de tags per implementar les tasques més comunes de recepció i enviament de dades. Per una banda, trobem eines per mostrar dades formatjades, així com iterar per col·leccions d’elements o escollir entre mostrar una informació o un altre sota certes condicions. Per altra banda, trobem etiquetes per gairebé qualsevol tipus d’entrada en format de camps de formulari HTML, despreocupant al desenvolupador de com els valors inserits en aquestos camps es codifiquen posteriorment per a que les classes Java del model rebin els tipus de paràmetre esperats.

V. ValueStack i OGNL

Tot i no ser un component especialment visible per al desenvolupador, cal fer especial menció al ValueStack. Sota aquest nom es troba el contenidor de dades que un Interceptor especial fa servir per codificar i descodificar les dades en el seu camí entre els formularis d’usuari i el codi de les accions.

Aquesta pila de dades es manté gràcies al OGNL, acrònim de Object-Graph Navigation Language. Igualment transparent en la majoria dels casos per al desenvolupador J2EE, es tracta d’un potent llenguatge d’expressions regulars idoni per referir-se i manipular la pila de valors.

4.4.2.4. Configuració

Tot i que amb la seva versió 2 Struts ha fet un esforç per oferir serveis de configuració mitjançant anotacions Java, el resultat no és lo suficientment perfecte per creure que aquesta és l’opció més còmode per la posada a punt d’un sistema que utilitza aquest framework. No obstant, la llibreria encara permet decidir quin tipus de configuració establir a la nostra aplicació. En concret les tres alternatives possibles són configuració clàssica amb XML, ús de convencions de noms o la referida configuració per anotacions.

I. Configuració per XML

Heretat directament de la seva versió 1, és el cas més típic on a partir de la instal·lació del controlador principal (el FilterDispatcher, en aquest cas) al descriptor web.xml de l’aplicació, la resta de paràmetres configurables queda definida en una segona font XML que per defecte s’espera amb el nom de struts.xml.

En aquest fitxer XML es troben, principalment, els punts d’instal·lació de les diferents Actions que el desenvolupador habilita per a tractar les peticions de l’usuari. Aquestos punts d’instal·lació poden venir acompanyats de certes possibilitats addicionals, como ara fer que un Action atengui a diferents punts d’entrada, indicats estàticament o bé en funció de valors de la URI de la petició rebuda.

II. Configuració per convenció de noms

Una de les novetats en matèria de configuració on Struts 2 ha fet un esforç notable és en la de establir certs aspectes del sistema de forma implícita en base als noms atribuïts als seus components.

D'aquesta manera podem delegar a la llibreria decidir aspectes como ara la URI que até cada Action, en base al nom que li donem a la seva classe. De la mateixa manera, podem associar una vista i el nom del fitxer plantilla que utilitza utilitzant noms comuns, creant així un Result implícit que el framework s'encarrega de gestionar.

Per exemple i si no indiquem lo contrari, una classe anomenada HelloWorld que implementa la interfície Action de Struts escoltarà peticions contra la URI /hello-world, delegant la lògica que tracta la petició al mètode execute() que defineix la interfície. Per altra banda, si la resposta no és un error el controlador cercarà per a generar la vista un fitxer anomenat hello-world.jsp. En aquest cas l'únic que ha fet el desenvolupador és anomenar "HelloWorld" a la classe Action, i la resta de noms venen donats per la esmentada convenció.

III. Configuració per anotacions Java

En darrera instància, ens queda l'opció d'evitar els fitxers XML però sense lligar-nos d'aquesta manera a uns noms concrets per als nostres components. La manera d'aconseguir-ho es fer ús d'anotacions Java però, i aquí on es troba la principal contradicció, encara estarem obligats a complir un petit conjunt de convencions de noms. La més evident és que les nostres Action deuen estar definides en classes Java amb un nom acabat en, precisament, "Action".

Posteriorment apareixen les anotacions com @Action o @Result per configurar les URI i vistes que volem associar a cada punt d'entrada, però si no hem cobert el primer requisit de la convenció de noms, no serà suficient per aconseguir el comportament desitjat.

4.4.2.5. Validació de dades

Per a la validació de dades, Struts 2 opta per un sistema més senzill però de la mateixa manera de més fàcil configuració que, per exemple, Spring MVC.

Per a validar dades el punt d'entrada és un Interceptor. La "pila d'interceptors" que Struts ofereix per defecte ja inclou una etapa de validació, que s'activa en els moments adequats sempre i quan la Action associada a la petició ofereixi eines de validació (això és, implementa la interfície Validateable).

Quan es donen aquestes condicions, el controlador delega en el mètode validate() de l'acció la tasca de comprovar les dades (que ja són accessibles des de la instància de l'acció) i detectar i notificar els possibles errors de validació. Aquestos errors queden inserits a un contenidor especial de dades

que el propi Interceptor s'encarrega de comprovar per conèixer si la validació ha estat satisfactòria.

4.4.2.6. Característiques fora d'estudi

Tot i haver tractat els components més habituals a l'hora d'emprar Struts 2, certes parts que ofereix el framework queden fora de l'objecte de l'estudi per una qüestió de recursos i temps disponibles.

I. Struts Tiles

A banda de la compatibilitat amb tecnologies com JSP o Velocity, Struts va desenvolupar i continua insistint en la seva pròpia tecnologia de presentació, anomenada Tiles. La idea principal que persegueix Struts Tiles és la de crear petits fragments de presentació reutilitzables, cadascun amb una finalitat molt concreta (capçalera, peu de pàgina, contingut, menú, etc.) i combinats degudament per a formar el resultat final desitjat.

4.4.2.7. Cas pràctic

Per a familiaritzar-se amb el desenvolupament d'una aplicació J2EE utilitzant el framework Struts2, passem a implementar la aplicació demo que hem fet servir al llarg del projecte, consistent en una agenda de contactes via web.

Tant el codi de base (DemoAgenda) com el de la pròpia aplicació del cas pràctic es troba disponible als annexos d'aquesta memòria.

Exposarem els punts més importants del desenvolupament del nostre cas, fent especial incís en aquelles característiques que diferencien la solució Struts2 de la resta d'alternatives estudiades.

En aquest cas optem per una configuració basada en els fitxers XML. La configuració d'un framework a través d'anotacions ja l'hem vist per altres casos de l'estudi, i en el cas de Struts2 presenta certs aspectes que fan d'aquesta una opció més confusa.

I. Creació del projecte i llibreries necessàries

Com a qualsevol posada en marxa d'un projecte J2EE, comencem configurant el projecte així com el conjunt de llibreries necessàries per fer córrer la nostra aplicació sota el framework Struts2. Tornem a l'assistent de nous projectes de NetBeans7, utilitzant en aquest cas el nom "StrutsExample".

El paquet complet de llibreries de Struts2 té un pes superior als 80mb i inclou un nombre elevat de fitxers .jars que només són necessàries sobre unes

condicions molt particulars. Pel cas que ens ocupa, n'hi ha prou amb conservar i afegir al projecte del IDE les següents llibreries:

- struts2-core-2.2.3.1.jar

A més de la base del framework, i també incloses a la descàrrega del paquet complet, necessitem les següents llibreries externes que Struts2 utilitza internament per al seu funcionament:

- commons-fileupload-1.2.2.jar
- commons-io-2.0.1.jar
- commons-lang-2.5.jar
- freemarker-2.3.16.jar
- javassist-3.11.0.GA.jar
- ognl-3.0.1.jar
- xwork-core-2.2.3.1

Evidentment, en darrera instància afegim com a dependència el projecte "DemoAgenda" amb les classes necessàries per accedir i operar amb la nostra agenda de contactes.

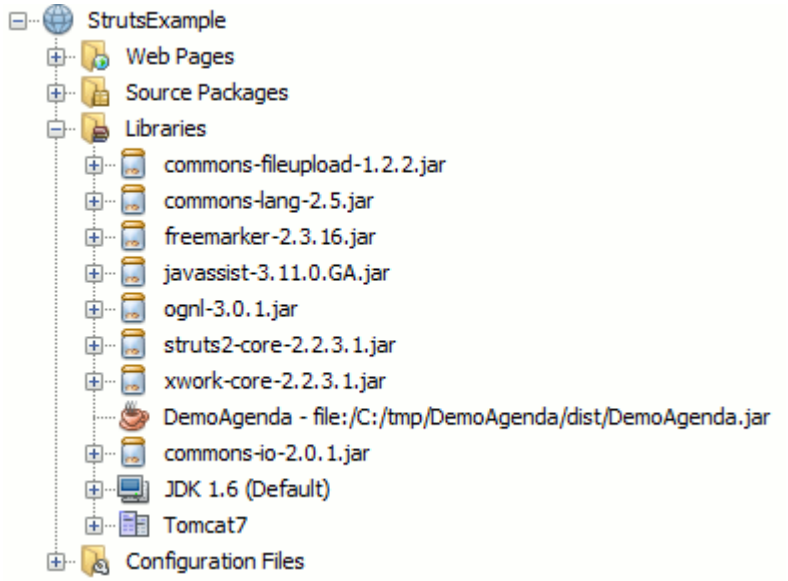

Figura 9. Llibreries del cas pràctic amb Struts 2.

II. Configuració del FilterDispatcher

Una de les principals diferències de Struts2 respecte a la seva competència, i fins i tot respecte a la seva versió anterior, és l'ús d'un component Filter com a component central del sistema, en comptes d'un Servlet com acostuma a ser el cas majoritari.

Struts ofereix una sèrie d'implementacions d'aquest Filtre central, incorporant comportaments que s'ajusten a certes preferències (per exemple, aprofitant les convencions de noms). Pel nostre cas ens limitem a la versió més bàsica

d'aquest controlador principal, implementada per la classe org.apache.struts2.dispatcher.FilterDispatcher.

Instal·lem aquest filtre en el nostre descriptor J2EE (web.xml), capturant absolutament totes les peticions dirigides a l'aplicació. Com veurem més endavant, Struts2 dona un bon rendiment per defecte situant els recursos estàtics (estils CSS, JavaScript, etc) a la mateixa arrel que els controladors dinàmics.

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-
class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

A partir d'aquest moment no necessitarem tornar al nostre fitxer web.xml, ja que tota la configuració pròpia de Struts2 es realitzarà a un fitxer xml de configuració especial anomenat struts.xml, situat a l'arrel del Classpath del projecte.

III. Creació de les accions de l'aplicació

Com comentàvem, la configuració d'accions de Struts2 té lloc a un fitxer struts.xml situat a l'arrel del Classpath visible per l'aplicació.

Ens decidim per una arquitectura en la que un mateix controlador accepta totes les possible peticions que contempla el sistema. Per configurar aquesta situació, definim tota una sèrie d'accions implementades per la mateixa classe, anomenada AgendaAction. La distinció entre les diferents peticions possibles es realitza amb el paràmetre "method" que, fent ús d'una de les convencions de Struts2, es referirà tant a la URI on queda instal·lada l'acció com al mètode Java que implementa el seu procés dins de la classe controladora.

Per exemple, la següent línea defineix que el mètode AgendaAction.execute_list rebrà les peticions a "/list":

```
<action
    name="list"
    class="edu.uoc.examples.struts.action.AgendaAction"
    method="execute_list" />
```

Definides les accions, encara cal configurar el sistema de vistes que generen la resposta per a cada petició. Per exemple, l'acció de llistat sempre delegarà en la mateixa vista, però l'acció de guardar dades pot desitjar sortides diferents en funció de les dades d'origen.

Per altra banda, hem de distingir entre les vistes “normals”, en les que partim d’una plantilla per omplir les dades, i les vistes que fan referència a una redirecció interna. Per a les primeres el valor del contenidor “result” fa referència a la plantilla, i amb l’atribut type indiquem el tipus de tecnologia de renderitzat que emprem (en aquesta ocasió ens tornem a decidir per FreeMarker). Per el cas de les redireccions, l’atribut type indica “redirect” i el valor del resultat és la URI relativa a on volem enviar la redirecció.

El següent contenidor defineix les tres possibles vistes que generen la resposta a una petició de guardat de dades. Una d’elles renderitza la plantilla “edit.tmpl”, i les altres dos fan una redirecció cap a la pàgina de llistat.

```
<action
 name="save"
 class="edu.uoc.examples.struts.action.AgendaAction"
 method="execute_save">
    <result name="EDIT" type="freemarker">/edit.tmpl</result>
    <result name="INSERTOK"
type="redirect">/list?message=insertok</result>
    <result name="UPDATEOK"
type="redirect">/list?message=updateok</result>
</action>
```

IV. Creació de la classe controlador

Com hem decidit que una sola classe controladora atendrà totes les peticions, implementem una única classe Java anomenada AgendaAction. Tot i no ser necessari, construïm aquesta classe heretant de ActionSupport, ja que aquesta ja implementa la majoria de comportaments estàndard que no volem modificar.

```
public class AgendaAction extends ActionSupport {
    ...
}
```

Creem mètodes per a cada punt d’accés de l’aplicació, amb noms que coincideixen amb els indicats en els atributs “action” del fitxer de configuració, i retornant sempre un String que serà l’identificador del Result que volem utilitzar.

```
public String execute_list() {
    ...
}

public String execute_save() throws AgendaException{
    ...
}
```

Arriba ara un dels punts més diferenciadors de Struts2, i que poden esdevenir una virtut o un problema segons l’escenari. En Struts2 (i també a la versió anterior), el propi controlador exerceix les funcions de contenidor de dades. És a dir, el controlador manté la informació vinculada a una petició en forma

d'atributs amb mètodes getter i setter, i el framework s'encarregarà de fer les crides pertinents a aquestos setter abans de invocar el mètode que processa cada petició. Per exemple, definim una llista Java que contindrà els contactes a mostrar a la pantalla de llistat.

```
private List<AgendaEntry> entryList;

public List<AgendaEntry> getEntryList() {
    return entryList;
}

public void setEntryList(List<AgendaEntry> entryList) {
    this.entryList = entryList;
}
```

Això inicialment ens simplifica el codi, però per controladors d'accions que impliquen entrada de dades complexes o de volum considerable, pot esdevenir un problema. Hem d'estudiar les accions a implementar i pensar en quines dades necessitem presentar o rebre per a cada cas. Els atributs que creem pertanyen a l'àmbit de petició, i per tant no mantenen el seu valor entre diferents peticions. No és el nostre cas, però si vulguem mantenir cert estat de l'aplicació caldria recórrer a alternatives per accedir a l'àmbit de sessió o aplicació.

V. Creació de les vistes

Les vistes, com hem decidit continuar amb la tecnologia FreeMarker de casos anteriors, no presenta majors dificultats. En cas d'haver escollit JavaServer Pages, en aquest moment podríem aprofitar la potència del Struts TagLib amb solucions per construir pàgines dinàmiques accedint a les dades contingudes al controlador.

VI. Validació de dades

En quant a la validació, disposem de dos alternatives: aprofitar al màxim la infraestructura que ja ofereix Struts2, o bé optar per una solució més manual però que ens permet no haver de modificar tota una sèrie de configuracions per defecte.

En cas d'escollir la primera opció, aprofitaríem el interceptor de validació que ja es troba desplegat per defecte, però caldria tenir cura de en quines accions volem que les dades siguin prèviament validades i en quines no. Com que hem optat per un controlador comú per totes les accions, això ens afegeix una feina addicional. En aquesta opció, cal implementar el procés de validació en un mètode anomenat "validate", però també necessitaríem reconfigurar tota la pila d'interceptors (sobreescrivint la que Struts2 disposa per defecte) per tal de definir quins mètodes del controlador requereixen validació en el nostre cas.

Per altra banda, podem optar per invocar manualment la validació. Si no implementem el mètode "validate" del nostre controlador, la tasca configurada als interceptors mai tindrà efecte ja que no trobarà com executar la comprovació de les dades. D'aquesta manera, el propi controlador assumeix la responsabilitat de validar dades només quan és necessari, i crida al seu mètode intern de validació al inici d'aquestes peticions. Després d'invocar aquest mètode, fem el que faria l'interceptor en el seu cas: comprovar si s'han detectat errors i, en aquest cas, delegar en la vista que els pot mostrar.

```
public String execute_save() throws AgendaException {
    myValidation();
    if(!this.getFieldErrors().isEmpty()) { return "EDIT"; }
    ....
}
```

El mètode de validació implementa totes les comprovacions necessàries i, per cada error detectat, afegeix una entrada a la pila d'errors a través del mètode "addFieldError" heretat de ActionSupport.

```
if(
  getInputData().get("surname") == null ||
  "".equals(getInputData().get("surname").trim()))
{
    addFieldError("save_surname", "Blank surname");
}
```

Per altra banda, la validació de dades presenta un altre complicació fruit de compartir controlador i contenidor de dades. El camí de la petició és tal que primer s'executen els mètodes setter necessaris per enviar les dades del formulari, després s'executa la validació i en darrer lloc executem el codi de la petició pròpiament dit. Per tant, necessitem un contenidor de dades que ens permeti mantenir la informació inserida per l'usuari sense cap mena de tractament.

Per exemple, si tenim un camp que ha de ser de tipus numèric, de totes maneres necessitem guardar el seu valor de tipus String per poder fer la comprovació pertinent en el mètode de validació. Si fem directament la conversió a int amb tractament de l'excepció, perdrem el valor original rebut i no podrem detectar segons quins tipus d'errors de validació. Per això el nostre controlador inclou un Map que emmagatzema aquestes dades. El formulari invoca a uns setter que s'encarreguen de guardar la informació tant sense tractament (en aquest Map), com a l'objecte que utilitzem després al tractament de la petició.

```
private Map<String, String> inputData;

public void setSave_surname(String s) {
    getInputData().put("surname", s);
    this.getEntry().setSurname(s);
}
```

En darrer lloc, cal ampliar la nostra vista per mostrar els possibles errors de validació. La plantilla rep un objecte fieldErrors on utilitzant com a clau el nom de cada camp, recuperem els eventuals errors de validació que puguin existir.

```
<@_show_errors fieldName='save_surname" />
```

VII. Tractament d'excepcions

En darrer lloc, desitgem que les excepcions provocades per les peticions, i més concretament les provocades per la gestió de la nostra Agenda en concret, tinguin un tractament especial que permeti a l'usuari visualitzar l'error.

Per fer-ho cal ampliar el fitxer de configuració d'Struts2 indicant la vista encarregada de renderitzar cada tipus d'excepció.

```
<global-results>
    <result name="ERROR_AGENDA"
type="freemarker">/error_agenda.tmpl</result>
    <result name="ERROR" type="freemarker">/error.tmpl</result>
</global-results>

<global-exception-mappings>
    <exception-mapping
     exception="edu.uoc.examples.agenda.AgendaException"
result="ERROR_AGENDA" />
    <exception-mapping exception="java.lang.Exception" result="ERROR"
/>
</global-exception-mappings>
```

A les plantilles d'error, disposem d'una variable "exception" amb tota la informació necessària de l'excepció generada.

```
<p style="color:red;">${exception.message}</p>
```

4.4.2.8. Conclusions

La experiència amb Struts2 esdevé només parcialment satisfactòria. El seu punt de partida en forma de Filter en lloc de Servlet no comporta cap diferència radical en el desenvolupament. La configuració per XML es clara i senzilla, però no així la seva oferta d'anotacions Java per evitar arxius addicionals. En aquest aspecte ja durant l'estudi de l'eina ens ha semblat que la solució no resultava atractiva pel desenvolupador.

Per altra banda, és impossible parlar de Struts2 sense fer referència a una de les seves particularitats: la convivència de controlador i dades a la mateixa classe Action. En el moment que volem interactuar amb estructures de dades més complexes que els tipus bàsics, el manteniment dels controladors es complica exponencialment. Afegir controls de validació no fa més que afegir encara més complexitat a les nostres classes.

En l'aspecte positiu, la llibreria inclou tota una sèrie de comportaments per defecte que, per a gran part dels escenaris possibles, poden ser més que suficients, com ara la configuració d'interceptors o el suport gairebé nadiu per plantilles amb FreeMarker.

4.4.3. Java Server Faces

4.4.3.1. Identificació i història

El projecte Java Server Faces (JSF) veu la llum amb la seva primera versió l'any 2004, a mans d'una selecció d'experts dins de la Java Community Process entre la que es troben, i no casualment, noms clau dins el desenvolupament J2EE com ara el creador de Struts, Craig McClanahan.

