Projecte Final de Carrera d'Enginyeria en Informàtica

Estudi i disseny d'un bastiment arquitectònic en J2EE

Autor: Sergi Acebes Villanueva
Consultor: Javier Ferró García

Barcelona, de Gener de 2008

Aquest projecte està dedicat a tots els meus amics, als companys de feina, i la meva família, sobre tot per la paciència demostrada durant aquests anys d'estudi, especialment la de la meva parella sentimental, Maria, i de la meva mare Isabel.

- Sergi

El projecte que es presenta a continuació és un estudi de diferents bastiments arquitectònics que ofereixen solucions bassades en la plataforma J2EE. Es fa una comparativa seguint el seu model en capes i s'analitzen des d'aquesta perspectiva quines propostes podem escollir com les més adequades per oferir una producte de qualitat sota una perspectiva funcional i no funcional.

Per tant, es planteja l'estudi i viabilitat dels actuals marcs de desenvolupament i la seva possible aplicació dins d'un entorn empresarial, analitzant el complex conjunt d'eleccions que hi ha en el mercat sobre aquesta arquitectura, i en particular, l'ús de frameworks d'aplicació disponibles que donen una solució completa a les diferents capes estructurals en les que es divideix aquesta plataforma.

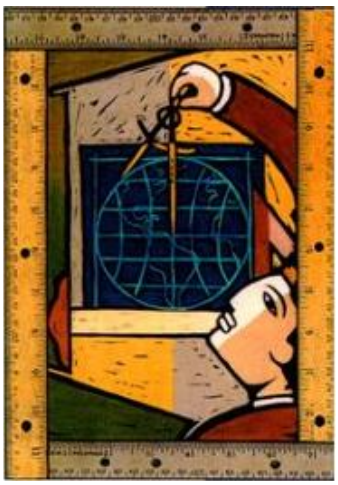
En definitiva, l'objectiu final serà donar un model a mode de solució introductòria per a poder realitzar aplicacions empresarials sota l'arquitectura J2EE, que servirà com a guia o plantilla per equips de desenvolupament.

Àrea PFC: J2EE

Índex

Índex de figures

Pla de treball



Objected-Oriented Metrics

“No es pot controlar el que no es pot mesurar”.

1982, Tom deMarco, autor i professor d’Enginyeria del Programari.

1 Introducció

1.1 Justificació

Les necessitats empresarials per oferir productes cada vegada més competitius dins de cada sector fa que les empreses es centrin en trobar solucions estratègiques a les seves necessitats. En l'actualitat l'elecció de la plataforma J2EE, *Java 2 Enterprise Edition*, està cada vegada més estesa i és una de les tecnologies més utilitzades com a plataforma d'aplicacions distribuïdes empresarial.

Dins d'aquesta plataforma es presenta un gran ventall de possibilitats per escollir diferents tecnologies apropiades per desenvolupar programari. La seva elecció cada vegada és més complicada i la quantitat de models de construcció, marcs de treball, bastiments[1] i APIs presents, afegeix cada vegada més confusió.

És per aquest motiu que avui en dia és molt important poder entendre les necessitats reals de la nostra aplicació, i poder adaptar el seu desenvolupament als continus canvis que trobem a les tecnologies d'implementació, que implica un major esforç en el disseny de l'aplicació.

El projecte desglossa una proposta per desenvolupar una solució que integra de forma adequada diferents tecnologies de presentació, de negoci i de persistència on l'estudi de solucions desenvolupades com a bastiments d'aplicació, és la part essencial i més important.

Amb la presentació d'aquest projecte final de carrera es pretén demostrar els coneixements adquirits per l'alumne durant la carrera, aprendre'n de nous, i documentar correctament un projecte informàtic.

1.2 Objectius

Aquest projecte té com a objectiu l'anàlisi, el disseny i la implementació d'una solució tecnològica adequada bassada en la plataforma J2EE per a un projecte de programari empresarial que inclogui estàndards, reutilització de components i utilització de patrons amb l'objectiu final de reduir costos en la construcció de programari.

Així el projecte abasta des de l'estudi i anàlisi de les capes arquitectòniques de la plataforma J2EE i aquells frameworks disponibles que donen solucions funcionals a aquestes capes, fins la definició d'una solució pràctica que consideri l'optimització de costos en el desenvolupament de programari, la característica més important sota una perspectiva concreta funcional i no funcional. Per tant, es centra en l'estudi i anàlisi d'una solució i no pas en la seva implementació.

1.3 Enfocament i mètode seguit

Una vegada dut a terme aquest projecte s'espera haver:

- Aprofundit en els coneixements de la plataforma J2EE i les seves capes.
- Conèixer els frameworks més coneguts per a cada capa de la plataforma.
- Veure quin és el model de programari J2EE proposat i establir la correspondència arquitectònica entre aquest model i les solucions funcionals proposades.
- Aplicar aquests coneixements en un exemple pràctic que:
    - Permeti disminuir els costos en la construcció de programari.
    - Elimini tasques repetitives de programació.
    - Optimitzi processos de construcció de programari.
    - Ofereixi una resposta orientada a l'àmbit empresarial.

[1] En anglès, *framework*.

Es considera dintre de l'abast d'aquest projecte:

- Descripció de l'arquitectura J2EE i les seves particularitats.
- Estudi de les capes de l'arquitectura.
- Anàlisi de solucions tecnològiques per aquestes capes.
    - Frameworks de presentació.
    - Frameworks de negoci.
    - Frameworks de persistència.
- Conclusions sobre les solucions estudiades.
- Elecció justificada de tres tecnologies per la capa de presentació, negoci i persistència.
- Implementar un exemple pràctic que integri les tres solucions escollides.
- Documentar de forma correcte tots els apartats.

1.4 Planificació del projecte

El projecte es desglossa en les següents activitats:

Llançament del projecte i tasques inicials

Consta de les primeres reunions entre els participants del projecte i es fa un acostament a la concepció del projecte. El document final que s'entrega és el Pla de Treball.

> Durada esperada: 1,2 setmana.
> Participaran en aquesta activitat: consultor i projectista.

Anàlisi de la tecnologia J2EE i estudi de frameworks

Durant aquesta etapa s'estudiaran les particularitats de la plataforma J2EE les seves capes arquitectòniques i es realitzarà un estudi en profunditat dels patrons i frameworks disponibles que els hi donen suport funcional.

> Durada esperada: 3 setmanes.
> Participaran en aquesta activitat: consultor i projectista.

Disseny d'una solució

A partir dels coneixements adquirits durant l'etapa d'anàlisi, proporcionar un disseny que inclogui una solució adequada segons els requeriments funcionals i no funcionals definits. El document final que s'entrega és l'anàlisi i disseny de l'aplicació.