En realitat, quan parlem de JSF no ens estem referint a una solució concreta, si no a l'especificació d'un tipus de solució per l'àmbit dels framework J2EE. La JCP defineix els estàndards que han de complir les possibles implementacions, però no obliga a utilitzar cap en particular. De fet, l'esperit de JSF és definir una base que permeti desenvolupar eines de suport (com ara entorns IDE) sense necessitat de dependre d'una solució concreta, i ofereix tota una sèrie de facilitats per afegir components de collita pròpia.

Des de la seva primera aparició sota la versió 1.0 ha patit fins a quatre grans actualitzacions, essent la darrera i potser la més rellevant la de l'any 2009 donant un salt de primer nivell fins a la denominació de JSF 2.0.

Existeixen diverses implementacions de l'especificació de Java Server Faces, començant per la pròpia de referència llençada per Sun Microsystems (ara en propietat d'Oracle). Altres opcions que gaudeixen de popularitat són MyFaces, la proposta de la Apache Software Foundation, RichFaces, que té un especial interès en desenvolupar aplicacions més dinàmiques sota la filosofia AJAX, o ICEFaces, que ofereix tota una sèrie de components avançats enfocats a una experiència de l'usuari més rica i atractiva.

4.4.3.2. Estil i arquitectura

A diferència d'altres grans noms dins el món dels frameworks J2EE (Struts, Spring...), JavaServer Faces defineix un framework orientat a components, i no a petició. Això vol dir que el servidor instal·la una sèrie de components que composen el sistema, i l'usuari es limita a produït els esdeveniments que aquestos components reben i transformen, després de tota una lògica implementada, en una resposta en forma de transformació de la interfície. Per tant, podem dir que la interfície en aquesta ocasió "s'executa" totalment a la part del servidor.

Com que no neix tan fortament lligada al protocol de peticions HTTP, ofereix un millor suport nadiu per a cicles de vida dels components, així com pel desenvolupament de sistemes amb memòria.

No obstant aquesta notable diferència, si que comparteix amb els seus "competidors" l'ús del patró FrontController com a pedra angular del sistema. En aquesta ocasió, a través d'un FacesServlet que s'encarrega de redirigir tot el tràfic de l'aplicació en ambdós sentits.

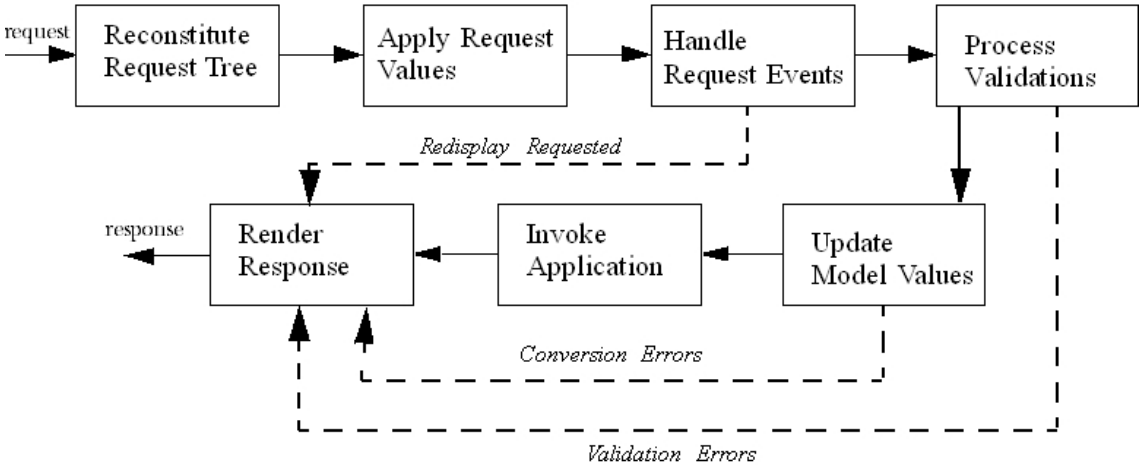

Figura 10. Cicle de vida a Java Server Faces.

El servidor manté una estructura interna de components i el seu estat. Quan l'usuari genera un esdeveniment, FacesServlet el rep i transmet la petició cap als components afectats, que a la seva vegada decideixen quina resposta generar operant amb els Beans que gestiona.

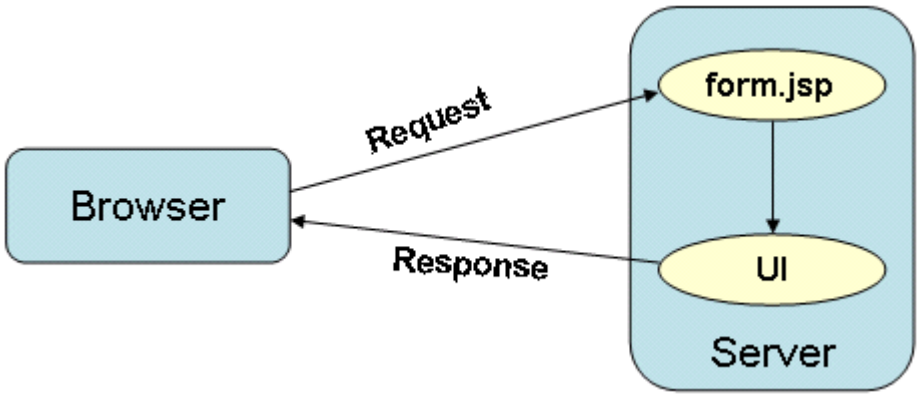

Figura 11. Camí de les dades a Java Server Faces.

4.4.3.3. Components del framework

JavaServer Faces presenta dues parts ben diferenciades: els UI Components que s'encarreguen de rebre, processar i donar resposta a les peticions, i dos llibreries d'etiquetes que permeten la generació de la vista fortament lligada a aquestos components.

FacesServlet

Com hem introduït abans, el FacesServlet és el controlador principal i s'encarrega d'analitzar quins components són responsables dels esdeveniments produïts per l'usuari final. Es tracta d'un component que el desenvolupador no acostuma a manipular, limitant-se a desplegar-ho.

I. Taglibs

Els taglibs són part essencial del funcionament de Java Server Faces. En diferenciem dos llibreries: una encarregada d'oferir eines per a la presentació de dades i altres missatges a l'usuari, i altre amb tot lo necessari per dissenyar aquestes interfícies connectades amb el servidor en forma de generació d'esdeveniments.

En una aplicació desenvolupada amb JSF, l'ús dels taglibs a les vistes acostuma a ser intensiu, fins i tot amb més presència que el propi codi HTML dins de les plantilles JSP que defineixen la vista. Això és degut a que gran part del codi HTML final que es presenta ha de ser generat de forma molt dependent als comportaments del sistema, i per tant la seva creació és responsabilitat de diverses etiquetes dels taglibs de JSF.

II. Components UI

Els components UI, per la seva banda, són els petits fragments de software que permeten implementar tota una sèrie de comportaments típics d'una aplicació web. Els més evidents són els encarregats de processar esdeveniments o validar dades, però també els trobem per altres serveis com ara la internacionalització o la creació d'una aplicació accessible.

La especificació de Java Server Faces ja inclou tota una sèrie de components UI que tota implementació de l'estàndard es troba obligat a implementar. En canvi, no es tracta d'una col·lecció tancada i cada implementació pot aportar els seus propis components, ampliant així l'atractiu i la potència del framework.

III. Managed Beans

En darrer lloc, els Managed Beans son aquelles peces de codi que queden acoblades als components UI per tal d'oferir propietats d'entrada i sortida i mètodes per implementar els comportaments desitjats. Podríem dir que els components UI exerceixen el rol d'intermediari o interceptor de les peticions, i són finalment els Managed Beans els qui s'encarreguen d'implementar la lògica de la nostra aplicació.

4.4.3.4. Configuració

La configuració d'una aplicació amb JavaServer Faces ha estat una de les parts més afectades pel més recent canvi de versió. Fins a JSF1.2, es seguia el tradicional principi d'un arxiu de configuració (faces-config.xml) on definir i instal·lar tot lo necessari: managed beans, regles de navegació, validacions, etc.

En canvi, amb el salt a JSF2.0 l'especificació de JSF ha aprofitat la potència de les anotacions Java i augmentat la potència de les seves taglibs amb l'objectiu de fer més comprensible i senzilla l'arquitectura del nostre sistema sense deixar de mirar el seu codi.

Així, per exemple, ja no és estrictament necessari un fitxer faces-config.xml, tot i que per sistemes complexos amb una alta oferta de serveis i navegació encara es recomana disposar d'aquest emplaçament o concentrar aquesta informació. En qualsevol cas, el desenvolupador té l'opció de, entre d'altres, indicar les regles de navegació com a atributs a les pròpies etiquetes dels taglibs, o indicar que una classe Java fa les funcions de Managed Bean a través de l'anotació @ManagedBean.

El que no canvia, però, és l'òbvia necessitat d'indicar al descriptor de l'aplicació web.xml la instal·lació del FacesServlet que s'encarrega de rebre i processar totes les peticions dirigides al framework.

4.4.3.5. Validació de dades

La validació de les dades es du a terme en els propis Managed Beans que contenen la informació que l'usuari envia. En concret, el desenvolupador crea un mètode (sense cap imposició en quant al nom) que s'encarrega de realitzar la validació per a cada camp. Es tracta d'un mètode sense valor de retorn i que, en cas de trobar errors, genera una ValidatorException que els components UI s'encarregaran de detectar.

L'associació entre camps de formulari, validacions i missatges d'error té lloc, com gairebé tot en Java Server Faces, en els atributs de les etiquetes que el framework ofereix.

4.4.3.6. Característiques fora d'estudi

Per la seva pròpia naturalesa, Java Server Faces és un framework que inclou un ample catàleg de possibilitats, el qual encara creix més si comencem a estudiar les implementacions disponibles de l'especificació. Algunes d'aquestes característiques poden ser d'utilitat i d'ús freqüent en la gran majoria de desenvolupaments J2EE, però queden fora d'aquest estudi per una qüestió d'economia de recursos i abast del projecte.

I. Suport AJAX (RichFaces, etc.)

Algunes de les implementacions de Java Server Faces es troben molt orientades a la nova filosofia d'aplicacions web més dinàmiques i no tant lligades al estereotip d'una plana nova per a cada petició, actualitzant petites parts de la pàgina ja carregada quan es dona resposta als esdeveniments de l'usuari.

Una de les implementacions d'aquest tipus més coneguda és RichFaces, que implementa components UI que permeten que la interfície de l'usuari s'alteri només parcialment a través de transformacions de l'arbre HTML amb JavaScript.

II. Suport per temes

Donat que gran part del codi de presentació queda delegat en l'ús de taglibs del framework, l'aspecte final d'aquesta presentació es troba molt lligat al marc de treball i, aprofitant aquesta situació, el suport per a disposar de temes intercanviables és molt elevat.

4.4.3.7. Cas pràctic

Java Server Faces presenta certes característiques que no constituirien un obstacle en el desenvolupament real d'aplicacions amb certa complexitat, però que esdevé un handicap pel nostre cas d'una senzilla aplicació d'exemple per provar de primera mà l'experiència del seu ús: es tracta d'una llibreria amb esperit de donar suport a aplicacions completes i, el més important, amb la intenció de treballar conjuntament amb editors i assistents que alliberin el desenvolupador de les tasques més mecàniques o que suposen un menor repte.

El resultat és que, quan cerquem la documentació relativa a crear un projecte des de zero amb Java Server Faces, la major quantitat d'informació que trobem fa referència a assistents d'editors com ara Eclipse o NetBeans que permeten crear tota una estructura a partir d'un model pre-existent, com ara un esquema de base de dades.

Aquest no és el nostre cas, ja que per la nostra aplicació d'exemple fem servir un conjunt de classes Java que simulen l'accés a un magatzem de dades, però no l'implementen realment. Per tant, totes aquestes solucions "màgiques" no resulten aplicables pel nostre cas i ens veiem obligats a crear la estructura del projecte "a mà".

I. Creació del projecte

Comencem amb la única fase en la que efectivament el nostre IDE ens pot ajudar més enllà de l'edició de fitxers: al crear un projecte amb NetBeans, indiquem en el darrer pas de l'assistent que volem treballar amb el framework "Java Server Faces". Quan fem això, el projecte resultant presenta dos afegits respecte al cas bàsic projecte web amb Java: en l'apartat de llibreries s'han afegit els arxius .jar corresponents a la API i implementació de JSF2.0 i la JSTL1.1, i el descriptor web.xml ja instal·la el controlador principal Faces Servlet contra totes les peticions realitzades a la URI "/faces/*".

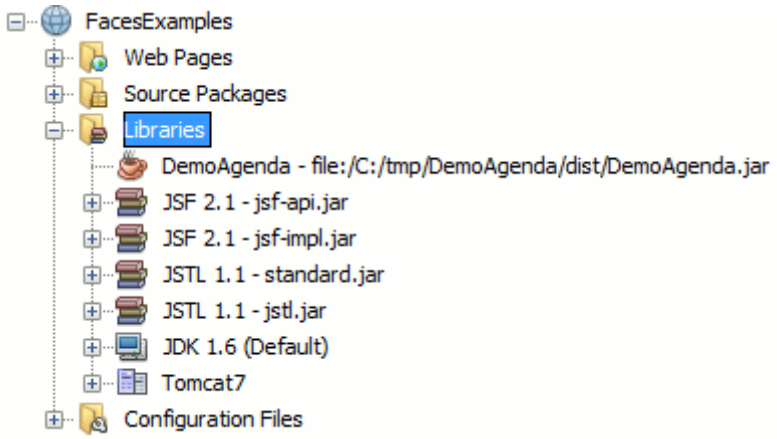

Figura 12. Llibreries del cas pràctic amb Java Server Faces.

II. Creació del Managed Bean

Observant l'arquitectura d'una aplicació desenvolupada amb JSF, veiem que la tasca del desenvolupador es troba molt focalitzada a dos punts: les plantilles (en aquest cas arxius .xhtml) que presenten la vista, i els ManagedBeans que s'encarreguen d'albergar les dades amb les que treballa la vista així com les accions que aquesta pot invocar.

Podem concloure doncs que els Managed Bean és, si no més, un 50% del gruix de la nostra aplicació. Una decisió habitual és implementar un Managed Bean per cada àmbit del sistema (per exemple, un que gestiona empleats, altre que gestiona departaments, etc). Com que la nostra aplicació és molt senzilla i només opera amb un tipus de registre (entrades d'una agenda), en tenim prou amb un sol contenidor.

Arriba el torn de decidir quin principi de configuració aplicarem en la nostra aplicació. Podem optar per la fórmula clàssica creant i omplint el fitxer faces-config.xml, o bé optar per la tècnica més moderna de les anotacions Java. Per un equilibri entre usabilitat, comprensió del codi i preferències personals, ens decidim per les anotacions.

Podem convertir qualsevol objecte Java en un Managed Bean amb tan sols afegir l'anotació @ManagedBean junt a la definició de la classe. Podem indicar

un nom que referirà al Bean durant la vista, o deixar que el framework li assigni un segons les seves convencions. Addicionalment, el nostre Bean pot tenir cert suport per mantenir estats o no, segons si el seu àmbit és la petició, la sessió d'usuari, o tota la aplicació. Pel nostre cas ens convé l'anotació @SessionScope.

```
@ManagedBean(name="agenda")
@SessionScoped
public class AgendaManagedBean {
    ...
}
```

A partir d'aquí, tot el que afegim al nostre objecte poden ser atributs visibles per la vista (sempre que implementen els corresponents getters o setters), o bé punts d'entrada per accions invocades des d'aquestes. Per aquest segon cas, els mètodes corresponents han de retornar un String corresponent a la vista que volen processar.

```
public String action_list() {
    ...
}
```

Tornarem a aquest ManagedBean per certes accions requerides per algunes funcionalitats que afegirem.

III. Creació de les vistes

Si el Managed Bean era un 50% del nostre sistema, les plantilles que conformen la vista són l'altre meitat. Al contrari que en altres frameworks, en aquest cas no té gaire sentit debatre sobre quina tecnologia de generació de vistes emprar: gran part de l'atractiu de Java Server Faces és la seva col·lecció de taglibs que implementa la majoria de necessitats d'una vista HTML amb entrada i sortida de dades. A partir de la versió 2.0, la recomanació és crear les plantilles com a arxius XHTML, abandonant la pràctica anterior de definir-les com a Java Server Pages (JSP).

Totes les vistes de l'aplicació tenen un punt de partida comú: afegir les referències necessàries per utilitzar les llibreries de tags h:* (html) i f:* (core). Amb aquestes dues línies, ja podem aprofitar tota la potència dels TagLibs de JSF per generar la nostra resposta HTML.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
```

Hi ha certes normes que una plantilla de Faces ben construïda ha de complir. Per exemple, totes aquelles etiquetes involucrades en l'entrada/sortida i invocació d'accions de manipulació de dades es troben anidades per un contenidor de l'etiqueta h:form.

L'ús d'etiquetes de JSF fan que les plantilles per aquest cas d'exemple siguin radicalment diferents que les dels casos estudiats amb anterioritat. Per exemple, ja no definim explícitament taules HTML, si no que l'etiqueta h:dataTable juntament a altres com h:column o f:facet s'encarregaren de tabular les dades i, finalment, presentar-les com una taula HTML.

```
<h:dataTable value="..." var="item">
    <h:column>
        <f:facet name="header...</f:facet>
        ...
    </h:column>
    ...
</h:dataTable>
```

Quan volem mostrar a la vista dades contingudes en un Managed Bean, farem ús d'etiquetes com h:outputText, que permeten indicar la font d'informació on es troba el valor que volem mostrar. Cada cop que fem referència a un Managed Bean, ho farem amb la sintaxi #{beanName.[attribute/method].

```
<h:outputText value="#{entry.name}" />
```

Un cas similar el trobem als camps de formulari que permeten l'entrada de dades. Etiquetes com h:selectBooleanCheckbox o h:inputText crearan camps input html que reben les dades del contenidor, i les envien quan s'invoca alguna acció.

```
<h:inputText id="Name" value="#{agenda.entry.name}"
required="true"></h:inputText>
```

En darrer lloc, podem invocar accions d'un Managed Bean a través de, per exemple, les etiquetes h:commandButton (si volem un botó) o h:commandLink (si volem un anchor). Per tots dos casos indiquem el Bean i acció associat amb la sintaxi prèviament esmentada.

```
<h:commandButton action="#{agenda.action_save}" value="Save" />
```

IV. Estructura DataModel

Al llarg de la implementació de la vista corresponent a una llista dels registres de l'agenda, ens trobem ja una peculiaritat de JSF.

El nostre sistema ha de mostrar una taula per la qual, si l'usuari fa clic sobre un botó d'una fila, s'ha de rebre aquesta petició i mostrar un formulari d'edició del registre seleccionat. Dit d'una altra manera, necessitem una acció que pugui identificar el registre seleccionat, carregar les seves dades i delegar a una nova vista amb el formulari esmentat.

Per poder implementar aquest escenari, i més particularment per poder detectar quin dels registres llistats és el que ha seleccionat l'usuari, la nostra col·lecció de dades ha de ser implementada per una classe que implementa la

interfície DataModel¬<T>, on T fa referència al tipus de dada que gestiona la col·lecció.

```
private DataModel<AgendaEntry> entries;
```

Pel nostre cas la implementació més adient és ArrayDataModel, que permet crear una col·lecció d'aquest tipus a partir d'un array de Java.

Quan rebem l'acció d'editar un registre, podem identificar quin és el registre involucrat invocant el mètode getRowData() del nostre DataModel. El framework ja s'haurà encarregat de mapejar internament les dades de la fila corresponent.

```
public String action_edit() throws AgendaException {
    this.setEntry(getEntries().getRowData());
    return "edit";
}
```

V. Atribut "rendered"

Una funcionalitat especialment interessant en el moment de dissenyar vistes és poder configurar fàcilment que certes parts de la interfície es mostrin o no segons certes condicions calculades en temps d'execució. Per exemple, si volem aprofitar una mateixa vista per l'edició de registres existents o la inserció de nous, certes parts com ara un botó "d'esborrat" només tenen sentit pel primer dels casos.

Aquest servei l'ofereix l'atribut "rendered" present a gran part de les etiquetes dels taglibs de JSF. Quan s'indica, el codi HTML generat per l'etiqueta en qüestió només es crearà en cas que es compleixi la condició indicada per aquest atribut. Aquesta condició pot ser, per exemple, un atribut del Bean calculat en base a altres dades del seu àmbit (per exemple, a partir de l'existència o no de codi del registre actual).

```
public boolean getNewEntry() {
    return !(this.getEntry() != null && this.getEntry().getId() != 0);
}

<h:outputText rendered="#{agenda.newEntry}" value="New entry" />
```

VI. Validació de dades

Com és desitjat, els taglibs de JSF també ofereixen eines per a la validació de dades prèvia a invocar una acció. Aquesta ve integrada dins les etiquetes corresponents a camps de formulari, i s'indica afegint en el seu cos altres etiquetes com ara f:validateRequired o f:validateRegex. Pel cas de camps obligatoris, tenim una fórmula encara més senzilla: utilitzar l'atribut "required" de la pròpia etiqueta que genera el input HTML.

```
<h:inputText id="E-mail" value="#{agenda.entry.mail}" required="true">
    <f:validateRegex pattern=".*@.*" />
</h:inputText>
```

Si alguna de les validacions especificades no es compleix, l'acció derivarà en la recàrrega de la vista actual, però en aquest cas podem accedir als missatges d'error propis de cada camp gràcies a l'etiqueta h:messages, indicant a l'atribut "for" l'identificador del camp del qual volem conèixer (i mostrar si existeixen) els errors que ha provocat.

```
<h:messages for="E-mail" style="color:red;" />
```

VII. Tractament d'excepcions

No tot el procés d'implementar la nostra aplicació d'exemple amb Java Server Faces ha estat satisfactori. Hi ha un aspecte, el del tractament d'excepcions, en el que no hem trobat una solució adient dins el temps destinat a aquesta tasca.

La documentació disponible a la xarxa ofereix dues possibilitats pel tractament d'excepcions. Una, la més bàsica, consisteix a configurar una pàgina d'error convencional a través del contenidor "error-page" del descriptor web.xml. En realitat, Faces no afegeix cap tractament especial en aquest cas, i es limita a propagar l'excepció per tal que el servidor d'aplicacions s'encarregui de processar-la.

La segona opció, més sofisticada, és crear el nostre propi ExceptionHandler que s'encarregui d'assumir la responsabilitat de detectar aquestes excepcions i actuar en conseqüència.

En el nostre entorn, en canvi, cap de les solucions ha funcionat com s'esperava, obtenint com a resultat una pàgina d'error del servidor d'aplicacions que ens indica que l'excepció no ha estat capturada degudament.

4.4.3.8. Conclusions

És difícil tractar Java Server Faces en igualtat de condicions respecte a la resta d'eines estudiades. Tant el seu esperit (especificació oberta amb implementacions independents), com la seva arquitectura bàsica (orientada a components, no a peticions), fan que esdevingui una solució radicalment diferent a la proposada per altres alternatives com Spring o Struts. De fet, la documentació oficial del framework ofereix tutorials per a permetre conviure dins la mateixa aplicació Java Server Faces amb altres solucions per a la capa de presentació.

L'experiència ha estat positiva, millor fins i tot que lo esperat abans d'estudiar i posar en marxa la llibreria. La seva configuració resulta còmoda i intuïtiva, i el codi resultant és altament comprensible i permet una clara separació de rols dins d'un departament de desenvolupament.

Per altra banda, una de les seves grans virtuts (la llibreria de tags) pot esdevenir també un dels grans arguments en contra, ja que condiciona molt a l'encarregat de dissenyar la interfície en quant a dissenyar les vistes, sempre lligat al codi generat pels esmentats taglibs, i sense més possibilitat per a la personalització que fer servir un sistema de temes configurables proposat pel propi framework.

4.4.4. Grails

4.4.4.1. Identificació i història

Grails, nom que ha evolucionat de l'original "Groovy on Rails", és un framework per a aplicacions web implementat sobre el llenguatge de programació Groovy, compatible amb les màquines virtuals Java.

Situem el seu inici a l'any 2003, quan arrenca el projecte i es fa visible als cercles de desenvolupament. El seu creador abandona el projecte a 2007, poc abans de que es publiqui la seva primera versió definitiva que en un curt període de temps evolucionaria fins a la 1.5. Dos anys després de la seva marxa, aquest reconeix que el projecte mai hagués existit d'haver conegut els progressos de Scala, un altre llenguatge que ofereix unes característiques similars a les que Groovy en general i Grails en particular volen defensar.

Groovy pertany inicialment a la companyia G2ONE, però aquesta va ser adquirida per SpringSource (responsable del framework Spring), i aquesta a la seva vegada és actualment propietat de VMWare (companyia especialitzada en virtualització de sistemes).

4.4.4.2. Groovy

Groovy és un llenguatge dinàmic que posposa fins el temps d'execució moltes de les fases que en els llenguatges més convencionals queden reservades per a la fase de compilació. En una mateixa acció el codi font es prepara i executa com a bytecode Java, sense deixar cap sortida intermèdia com ara classes precompilades.

Pel seu propi paradigma i gràcies a la seva compatibilitat en ambdues direccions en Java, una considerable quantitat de desenvolupadors del llenguatge de Sun ha adoptat Groovy com a llenguatge complementari, considerant-lo en ocasions un llenguatge de scripting per treballar conjuntament amb el Java original.

El llenguatge neix amb una molt clara premissa: facilitar la tasca de desenvolupador, alliberant-lo de tasques com ara configuració o preparació d'un entorn de desenvolupament. En resum, més automatització, menys configuració. Aquest principi queda especialment visible en el cas del framework Grails.

4.4.4.3. Estil i arquitectura

Grails es basa en el clàssic paradigma Model Vista Controlador. En canvi, l'experiència de desenvolupar amb aquest framework és molt diferent i, si es permet l'expressió, molt més senzilla que amb molts altres frameworks que segueixen el mateix paradigma.
Els principals arguments de l'existència de Grails són el foment d'una eina idònia per a l'alta productivitat, amb una notable presència de reutilització de components i sempre amb el principi de "Don't Repeat Yourself" (DRY) en l'horitzó. El framework aprofita certes eines ja suficientment provades i acceptades per la comunitat, com Hibernate o Spring.

Tot i que els seus principis fan que l'arquitectura interna de Grails no sigui tan visible durant el desenvolupament, si es pot afirmar que conserva una estructura clàssica on un controlador central s'encarrega de redirigir la informació cap als controladors propis del desenvolupament, i aquestos a la seva vegada generen la resposta a la petició cap amb l'ajut d'una plantilla de vista.

4.4.4.4. Components del framework

I. Classes de domini

Les classes de domini són l'equivalent als Beans de Java, exercint de contenidor per al model de dades. Les eines incorporades a Grails permeten que, només amb la classe de domini, sigui possible generar automàticament tot un comportament típic de pàgines per llistat, creació, edició i esborrat de registres del tipus del domini, en la tècnica coneguda com "scaffolding".

El següent fragment codifica la classe de domini per un empleat:

```
class Employee {
    Department department;
    String firstName;
    String lastName;
    int age;
}
```

II. Controller

Els controladors s'encarreguen de definir de forma agrupada les accions que implementa el sistema. Com veurem més endavant, el seu mapeig queda definit per la ubicació del propi controlador dins l'estructura del projecte, i en el seu codi queden establertes totes les accions que s'instal·len a partir d'aquest punt.

El següent codi pertany a un controlador que sobreescriu la versió "autogenerada" de l'acció per guardar un empleat.

```
class EmployeeController {
    def scaffold = Employee

    def save = {
        def employee = new Employee(params)
        if(!employee.hasErrors() && employee.save()) {
            flash.message = "Employee ${employee} created"
            redirect(action:show,id:employee.id)
        } else {
            render(view:'create',model:[employee:employee])
        }
    }
}
```

III. View

Les vistes de Grails són arxius .groovy amb format GSP (Groovy Server Pages), el qual es podria definir com una ampliació de les clàssiques JSP amb la inclusió de noves etiquetes i funcions creades a propòsit pel seu funcionament sota frameworks com Grails.

El següent fragment pertany a una vista que, entre d'altres, mostra una columna amb un desplegable per seleccionar el departament d'un empleat:

```
<td
 valign='top'
 class='value
${hasErrors(bean:employee,field:'department','errors')}'>
    <g:select
     optionKey="id"
     from="${Department.list()}"
     name='department.id'
     value="${employee?.department?.id}">
    </g:select>
</td>
```

4.4.4.5. Configuració

Quan parlem de configuració les grans virtuts de Grails surten a relluir, ja que precisament la posada en marxa immediata i l'automatització de certs processos són alguns dels principals atractius del framework.

Grails compta amb un servidor d'aplicacions integrat, per la qual el desenvolupador no necessita invertir el seu temps en la posada en marxa del seu propi servidor. La instal·lació de les aplicacions, com és de suposar, és també quasi totalment automàtica.

Per aconseguir rebaixar la càrrega de treball, Grails es basa únicament en la tècnica del conveni de noms. Per tant, no trobarem entre els seus requisits cap

fitxer extern XML ni cap mena d'anotació especial per tasques de posada a punt.

Per exemple, la manera d'indicar que una classe Groovy és un controlador es que el seu nom finalitzi amb "Controller". La URI base contra la que queden instal·lades les accions d'un controlador es decideix en funció de la ubicació física d'aquest controlador dins l'estructura de directoris del projecte. Les vistes es resolen per la clàssica estratègia de cercar arxius amb la forma [NomVista].gsp. Per definir components addicionals com ara taglibs, les classes que creem s'han de trobar dins la carpeta "taglib" del projecte.

Gràcies al seu caràcter de llenguatge sense fase de compilació, els components com ara controladors i taglibs afegeixen serveis de forma declarativa, de manera que no és necessària cap altre configuració per tal que el framework s'assabenti de quins recursos disposa el sistema.

4.4.4.6. Scaffolding

Sota el nom de "scaffolding" s'amaga una tècnica cada cop més estesa i que Grails, clarament influenciat per frameworks en altres llenguatges com ara Ruby on Rails, també implementa.

Podríem definir el scaffolding de Grails com la possibilitat d'utilitzar una implementació per defecte per les operacions més habituals que un sistema desitja realitzar sobre els objectes d'un domini. D'aquesta manera en, literalment, un parell de línies de codi la nostra aplicació disposarà d'un complet sistema de llistat, creació, edició i esborrat de registres per a un tipus de dada concret.

```
class EmployeeController {
    def scaffold = Employee
}
```

Aquesta funció allibera enormement al desenvolupador de tasques repetitives com ara pantalles amb un estil sempre igual en les que només varia el tipus de dada amb que s'opera. No obstant, Grails incorpora certes característiques que estenen la utilitat del scaffolding més enllà de les pàgines "senzilles", com ara la possibilitat de "heretar" i ampliar l'esquelet base.

En altres llenguatges com Ruby on Rails, si volem partir d'un esquelet autogenerat però amb les nostres modificacions, no ens queda més remei que importar el codi complet d'aquest esquelet per posteriorment aplicar els nostres canvis. En Grails podem optar per una solució intermèdia, en la que incloem l'esquelet amb les dos línies anteriorment esmentades, però estenem o sobreescrivim certs comportaments sense necessitat de copiar tot el controlador bàsic en el nostre codi.

De tota manera, la opció de generar estàticament el controlador i les vistes que la tècnica de scaffolding utilitza encara es troba disponible, si ho preferim així.

Validació de dades

La validació de dades de domini en Grails es basa en la definició de "constraints" dins d'aquestes mateixes classes de domini. Ubicada normalment immediatament després de la definició d'atributs, s'inclou un atribut static anomenat "constraints" on queden definides les regles que cada atribut del domini ha de complir.

```
class Employee {
    Department department;
    String firstName;
    String lastName;
    int age;

    static constraints = {
        firstName(blank:false, nullable:false, size:1..150)
        lastName(blank:false, nullable:false, size:1..150)
        age(min:18,max:120)
    }
}
```

Per realitzar la validació, tota classe de domini porta inclòs un mètode validate() que s'invoca per fer les pertinents comprovacions. Anàlogament, es disposa d'un mètode hasErrors() que contempla errors que es poden detectar més enllà de les constraint definides, com ara errors de conversió de dades procedents de la petició de l'usuari.

En ambdós casos, tenim un valor de retorn booleà que determina si el l'objecte de domini és correcte. A partir de l'atribut de l'objecte "errors" podem fer un recorregut pels diferents problemes detectats.

```
def user =  new User(params)

if(user.validate()) {
      …
} else {
user.errors.allErrors.each { println it }
}
```

En quant a la vista, Grails ja incorpora tota una sèrie d'etiquetes per a la gestió d'errors, com ara <g:renderErrors bean="xxx" />, <g:hasErrors bean="xxx" /> o <g:eachError var="vvv" bean="xxx">.

4.4.4.7. Característiques fora d'estudi

I. Cas pràctic

Dins el marc d'aquest projecte, la inclusió de Grails té un caràcter molt experimental en comparació amb la resta d'eines estudiades. El seu funcionament sota el llenguatge de programació Groovy fa que es pugui

classificar només parcialment com a Framework J2EE, basat únicament en una clara similitud de sintaxi i en la seva alta compatibilitat amb la plataforma Java.

No obstant, per l'objecte del nostre estudi la part que més ens interessa de Grails és el seu principi d'alta productivitat. La realització d'un cas pràctic amb una aplicació de prova no esdevé tan necessària en aquest cas, ja que al cap i a la fi tractaria de treballar amb unes tecnologies que no emprarem posteriorment quan dissenyem el nostre propi Framework J2EE.

Per aquest motiu, la presència de Grails dins el nostre projecte es limita al camp teòric i queda exclosa la implementació d'una aplicació de proves utilitzant com a base aquest framework.

4.4.4.8. Conclusions

Grails és un clar exemple d'una nova tendència al món dels frameworks: la d'alliberar al desenvolupador de tasques que fins ara requerien un alt percentatge dels recursos totals destinats a la creació d'una aplicació.

En un moment en que les tècniques per millorar la productivitat guanyen força, sembla lògic pensar que algunes de les idees i principis de frameworks com ara Grails es consolidaran i s'estendran a les noves versions d'altres frameworks fins el moment més "tradicionals".

Com era d'esperar en el moment d'incloure Grails en l'estudi d'eines, aquesta és la principal conclusió que extraiem del seu anàlisi: la necessitat d'implantar mecanismes en que el propi entorn alliberi al desenvolupador de tasques que ja no requereixen intervenció humana.

4.4.5. Conclusions de l'estudi

Com a darrera fase del nostre estudi de marcs de treball de presentació existents, fem una mirada general a totes les característiques, pros i contres observats i extraiem aquelles idees o tècniques que ens semblen més adients i per tant adoptarem per a la nostra pròpia proposta de framework.

Spring MVC ha estat una de les eines més completes i amb un estil més proper al que desitgem per a la nostra implementació. L'ús d'anotacions per estalviar tasques de configuració amb arxius XML és excel·lent, intuïtiu, usable i fàcil d'emprar. La manera en que es defineixen punts d'entrada del framework amb paràmetres d'entrada i sortida és molt propera a la que implementarem pel nostre cas.

En canvi, com a part negativa de Spring MVC tenim la seva excessiva dimensió fruit de patir una forta integració amb el seu framework "continent", i els problemes soferts durant la configuració de validacions dels valors d'entrada.

Un dels majors trets diferenciadors de Struts2 (també ho era de Struts1) és la seva gran virtut i alhora gran handicap: la forta dependència entre els controladors i l'àmbit on s'emmagatzemen les dades. Quan la interacció entre l'usuari i el sistema és complexa, aspectes com la conversió, manipulació i processament de les dades que viatgen entre ambdós actors esdevé dificultosa. Això és clarament un punt a evitar en el nostre framework ideal.

Java Server Faces aporta una notable quantitat d'aspectes interessants i dignes a ser candidats a la nostra proposta. El concepte de Managed Bean és interessant i combina, de forma molt més encertada que en el cas de Struts, la inclusió de dades a gestionar i mètodes amb lògica de negoci dins de la mateixa classe. A més a més, permetre disposar de nombrosos Managed Beans tots simultàniament accessibles des de la mateixa vista incrementa la potència d'aquesta.

En darrer lloc, només ens hem apropat a Grails des d'un punt de vista molt teòric, però suficient per remarcar els avantatges que suposa pel desenvolupador disposar de tot un conjunt de comportaments "automatitzats". Seria interessant, en la mesura de lo possible, incloure alguns serveis similars al "scaffolding" en el nostre propi framework.

4.5. Desenvolupament d'un framework: FFramework

4.5.1. Principis bàsics

Com a punt de partida del desenvolupament d'un marc de treball de presentació, una opció interessant és enumerar que es el que desitgem. Abans d'entrar a detalls tècnics i d'implementació, fer un primer esforç de síntesi i acotar allò que ha de tenir prioritat durant tot el procés. Només en el moment que es té clar el que es vol fer, estem en condicions de preguntar-nos com fer-ho.

A continuació s'enumeren les idees més rellevants que es desitja tenir presents durant el desenvolupament del nostre framework, algunes fruït d'una opinió personal, i d'altres com a conseqüència de les observacions i conclusions obtingudes a l'estudi previ d'eines ja existents al mercat.

4.5.1.1. Fàcil configuració

Dues paraules clau que han de predominar tant en el desenvolupament com a l'eina resultant són "usabilitat" i "simple", sempre referint-nos a l'experiència final que tindrà un desenvolupador web que vulgui implementar una aplicació utilitzant com a base de programari el nostre framework.

Aquesta línea de pensament ha de començar en la configuració. Si volem una eina fàcil d'utilitzar, aquesta ha de ser fàcil de configurar. I fins i tot, sempre que sigui possible, que ni tan sols requereixi una configuració explícita.

4.5.1.2. Innovació: més anotacions, menys herències

Sempre és desitjable fer ús de les tècniques i serveis més moderns dels que es disposen en el moment d'iniciar un desenvolupament. Això, aplicat al cas d'una configuració simple, ens porta directament a la decisió de donar prioritat a l'ús d'anotacions Java per sobre de tècniques més tradicionals com ara l'herència o l'ús d'arxius de configuració en format XML.

Si bé no sempre serà possible, desenvolupar una eina amb el nostre framework rarament ha de requerir que les classes desenvolupades heretin de cap interfície o classe abstracta. La manera d'assignar competències o peculiaritats a una classe, acció o paràmetre serà mitjançant anotacions degudament definides.

Si podrem, en canvi, prescindir totalment de l'ús d'arxius XML per a la configuració, tal i com hem observat en certs casos durant l'estudi d'eines com Spring o Struts.

4.5.1.3. No requerir molts arxius: convivència de dades i accions

Durant l'estudi de Struts, una de les observacions que varem extraure va ser la convivència de dades i accions en les mateixes classes Java. Això suposava al mateix temps un avantatge i un problema. Per la banda positiva, permetia reduir el nombre d'arxius que composen una aplicació i afavoria el principi "d'alta cohesió": les accions es trobaven fortament lligades a les dades que manipulen. Per altra banda, en el moment de la pràctica comportava molts problemes inesperats que perjudicaven el ritme de producció del desenvolupador, usualment errors relacionats amb l'actualització i conversió de les dades.

En el cas del nostre framework, tenim la intenció d'aprofitar les avantatges d'aquesta tècnica, però mirant de superar els problemes que presenten en el cas de Struts.

4.5.1.4. Si vols accedir, pots accedir

Un problema que hem trobat sovint en els casos pràctics d'eines existents eren les limitacions per accedir a dades que no estiguessin directament lligades a l'acció que s'estava executant. En un cas real, acostuma a ser habitual la necessitat d'accedir transversalment a les dades. De forma puntual, un mòdul de la nostra aplicació pot desitjar o veure's beneficiat de conèixer certa informació pertanyent a un segon mòdul del mateix sistema.

La nostra eina permetrà fàcilment cobrir aquesta necessitat, permetent que una acció pugui rebre com a entrada, a més de les dades implícites ja actualitzades en el seu propi contenidor, referències a altres contenidors del sistema amb les seves pròpies dades.

En un segon apartat i encara parlant d'accés a dades, un altre punt interessant és poder recuperar en cas de necessitat les dades "crues" que l'usuari ha enviat. És a dir, aquelles que queden fora de qualsevol tractament o conversió de tipus efectuada abans d'invocar a l'acció. Aquest era un dels punts més febles de Struts, en el que era necessària una feina addicional i poc agraïda per tal de conservar les dades originals que viatgen amb la petició d'executar una acció.

El nostre framework permetrà, sempre que el desenvolupador ho desitgi, accedir directament a la instància ServletRequest que ha originat la invocació de l'acció, de mode que sempre serà possible recuperar les dades originals de la petició.

4.5.1.5. Una vista, molts contenidors

Una de les característiques que més destacàvem a l'estudi de Java Server Faces era la possibilitat de, des d'una mateixa vista, poder accedir a dades i accions ubicades a diferents components dels que fa ús l'aplicació. Així

fomentava l'accés transversal a dades i feia molt més flexible el desenvolupament.

En el nostre framework tindrem cura de que qualsevol vista pugui accedir a dades i accions de qualsevol dels controladors definits.

4.5.1.6. Potenciar la vista amb automatismes

Quan es tracta de dissenyar i implementar vistes, no són poques les tasques repetitives que ens trobem. Qüestions com obtenir el valor de una dada o habilitar un camp de formulari per actualitzar-la poden complicar-se degut a un excés de comprovacions prèvies necessàries per no caure en excepcions com la habitual NullPointerException.

En el cas de les JSP, aquest obstacle es pot superar gràcies a les diferents llibreries d'etiquetes (TagLibs) que els frameworks ofereixen. Pel nostre cas, on la tecnologia de vistes per defecte serà FreeMarker, disposem de les anomenades "FreeMarker Directives", que permeten implementar funcions (a vegades anomenades "macros") que posteriorment poden ser invocades des de les plantilles.

4.5.2. Tecnologies clau

Passem ara a enumerar les tecnologies que cal conèixer i dominar per entendre el funcionament de la nostra eina, ja que en elles radica gran part de la implementació.

4.5.2.1. Anotacions Java

Com hem esmentat al enumerar els principis de la nostra eina, farem ús d'anotacions Java per tal de simplificar les tasques de configuració i permetre que aquesta estigui fortament lligada a allò que es configura.

Les anotacions Java són interfícies especials que es declaren amb el prefix "@", i permeten acompanyar paquets, classes, atributs, mètodes o paràmetres per tal d'aportar informació addicional. Aquest component existeix des de la versió 1.5.

Un exemple d'una anotació definida per aportar informació sobre una classe seria la següent:

```
@MyAnnotation(type="HighPriority", priority=5)
public class MyClass...
```

Per crear una anotació Java, cal seguir la sintaxi següent:

```
@interface Nom_anotacio {
      Tipus_atribut_1 nom_atribut_1() [default valor_per_defecte_1];
      ...
      Tipus_atribut_N nom_atribut_N() [default valor_per_defecte_N];
}
```

És a dir, una anotació té un nom i una sèrie d'atributs, que poden ser obligatoris o opcionals (en aquest cas, es proporciona un valor per defecte). Els atributs poden ser de qualsevol tipus, des d'un String fins a un Class, o fins i tot arrays o col·leccions.

És important conèixer certes anotacions ja predefinides que permeten indicar certes característiques de les anotacions que creem. Per exemple, l'anotació @Target ens permet restringir què elements pot acompanyar la nostra anotació, i l'anotació @Retention permet configurar que les anotacions es mantinguin després de la compilació del codi, ja que el seu comportament per defecte és no sobreviure quan es genera el bytecode de Java.

Per exemple, la següent anotació sobreviu més enllà de la compilació, pot acompanyar a classes i mètodes, i té dos atributs: un nom obligatori, i un nombre sencer opcional.

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD,ElementType.TYPE})
public @interface MyAnnotation {
      String type();
      int priority() default 0;
}
```

4.5.2.2. Java Reflection

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

En una eina de la naturalesa d'un framework, és habitual emprar els serveis que ofereix aquest paquet per tal d'estudiar les classes de l'aplicació client.

El nostre framework farà servir las classes del paquet java.lang.reflect per detectar anotacions existents, tipus de dades, valors de retorn, invocar mètodes, etc.

4.5.2.3. Reflection avançada per conèixer genèrics

Des de la versió 5, Java permet l'ús del que han decidit anomenar "Genèrics". Sota aquest nom trobem una sèrie de tècniques que ens permeten concretar

més sobre certs tipus de dades en temps de compilació. Per exemple, el que fins el moment era una llista “List” que incloïa objectes de qualsevol tipus, ara pot ser una llista “List<Integer>” definida per acceptar exclusivament tipus enters.

Aquesta peculiaritat, si bé un gran avantatge pel desenvolupador, esdevé una complicació quan desenvolupem una “meta-eina” com ara un framework, especialment en el cas de la necessitat de convertir dades entre tipus. Per exemple, podem disposar d’un array de cadenes de text (String) i sabem que el seu destí és una llista, però de quin tipus han de ser els objectes que afegim a aquesta col·lecció?

Per a tractar aquest cas és necessari fer un ús més complex dels serveis que ofereix la API de Reflection, procés que queda exposat en un bloc de l’aparat “Diari de desenvolupament” d’aquest mateix document.

4.5.2.4. FreeMarker

FreeMarker és un motor de generació de plantilles. Utilitza una sintaxi molt propera a Java, per la qual cosa assolir un domini suficient no esdevé un problema per desenvolupadors que ja acostumen a treballar amb aquesta tecnologia.

Els passos que cal configurar per treballar amb FreeMarker són escassos i senzills: només cal incloure la llibreria en el projecte, i crear inicialment una instància de configuració on indicar els aspectes necessaris, com ara la ubicació base de les plantilles, detalls de la codificació o quins afegits incloure en el procés de generació.

Un cop configurat, donat un arxiu de tipus FreeMarker template (usualment amb extensió .tpl) només cal indicar a aquesta instància de configuració la seva ubicació i el conjunt de dades que volem incloure al contenidor que el motor utilitza per resoldre els valors dinàmics de la plantilla.

Una característica de FreeMarker és la seva política altament restrictiva amb els valors nuls. Un cas habitual en motors de generació és que, si en resoldre un valor aquest no és troba inicialitzat o no existeix, es decideix utilitzar un valor considerat “buit”, com pot ser una cadena de text buida. No és el cas de FreeMarker, que per un cas així provocarà una excepció i no continuarà amb el processat de la plantilla. Això, que pot ser interpretat com un argument en contra, és també una virtut ja que evita la propagació d’errors (molts d’ells tipogràfics) que queden amagats ja que no es pot distingir si un valor buit és realment el que esperàvem o bé és conseqüència d’una referència incorrecta.

Fins a quin punt aprofitar la potència de FreeMarker queda a decisió del desenvolupador final que construeix una aplicació web sobre el nostre framework. Per part de la pròpia eina no s’imposa cap restricció, i l’única funcionalitat més avançada que s’utilitza és la de definir “directives”, que podríem considerar l’equivalent a les JSP TagLibs.

4.5.3. Casos d'ús

En una aproximació a lo que consideraríem la fase d'anàlisi d'un desenvolupament de software, fem ara una enumeració de totes les funcionalitats que el nostre framework oferirà al desenvolupador que ho empri per construir la seva pròpia aplicació web. Aquestes funcionalitats s'exposen estructurades en forma de casos d'ús.

4.5.3.1. CUS1. Instal·lar framework

Precondició:

El desenvolupador disposa d'un projecte d'aplicació web.

Postcondició:

L'aplicació habilita un punt d'accés base per a peticions dirigides al framework, inicialment sense cap acció ni contenidor definit.

Seqüència d'esdeveniments:

1. El desenvolupador afegeix a les llibreries del projecte els .jar corresponents al framework i les seves dependències.
2. El desenvolupador afegeix una entrada al descriptor web.xml corresponent al controlador principal del framework, indicant opcionalment la ruta base on es trobaran disponibles les classes que el framework deu explorar.
3. El desenvolupador afegeix una entrada al descriptor web.xml corresponent al punt d'instal·lació del framework. La ruta indicada serà aquella de la qual l'eina capturi i processi les peticions.
4. El sistema, bé en arrencar el servidor d'aplicacions o bé en rebre una primera petició (segons el paràmetre load-on-startup), executarà el codi d'inicialització del controlador principal que configura el model de dades del framework.

4.5.3.2. CUS2. Afegir contenidor de dades

Precondició:

El projecte disposa de les llibreries del framework i aquest es troba correctament instal·lat.

Postcondició:

S'ha habilitat un contenidor de dades, identificar per un nom, que pot albergar atributs o accions.

Seqüència d'esdeveniments:

1. El desenvolupador crea una classe Java sense cap mena de requisits de herència ni implementació d'interfície. Si en CUS1 s'ha indicat un paquet arrel de classes del framework, ubica aquesta classe en aquest paquet o bé un subpaquet de nivell inferior.
2. El desenvolupador afegeix a la capçalera de la nova classe l'anotació que la defineix com a contenidor de dades, incloent un nom que l'identificarà en futures referències i a quin àmbit correspon (sessió o petició). Si no especifica un nom, se li assignarà un derivat del nom de la pròpia classe. Si no especifica un àmbit, es considerarà de sessió.
3. Quan s'inicialitza, el framework detectarà l'existència d'un contenidor de dades implementat en la nova classe i la processarà i instal·larà sota el nom corresponent, respectant el cicle de vida corresponent a l'àmbit d'aquest.

Errors:

3e. Si ja s'havia definit prèviament un contenidor amb el mateix nom, el framework genera un error i avorta la seva inicialització.

4.5.3.3. CUS3. Afegir dades d'entrada i sortida

Precondició:

El framework es troba configurat i existeix un contenidor de dades habilitat segons les passes de CUS2.

Postcondició:

S'han afegit dades d'entrada i sortida dins d'un dels contenidor de dades.

Seqüència d'esdeveniments:

1. El desenvolupador identifica una classe corresponent a un contenidor de dades, definida tal i com descriu CUS2.
2. El desenvolupador afegeix una sèrie d'atributs a la classe contenidora, essent aquestos de tipus bàsics (cadenes, numèrics, booleans), temporals (Date), complexos (objectes POJO), arrays, o col·leccions (genèriques o no).
3. El desenvolupador crea els mètodes getter i setter corresponents als atributs que són visibles per al controlador.
4. Quan es rep una petició al framework, aquesta és susceptible d'incloure dades que han d'actualitzar els atributs creats. De la mateixa manera, les vistes poden consultar i mostrar els valors d'aquestos atributs.

4.5.3.4. CUS4. Afegir punts d'entrada o accions

Precondició:

El framework es troba configurat i existeix un contenidor de dades habilitat segons les passes de CUS2.

Postcondició:

S'ha afegit una nova acció que pot ser invocada en una petició dirigida al framework.

Seqüència d'esdeveniments:

1. El desenvolupador identifica una classe corresponent a un contenidor de dades, definida tal i com descriu CUS2.
2. El desenvolupador afegeix un mètode públic sense paràmetres d'entrada i que retorna una cadena de text. No hi ha cap restricció respecte al cos del mètode.
3. El desenvolupador afegeix a la capçalera del mètode l'anotació que el defineix com a acció del controlador, incloent el nom que la identificarà en futures referències. Si no especifica nom, el framework utilitzarà el nom del propi mètode.
4. Quan es rep una petició al framework, la URI original pot fer referència a l'acció creada i, si és el cas, el controlador principal invocarà a aquest mètode després d'haver actualitzat les dades d'entrada.

Errors:

4e.1. Si ja existeix una acció amb aquest nom dins del mateix contenidor de dades, el framework genera un error i avorta la seva inicialització.
4e.2. Si l'acció no retorna el tipus de dada esperat, el framework genera un error i avorta la seva inicialització.

4.5.3.5. CUS5. Afegir paràmetres d'entrada a una acció

Precondició:

El framework es troba configurat i existeix un contenidor amb accions habilitat segons les passes de CUS2 i CUS4.

Postcondició:

S'ha modificat una acció per tal que rebi paràmetres explícits a més dels ja implícits del propi contenidor de dades.

Seqüència d'esdeveniments:

1. El desenvolupador identifica una classe corresponent a un contenidor de dades definida tal i com descriu CUS2 i, entre els seus mètodes, una acció del framework definida tal i com descriu CUS4.
2. El desenvolupador afegeix paràmetres al mètode en qüestió, limitant-se sempre a tres orígens de dades possible indicant cadascun mitjançant l'anotació Java creada per aquest propòsit:

- Dada independent procedent de la petició però sense relació amb cap contenidor de dades.
- Instància de la petició sencera rebuda pel servidor, sense passar per cap filtre del framework.
- Referència a un contenidor de dades diferent de l'implícit.

3. Per a cada possible origen de dades, el tipus de dada del paràmetre ha de ser compatible amb l'origen escollit.
4. Quan una petició provoqui la invocació de l'acció modificada, el framework resoldrà els valors corresponents a cadascun dels paràmetres creats.

Errors:

3e.1. Si l'origen de dades indicat a un paràmetre és desconegut, el framework genera un error i avorta la seva inicialització.
3e.2. Si el tipus de dada d'un dels paràmetres no es compatible amb l'origen de dades indicat, el framework genera un error i avorta la seva inicialització.

4.5.3.6. CUS6. Afegir i instal·lar conversors globals

Precondició:

El framework es troba instal·lat.

Postcondició:

Queda configurat per a tota la aplicació un tipus de conversió d'entrada diferent del que el framework proposa per defecte.

Seqüència d'esdeveniments:

1. El desenvolupador crea una nova classe que implementa la interfície de conversor d'entrada definida pel framework. En la seva creació, a més de la pròpia conversió, indica quin tipus de dada converteix.
2. El desenvolupador afegeix, a la capçalera de la classe, l'anotació corresponent a definir un conversor global de dades aplicable a tot el sistema.
3. Quan el framework rep una petició i codifica les seves dades d'entrada, emprarà com a darrera opció el conversor global definit per cada tipus si no ha trobat abans un conversor específic definit a nivell d'acció o contenidor.

Errors:

3b. Si ja s'ha definit prèviament un conversor de dades capaç de tractar el tipus indicat, el sistema notifica el conflicte i dona prioritat al primer conversor definit, descartant el darrer especificat.

4.5.3.7. CUS7. Afegir i instal·lar conversors específics

Precondició:

El framework es troba instal·lat i existeix un contenidor de dades segons allò exposat a CUS2 i CUS4.

Postcondició:

Queda configurat per un contenidor de dades o bé per una acció concreta un tipus de conversió d'entrada diferent del que el framework proposa per defecte.

Seqüència d'esdeveniments:

1. El desenvolupador crea una nova classe que implementa la interfície de conversor d'entrada definida pel framework. En la seva creació, a més de la pròpia conversió, indica quin tipus de dada converteix.
2. El desenvolupador identifica una classe definida com a contenidor de dades segons allò exposat a CUS2.
3. El desenvolupador afegeix, a la capçalera de la classe, l'anotació corresponent a definir conversors de dades explícits per a les accions del contenidor, afegint a la col·lecció de conversors el que acaba de crear.
4. Quan el framework rep una petició dirigida cap a una acció del contenidor implicat, les dades d'entrada (ja siguin atributs de contenidor o be paràmetres de l'acció) del tipus que accepta el conversor utilitzaran aquest per codificar les dades enviades per l'usuari.

Alternatives:

3b. El desenvolupador identifica una acció concreta dins del contenidor de dades i aplica l'anotació de conversors explícits a la seva capçalera. A partir d'aquest moment, el cas d'ús segueix el seu camí convencional però veient-se afectat per la nova norma només les peticions dirigides a aquesta acció concreta, i no a qualsevol acció del contenidor.

Errors:

4b. Si pel contenidor o acció específic ja s'ha definit prèviament un conversor que accepta el mateix tipus de dada, el sistema notifica el conflicte i dona preferència al primer conversor definit, descartant el darrer en detectar.

4.5.3.8. CUS8. Afegir i instal·lar validadors de dades

Precondició:

El framework es troba configurat i existeix un contenidor de dades amb una acció instal·lada tal com defineixen CUS2 i CUS4.

Postcondició:

L'acció incorpora una fase de validació de dades prèvia a l'execució del seu contingut.

Seqüència d'esdeveniments:

1. El desenvolupador crea una nova classe que implementa la interfície de validador de dades definida pel framework.
2. El desenvolupador identifica un contenidor de dades amb una acció definides segons allò exposat a CUS2 i CUS4.
3. El desenvolupador afegeix a la capçalera del mètode corresponent a l'acció una anotació que defineix quin validador de dades precedeix a la seva execució, indicant com a classe validadora la creada anteriorment.
4. Quan el framework rep una petició dirigida cap a l'acció implicada, després de la fase de conversió de dades però abans de la invocació de l'acció crida al validador configurat. En el cos del validador, el desenvolupador afegeix els eventuals errors i configura la vista encarregada de mostrar els errors en cas d'existir.

4.5.3.9. CUS9. Afegir i instal·lar gestors d'excepcions globals

Precondició:

El framework es troba instal·lat.

Postcondició:

Queda instal·lat un gestor d'excepcions que, en cas que una acció provoqui una excepció dels tipus que accepta, serà l'encarregat de processar-la si no existeix un altre gestor més específic a nivell de contenidor o acció.

Seqüència d'esdeveniments:

1. El desenvolupador crea una nova classe que implementa la interfície de gestor d'excepcions, definit el seu comportament i la relació de tipus d'excepcions que és capaç de processar.
2. El desenvolupador afegeix a la capçalera de la classe l'anotació que defineix que es tracta d'un gestor d'excepcions global que afecta a tot el sistema.

3. El framework enregistra la classe com a gestor d'excepcions global i aquesta romandrà disponible per processar excepcions del tipus definit en cas de produir-se i no existir cap altre gestor a un nivell més específic.

4.5.3.10. CUS10. Afegir i instal·lar gestors d'excepcions específics

Precondició:

El framework es troba instal·lat i existeix un contenidor de dades amb alguna acció tal com defineixen CUS2 i CUS4.

Postcondició:

Queda instal·lat un gestor d'excepcions que, en cas de fallida a la invocació de l'acció indicada o una acció qualsevol del contenidor indicat, serà l'encarregat de processar-la si accepta el seu tipus.

Seqüència d'esdeveniments:

1. El desenvolupador crea una nova classe que implementa la interfície de gestor d'excepcions, definit el seu comportament i la relació de tipus d'excepcions que és capaç de processar.
2. El desenvolupador identifica un contenidor de dades existent i definit segons les pautes de CUS2.
3. El desenvolupador afegeix a la capçalera del contenidor l'anotació que permet indicar els gestors d'excepcions actius per a les accions del contenidor, i afegeix una referència a la classe creada.
4. Quan es produeix una excepció durant la invocació d'una acció del contenidor, si el tipus es troba entre els acceptats el gestor serà l'encarregat de processar-la.

Alternatives:

3b. El desenvolupador identifica una acció concreta del contenidor i afegeix a la seva capçalera la anotació que fa referència al gestor d'excepcions. A partir d'aquest moment, el gestor s'aplica només a l'acció concreta, i no totes les del contenidor.

Errors:

4b. Si el framework detecta que per un dels tipus d'excepció acceptats ja existeix un gestor al mateix nivell capaç de processar-lo, notifica el conflicte i dona preferència al primer definit.

4.5.3.11. CUS11. Afegir directives pròpies

Precondició:

El framework es troba instal·lat tal com defineix CUS1, indicant (o utilitzant l'arrel altrament) un paquet base on es troben les classes afectades.

Postcondició:

S'ha afegit una directiva de vista que pot ser invocada des de les vistes ofertes al framework.

Seqüència d'esdeveniments:

1. El desenvolupador implementa una directiva de FreeMarker, segons les passes indicades a la documentació de la pròpia llibreria.
2. El desenvolupador afegeix a la classe de la directiva l'anotació que determina que es tracta d'una directiva que pot ser invocada des de les vistes. Indica explícitament el nom amb el que pot ser referida.
3. El framework, en el moment de la seva inicialització, troba la classe anotada i la instal·la en el motor de plantilles per tal que la directiva corresponent pugui ser invocada amb el nom indicat.

Errors:

3e. Si la classe anotada per indicar que és una directiva no implementa la interfície definida pel motor de plantilles, el framework genera un error i avorta la seva inicialització.

4.5.3.12. CUS12. Afegir vista

Precondició:

El framework es troba instal·lat amb un contenidor de dades i una acció definides segons CUS2 i CUS4.

Postcondició:

Queda afegida una plantilla de vista que serà carregada i processada quan una acció retorni la seva referència.

Seqüència d'esdeveniments:

1. El desenvolupador crea un arxiu de plantilla seguint la sintaxi del motor de vistes que utilitza el framework.
2. El desenvolupador dona nom a la plantilla amb l'extensió corresponent i el desa al directori prèviament definit com a magatzem de plantilles.

3. Quan una acció demanada al framework retorni com a resultat de la seva execució un identificador de vista igual al nom de la plantilla, carregarà i processarà l'arxiu per a generar la vista en resposta a la petició de l'usuari.

4.5.4. Classes del sistema

Com a primera aproximació al framework que volem implementar, definim les classes que compondran el sistema aportant una breu descripció de les competències de cadascuna d'elles.