> Durada esperada: 1 setmana
> Participaran en aquesta activitat: consultor i projectista.

Construcció del model operatiu

Obtenir una proposta a mode de solució pràctica que inclogui la utilització dels frameworks escollits i atengui degudament els requeriments. L'objectiu d'aquesta etapa és la construcció dels components definitius de l'aplicació. El document a entregar serà la implementació de la solució.

> Durada esperada: 6,8 setmanes.
> Participaran en aquesta activitat: projectista.

Confecció de la memòria

En aquesta etapa es prepara tota la documentació a presentar de la memòria del projecte, així com la presentació virtual. El document a entregar serà la memòria complerta amb el codi implementat i una presentació en defensa del projecte

> Durada esperada: 4 setmanes.
> Participaran en aquesta activitat: projectista.

1.4.1 Calendari de Treball

	Nombre de tarea	Duración	Comienzo	Fin	Pred
1	Fita externa 1: trobada presencial	0 días	lun 08/10/07	lun 08/10/07	
2	**- Inici del projecte PFC**	**86 días?**	**mié 19/09/07**	**mié 16/01/08**	
3	**- Llançament del projecte**	**8 días?**	**mié 19/09/07**	**vie 28/09/07**	
4	1a Reunió Comité Coordinació	1 día?	mié 19/09/07	mié 19/09/07	
5	2a Reunió Comité Coordinació	1 día?	vie 21/09/07	vie 21/09/07	
6	3a Reunió Comité Coordinació	1 día?	lun 24/09/07	lun 24/09/07	
7	Definició Pla del Projecte	4 días	mar 25/09/07	vie 28/09/07	6
8	Presentació del Pla de Projecte	0 días	vie 28/09/07	vie 28/09/07	3
9	**- Anàlisi i estudi J2EE**	**16 días**	**lun 01/10/07**	**lun 22/10/07**	
10	Aproximació de l'arquitectura J2EE	2 días	lun 01/10/07	mar 02/10/07	8
11	Anàlisi de patrons en J2EE	3 días	mié 03/10/07	vie 05/10/07	10
12	El model MVC	3 días	lun 08/10/07	mié 10/10/07	11
13	Solucions per la capa de negoci	4 días	jue 11/10/07	mar 16/10/07	12
14	Frameworks de persistència	4 días	mié 17/10/07	lun 22/10/07	13
15	**- Disseny solució**	**5 días**	**mar 23/10/07**	**lun 29/10/07**	**14**
16	Disseny de la solució escollida	3 días	mar 23/10/07	jue 25/10/07	
17	Confecció entregable PAC2	2 días	vie 26/10/07	lun 29/10/07	16
18	Presentació Anàlisi i Disseny	0 días	lun 29/10/07	lun 29/10/07	17
19	**- Construcció del model**	**34 días**	**mar 30/10/07**	**vie 14/12/07**	
20	Construcció capa presentació	5 días	mar 30/10/07	lun 05/11/07	18
21	Construcció capa de negoci	7 días	mar 06/11/07	mié 14/11/07	20
22	Construcció capa de persistència	7 días	jue 15/11/07	vie 23/11/07	21
23	Integració de les solucions i proves	13 días	lun 26/11/07	mié 12/12/07	22
24	Confecció entregable PAC3	2 días	jue 13/12/07	vie 14/12/07	23
25	Presentació Implementació	0 días	vie 14/12/07	vie 14/12/07	24
26	**Confecció entregable de la memòria**	20 días	lun 17/12/07	vie 11/01/08	25
27	Presentació de la memòria	0 días	vie 11/01/08	vie 11/01/08	26
28	Defensa del projecte	3 días	lun 14/01/08	mié 16/01/08	27
29	Tancament del projecte	0 días	mié 16/01/08	mié 16/01/08	28

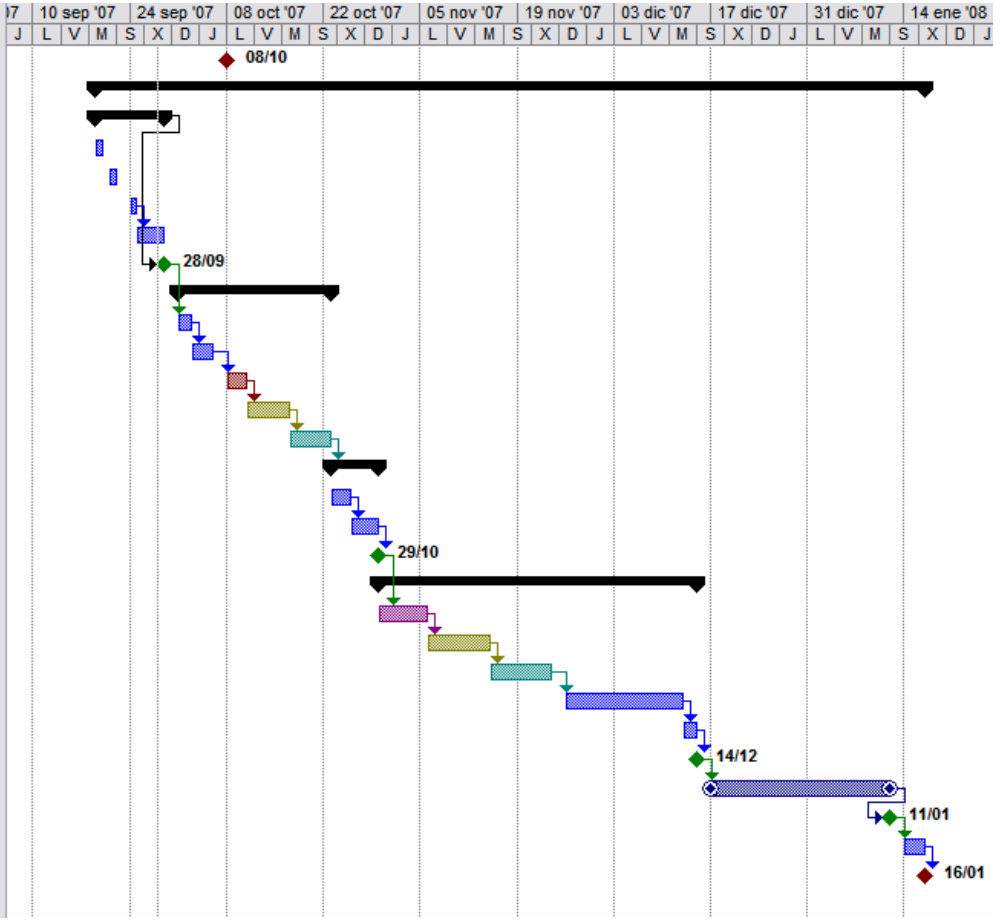


Figura 1. Calendari de treball

1.4.2 Fites Principals

A partir de la planificació anterior s'estableixen les següents fites de control, que ens serveixen per a revisar l'estat del projecte i establir accions de control sobre el seu desenvolupament, de manera que puguem mesurar les desviacions entre la realitat i la planificació prèvia i prendre les mesures correctores adequades per a assolir els objectius del projecte.