4.5.4.1. Diagrama de classes

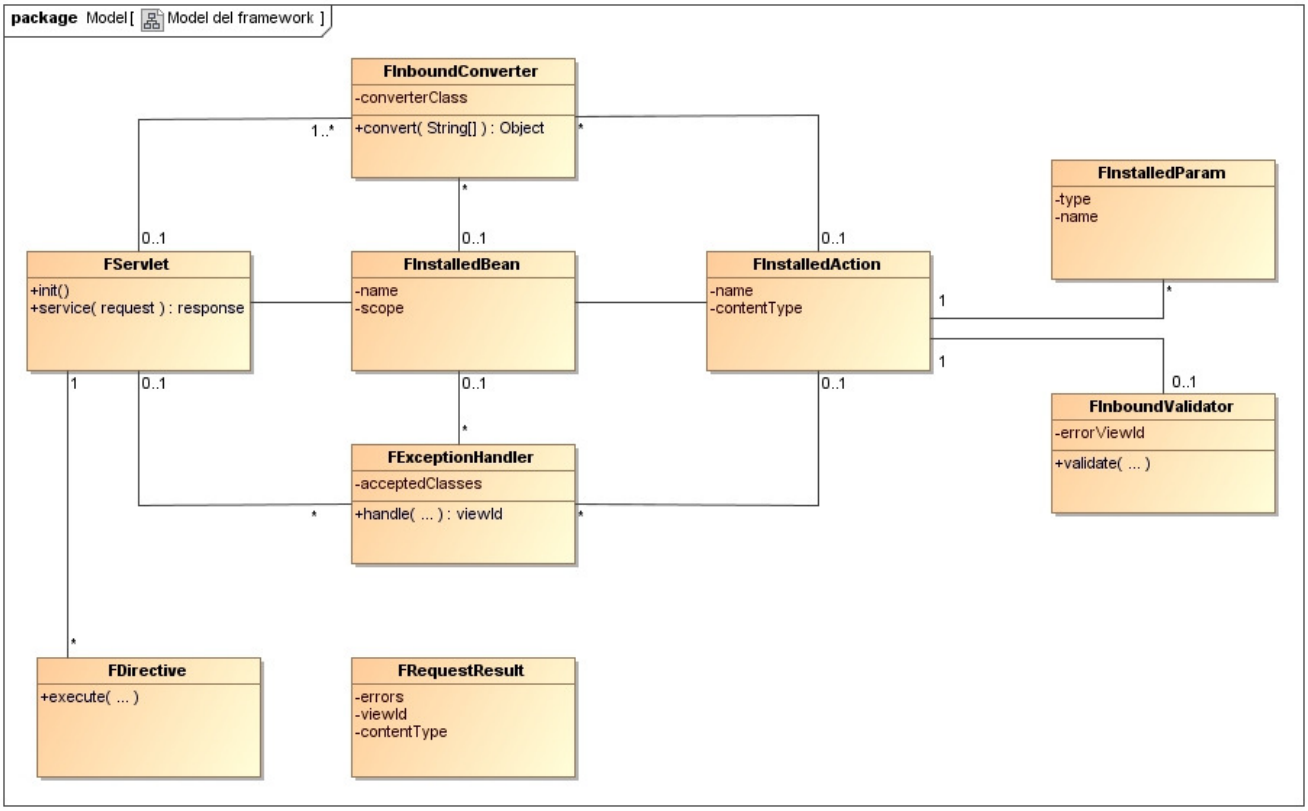

Figura 13. Diagrama de classes de FFramework.

4.5.4.2. FServlet

És el component central del framework i l'encarregat d'implementar el patró FrontController. Com a eix central del sistema, inclou l'arbre de components instal·lats sobre el framework i s'encarrega de rebre les peticions de l'usuari per generar una resposta. La major part del codi del nostre framework forma part de les rutines d'inicialització i servei de peticions d'aquest Servlet.

4.5.4.3. FInstalledBean

Representa els controladors desplegats sobre el framework, i poden exercir alhora les funcions de contenidor de dades (per la qual cosa inclou atributs) i contenidor d'accions. Tenen un cicle de vida definit per l'àmbit al que pertanyen: sessió o petició. Ja en aquest nivell queden definits conversors de dades i gestors d'excepcions que afecten en cascada a totes les accions definides en el seu interior.

4.5.4.4. FInstalledAction

Equival als punts d'entrada que el framework habilita. Tota acció es troba definida en el context d'un controlador (bean), i pot definir els seus propis conversors de dades i gestors d'excepcions que tenen preferència sobre els definits pel controlador. Opcionalment, una acció pot definir un procés de validació previ i una sèrie de paràmetres d'entrada.

4.5.4.5. FInstalledParam

Suposa els paràmetres d'entrada que accepta una acció. L'origen d'aquestos paràmetres es limita a tres possibilitats: tota la petició de l'usuari, un únic valor d'entrada de l'usuari o tot un controlador (per tal que la acció pugui accedir a dades d'un contenidor que no és al que pertany).

4.5.4.6. FInboundConverter

Són els conversors necessaris per traduir els valors que l'usuari envia en forma de cadenes de text. El conversor rep la cadena i la transforma en una instància del tipus desitjat. Es poden definir conversors a nivell de tota l'aplicació, un sol controlador o una sola acció, sent més preferent aquell d'àmbit més concret.

4.5.4.7. FInboundValidator

Representa els processos de validació de dades previs a l'execució d'una acció del framework. Tenen accés a les mateixes dades que l'acció que precedeixen, és a dir, els atributs del controlador al que pertany l'acció, i els paràmetres que la pròpia acció defineix. El procés de validació pot tenir efecte en la resposta generada pel framework, afegint errors o decidint la vista encarregada de donar resposta.

4.5.4.8. FExceptionHandler

Fragments de codi encarregats de processar una excepció generada durant l'execució d'una acció del framework. Tenen accés, a més de a la pròpia excepció, a les mateixes dades que pot accedir l'acció que l'ha provocat. Poden

decidir propagar l'error original o bé assumir que ja s'ha tractat i informar de la vista encarregada de donar resposta.

4.5.4.9. FRequestResult

Conté tota la informació relativa a l'execució d'una acció del framework, és a dir: els possibles errors de conversió i validació, la vista encarregada de donar resposta, i el format de la resposta que s'ha de generar. Cada FRequestResult neix i mor en el procés d'una petició de l'usuari.

4.5.4.10. FDirective

Fragments de codi que permeten ampliar la potències de les vistes, afegint funcionalitats avançades, essencialment, d'accés a dades contingudes pels controladors.

4.5.5. Inicialització del sistema

Passem ara a definir la seqüència d'esdeveniments que tenen lloc quan s'inicialitza el framework, resumits en com l'eina analitza l'aplicació per detectar elements i desplegar-los al sistema. S'inclou un guió informal amb una descripció dels passos que segueix el component FServlet (peça central del framework) durant aquest procés.

4.5.5.1. Diagrama de seqüència

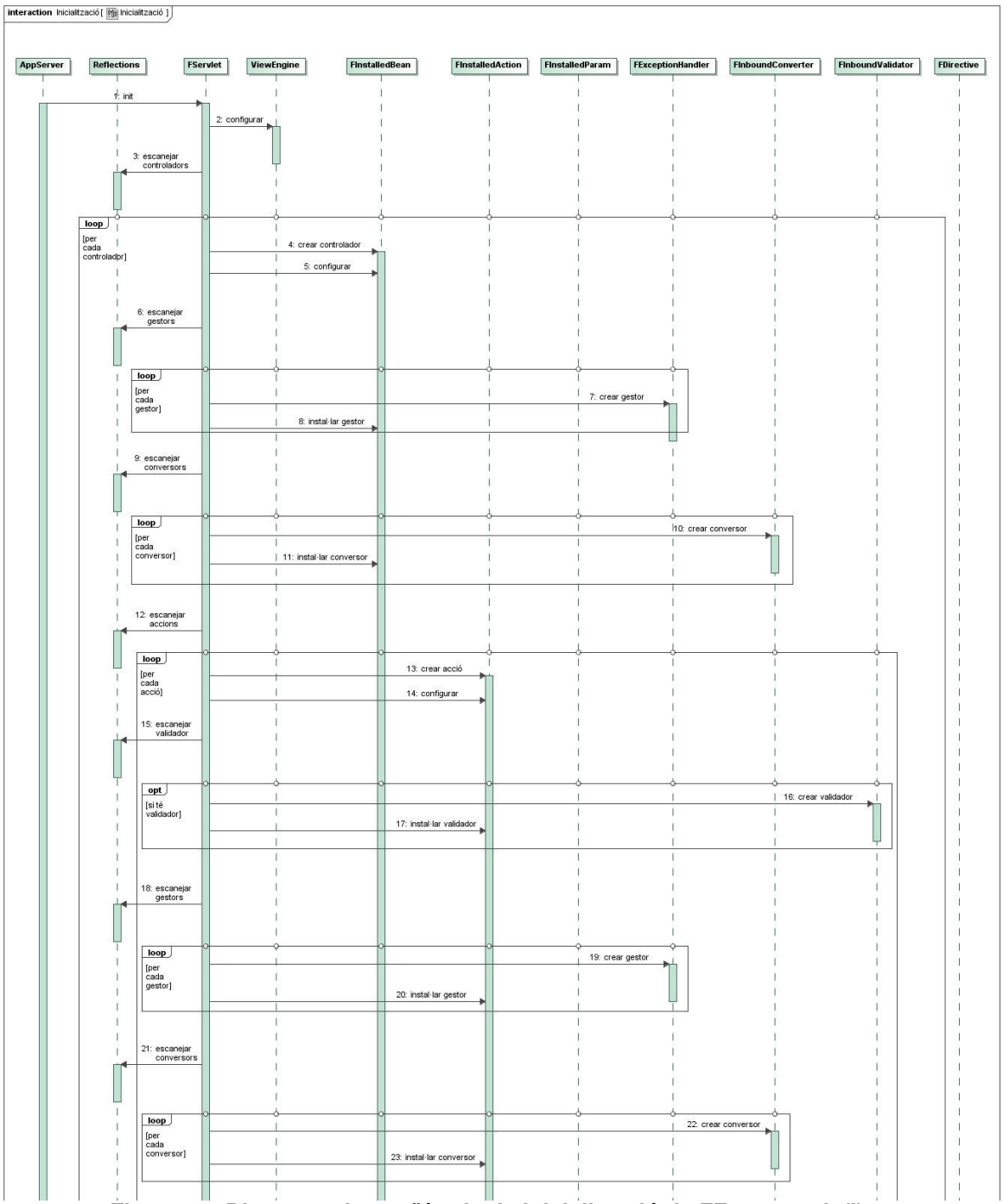

Figura 14. Diagrama de seqüència de inicialització de FFramework (I).

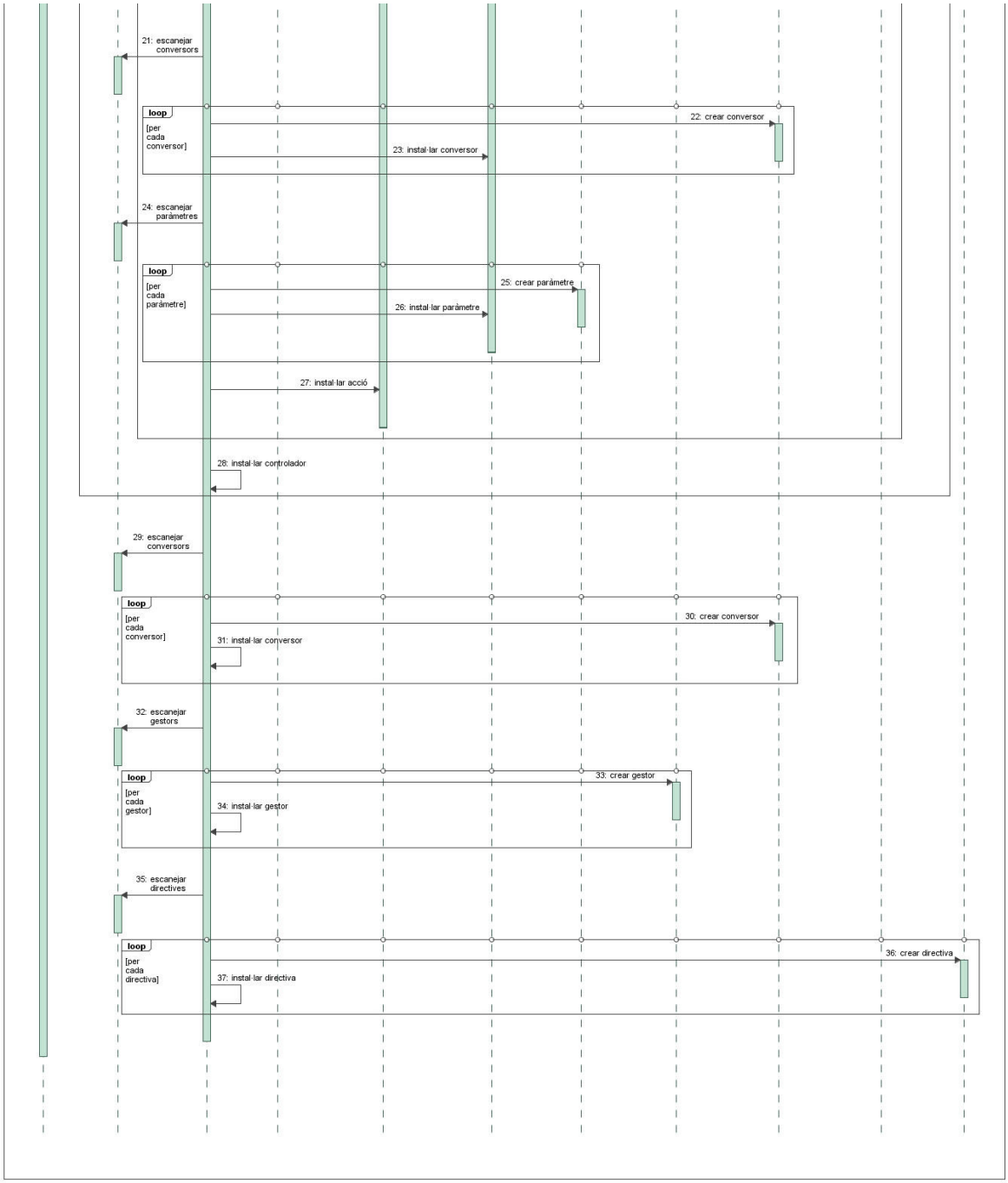

Figura 15. Diagrama de seqüència de inicialització de FFramework (II).

4.5.5.2. Descripció de la inicialització

1. Ordre d'inicialitzar el sistema.
2. Configurar el motor de generació de vistes (rutes, directives, etc.) [2].
3. Escaneig del sistema cercant candidats a controlador [3].
4. Per a cada candidat:
   4.1. Creació i configuració de la instància de controlador (nom, àmbit, etc.) [4 i 5].
   4.2. Cerca i instal·lació de gestors d'excepcions de l'àmbit del controlador [6, 7 i 8].
   4.3. Cerca i instal·lació de conversors de dades de l'àmbit del controlador [9, 10 i 11].
   4.4. Cerca de candidats a acció oferta pel controlador [12].
   4.5. Per a cada candidat:
      4.5.1. Creació i configuració de la instància d'acció (nom, tipus de resposta, etc.) [13 i 14]
      4.5.2. Detecció i instal·lació de validador previ a l'execució [16 i 17].
      4.5.3. Cerca i instal·lació de gestors d'excepcions de l'àmbit de l'acció [18, 19 i 20].
      4.5.4. Cerca i instal·lació de conversors de dades de l'àmbit de l'acció [21, 22 i 23].
      4.5.5. Cerca i instal·lació de paràmetres que accepta l'acció [24, 25 i 26].
      4.5.6. Instal·lació de l'acció dins del controlador [27].
   4.6. Instal·lació del controlador dins del sistema [28].
5. Cerca i instal·lació de conversors de dades de l'àmbit de l'aplicació [29, 30 i 31].
6. Cerca i instal·lació de gestors d'excepcions de l'àmbit de l'aplicació [32, 33 i 34].
7. Cerca i instal·lació de directives de la vista pel motor de generació [35, 36 i 37].

4.5.6. Processat de petició

Definim ara la sèrie d'esdeveniments que tenen lloc quan el framework rep una petició per part de l'usuari i cal actualitzar dades dels contenidors i executar una acció.

4.5.6.1. Diagrama d'activitat

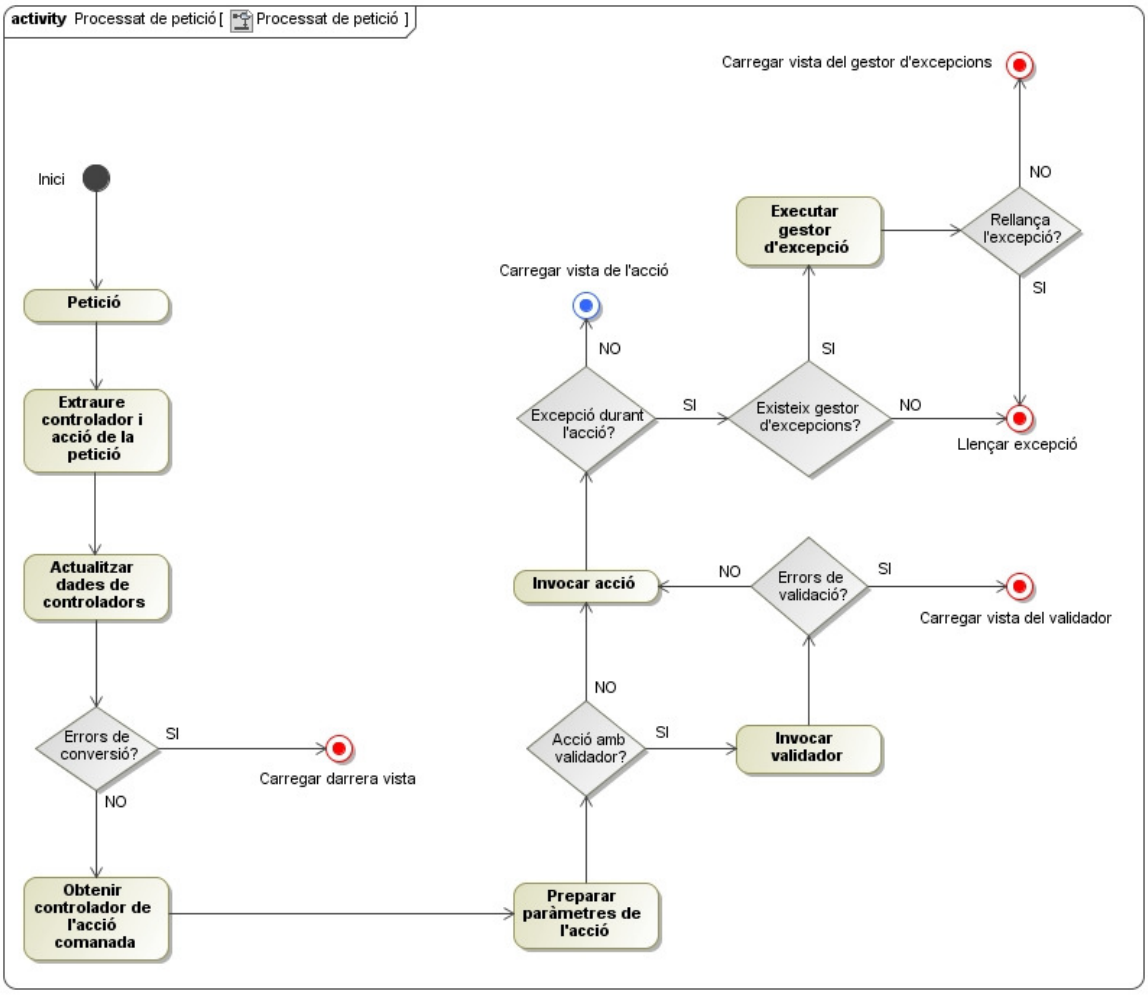

Figura 16. Diagrama d'activitat del processament d'una petició a FFramework.

4.5.6.2. Descripció del processat

A continuació s'explica breument en què consisteix cada acció i decisió exposada en el diagrama anterior:

- Petició: el framework rep una petició HTTP de l'usuari.

- Extraure controlador i acció de la petició: processar el contingut de la petició de l'usuari per concloure quina acció de quin controlador s'està comanant.

- Actualitzar dades de controladors: repassar el contingut de la petició de l'usuari per detectar valors que han de modificar els atributs dels controladors de l'aplicació, i executar l'actualització en base als conversors de dades instal·lats.

- Errors de conversió? Determina si s'ha produït un o més errors de conversió durant l'actualització de dades. En aquest cas, aturem el processat i carreguem novament la vista de la que procedeix la petició, que és també l'encarregada de mostrar els possibles errors de conversió.

- Obtenir controlador de l'acció comanada: segons l'àmbit del controlador objectiu, cercar la seva instància o inicialitzar-la si és el primer cop que es vol accedir a ella.

- Preparar paràmetres de l'acció: estudiar els paràmetres explícits que rep l'acció objectiu i resoldre els valors que han de prendre segons el seu origen de dades.

- Acció amb validador? Determina si l'acció objectiu té configurat un validador previ a la seva execució.

- Invocar validador: Executa el mètode de validació previ a l'acció.

- Errors de validació? Determina si l'execució de la validació ha provocat l'aparició d'errors de validació i, en aquest cas, s'atura el processament i es carrega la vista que el validador determina.

- Invocar acció: Executa el cos de l'acció objectiu, ja amb els controladors actualitzats i passant a la invocació els valors dels possibles paràmetres explícits.

- Excepció durant l'acció? Determina si en el cos de l'excepció s'ha produït una excepció.

- Existeix gestor d'excepcions? Determina si existeix un gestor d'excepcions instal·lat capaç de processar una excepció del tipus que s'ha produït.

- Executar gestor d'excepcions: Executa el cos del gestor d'excepcions per a l'error produït durant l'acció.

- Rellança excepció? Determina si el gestor d'excepció ha capturar l'error per complet o bé després de la seva execució l'ha tornat a llençar per a la seva propagació.

Per altra banda, els possibles estats finals són llençar l'excepció (si cap gestor la podia processar o bé el gestor escollit l'ha propagat) i carregar una vista, referint-se en tal cas a preparar tots els controladors de l'aplicació per a que els seus atributs siguin accessibles i invocar al motor de generació de vistes indicant la plantilla corresponent.

4.5.7. Guia d'anotacions

Un dels pilars bàsics per a la configuració del nostre framework són les anotacions Java5. A través d'elles indiquem a l'eina quines classes són de quin tipus, i configurem certs aspectes de cada component. A continuació s'enumeren totes les anotacions que el nostre framework ofereix al desenvolupador i una explicació i exemples d'ús per a cadascuna.

4.5.7.1. @FBean

Afegit just abans de la definició d'una classe, indica que aquesta representa un controlador que pot incloure atributs (dades) i accions. Accepta dos paràmetres:

- name: cadena de text que identificarà al controlador a les vistes per referir-se a atributs (name.nomAtribut) i a la URI per referir-se a accions (/name.nomAccio). Si no s'indica, per defecte s'utilitzarà el nom de la classe en minúscules.
- scope: constant (disponible a FConstants) que identifica l'àmbit i cicle de vida del controlador, que pot ser la sessió d'usuari o la petició. Per defecte es considera la sessió d'usuari.

4.5.7.2. @FAction

S'afegeix abans de la signatura d'un mètode definit dins d'un controlador, i provoca que el mètode en qüestió es consideri una acció invocable per l'usuari. El mètode ha de complir certs requisits: retornar un valor de tipus String (corresponent a la vista a renderitzar), i acceptar només paràmetres convenientment configurats amb l'anotació @FParam. Accepta tres paràmetres.

- name: cadena de text que identificarà l'acció quan ens volem adreçar a ella a la URI de la petició (/nomBean.name). Si no s'indica, per defecte s'utilitzarà el nom del mètode.
- validator: referència a una classe que implementa la interfície FInboundValidator i que serà l'encarregada d'executar una validació de dades prèvia a l'execució de l'acció. Si no s'indica, no s'executarà cap validació prèvia.
- contentType: identificador MIME que indica el tipus de contingut que es generarà després de renderitzar la vista que doni l'acció per resposta. Per defecte és "text/html", corresponent a planes HTML.

4.5.7.3. @FParam

Anotació que permet configurar els possibles paràmetres d'entrades d'una acció del framework. Més concretament permet indicar l'origen de dades que ha de donar valor al paràmetre. Accepta tres paràmetres:

- type: constant (disponible a FConstants) que identifica l'origen de dades del paràmetre. Pot ser de tres tipus: carregar en el paràmetre tota la petició de l'usuari, carregar un valor concret dels que envia l'usuari, o carregar tot un controlador de dades.
- paramName: només aplicable pel cas de paràmetres que tenen com a origen un valor concret de la petició, indica el nom del paràmetre en qüestió.
- beanName: només aplicable pel cas de paràmetres que tenen com a origen un controlador, indica el nom del controlador en qüestió.

4.5.7.4. @FGlobalConverter

Indica que una classe que implementa la interfície FInboundConverter és un conversor de dades que s'aplica per a totes les conversions de l'aplicació. Només cal afegir-ho abans de la classe en qüestió, no rep paràmetres.

4.5.7.5. @FGlobalExceptionHandler

Indica que una classe que implementa la interfície FExceptionHandler és un gestor d'excepcions susceptible de processar excepcions provocades per qualsevol acció del framework. Només cal afegir-ho abans de la classe en qüestió, no rep paràmetres.

4.5.7.6. @FConverters

Afegit conjuntament a @FBean o @FAction, indica que tot un controlador o bé una acció concreta utilitza els conversors de dades indicats, tenint aquestos preferència sobre els definits a un àmbit més general. Només rep un paràmetre:

- converters: array de classes que implementen la interfície FInboundConverter i que són els conversors a aplicar per a l'acció concreta o per a totes les accions del controlador, segons on s'afegeix l'anotació.

4.5.7.7. @FExceptionHandlers

Afegit conjuntament a @FBean o @FAction, indica que tot un controlador o bé una acció concreta utilitza els gestors d'excepcions indicats, tenint aquestos preferència sobre els definits a un àmbit més general. Només rep un paràmetre:

- handlers: array de classes que implementen la interfície FExceptionHandler i que són els gestors d'excepcions a aplicar per a l'acció concreta o per a totes les accions del controlador, segons on s'afegeix l'anotació.

4.5.7.8. @FDirective

Anotació que permet identificar configurar classes que implementen la interfície TemplateDirectiveModel (pertanyent a FreeMarker) per a que quedin instal·lades com a directives del motor de generació disponibles a la vista. Només rep un paràmetre:

- name: nom que identificarà a la directiva a les vistes de l'aplicació.

4.5.8. Guia d'interfícies

Certs aspectes del framework queden lligats a certes interfícies que asseguren la implementació de mètodes amb un objectiu i signatura determinats. És el cas de la validació de dades, la conversió d'aquestes, i la gestió d'excepcions. A continuació s'enumeren les interfícies definides pel nostre framework.

4.5.8.1. FInboundConverter

Interfície per a conversors de dades que reben un o més String (pertanyents a la petició de l'usuari) i transformen el seu valor en una instància d'un objecte concret. Defineix dos mètodes:

- Class getConverterClass(): retorna la referència a la classe objectiu que el conversor és capaç de transformar.
- Object convert(String[] in) throws FConverterException: executa la conversió i retorna una instància (o nul, segons el cas) del resultat d'aquesta.

4.5.8.2. FInboundValidator

Interfície per a validadors de dades que reben les dades visibles per una acció i executen comprovacions sobre aquestes dades, afegint errors si és el cas i decidint quina vista és responsable de mostrar aquestos errors. Defineix dos mètodes:

- validate(FRequestResult result, Object bean, Object[] actionParams): executa la validació sobre les dades visibles i afegeix els errors que detecti en el resultat de la petició.
- String getErrorView(): retorna l'identificador de la vista responsable de mostrar els errors que s'hagin detectat.

4.5.8.3. FExceptionHandler

Interfície per a gestors d'excepcions que reben un error llençat per una acció, així com les dades que aquesta rep, i fa tasques de tractament de l'excepció esmentada. Defineix dos mètodes:

- List<Class> getAcceptedClasses(): retorna la llista de classes d'excepció que el gestor és susceptible de capturar i processar.
- String handle(Throwable t, Object bean, Object[] actionParams) throws Throwable: fa el tractament de l'excepció amb accés a les mateixes dades que l'acció i provoca dos resultats: o propagar l'excepció perquè continuï el seu camí per defecte, o bé retornar un identificador de vista que serà la que es generarà.

4.5.9. Manual d'ús

A continuació s'exposen les característiques del framework que un desenvolupador ha de conèixer per tal de implementar la seva pròpia aplicació web emprant l'eina com a eina de fons. La guia comença per punts molts generals com la instal·lació del framework, els controladors i les accions, però també detalla certs aspectes més concrets com la convenció de noms establerta per a les peticions que emet l'usuari.

4.5.9.1. Instal·lació del framework

Com a conseqüència del principi de configuració mínima, disposar del framework en un nou projecte web porta tan sols uns petits passos:

1. Disposar al classpath de la llibreria del nostre framework, així com les seves dependències (Google Reflections, FreeMarker).
2. Indicar al descriptor de l'aplicació web.xml la instal·lació del controlador principal (FServlet), determinant en aquest moment la ruta base de la que penjaran les accions del controlador.

3. Opcionalment, indicar la ruta de paquets arrel a la que es troben tots els components a instal·lar sobre el framework (controladors, conversors, etc). Si no s'especifica, el escanejat de components inclourà tot l'arbre de paquets. El paràmetre per indicar aquesta ruta és "root.package".
4. Opcionalment, indicar la ruta on es troben els arxius de plantilla que cercarà el motor de vistes. Si no s'especifica, la ruta per defecte és l'arrel del directori "WEB-INF/" de l'aplicació web. El paràmetre per indicar aquesta ruta és "template.path".
5. Opcionalment, indicar l'extensió dels arxius de plantilla de vista, que tenen com a nom els identificadors de vista que els components retornaran. Si no s'especifica, l'extensió esperada serà ".ftl". El paràmetre per indicar aquesta extensió és "template.extension".
6. Es recomanable, però no imperatiu, indicar que FServlet es carregi al inici, ja que d'aquesta manera s'estalvia el temps d'inicialització a la primera petició rebuda, i es pot detectar prematurament eventuals errors durant la posada en marxa del framework.

4.5.9.2. Creació de components satèl·lits

Anomenem components satèl·lits aquells que no formen part de l'estructura base, però fan les funcions de petits afegits opcionals que complementen o modifiquen certs comportaments del framework. En aquesta categoria englobem els conversors, gestors d'excepcions, validadors i directives de vista aportades pel desenvolupador.

Per crear els components n'hi ha prou amb disposar les classes implementant les interfícies requerides per a cada cas i, si és el cas, afegir ja les primeres anotacions Java que indiquen que certs components tenen un àmbit d'actuació global.

És important que els components pengin de paquets inclosos a la ruta base indicada durant la instal·lació del framework.
Creació de components principals

Són els components principals aquells necessaris per donar contingut al framework, és a dir, els controladors i les seves accions.

Per configurar una classe com a controlador, cal afegir l'anotació necessària a la seva capçalera, sempre assegurant que la classe es troba a un paquet inclòs a la ruta indicada durant la configuració del framework.

El controlador pot tenir tants atributs com vulgui, però només aquells que disposen de mètodes getter i setter convencionals es podran comunicar-se amb la vista. Per raons internes, ambdós mètodes són necessaris, fins i tot per casos de variables de només lectura.

Per configurar un mètode d'un controlador com a acció, cal afegir l'anotació necessària abans de la seva signatura, sempre assegurant que el mètode retorni un valor del tipus esperat, i que els possibles paràmetres explícits

incloguin degudament especificat el seu origen mitjançant l'anotació ideada per aquest fi.

També en aquest moment tant controladors com accions poden incloure característiques addicionals com ara conversors, gestors d'excepcions o validadors (pel cas de les accions), fent sempre referència als components satèl·lit definits al pas anterior.

4.5.9.3. Creació de les vistes

Coneguts els identificador de vista que les accions generaran, cal disposar a la ruta indicada durant la inicialització del framework els arxius de plantilla corresponent a cada vista. Aquestos arxius tenen sintaxi de FreeMarker i poden fer crides tant a les directives que el framework ofereix per defecte com a aquelles definides per l'usuari durant la fase de creació de components satèl·lits.

4.5.9.4. Posada en marxa

Durant la posada en marxa (o al rebre la primera petició, segons com s'hagi indicat a la inicialització), el framework farà la càrrega de tots els components que composen el sistema. És recomanable observar la sortida d'aquest procés, ja que informa de les passes executades, el nombre de components detectats, i alerta sobre possibles problemes trobats durant la inicialització. Aquestos problemes poden ser lleus (el framework pren certes decisions però no decideix avortar la posada en marxa), o greus (el framework informa de l'error i cap component queda desplegat).

4.5.9.5. Com referir-se a una acció

Durant el procés de disseny i construcció de les vistes, arriba un punt on l'usuari provoca una acció que acaba amb l'enviament d'una petició HTTP al servidor. Per a que aquesta petició provoqui la invocació d'una acció del framework, la seva ruta ha de seguir la forma:

```
<url_framework>/<nom_bean>.<nom_accio>[?possibles_paràmetres_per_get]
```

On <url_framework> és la ruta base a la que s'ha instal·lat el controlador principal, <nom_bean> és el nom d'un dels components controlador instal·lats, i <nom_accio> és el nom d'una acció inclosa en l'esmentat controlador. El framework accepta tant peticions de tipus GET com de tipus POST.

4.5.9.6. Com actualitzar a atributs del controlador

Quan es realitza una petició que provoca la crida d'una acció del framework, cap la possibilitat de que la mateixa ordre inclogui nou valors corresponents a atributs de controladors instal·lats. Per a indicar que un paràmetre de la petició fa referència a un atribut d'un controlador, el seu nom ha de ser:

```
f:<nom_bean>.<nom_atribut>
```

On "f:" és un prefix invariable, <nom_bean> és el nom d'un controlador instal·lat, i <nom_atribut> és el nom d'un dels atributs del controlador. L'atribut pot ser simple o compost, en forma de ruta separada per punts. En aquestos casos és necessari disposar d'un mètode getter i setter per a cadascuna de les passes de la ruta. Per exemple, si a la petició viatja el següent paràmetre:

```
f:employees.employee.address.number
```

És necessari que el controlador amb nom "employees" disposi dels mètodes getEmployee() i setEmployee(...), el tipus empleat disposi dels mètodes getAddress() i setAddress(...), i el tipus adreça disposi dels mètodes getNumber() i setNumber(). Un cop més, per raons tècniques és necessari disposar de tots dos mètodes, tot i que només es desitgi fer una escriptura de l'atribut.

4.5.9.7. Accions que terminen amb una redirecció

Considerem com a més habitual el cas en que una acció, enllestida la seva execució, dictamina quina vista s'ha d'encarregar de presentar la resposta a l'usuari, retornant al finalitzar el seu cos un identificador de vista.

En canvi, existeix la possibilitat de que una acció desitgi que, en acabar la seva execució, el framework "salti" cap a una nova acció en forma de redirecció. Per aquestos casos, el que ha de retornar l'acció en comptes d'un identificador de vista és una cadena de la forma:

```
r:<nom_bean>.<nom_accio>
```

On "r:" és un prefix que determina que es tracta d'una redirecció, i <nom_bean> i <nom_accio> es refereixen als noms per que es coneixen el controlador i l'acció als que es desitja fer el salt.

4.5.10. Components per defecte

El nostre framework permet la implementació e instal·lació de components "satèl·lit" de la collita del desenvolupador. No obstant, ofereix una sèrie d'aquestos components per defecte amb la finalitat de cobrir alguns dels escenaris més habituals.

4.5.10.1. Conversors de dades

L'eina ofereix una sèrie de conversors de dades bàsics per als tipus més elementals de Java: nombres (sencers i amb decimals), booleans i dates. Pels casos dels Integer, Long, Double, Float i Boolean, s'habiliten conversors tant per les primitives (int, long, etc.) com per les classes continents.

Tots aquestos conversors implementen el mateix patró: rebre una cadena de text i executar la conversió al tipus destí assumint un format d'entrada. En el cas de les dates, s'habiliten dos possibles conversors, un que espera el format dd/mm/aaaa i un que espera una data amb hores i minuts amb el format dd/mm/aaaa hh:mi. Com a conversor de dates per defecte s'habilita el primer dels casos.

Addicionalment i per homogeneïtat del codi, s'habilita un conversor "fictici" per a cadenes de text que retorna el mateix que rep com a entrada.

4.5.10.2. Gestor d'excepcions

Per defecte, l'eina no disposa de cap gestor d'excepcions per al tractament dels errors produïts durant l'execució d'una acció. Això vol dir que els eventuals errors produïts es propaguen fins a la capa més superior del framework, i per aquest cas el tractament que es fa és presentar les dades de l'excepció en un format amigable.

Concretament, es mostra amb un HTML amb un mínim d'estils aplicats la informació general i la traça de l'error, amb un màxim de 10 nivells.

4.5.10.3. Directives

La més completa i sofisticada de les propostes que ofereix per defecte el framework és el catàleg de directives ja implementades. Aquestes directives, invocables des de la vista, cobreixen diferents objectius: mostrar dades de només lectura, mostrar dades en forma de camp editable, mostrar errors de l'aplicació, o iterar a dades de col·lecció.

I. Directives per a camps de formulari

Són directives que, donat el nom d'un controlador i un atribut, presenten el seu valor actual en forma de camp de formulari, proveint l'opció de camps de text (TextField), caselles seleccionables (CheckboxField) o llistats desplegables (SelectField). Per a cada possibilitat s'habiliten diferents paràmetres configurables.

Nom: FTextField
Descripció: Directiva per mostrar camps de text.

Atributs:

Nom de l'atribut	Descripció	Obl.
field	Nom de l'atribut en forma "nom_controlador.nom_atribut"	✓
dateFormat	Format en que es vol mostrar el valor actual, només aplicable quan l'atribut referit és del tipus Date.	
class	Estils CSS que cal aplicar al input HTML generat.	
style	Estils explícits que cal aplicar al input HTML generat.	
onchange	Comportament que cal vincular al input HTML generat quan es modifica el seu valor.	
forceValue	Valor per defecte en cas que l'atribut sigui nul o buit (cadena buida).	

Nom: FCheckboxField
Descripció: Directiva per mostrar caselles seleccionables.

Atributs:

Nom de l'atribut	Descripció	Obl.
field	Nom de l'atribut en forma "nom_controlador.nom_atribut".	✓
checkedValue	Valor de l'atribut que provoca que la casella quedi seleccionada. Si no s'especifica, el seu valor és "true".	
class	Estils CSS que cal aplicar al input HTML generat.	
style	Estils explícits que cal aplicar al input HTML generat.	
onclick	Comportament que cal vincular al input HTML generat quan es fa clic sobre ell.	
forceValue	Valor per defecte en cas que l'atribut sigui nul o buit (cadena buida).	

Nom: FSelectField
Descripció: Directiva per mostrar llistes desplegables

Atributs:

Nom de l'atribut	Descripció	Obl.
field	Nom de l'atribut en forma "nom_controlador.nom_atribut".	✓ (un dels dos)
fieldPost	Nom del paràmetre que crearà amb el seu valor el desplegable generat.	
optField	Nom de l'atribut que esdevé la llista d'opcions a presentar.	✓
optValue optText	Nom dels subatributs per cada element de la llista que es corresponen amb el valor que assigna al camp i el text que es mostra com a opció respectivament. Si no s'indiquen, s'aplica un .toString().	
emptyOptValue emptyOptText	Valor i text de l'opció "buida" de la llista. Si no s'especifiquen els dos, no es mostra opció buida.	
class	Estils CSS que cal aplicar al input HTML generat.	
style	Estils explícits que cal aplicar al input HTML generat.	
onchange	Comportament que cal vincular al input HTML generat quan es modifica el seu valor.	
forceValue	Valor per defecte en cas que l'atribut sigui nul o buit (cadena buida).	

II. Directives per a mostrar errors

Es tracta de directives que accedeixen els diferents contenidors d'errors (de conversió i de validació) i permeten mostrar-los en format HTML.

Nom: FFieldError
Descripció: Mostra, cas d'existir, l'error de conversió provocat per un camp del formulari d'entrada.

Atributs:

Nom de l'atribut	Descripció	Obl.
field	Nom de l'atribut en forma "nom_controlador.nom_atribut".	✓
pre, post	Fragments de codi HTML a afegir abans i després de l'error, en cas d'existir.	