- **Llançament del projecte.**

Moment d'inici del projecte en què s'ha detectat si es disposa dels recursos necessaris per dur-lo a terme.

- **Presentació del Pla de Treball.**

Es dóna conformitat a la definició prèvia realitzada entre els participants. PAC1 (28-09-2007)

- **Trobada presencial**

Reunió entre el consultor i el projectista (6-10-2007).

- **Revisió i validació de l'anàlisi i disseny.**

Es validarà l'anàlisi realitzat del projecte i es donarà confirmació de la finalització del disseny del model. PAC2 (29-10-2007).

- **Consolidació de la solució.**

Entrega de la implementació del model. PAC3 (17-12-2007).

- **Presentació de la memòria**

Entrega de tota la documentació del projecte (14-01-2008).

- **Defensa del projecte**

Presentació i defensa del projecte.

- **Final del projecte**

Es donarà l'aprovació del projecte pels equips de treball.

1.4.3 Equip de Treball

Així doncs s'identifiquen els següents equips de treball:

- Un comitè de coordinació del projecte amb:
  - Un coordinador global del projecte.
  - Un director de projecte

1.4.4 Definició de Rols

Coordinador del Projecte:

- Javier Ferró Garcia, consultor UOC.

Director del Projecte:

- Sergi Acebes Villanueva, projectiste UOC.

Comitè de Coordinació del Projecte:

- Javier Ferró Garcia, consultor UOC.
- Sergi Acebes Villanueva, projectiste UOC.

1.4.5 Mecanismes de Control

Atesa la criticitat del projecte i com són d'ajustades les dates, se'n proposa un control bastant estricte. Es faran reunions de seguiment mensuals per part de qualsevol dels membres del comitè de coordinació.

En cas d'esdeveniments extraordinaris que així ho aconsellin, serà facultat del cap de projecte, convocar reunions amb una altra periodicitat.

El cap de projecte s'encarregarà de la documentació necessària, pel que fa a l'avanç del projecte, problemes detectats i solucions proposades.

1.5 Anàlisi de riscos

Amb la informació disponible s'identifiquen els següents riscos:

- **Temporalitat.**
  El projecte està marcat dins l'estudi i anàlisi de bastiments J2EE però, en el mercat hi ha molts, fins i tot qualsevol pot construir-ne un, llavors, és força important limitar la temporalitat del projecte per tal de no allargar-lo amb escreix.

- **Manca d'una anàlisi en profunditat.**
  Per a l'elaboració d'aquesta proposta s'han fet servir estimacions preliminars. Si bé, no esperem canvis significatius, en cas de produir-se seria necessari replantejar el projecte.

- **Coordinació general.**
  El projecte implica diferents estudis de vàries àrees amb una forta interdependència entre les tasques a ser desenvolupades.

1.6 Productes obtinguts

En primer lloc hi ha tres entregables amb la memòria:

- El pla de treball del projecte on es defineixen els objectius, abast i planificació.
- L'anàlisi dels bastiments estudiats: descripció, comparativa i conclusions.
- Disseny d'una solució i aplicació exemple.

Amb aquest projecte es pretén dotar d'una mètode de desenvolupament de programari d'aplicacions empresarials i l'aplicació d'exemple implementada reflecteix un model de construcció seguint les pautes estudiades dels bastiments i les bones maneres de programació de l'estudi plantejat. D'aquesta manera pretenem reflectir els coneixements adquirits en un exemple d'implementació.

El requeriment del producte és la construcció d'una prestació online de bloqueig de targetes per a donar suport a una empresa proveïdora de serveis Call Center d'una entitat financera.

L'aplicació de suport al Call Center determina una solució online de suport als clients d'una entitat bancària fora dels canals operatius humans tradicionals (oficina) en situacions problemàtiques (incidències, robatoris).

L'exemple donarà una guia per la construcció de projectes J2EE utilitzant Spring Web Flow, Hibernate i Spring Framework.

1.7 Descripció dels capítols de la memòria

La primera part del projecte ens introdueix en el món J2EE i les seves particularitats, es parla d'algunes tecnologies de l'arquitectura J2EE, i s'expliquen els patrons de disseny i la seva utilització. Aquesta part ajudarà a introduir-se en la programació d'aplicacions Web i entendre les seves particularitats.

A continuació, es fa una descripció del catàleg de patrons de l'arquitectura J2EE i s'expliquen les diferents capes que la formen. Així, ens introduïm en l'estudi i anàlisi del patró MVC (Model-Vista-Controlador), i s'analitzen algunes solucions diferents existents en l'actualitat que implementen aquesta funcionalitat. Posteriorment, parlarem de la capa de negoci i estudiarem quins frameworks hi ha en el mercat que donen solució a aquesta funcionalitat. En aquest estudi per capes finalitzem amb la de persistència.

Per a cada capa es considerarà l'ús de solucions pràctiques i efectives en el desenvolupament de programari a partir dels resultats obtinguts en la seva investigació. Es justificarà els avantatges i inconvenients de cada exemple i es mostrarà una solució per a cada tipus d'aplicació.

La darrera part del projecte finalitza amb una proposta pràctica de desenvolupament on es considera l'ús de frameworks d'aplicació per a cada capa de l'arquitectura J2EE estudiada, on cada solució utilitza els serveis de les capes inferiors. Aquestes solucions proposades sortiran de l'anàlisi dels apartats anteriors.

Estudi Arquitectònic

Germans Wright

“Tot el que podia ser inventat, ja ha estat inventat”.

1899, Charles Duell, comissionat de la Oficina de Patents Nord-americana.

Entendre les necessitats d'una aplicació és un llarg camí que ens acosta a la disminució de la complexitat de J2EE. Per tant, entendre les solucions que el mercat ens proporciona pel desenvolupament de projectes sota aquesta arquitectura és prerequisit per poder construir aplicacions de qualitat.

Els programadors d'aplicacions J2EE disposen d'un complex conjunt d'eleccions que van des de contenidors de pes lleuger com Spring, *NanoContainer* o HiveMind, a marcs de desenvolupament com WebWork, Tapestry, un UI basat amb JSF com el nou *ApplicationDevelopment Framework* d'Oracle, o Velocity. Si li afegim a aquesta elecció un nou conjunt d'especificacions J2EE, o el nou èmfasi dels serveis web i el corresponent jeroglífic de l'Arquitectura Orientada al Servei utilitzant JAXM, SAAJ, JAX-RPC, o JAX (sense especificar les llibreries o eines "WS-*") és un miracle que els desenvolupadors Java puguin fer alguna cosa.

En paraules de **Ben Galbraith**, participant de la Simposi sobre "*No Fluff Just Stuff Software*", "*Tan aviat com decideixes utilitzar un marc de treball, surt una nova versió d'una altre marc, així una i altre vegada*", això li anomena: **El principi d'incertesa del Marc de Treball Java**.

La major part del problema ve de no conèixer quina tecnologia cobreix millor cada una de les nostres necessitats, i la millor forma de fer això és definir conceptualment la nostra aplicació i distingir quines són les seves necessitats. Una vegada fet això, l'elecció de la tecnologia apropiada a utilitzar es veu més clarament definida i és més fàcil d'entendre.

En el capítol que presentem, es veurà una amplia introducció al estat actual de J2EE, les tecnologies que l'envolten, i alguns marcs de treball que un arquitecte ha d'encarar avui en dia.

Analitzem què és la plataforma J2EE, quins elements la formen i quin estil arquitectònic segueixen les aplicacions J2EE. Sota el terme (*Enterprise* Java) o aplicacions "Java empresarials" es defineix l'aproximació d'una aplicació per donar solució a l'entorn empresarial seguint el model tradicional de les tres capes, on es separen la **Presentació**, la **Lògica de Negoci** i l'**Accés a dades**. Estudiarem aquesta divisió que proposa el model J2EE i veurem els components que ofereix aquesta plataforma per implementar les funcionalitat de cadascuna de les capes.

Farem especial menció al model del **catàleg de patrons que proposa Sun** i les seves bones maneres de la programació definida a l'especificació ***Blueprints*** (http://java.sun.com/reference/blueprints/) que ens servirà per introduir-nos en el món abstracte dels patrons i l'ús apropiat de bastiments. És aquest darrer punt, la part més important de l'estudi. Per a què utilitzar un bastiments? Quins bastiments hi ha en el mercat? Quins utilitzar? On utilitzar-los?

Analitzarem, per tant, per a cada separació del model arquitectònic quines solucions podem seleccionar pel seu disseny i implementació. Aprofundirem en el patró MVC, la seva evolució i els marcs possibles que podem escollir. Parlarem de les diferents solucions de la capa de negoci i per últim, explicarem la necessitat d'utilitzar un bastiment de persistència o mapatge objecte relacional (ORM).

Una vegada conegudes les característiques d'aquesta plataforma i vistes les solucions que podem seleccionar, descriurem les característiques funcionals i no funcionals que una aplicació empresarial J2EE ha de cobrir per obtenir un producte de qualitat. Amb aquests paràmetres donarem una visió de la solució arquitectònica escollida i seleccionarem els candidats de bastiments per poder-la implementar.

2 Arquitectura bàsica J2EE

El capítol que comencem a continuació ens aproxima a la plataforma J2EE per poder donar una ullada a les propostes que ens fa Sun amb el seu catàleg de patrons i bones maneres de treballar que ens servirà per poder avaluar els diferents bastiments que podem trobar en el mercat i analitzar-los des d'un escenari empresarial.

Una vegada vista l'arquitectura tractarem amb les tres capes amb que podem dividir la capa intermèdia. Les estudiarem segons les propostes de J2EE i farem una breu introducció a les recomanacions de disseny que cal tenir en compte per dissenyar aplicacions.

2.1 Arquitectura multicapa

Java 2 Enterprise Edition (J2EE) és un exemple de plataforma de desenvolupament empresarial que té com a objectiu primordial fer més senzill i àgil el desenvolupament i la posta en marxa d'aplicacions distribuïdes. És una plataforma de desenvolupament (proposada per Sun Microsystems l'any 1997) que defineix un estàndard per al desenvolupament d'aplicacions empresarials multicapa.

J2EE simplifica el desenvolupament d'aquestes aplicacions basant-les en components modulars estandarditzats, proporcionant un conjunt molt complert de serveis a aquests components i gestionant automàticament moltes de les funcionalitats o característiques complexes que requereix qualsevol aplicació empresarial (seguretat, transaccions, etc.), sense necessitat d'una programació complexa.

J2EE segueix un estil arquitectònic client-servidor multicapa amb n-nivells i les aplicacions es poden dividir, bàsicament en tres capes:

- **Capa client.** Conté els components que s'executen a les màquines client. Per exemple, pàgines HTML que s'executen als navegadors dels usuaris.
- **Capa intermèdia** (***middle layer***). Normalment conté el processament de la lògica de negoci, situada entre la capa client i la capa EIS. Els components d'aquesta capa s'executen als diferents contenidors que formen el servidor d'aplicacions. En l'arquitectura J2EE podem dividir la capa intermèdia en tres subcapes formades per components diferents que s'executen a contenidors diferents i que tenen funcionalitats diferents.
- **Capa EIS.** En aquesta capa hi trobem la resta de components del sistema d'informació que hi ha a la empresa. Per exemple, conté els components de la capa de dades que resideixen a un servidor de dades (com podrien ser les taules d'una base de dades relacional).

2.2 Capa Intermèdia

La capa intermèdia contindrà els components que s'executaran als contenidors del servidor d'aplicacions. Aquest capa es pot dividir lògicament en tres capes més:

- Capa de presentació
- Capa de negoci
- Capa d'integració

2.2.1 Capa presentació

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

El servidor de la capa web al rebre una petició si va dirigida a un component web, la passa al contenidor web, que s'encarregarà de processar-la i torna la resposta al client, sinó el servidor envia la resposta al client sense passar pel contenidor web.

Per a proporcionar les funcionalitats anteriors cal desenvolupar, empaquetar i desplegar el que s'anomena aplicació web.

La plataforma J2EE defineix dos tipus de components web:

- **Servlets.** Classe Java que rep peticions HTTP i genera contingut dinàmic en resposta a aquestes peticions.
- **Java Server Pages (JSP).** Documents de text que s'executen com a Servlets, però que permeten una aproximació més natural a la creació de contingut estàtic. Les JSP tenen un aspecte molt similar als fitxers HTML però s'hi pot posar codi Java.