Nom: FValidationErrors
Descripció: Mostra, en cas d'existir, la relació d'errors de validació que s'han produït durant la darrera acció.

Atributs:

Nom de l'atribut	Descripció	Obl.
pre, post	Fragments de codi HTML a afegir abans i després de la llista d'errors, en cas d'existir.	
itemPre, itemPost	Fragments de codi HTML a afegir abans i després de cadascun dels errors llistats, en cas d'existir.	

Nom: FErrorCount
Descripció: Presenta el nombre d'errors (incloent tant els de conversió com els de validació) que s'han produït amb la darrera acció.

Atributs:

Nom de l'atribut	Descripció	Obl.
showIfZero	Si el seu valor és "yes", provoca que la directiva generi la seva sortida tot i que el contador d'errors sigui zero.	
pre, post	Fragments de codi HTML a afegir abans i després del contador.	

III. Directiva per a mostrar valors

Es tracta d'una directiva que accedeix a valors d'atributs de contenidors (tal i com fan les directives de camps de formulari), però en aquest cas només mostren el seu valor actual com a text pla.

Nom: FTextOutput
Descripció: Mostra el valor actual d'un atribut.

Atributs:

Nom de l'atribut	Descripció	Obl.
field	Nom de l'atribut en forma "nom_controlador.nom_atribut"	✓
dateFormat	Format en que es vol mostrar el valor actual, només aplicable quan l'atribut referit és del tipus Date.	
pre, post	Fragments de codi HTML a mostrar abans i després del valor de l'atribut.	
forceValue	Valor per defecte en cas que l'atribut sigui nul o buit (cadena buida).	

IV. Directiva per iterar col·leccions

Es tracta d'una directiva especial que accedeix a atributs de controladors que es poden recórrer (arrays, col·leccions) i permet executar un fragment de codi de plantilla iterativament per a cada element de l'atribut (de manera similar a la directiva implícita [#list] de FreeMarker).

Nom: FLoop
Descripció: Permet fer un recorregut iterativament d'un atribut de controlador corresponent a una col·lecció.

Atributs:

Nom de l'atribut	Descripció	Obl.
field	Nom de l'atribut en forma "nom_controlador.nom_atribut" que inclou la llista que es vol recórrer.	✓
emptyValue	Codi HTML que cal mostrar en cas que la col·lecció no existeixi o sigui buida i mai es processi el contingut de la directiva.	

Addicionalment, aquesta directiva necessita rebre un paràmetre amb el nom de la variable que, al cos de la directiva, farà referència a l'element de cada iteració. Un exemple d'ús seria el següent:

```
<@FLoop
 field="empleats.llistaEmpleats"
 emptyValue="<li>No existeixen empleats</li>"; empleat>
      <li>${empleat.nom}</li>
</@FLoop>
```

4.5.11. Diari de desenvolupament

El següent apartat prova d'enumerar aquelles situacions succeïdes durant el desenvolupament del framework que han suposat un major repte, bé per la complexitat del problema al que ens enfrontàvem o bé per la necessitat de fer un esforç de recerca per a trobar la solució més adient. No es tracta de fer un seguiment exhaustiu de totes les fases del desenvolupament, si no només en aquelles que pels motius esmentats sembla remarcable documentar per aprofitar el coneixement adquirit.

4.5.11.1. Escaneig d'anotacions

Una de les primeres necessitats a cobrir en iniciar la fase d'implementació fou la de trobar un mecanisme per detectar, donat un paquet arrel, totes les classes visibles per la màquina virtual de Java que incloguin una determinada anotació. Aquest és el principi que utilitza el nostre framework per detectar certs components que poden estar ubicats a qualsevol part dins del paquet indicat a la configuració.

L'API de Reflection, tot i ser molt potent, no ofereix eines suficients pel concepte "d'escaneig". Un cop hem localitzat una classe que volem processar, si disposem de tot un conjunt d'eines suficients per conèixer qualsevol aspecte d'aquesta. En canvi, les solucions no són tantes quan es tracta de proveir mètodes que permetin fer un recorregut exhaustiu d'un paquet i tots els seus paquets. Entre els motius donats per l'absència d'aquesta funcionalitat, el més robust sembla ser la complexitat de la JVM i com aquesta té una estructura de registre de classes més complicada que un simple contenidor.

Després d'una breu recerca, trobem tres candidats per assolir el nostre objectiu. El primer és el petit mòdul inclòs a Spring que aquest utilitza internament per aquesta mateixa finalitat. Es tracta d'una classe anomenada :

org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider

La qual presenta l'inconvenient de requerir incloure nombroses llibreries i paquets de Spring al nostre propi framework, una situació que per ara descartem.

La segona opció aprofita els registres i loaders de serveis de Java. No obstant, el seu ús requereix de la creació de fitxers auxiliars (normalment inclosos a la carpeta META-INF del projecte compilat), llistats de referències que cal construir en temps de compilació. Això complicaria tant el nostre desenvolupament com les instruccions que cal aportar al desenvolupador de l'aplicació final, la qual cosa col·lisiona frontalment amb el nostre principi bàsic de mínima configuració.

La darrera opció ens porta a Mountain View. L'empresa Google, gegant el sector informàtic i clara referència de l'explosió del món a la xarxa, ofereix la seva pròpia llibreria Java per l'estudi i processament de metainformació de les nostres classes. S'anomena Google Reflections, i ens ofereix en dues línies de codi precisament allò que estem cercant. L'inconvenient és la necessitat d'incloure una quantitat considerable de dependències en forma de llibreries externes, però ens resulta preferible a l'opció d'incloure llibreries d'un altre framework com Spring.

Una cop que hem inclòs Google Reflections i les seves dependències al projecte del nostre framework, un codi tan simple com el següent cerca totes les classes d'un paquet i els seus subpaquets i ens retorna aquelles que compleixen un criteri determinat, com ara estar acompanyades d'una anotació determinada.

```
Reflections reflections =
new Reflections(rootPackage, new TypeAnnotationsScanner());
Set<Class<?>> beans = reflections.getTypesAnnotatedWith(FBean.class);
```

4.5.11.2. Detecció de genèrics

Des de la seva versió 5, Java ha millorat els seus tipus iterables com ara les col·leccions afegint el que han decidit anomenar "genèrics". Fins llavors, tota llista, conjunt, mapa o d'altres, es definien sense donar cap altre informació respecte al tipus de dada que acceptaven com a elements de la col·lecció. Això obria una finestra a possibles errors indetectables en temps de compilació, com ara afegir a una llista suposadament de nombres sencers una cadena alfanumèrica.

Amb els genèrics, el desenvolupador pot definir, a més del tipus de col·lecció, el tipus de dades dels elements que la composen. Així, per exemple, queda indicat que una llista només accepta elements de tipus Integer, o que les claus d'un mapa són de tipus String, mentre els seus valors són de tipus Class. D'aquesta manera, el compilador pot detectar si s'està afegint a una col·lecció un tipus de dada que no és el que s'esperava, millorant així l'eficiència del desenvolupador en termes de detecció d'errors.

Arribant ja al nostre cas, el nostre framework té un apartat en que requereix conèixer el tipus de dada objectiu que es vol actualitzar dins d'un controlador. Per tipus no iterables n'hi ha prou amb un ús bàsic de l'API de Reflection per tal de conèixer el tipus de l'atribut objectiu i fer la conversió pertinent. En canvi, quan volem afegir objectes a una col·lecció la cosa és complica sensiblement.

Juntament amb el concepte de genèrics, Java afegeix el que es coneix com a "type erasure". Com a norma, les definicions de genèrics del codi font no tenen necessitat de sobreviure en temps d'execució, ja que la seva finalitat és només la detecció d'error en temps de compilació. És a dir, tot i que una llista es declari com a List<String>, en el bytecode generat pel compilador no hi ha necessàriament diferència amb una llista declarada només com a "List".

Això, per altra banda, no és del tot cert. És veritat que, amb les eines necessàries (emmascarant el tipus real d'una dada i utilitzant Reflection, principalment), podem enganyar al codi font per tal que no doni problemes de compilació i ens permeti executar insercions d'elements a priori no permesos en una col·lecció. En canvi, si és possible recuperar la informació de genèrics que es va declarar en el seu moment, gràcies novament a l'API de Reflection.

Entra en lloc un nou concepte fortament lligat al de Classe: el concepte de Tipus. Amb la classe Type i les seves subclasses, pertanyents a l'API de Reflection, és possible interrogar en temps d'execució una col·lecció per esbrinar quin és el tipus objectiu que va declarar, o bé si no va declarar cap tipus específic.

Pel cas d'una col·lecció declarada com a tipus de retorn d'un mètode, el següent codi seria l'encarregat de, en primera instància detectar que és una col·lecció, i en segona conèixer el tipus que accepta.

```
Method m = …;
Class returnType = m.getReturnType();
Type genericReturnType = m.getGenericReturnType();
if(Collection.class.isAssignableFrom(returnType)) {
Class colClass = (Class) (((ParameterizedType)
            genericType).getActualTypeArguments()[0]);
      …
}
```

En el fragment anterior, colClass és la Classe del tipus de dada que accepta la col·lecció. En cas que aquesta no fou definida amb genèrics, la crida a getActualTypeArguments provocaria una excepció i ha de ser capturada per fer les accions corresponents a una col·lecció sense tipus específic.

4.6. Aplicació de prova

El següent mòdul presenta l'experiència de desenvolupar una senzilla aplicació de prova amb l'ajut del nostre propi marc de presentació FFramework.

Després d'una breu introducció on exposar la temàtica i les funcionalitats de l'aplicació dissenyada, passarem a enumerar aquelles situacions més rellevants del desenvolupament, fent especial incís en les complicacions sorgides durant el procés i si aquestos descobriments han suposat alguna modificació, millora o observació a fer sobre el framework desenvolupat.

4.6.1. L'aplicació: base de dades cinematogràfica

Per a l'exemple escollim una senzilla base de dades de l'àmbit del cinema. Una eina que permet llistar, editar i afegir tant pel·lícules com persones pertanyents al sector, amb la possibilitat de creuar les dades per indicar quines persones han treballat a una cinta concreta.

El sistema presentarà dos espais de llistat + fitxa de dades independents: un per a les persones i un altre per a les pel·lícules. A la fitxa de cadascun, s'habilitarà una secció des de la qual consultar i modificar la relació entre les dues entitats.

L'aplicació inclou un rudimentari control d'accés. S'emula una acció d'autenticació contra el sistema, on en cas que l'usuari d'accés sigui "admin" es suposarà que es tracta d'un usuari administrador. Només en aquest cas permetrem certes accions contra la base de dades.

Per a facilitar la prova de l'aplicació, s'habilitarà una opció de "Crear base de dades de prova" que ompli el sistema amb una col·lecció de pel·lícules, persones i relacions entre aquestes suficients per poder provar totes les funcionalitats ofertes.

Tot i no ser l'objecte d'aquesta fase, val a dir que partim de la base d'una sèrie de classes Java que ja implementen tant el model com l'accés a la "base de dades", encara que es tracta d'una solució rudimentària que manté les dades a memòria.

4.6.2. Dissenyant el sistema

Per adaptar el nostre sistema a la solució que ofereix FFramework, cal decidir la estructura de controladors, accions i vistes que composen la nostra aplicació, a més d'altres components satèl·lits com ara els validadors de dades.

Començant per lo més bàsic, decidim incloure tres controladors al nostre sistema: un controlador d'aplicació encarregat de gestionar l'usuari autenticat, i controladors per a pel·lícules i persones, cadascun amb suport per a les operacions suportades sobre cada tipus d'entitat.

En quant a les vistes, habilitem una vista de portada a la que anomenarem “home” i vistes de llista i fitxa per a cada entitat: “movieList”, “movieItem”, “personList”, “personItem”. Aprofitem les possibilitats de FreeMarker per a incloure a totes les pàgines un “header” que inclourà informació relativa al login de l’aplicació.

4.6.3. Directiva de bucle i mostrar dades iterades

Una de les primeres carències de la nostra proposta de framework la trobem al moment d’iterar sobre una llista inclosa en un atribut d’un controlador. Entre les directives per a FreeMarker que ofereix FFramework trobem “FLoop”, que ens permet fer el recorregut en bucle per la col·lecció corresponent, i tota una sèrie de directives que permeten mostrar el valor d’un atribut de forma “segura”, amb certa protecció davant de referències nul·les o valors buits.

El problema el trobem en la incompatibilitat entre aquestes directives. Les que s’ofereixen per mostrar valors esperen com a paràmetre d’entrada la ruta de l’atribut en qüestió dins del conjunt de controladors del framework. En canvi, si volem mostrar valors de l’element concret sobre el que estem iterant, aquesta referència és local al bucle, i no hi ha manera de referir-la per a que les directives que volem emprar localitzin i processin el seu valor.

La solució passa per continuar explotant les possibilitats de les user-defined directives de FreeMarker. Observar si existeix una via per recuperar, en una directiva, el valor d’una variable de bucle instanciada en una directiva de nivell superior. En una primera aproximació, no hem estat capaços de trobar una solució que s’ajusti a aquesta descripció.

4.6.4. Directiva de llistes desplegables sense atribut de controlador vinculat

Amb diferents objectius, a les pàgines de fitxa de pel·lícula i persona desitgem mostrar camps de formulari desplegables on l’usuari selecciona un valor entre una sèrie d’opcions.

FFramework ofereix una directiva, “FSelectField”, que serveix per aquest propòsit. En canvi, la nostra primera versió d’aquesta directiva es limita únicament a camps desplegables que tenen un vincle amb un atribut de controlador concret.

En canvi, és d’esperar –de fet, la nostra aplicació de prova ja ho requereix- que es desitgi disposar de camps desplegables on en el moment d’enviar les dades del formulari no hi hagi un vincle directe amb cap controlador. És a dir, que el camp tingui com a destí un paràmetre de petició normal i corrent.

Per a solucionar-ho es decideix modificar la directiva FSelectField, de forma que no sigui obligatori indicar un atribut de controlador vinculat, i es permeti

indicar un nom de paràmetre “simple” que serà qui enregistri el valor seleccionat.

4.6.5. Reutilització de validadors entre accions de diferents controladors

Durant la implementació de l’aplicació de gestió trobem certes necessitats de validació de dades molt semblants entre si. És el cas de l’acció “Afegir rol”, que en el cas de la fitxa de pel·lícula afegeix una persona a la pel·lícula carregada, i en el cas de la fitxa de persona vincula una pel·lícula a la persona carregada. Com es pot observar, són situacions molt similars on cal validar que s’ha seleccionat un rol i una persona o pel·lícula, segons el cas.

Podem dissenyar un validador que pugui ser instal·lat contra ambdues accions, tot i pertànyer fins i tot a controladors diferents. Tot el que necessitem es distingir en quin escenari ens trobem gràcies a la classe del controlador implícit, rebut en el validador com a segon paràmetre de tipus Object.

Segons si aquest objecte és de tipus MovieBean o PersonBean, podem concloure en quin dels dos casos ens trobem i actuar en conseqüència resolent els valors implicats a la validació. Finalment, i encara discriminant per la classe de controlador que implementa l’acció a validar, es decideix la vista encarregada de mostrar els possibles errors, essent la fitxa de pel·lícula i persona respectivament.

4.6.6. Problemes amb tipus de dada complexos

Durant l’estudi previ d’eines existents una de les conclusions que varem extraure fou la complexitat de treballar amb certs tipus de dades amb el framework de Struts. Entre els objectius de cara a la nostra proposta de framework figurava emprar una estructura de Beans similar a la de Struts però evitant en la mesura possible aquestos problemes.

Es pot dir que no hem satisfet tot lo que voldríem aquest objectiu. Novament, l’aparició d’atributs complexos que, per exemple, inclouen referències a dominis en el seu interior, provoquen certes complicacions en el procés de transmetre la informació entre vista i controlador i en el sentit invers.

4.6.7. Control d’accés mitjançant validadors

Un dels apartats que inclou la nostra aplicació de prova és un mínim control d’accés basat en autenticació de l’usuari. Això significa que hi ha certes funcionalitats de l’aplicació que només un usuari amb els permisos suficients deu ser capaç d’accedir, controlant aquesta restricció a nivell de servidor ja que el nivell de client és susceptible d’atacs no desitjats.

En concret, desitgem que tota acció d'escriptura sobre la base de dades (editar, crear, esborrar, etc) quedi restringit a un rol d'administrador, figura que només considerem quan accedim mitjançant l'username "admin".

Donat que a FFramework no hem definit cap tipus de mecanisme explícitament dissenyat pel control d'accés, fem una aproximació a través del sistema de validació de dades.

Definim un validador que, entre els seus paràmetres d'entrada, sempre pugui rebre el controlador corresponent al control d'accés de l'aplicació (ApplicationBean, en el nostre cas). No establim cap restricció respecte a en quin ordre es rep aquest controlador: pot ser el controlador implícit de l'acció, o bé trobar-se en qualsevol dels paràmetres explícits d'aquesta.

Una vegada obtingut el controlador, recuperem l'usuari actual per confirmar si pertany al rol mínim suficient per executar les accions que acompanyarà aquesta validador. Si no és el cas, llavors simulem un error de validació que avortarà l'execució de l'acció i mostrarà (configurant les vistes amb aquest propòsit) el problema detectat.

Pot donar-se el cas que accions que s'inclouen en aquest control d'accés desitgin implementar la seva pròpia validació de dades. Donat que FFramework només permet un validador per acció, recorrem llavors a l'herència per crear validadors que hereten d'aquest validador mestre que s'encarrega del control d'accés. Aquestos segons validadors han de fer una crida a la seva superclasse per mantenir aquest servei de control d'accés.

4.6.8. Resum de l'experiència

Sense tornar a repetir allò que hem observat, podríem resumir l'experiència de desenvolupar una aplicació web amb FFramework com satisfactòria, però clarament millorable.

Aquesta simulació de "hands-on" sobre la nostra eina ens ha permet veure el resultat des d'un punt de vista molt diferent, lluny d'una perspectiva ja viciada i contaminada d'aquell que coneix totes les interioritats del sistema que està dissenyant.

En la pell d'un desenvolupador totalment aliè al projecte, hem pogut obtenir una resposta en un dels aspectes que hem tractat com a més prioritaris: la usabilitat. A banda de petites errades tècniques ja reparades, experimentar aquest nou rol ens ha obert tota una sèrie de portes en forma de possibles millores que detallarem en un apartat posterior.

4.7. Millores i ampliacions

Tant el temps com els recursos en forma d'hores/persona de dedicació destinats a aquest projecte són limitats, per la qual cosa sempre hi queda un marge per reforçar, completar i fins i tot millorar el treball fet.

Aquest grup d'observacions en forma de possibles ampliacions del projecte afecten essencialment a la part més pràctica d'aquesta, és a dir, al desenvolupament del nostre propi framework de presentació J2EE.

El resultat FFramework implementa diverses de les funcionalitats més bàsiques que s'esperen d'una eina d'aquest tipus, però pel camí han quedat descartades, per diferents motius, un bon nombre de possibilitats que aportarien un atractiu encara major al software produït. Addicionalment, durant la fase de proves i, sobretot, de desenvolupament d'una aplicació de prova sota FFramework, s'han detectat certes característiques o limitacions del producte ofert que caldria millorar per beneficiar l'experiència del desenvolupador que decideixi treballar amb el producte.

A continuació s'enumeren aquestes millores contemplades.

4.7.1. Suport per a llistes de dades complexes

Dins l'àmbit sempre obert a la millora com és la conversió de dades entre la petició del navegador i els objectes Java, una de les opcions més interessants és la de donar suport per a permetre enviar a través d'un formulari HTTP una col·lecció d'objectes Java complexos. Actualment FFramework només permet codificar llistes de tipus bàsics (integer, String, etc.), a través de la lectura d'un mateix paràmetre HTTP amb diferents valors. Aquesta solució ara per ara no és suficient si volem, per exemple, rebre en els controladors un llistat d'instàncies de la classe Empleat.

4.7.2. Multiidioma

En major o menor grau, totes les eines existents que hem estudiat incorporen serveis i funcionalitats relatives al suport d'aplicacions multiidioma, on la resposta del servidor pot adaptar-se al llenguatge que l'usuari estableixi entre un conjunt d'opcions disponibles. Aquest és un servei necessari per tal que el nostre framework pugui utilitzar-se en àmbits més globals no limitats a una única llengua, i passa per permetre substituir els literals de les vistes i valors de les dades per claus que fan referència a un contenidor de claus nom-valor (habitualment un arxiu de propietats Java).