2.2.2 Capa de negoci

La capa de negoci és, segurament, la capa més important dins d'una aplicació empresarial. Aquesta capa contindrà els components, les relacions i les regles que resolen els processos de negoci de l'empresa.

La capa de negoci proporciona als seus clients les funcionalitats de negoci específiques en un domini particular. A la plataforma J2EE la lògica de negoci s'implementa amb components de negoci reutilitzables anomenats Enterprise *JavaBeans* (EJB), que s'empaqueten, es despleguen i s'executen al contenidor d'EJB del servidor d'aplicacions.

Sun Microsystems recomana fer servir els EJB de sessió per implementar la capa de negoci, encara que cada cop més s'accepten altres alternatives com els *Plain Old Java Objects* (POJO). Amb la darrera especificació de Sun, EJB 3.0, es resolen molt dels inconvenients d'utilització de l'especificació 2.x.

La gran avantatja dels POJO és que són objectes molt lleugers que no afegeixen cap tipus de càrrega addicional (gestió de transaccions, seguretat, cicle de vida, etc.). Això els fa molt fàcils de desenvolupar, a més de proporcionar molt bon rendiment. El seu principal inconvenient és que no tenim cap de les característiques que tenen els EJB i si les necessitem les haurem d'implementar (o fer servir algun *framework* que els proporcioni).

2.2.3 Capa integració

La capa d'integració s'encarrega d'accedir i tractar les dades que poden estar emmagatzemades en fonts molt heterogènies (bases de dades relacionals, sistemes *legacy*, bases de dades orientades a objectes, transaccionals, ERP, etc.).

La capa d'integració obté les dades de les diferents fonts de dades i les passa a la capa de negoci perquè aquesta darrera faci les transformacions adients en cada cas.

La divisió de la capa intermèdia en capa de negoci i capa d'integració sol ser purament lògica, és a dir, els components de les dues capes són físicament a la mateixa màquina i, possiblement, desplegats dins el mateix contenidor.

3 Anàlisi de patrons en J2EE

No podríem començar un estudi i anàlisi de *frameworks* sense fer una introducció a una de les eines de més popularitat en el desenvolupament de programari: **els patrons**. Els dissenyadors i experts en l'orientació a objectes van formant un gran repertori de principis generals i d'expressions que ens guien al crear programari. A tots dos podem anomenar-los patrons.

Els patrons són una eina de creixent ús dins de la enginyeria del programari que ens permeten aplicar els avantatges de la reutilització a tots els àmbits del desenvolupament, especialment a l'anàlisi i al disseny. Els patrons ens presenten solucions ja provades i documentades que ens poden ajudar a refinar els artefactes resultants de les diferents etapes del cicle de vida del desenvolupament.

Segons la descripció del mòdul 1, d'enginyeria del programari orientat a objectes, *un patró és una plantilla que fem servir per a solucionar un problema durant el desenvolupament del programari per tal d'aconseguir que el sistema desenvolupat tingui les mateixes bones qualitats que tenen altres sistemes on s'ha aplicat la mateixa solució amb èxit anteriorment.*

3.1 Tipologies de patrons

Hi ha diferents classes de patrons i depenent del nivell conceptual del desenvolupament on s'apliquin, es distingeixen de més abstractes a més concrets: patrons d'anàlisi, patrons arquitectònics, patrons d'assignació de responsabilitats, patrons de disseny i patrons de programació (modismes).

Exemples de la classificació de patrons serien els següents:

- Patrons d'anàlisi: Objecte Compost (*Composite*).
- Patrons arquitectònics: MVC.
- Patrons d'assignació de responsabilitats: patró Expert (*Expert*), recomanem el llibre, UML y patrones de Craig Larman.
- Patrons de disseny: Baix acoblament (Low coupling).
- Patrons de programació o modismes: Enumeracions Java.

Podeu consultar http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html.

3.2 Catàleg de Patrons

Dins de J2EE, s'ha notat que apareixen problemes similars en tots els projectes. També han sorgit solucions similars per tots aquests problemes, encara que les estratègies d'implementació variaran, les solucions generals són bastant similars.

Els patrons segueixen una estructura que descriu una informació que normalment fa referència a un context-problema-solució. Els patrons J2EE s'han configurat d'acord a una plantilla de patró definida orientada a la estructura anterior, on cada secció de la plantilla contribueix a entendre al patró particularment.

La plantilla consta de les següents seccions.

- **Context:** Conjunt d'entorns sota els quals existeix el patró.
- **Problema:** Descriu els problemes de disseny que se ha trobat el desenvolupador.
- **Causes:** Llista les raons i motius que afecten al problema i la solució. La llista de causes il·lumina les raons per les que un podria haver escollit utilitzar el patró i proporciona una justificació del seu ús.
- **Solució:** Descriu breument la solució i els seus elements en més detall.

- **Conseqüències:** Aquí es descriuen les compensacions del patró, aquesta secció s'enfoca en els resultats de la utilització d'un patró en particular o de la seva estratègia, i anota els pros i els contres que podrien resultar de l'aplicació del patró.
- **Patrons Relacionats:** Aquesta secció llista altres patrons rellevants en el catàleg de patrones J2EE o altres recursos externs, com els patrons de disseny [GoF]. Per cada patró relacionat, hi ha una breu descripció de la seva relació amb l'altre patró que es descriu.

Descomposició explicada en la pàgina oficial de Sun: http://developer.java.sun

Per a més informació no dubteu en acudir al llibre "Patrones de Diseño" de E.Gamma et al., Ed.Addison-Wesley.

A continuació es pot veure una representació gràfica del Catàleg de Patrons Principals de J2EE (*Core J2EE Patterns*)

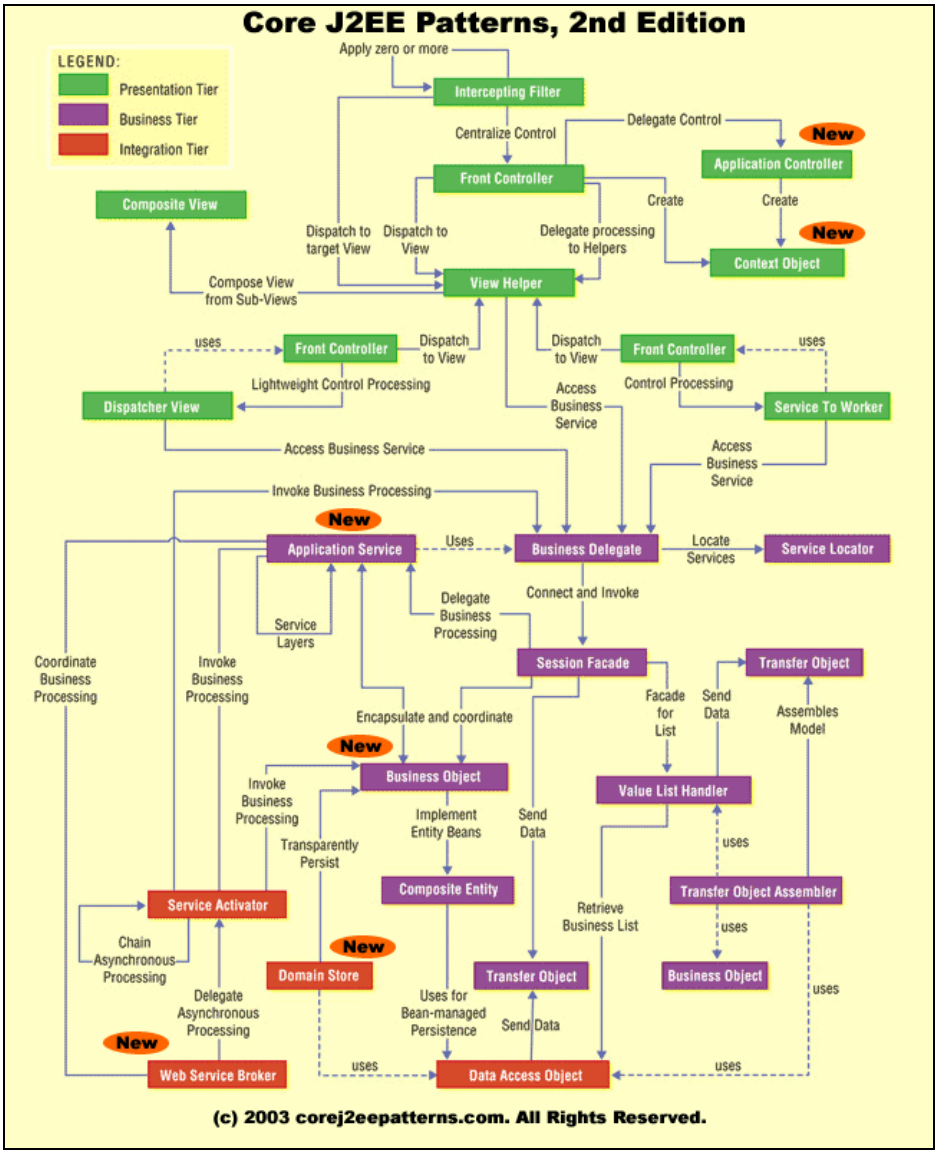


Figura 2. Patrons J2EE

No entra dins de l'àmbit del projecte descriure el catàleg sencer però, més endavant, podrem analitzar algun d'ells, sobre tot, a l'hora de trobar una solució a un problema plantejat dins del projecte.

3.3 Què és un bastiment?

Un bastiment (framework en anglès) és un conjunt de classes predefinides que hem d'especialitzar i/o instanciar per a implementar una aplicació o subsistema estalviant temps i errors. El bastiment ens ofereix una funcionalitat genèrica ja implementada i trobada que ens permet centrar-nos en allò específic de la nostra aplicació.

Un bastiment comparteix el control de l'execució amb les classes que el fan servir seguint el "principi de Hollywood": 'No cal que ens truquis, nosaltres et trucarem'. Això significa, que el bastiment implementarà el comportament genèric del sistema i cridarà les nostres classes en aquells punts on calgui definir un comportament específic.

Exemples de bastiments:

- Esquemes d'interfície gràfica d'usuari com poden ser les *Foundation Classes* de Microsoft, el *Model-View-Controller* de Smaltalk-80.
- Esquemes de persistència, com pot ser *Hibernate*.

3.4 Avantatges i inconvenients de l'ús de bastiments

Quan fem servir un bastiment, reutilitzem un disseny estalviant temps i esforç donant-nos una implementació parcial del disseny que volem reaprofitar.

El principal inconvenient de l'ús de bastiments és la necessitat d'un procés inicial d'aprenentatge ja que per reutilitzar el disseny del bastiment, primer l'hem d'entendre completament (corba d'aprenentatge).

3.5 Relació entre bastiments i patrons

La principal diferència entre un patró i un bastiment és que el bastiment és un element de programari, mentre que el patró és un element de coneixement sobre el programari, és a dir, un patró dóna una solució conceptual, que en ocasions pot trobar la seva representació en un bastiment. És a dir, l'aplicació d'un patró partirà d'una idea però no d'una implementació; de fet, el patró s'haurà d'implementar des de zero adaptant-lo al cas concret. En canvi, l'aplicació d'un bastiment partirà d'un conjunt de classes ja implementades la funcionalitat de les quals estendrem sense modificar-les.

Des del punt de vista dels patrons, un bastiment pot implementar per nosaltres un conjunt de patrons. És a dir, que el disseny que reaprofitem pot estar basat en patrons reconeguts igual que si l'haguéssim fet nosaltres. En aquest cas, entendre els patrons en els quals es basa un bastiment ens ajudarà a entendre el seu funcionament i a poder-lo utilitzar eficaçment.

4 El model MVC

4.1 Arquitectura en capes

En l'estudi anterior de l'arquitectura JEE , ens hem centrat en la manera comú en que es distribueix el sistema, una distribució en capes, que permet aïllar la lògica de l'aplicació i convertir-la en capa intermitja ben definida i lògica del programari. En la capa presentació es realitza relativament poc processament de l'aplicació; les finestres envien a la capa intermitja peticions de treball, i aquest es comunica amb la capa de persistència de l'extrem.

4.2 Visibilitat directa respecta a la capa presentació

Quin tipus de visibilitat haurien de tenir entre sí el negoci amb la capa de presentació? Com haurien de comunicar-se les pantalles amb les classes no relacionades amb elles? Sempre es busca disminuir l'acoblament directe dels components amb la presentació per què en teoria, també busquem la reutilització i poder aprofitar diferents components de negoci i vista per diferents aplicacions.

El principi que s'aplica en aquestes circumstàncies és el patró Separació Model-Vista, definint per primera vegada en "*A System of Pattern*" per Buchmann, Meunier, Rohnert, Sommerlad i Stal en 1996, amb el nom de **Model-Vista-Controlador**, MVC.

El patró de disseny MVC compleix amb un dels enunciats més antics de la programació: separar la presentació, de l'accés a les dades i de la lògica del negoci.

El disseny d'aplicacions web seguint l'arquitectura funcional abans descrita es reflexa en el model lògic. La capa presentació s'implementa segons el patró Model-Vista-Controlador, que té l'objectiu de separar la presentació de l'aplicació.