4.7.3. Múltiples tecnologies de vista

FFramework només suporta plantilles per a la vista implementades amb la tecnologia FreeMarker. Aquesta va ser una decisió basada en l'esperit de

recerca, ja que es tracta d'una potent opció no tan estesa en l'àmbit dels frameworks J2EE com ara JSP i les seves TagLibs. No obstant, disposar de la possibilitat de conviure amb diferents tecnologies de presentació simultàniament (com ja fan algunes de les eines estudiades) amplia el conjunt de desenvolupadors que es poden veure interessats pel nostre producte.

4.7.4. Major flexibilitat dels objectes Java

Actualment, per a que una classe Java pugui ser codificada a partir d'una petició web a FFramework és requisit indispensable que disposi d'una constructora buida. Entre d'altres, aquesta és una restricció imposada per motius tècnics que amb una revisió a fons de les possibilitats del llenguatge es pot arribar a alleugerar, aconseguint d'aquesta manera que l'experiència de desenvolupar amb FFramework sigui encara més ràpida i flexible.

4.7.5. Variacions del patró FrontController

En el moment d'imposar el patró FrontController per a FFramework es va decidir utilitzar un component J2EE de tipus Servlet per a desplegar el controlador principal. En canvi, durant l'estudi previ d'eines existents varem descobrir que hi ha opcions que han decidit implementar aquest controlador a través d'un component Filter. Una opció interessant seria estudiar més a fons els motius a favor i en contra de cadascuna de les dues opcions, i fins i tot si existeixen d'altres sense sortir dels components que J2EE ofereix.

4.7.6. Validadors amb paràmetres propis

En la implementació actual, els validadors suportats per a FFramework reben com a dades d'entrada exactament les mateixes que rep l'acció que prèviament validen. És a dir, no hi ha possibilitat de definir dades exclusivament per a validació sense afegir-les també a les accions que valida. Eliminar aquesta limitació no hauria de requerir un excessiu temps, concentrant la major part de l'esforç en modificar el codi del controlador principal que s'encarrega d'instal·lar i preparar els validadors definits.

4.7.7. Eina CASE

En el nostre objectiu prioritari de fer més fàcil la feina del desenvolupador J2EE, hi ha dues possibilitats per reduir la seva carrega de treball. La primera, i en la qual hem centrar els nostres esforços, és la d'implementar un programari base sobre el qual el desenvolupador pot començar a programar disposant ja de certs mecanismes habituals. La segona es oferir eines que automatitzen part del procés de codificació de l'aplicació. Aquestes eines, que reben el nom d'eines CASE, permeten rebre com a entrada certs paràmetres relatius al sistema que volem dissenyar (planes, tipus de dades, usuaris, etc) i creen una versió molt primària del sistema en base a un sistema de plantilles. Disposar

d'una eina CASE per generar projectes desenvolupats amb FFramework donaria un atractiu encara més gran a l'eina per tal de convèncer al seu públic potencial.

4.7.8. Tractament de recursos no relatius al framework

En lo que respecte a recursos estàtics de l'aplicació web resultant com ara fulls d'estils o imatges, a FFramework ens hem decidit per una solució tan senzilla com rudimentària: establir un fort vincle entre una ruta base i el nostre framework, de mode que una petició a qualsevol altre ruta que no inclogui aquest prefix serà ignorada per l'eina.

Aquesta és una aproximació idèntica a la d'altres frameworks de presentació existents, alguns d'ells considerats els més complets i potents. En canvi, condiciona el disseny de la jerarquia de directoris de l'aplicació web que el desenvolupador està creant.

Una bona millora, que a més oferiria un punt distintiu respecte d'altres alternatives, seria idear un mecanisme pel qual recursos dirigits o no cap al framework mantinguin una mateixa ruta, mantenint així un arbre de directoris més estàndard.

4.8. Conclusions

Els desenvolupadors estem acostumats a un procés d'anàlisi, disseny i implementació força definit, i sempre amb una idea notablement clara del resultat que volem obtenir. Aquesta visió a futur dels nostres objectius ens ajuden en el moment de prendre certes decisions o afrontar problemes que sorgeixen durant el procés, ja que normalment preguntar-nos què és el que volem ens donarà la resposta.

En canvi, dissenyar un programari que ha de servir de base per a futurs desenvolupaments esdevé un problema més complex. I desenvolupar un framework de presentació J2EE s'inclou en aquesta classificació.

L'esforç d'estudi de totes les possibilitats creix exponencialment, i el moment de prendre decisions és molt més delicat, ja que ens hem de basar en previsions i percepcions altament subjectives. El nostre producte s'ha de poder adaptar a tot un ventall d'opcions no sempre similars entre si, i és en el moment de la seva concepció on radica la major dificultat per aconseguir aquesta característica.

Totes aquestes observacions poden resultar relativament òbvies. Evidentment, el desenvolupament d'un "meta-producte" requereix d'un domini de l'àmbit en que es treballa més gran que no pas el disseny d'un producte, sense intenció de simplificar a través del llenguatge, "més senzill". Aquest projecte, no obstant, ens ha permès conèixer més a fons on radiquen aquestes dificultats afegides, sentir de primera ma quins aspectes d'un desenvolupament web són més variables i per tant més difícils de tipificar en un moment en que volem ser conscients de totes les possibilitats.

Entrant ja en matèria de tendències i opcions que semblen predominar, l'arquitectura bàsica dels framework no sembla presentar aquesta complexitat. Després d'un recorregut per algunes de les eines més populars, trobem una clara predilecció pel patró FrontController, que s'ajusta perfectament a la problemàtica d'habilitar un punt d'entrada a partir del qual es redistribueix la informació. Només petites variants canvien entre una proposta o altre, com ara l'ús d'un Servlet o un Filter per implementar aquest eix central del sistema, però en essència el principi escollit és sempre el mateix.

En quant a la configuració, si hem trobat un ampli catàleg de diferents decisions segons el framework estudiat. Alguns d'ells encara depenen fortament d'un dels sistemes més veterans, com ara la configuració a partir de fitxers externs, habitualment documents XML. Aquesta és una possibilitat prou contrastada i que per molt temps s'ha establert com a opció preferent, però en els darrers anys ha donat pas a altres alternatives que pretenen ser més eficients en diferents aspectes.

Per exemple, altres eines han decidit apostar plenament per la potència que les modernes anotacions Java ofereixen. D'aquesta manera les dades de configuració queden fortament lligades a l'objecte a configurar, facilitant així un manteniment posterior del sistema. En canvi, presenta també els seus

inconvenients, com ara la necessitat de tornar a compilar el codi (no necessari amb XML) o bé certes peculiaritats com a conseqüència de la tecnologia d'anotacions i la màquina virtual de Java (per exemple, el fenomen del "type erasure").

En darrer lloc, no cal oblidar la possibilitat d'evitar els aspectes de configuració tant com es pugui. Aquesta opció implica dos línies de treball: per una banda, establir tota una sèrie de convencions de noms per la qual quan denominem les peces de programari ja estem realitzant una configuració implícita. Addicionalment, cal definir i documentar degudament el comportament que tindrà cada component i el sistema en general en cas d'omissió de configuració, és a dir, ser més exigents en decidir els comportaments per defecte de l'eina.

No és fàcil parlar d'una opció de configuració millor que la resta, ja que totes presenten pros i contres, com ara la flexibilitat, la facilitat de manteniment, l'eficiència o la potència.

Com introduíem al inici d'aquestes conclusions, el desenvolupament d'un framework presenta tota una sèrie de complexitats fruits de la necessitat d'adaptar-se al ventall més gran possible d'usos que el desenvolupador final vulgui donar-li. Aquest escenari es presenta en diferents parts del programari, com ara les validacions o l'àmbit i cicle de vida dels components. No obstant, tal com suposàvem durant l'estudi d'eines i varem confirmar posteriorment en el nostre desenvolupament, la part més oberta a conflicte és la de la conversió de dades.

Per la pròpia limitació de treballar en l'àmbit d'aplicacions web amb tecnologia J2EE, és inevitable passar per una fase de traducció de dades en ambdues direccions: per presentar la informació, codificar instàncies de classes Java i els seus atributs en format text que un navegador web pugui mostrar per pantalla. Per rebre peticions d'un usuari, fer el camí invers construint i manipulant classes Java a partir de dades en forma de cadenes de text. És aquest segon punt el més conflictiu de tots, ja que les infinites possibilitats que ens ofereix la jerarquia i estructura de classes d'una aplicació Java col·lisiona frontalment amb l'aparent senzillesa d'un conjunt de parelles clau/valor que ens envia el navegador.

Establir una estratègia per correspondre el contingut de peticions HTTP amb les dades contingudes a tota una estructura d'objectes Java ha estat, sense discussió, la tasca més dificultosa a la que ens hem enfrontat durant el transcurs del projecte. De fet, certes limitacions o peculiaritats de com aquesta conversió de dades s'ha implementat en els frameworks estudiats prèviament ens porta a pensar que aquesta és una conclusió habitual quan s'afronta un projecte d'aquest tipus.

Per concloure i enllaçant amb alguns dels principis que ens van portar a la decisió de realitzar aquest projecte, considerem altament positiva l'experiència viscuda, reforçant la idea d'aprofundir i fer una petita retribució a una tecnologia i una comunitat que tant i durant tant de temps ha aportat el seu coneixement a

individus que pretenien assolir els coneixements i domini necessaris per produir aplicacions web de qualitat. Serveixi aquest projecte com un petit gra de sorra en una riba plena de petites i grans aportacions que composen el que és actualment la comunitat de Java en general i J2EE en concret: un gran nombre de persones preocupades per assolir i mantenir en constant millora una tecnologia que ha esdevingut clau en l'era moderna del desenvolupament de software.

5. Glossari

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

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

Lògica de negoci: Nom que rep aquella capa d'una aplicació que realitza essencialment el cos de les operacions de gestió de la informació.

Presentació: Nom que rep aquella capa d'una aplicació que s'encarrega de mostrar dades prèviament preparades a l'usuari final.

Vista: Component d'una aplicació corresponent a les interfícies d'usuari. Veure *presentació*.

HTML: Llenguatge de marcat desenvolupat pel World Wide Web Consortium i que permet presentar de forma estructurada continguts a l'usuari a través d'un intèrpret, habitualment un navegador web.

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

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

Etiqueta (HTML): Peça fonamental del llenguatge HTML que permet agrupar i separar els continguts segons el seu tipus i/o semàntica.

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

Workflow: Denominació que es refereix a un itinerari de passos ben definit i que permet conèixer les etapes per les que passarà un procés en base o no a que es satisfacin certes condicions.

Mapping: Traducció establerta entre dos conjunts de dades a priori sense relació, de forma que es manté la informació variant només la manera en que es troba emmagatzemada.

Tag: Veure *etiqueta (HTML)*.

Anchor: Tipus d'etiqueta que implementa un enllaç cap a un altre contingut de informació.

Precompilar: Procés pel qual peces de programari en codi font es tradueixen en objectes intermedis interpretables per una màquina virtual.

6. Bibliografia

SpringSource (27 autors) (2004-2011). *Spring Framework Reference Documentation, Part V. The Web. Web MVC Framework.*
<http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html>

Colin Sampaleanu (2011, gener). *Green Beans: Getting Started with Spring MVC.*
<http://blog.springsource.com/2011/01/04/green-beans-getting-started-with-spring-mvc>

SpringSource (27 autors) (2004-2011). *Spring Framework Reference Documentation, Part III. Core Technologies. Validation, Data Binding and Type Conversion.*
<http://static.springsource.org/spring/docs/current/spring-framework-reference/html/validation.html>

Wikipedia (diversos autors) (darrera modificació a 2011, desembre). *Apache Struts.*
<url>

The Apache Software Foundation (2055-2011). *Struts Documentation: Key Technologies Primer.*
<http://struts.apache.org/primer.html>

Vaannila.com (2009). *Struts 2Tutorial.* Apartats generals, d'anotacions, UI Tags, DispatchAction, validacions i "hola món".
<http://www.vaannila.com/struts-2/struts-2-tutorial/struts-2-tutorial.html>

Ed Burns, Ryan Lubke, Jim Driscoll (2010, gener). *Generating a JavaServer Faces 2.0 CRUD Application from a Database.*
<http://netbeans.org/kb/docs/web/jsf20-crud.html>

Mkyong.com (2010, setembre). *JSF 2.0 hello world example.*
<http://www.mkyong.com/jsf2/jsf-2-0-hello-world-example>

Cristóbal González Almirón (2009, març). *Introducción a JSF Java Server Faces.*
<http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IntroduccionJSFJava>

RunIT.com (2010, juny). *Hola mundo con Jsf, Eclipse y Tomcat.*
<http://www.runit.com.ar/pages/view/1307>

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

Qusay H. Mahmoud (2004, agost). *Developing Web Applications with JavaServer Faces.*
<http://www.oracle.com/technetwork/articles/javase/javaserverfaces-135231.html>

Wikipedia (diversos autors) (darrera modificació de gener 2012). *Groovy (programming language).*
<http://en.wikipedia.org/wiki/Groovy_%28programming_language%29>

Wikipedia (diversos autors) (darrera modificació de desembre 2011). *Grails (framework).*
<http://en.wikipedia.org/wiki/Grails_%28framework%29>

David Marco (2010, junio). *Introducción a Groovy.* Capítols 1 a 5 + annex.
<http://www.davidmarco.es/blog/entrada.php?id=181>
<http://www.davidmarco.es/blog/entrada.php?id=211>
<http://www.davidmarco.es/blog/entrada.php?id=225>
<http://www.davidmarco.es/blog/entrada.php?id=226>
<http://www.davidmarco.es/blog/entrada.php?id=227>
<http://www.davidmarco.es/tutoriales/anexos/anexo_introduccion_groovy.html>

Andrés Almiray (2007, agost). *Introducción a Groovy.*
<http://groovy.org.es/home/story/introduccioacuten-a-groovy>

Andrew Glover (2004, agost). *Alt.lang.jre: Feeling Groovy. Introducing a new standard language for the Java platform.*
<http://www.ibm.com/developerworks/java/library/j-alj08034/index.html>

Rick Reumann (2007, desembre). *Grails CRUD.*
<http://www.learntechnology.net/content/grails/grails_part1.jsp>

Graeme Rocher, Peter Ledbrook, Marc Palmer, Jeff Brown, Luke Daley, Birt Beckwith (data desconeguda). *Grails Validation – Reference Documentation.*
<http://www.grails.org/doc/1.3.7/guide/7.%20Validation.html>

FreeMarker (2011, maig). *Using FreeMarker with servlets.*
<http://freemarker.sourceforge.net/docs/pgui_misc_servlet.html>

FreeMarker (2011, maig). FreeMarker Data Model > *Directives.*
<http://freemarker.sourceforge.net/docs/pgui_datamodel_directive.html>

FreeMarker (2011, maig). *FreeMarker Directive Reference > User-defined directives.*
<http://freemarker.sourceforge.net/docs/ref_directive_userDefined.html>

Diversos participants (2008-2009). *Scanning Java annotations at runtime.* Fil de discussió amb diferents alternatives per l'escaneig d'anotacions Java en temps d'execució.
<http://stackoverflow.com/questions/259140/scanning-java-annotations-at-runtime>

Jakob Jenkov (data desconeguda). *Java Reflection: Arrays.*
<http://tutorials.jenkov.com/java-reflection/arrays.html>

Google (2011). *Reflections: A Java runtime metadata analysis, in the spirit of Scannotations.*
<http://code.google.com/p/reflections>

Diversos autors (2009-2011). *Java generics – type erasure – when and what happens.* Fil de discussió relatiu al fenòmen del "type erasure" utilitzant genèrics a Java.
<http://stackoverflow.com/questions/339699/java-generics-type-erasure-when-and-what-happens>

Ian Robertson (2007, juny). *Reflecting generics.*
<http://www.artima.com/weblogs/viewpost.jsp?thread=208860>

Sun Microsystems, Inc. (2002). *Front Controller.*
<http://java.sun.com/blueprints/patterns/FrontController.html>

Wikipedia (diversos autors) (darrera modificació de desembre 2011). *Front Controller pattern.*
<http://en.wikipedia.org/wiki/Front_Controller_pattern>

Sun Microsystems, Inc. (2001-2002). *Core J2EE Patterns – Front Controller.*
<http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html>

OXXus Hosting (2006, desembre). *J2EE Application Components.*
<http://www.oxxus.net/tutorials/j2ee/j2ee-components>

7. Annex I. Descàrrega de llibreries externes

Tal com es descriu a l'apartat de productes obtinguts, aquesta memòria ve acompanyada d'un fitxer comprimit que inclou certs desenvolupament produïts durant la realització del projecte. Per motius d'espai, alguns d'aquestos desenvolupaments requereixen llibreries de tercers que no es troben incloses en el paquet, per la qual cosa és necessari descarregar-les dels seus websites oficials.

7.1. SpringExample (cas pràctic amb Spring MVC)

La versió de Spring utilitzada és la 3.0.6.RELEASE, i tots els .jar especificats s'han de copiar al directori "lib/" del projecte de NetBeans "SpringExample".

Per obtenir els següents .jar és necessari accedir al repositori de SpringSource (http://www.springsource.com/download/community) i localitzar la versió sense documents **spring-framework-3.0.6.RELEASE.zip** (24,8 MB). Aquest comprimit inclou tota una sèrie de llibreries de les quals només necessitem les següents:

- org.springframework.asm-3.0.6.RELEASE.jar
- org.springframework.beans-3.0.6.RELEASE.jar
- org.springframework.context.support-3.0.6.RELEASE.jar
- org.springframework.context-3.0.6.RELEASE.jar
- org.springframework.core-3.0.6.RELEASE.jar
- org.springframework.expression-3.0.6.RELEASE.jar
- org.springframework.web.servlet-3.0.6.RELEASE.jar
- org.springframework.web-3.0.6.RELEASE.jar

Addicionalment, cal descarregar les següents llibreries third-party:

- asm-3.3.1.jar
  http://forge.ow2.org/projects/asm/

- cglib-2.2.2.jar
  http://sourceforge.net/projects/cglib/files/

- commons-logging-1.1.1.jar
  http://commons.apache.org/logging/download_logging.cgi

- freemarker-2.3.18.jar
  http://freemarker.sourceforge.net/freemarkerdownload.html

7.2. StrutsExample (cas pràctic amb Struts 2)

La versió de Struts 2 utilitzada és la 2.2.3.1, però en el moment de preparar el lliurament no es troba disponible en repriment de la 2.3.1.1 Tots els .jar especificats s'han de copiar al directori "lib/" del projecte de NetBeans "StrutsExample".

Per obtenir els següents .jar és necessari accedir descarregar la distribució completa de la versió corresponent de Struts 2:.

- commons-fileupload-1.2.2.jar
- commons-io.2.0.1.jar
- commons-lang.2.5.jar
- freemarker-2.3.18.jar
- javassist-3.11.0.GA.jar
- ognl-3.0.3.jar
- struts2-core-2.3.1.1.jar
- xwork-core-2.3.1.1.jar
  http://struts.apache.org/download.cgi#struts2231

7.3. FacesExample (cas pràctic amb JSF 2.0)

El cas pràctic amb Java Server Faces s'ha desenvolupat amb la versió del framework ja inclosa amb l'IDE NetBeans 7.0.1, motiu pel qual no ha estat necessari descarregar paquets addicionals. En cas de no voler dependre de l'IDE de Sun Microsystems, la versió del framework és la 2.1 i es pot obtenir a http://javaserverfaces.java.net/download.html.

- jsf-api.jar
- jsf-impl.jar

7.4. FFramework

La nostra pròpia proposta de framework de presentació J2EE fa ús de diferents llibreries per aconseguir certes funcionalitats. A continuació es detalla la manera d'obtenir cadascuna d'aquestes, que es divideixen essencialment entre FreeMarker, Google Reflections, i dependencies d'aquestos dos.

- reflections-0.9.5.jar
- dom4j-1.6.1.jar
- gson-1.4.jar
- guava-r08.jar
- javassist-3.12.1.GA.jar
- jboss-vfs-3.0.0.CR5.jar
- slf4j-api.1.6.1.jar
- slf4j-simple.1.6.1.jar
- xml-apis-1.0.b2.jar
  http://code.google.com/p/reflections/downloads/list
  (paquet **reflections-0.9.5.one-jar.jar**)

- freemarker-2.3.18.jar
  http://freemarker.sourceforge.net/freemarkerdownload.html

- log4j-1.2.16.jar
  http://logging.apache.org/log4j/1.2/download.html