- El **Model** disposa de les dades i del domini de l'aplicació, és independent de la representació de les dades.
- Les **Vistes**, mostren la informació a l'usuari segons les necessitats.
- El **Controlador**, rep les peticions de l'entrada i determina si la direcciona al model o bé mostra una vista determinada, és l'únic que pot rebre peticions externes.

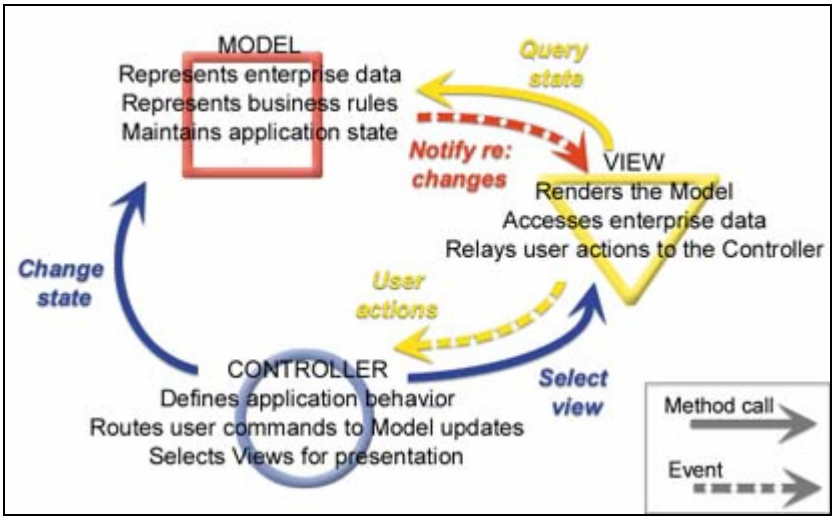


Figura 3. Funcionament del Model-Vista-Controlador

4.2.1 Orígens del model MVC: els models 1 i 2.

Tot i que un primer intent en el desenvolupament d'aplicacions web va ser utilitzar els *CGIs*, on és definia una interfície a través de la qual es podia comunicar entre sí un programa i un servidor Web, normalment escrit en C o *Perl* que corria al costat del servidors, l'aparició de pàgines dinàmiques i aplicacions més complexes van esdevenir l'aparició dels servidors d'aplicació Web. Va ser llavors on es va començar a veure la necessitat de separar la presentació de les dades.

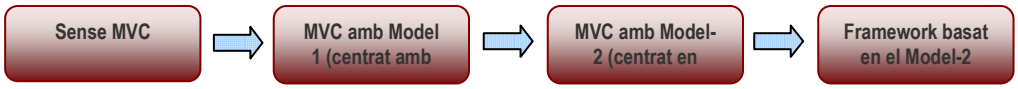


Figura 4. Historia i evolució del MVC

Els termes **Model 1** i **Model 2**, apareixen de les primeres especificacions de les *Java Server Pages* (JSP) on es descrivien dos patrons bàsics per construir aplicacions basades en aquesta tecnologia. El Model 1 és molt simple, el concepte de **Controlador** no existeix, i bàsicament tenim una pàgina JSP que accedeix a les classes del model i selecciona la vista segons la petició.

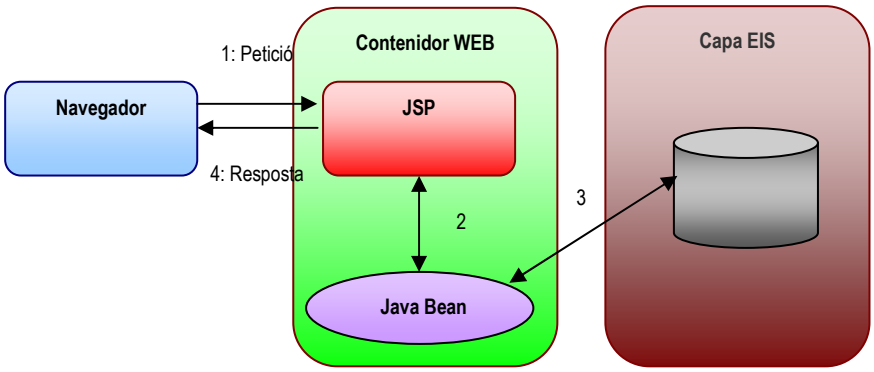


Figura 5. MVC: el Model-1

En una arquitectura amb Model 2 ja disposem d'un controlador. Existeix un servlet que actua com a controlador central que rep totes les peticions i a partir de la direcció origen, de les dades d'entrada, el estat actual de l'aplicació selecciona la següent vista a mostrar.

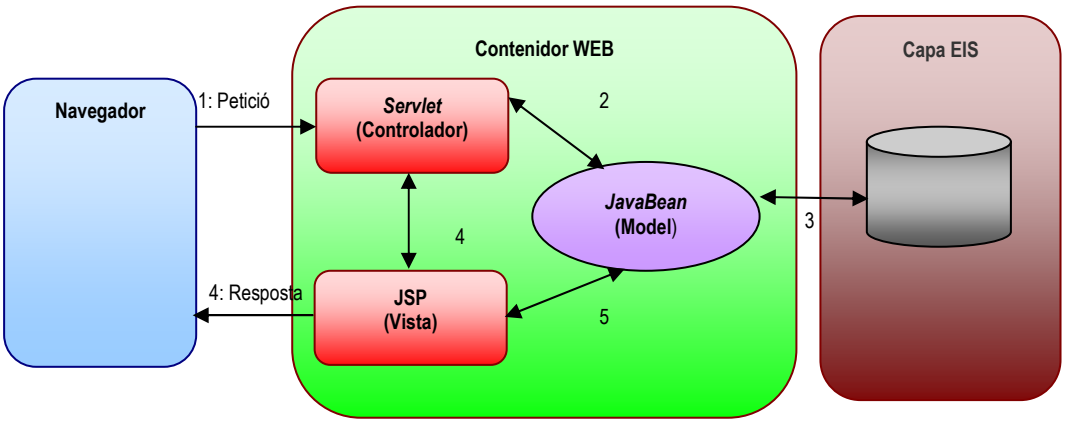


Figura 6. MVC: el Model-2

Les aplicacions basades en Model-2 permeten una separació clara de rols a la capa de presentació, són més flexibles, mantenibles i permeten tractaments centralitzats per a totes les peticions (per exemple, aplicar seguretat i *logs*). Per tot això, és recomanable seguir aquesta arquitectura per a la capa de presentació.

Si seguim una arquitectura basada en Model-2, hi ha un conjunt de tasques repetitives que s'han d'implementar per a totes les aplicacions, per exemple, rebre els paràmetres d'entrada de les peticions i fer-hi validacions, cridar a la lògica de negoci, triar la següent vista a mostrar, etc. Aquestes tasques es poden implementar en un framework per a la capa web que facin servir tots els desenvolupaments i d'aquesta manera no s'hagi d'implementar-les per a cada aplicació.

Els desenvolupadors crearan la capa web usant o estenent les classes i interfícies del *framework*. L'ús d'un *framework* per a la capa web basat en una arquitectura Model-2 us proporciona els següents avantatges:

- Desacobla la capa de presentació de la capa de negoci en components separats.
- Simplifica i estandarditza la validació del paràmetres d'entrada.
- Simplifica la gestió del flux de navegació de l'aplicació.
- Proporciona un punt central de control.
- Permet un nivell molt alt de reutilització.
- Imposa la mateixa arquitectura per a tots els desenvolupaments.
- Simplifica moltes tasques repetitives.

4.3 Exemples de bastiments per la capa web

Avaluar tots i cadascun dels bastiments webs excedeix en escreix aquest projecte, només sigui el cas, perquè qualsevol pot desenvolupar-ne un segons les seves directrius. És per aquest motiu que ens centrarem en aquells que per un o altre motiu han estat i seguiran estant en les solucions més utilitzades. Unes vegades simplement per la seva gran acceptació dins la seva comunitat d'usuaris, d'altres per la influència que les grans companyies en fan i d'altres per les solucions que ens donen.

4.4 Struts

Struts és un dels frameworks més utilitzats per aplicacions web java, desenvolupat per *Apache Software Foundation* (http://jakarta.apache.org/struts) té llicencia open-source i implementa el model 2 de MVC.

Struts proporciona un conjunt de classes, TAG-LIBS, un *Servlet* que actua com controlador, la integració amb el Model i facilita la construcció de vistes. El Model o lògica de negoci és la part que ens correspon desenvolupar. Es per aquest motiu que Struts és una plataforma sobre la qual muntem la lògica de negoci, i aquesta plataforma ens permet dividir la lògica de la presentació entre altres coses.

4.4.1 Implementació del patró MVC en Struts

Struts implementa el patró MVC utilitzant una combinació de JSP, etiquetes o *tags* JSP i un *Servlet* Java. En el següent gràfic es mostra el seu funcionament:

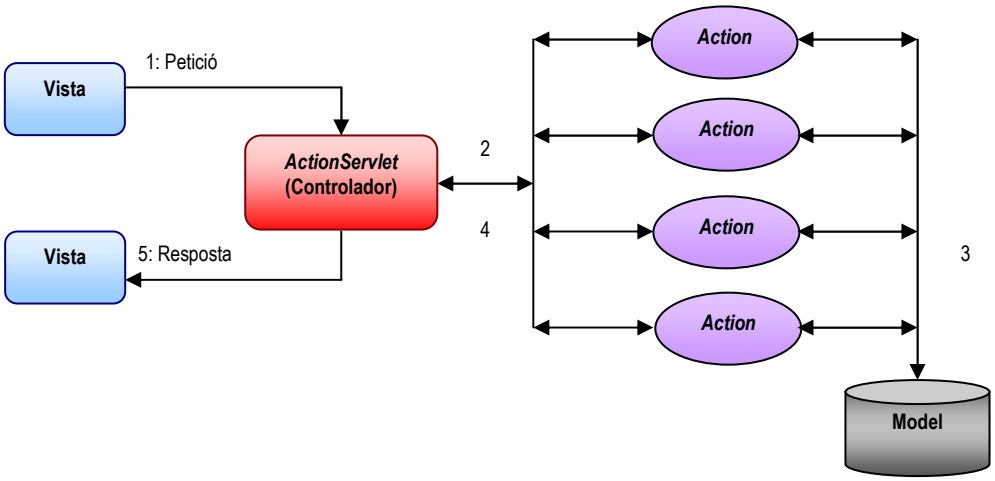


Figura 7. Funcionament d'Struts

4.4.2 Vocabulari específic

- ***Actions***: és la possible acció a invocar/realitzar. Són objectes que hereten de la classe *Action* on es descriu que és el que es farà.
- ***ActionMaping:*** mapeja les URLs a accions (objectes). És a dir, se li assigna a la classe *Action* de manera que puguin ser invocades des de el client com un *string*.
- ***ActionServlet:*** és el Servlet controlador.
- ***ActionForm:*** encapsen els paràmetres de les peticions dels clients presentant-los com a dades d'un formulari. Representen les dades d'entrada de l'acció a realitzar.

4.4.3 Funcionament bàsic

1. La Vista realitza una petició.
2. La petició és rebuda pel *ActionServlet* (patró *FrontController*), i aquest, després de verificar en un fitxer XML, determina el nom de la classe *Action* que portarà a executar la lògica de negoci.
3. La classe *Action* accedeix a la lògica de negoci del Model associat a l'aplicació.
4. Quan la *Action* és completada i processada retorna el control al *ActionServlet*. La classe *Action* retorna una clau que indica el resultat, i que és usada pel *ActionServlet* per determinar a on tindrà que ser dirigit el resultat per a la presentació.
5. La petició és completada quan el *ActionServlet* respon enviant una petició a la Vista que indica la clau retornada i la Vista 2 presenta el resultat.

El controlador és el responsable de detectar una entrada d'usuari, i de seleccionar la següent Vista pel client. El controlador ajuda a separar la presentació del model i és l'únic punt on totes les peticions de client són inicialment processades.

La intel·ligència del controlador es defineix en un arxiu *XML* anomenat *struts-config.xml* En aquest arxiu es guarden els mapejos, les accions i els formularis existents amb els que treballarà el *framework*.

```
<form-bean name="userForm" type="org.apache.struts.validator.DynaValidatorForm">
        <form-property name="user" type="org.appfuse.model.User"/>
</form-bean>
<action path="/user"type="org.springframework.web.struts.DelegatingActionProxy"
        name="userForm" scope="request" parameter="method" validate="false">
        <forward name="list" path="/userList.jsp"/>
        <forward name="edit" path="/userForm.jsp"/>
</action>
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation" value="/WEB-INF/action-
servlet.xml"/>
</plug-in>
```

Si pel costat del controlador existeix un *Servlet* que s'encarrega de tot, pel costat de la vista *Struts* disposa d'una biblioteca de *Tags* per embeir en el HTML de manera que sigui fàcil accedir als *beans* i generar les vistes.

```
<display:table name="users" class="list" requestURI="" id="userList" export="true">
        <display:column property="id" sort="true" href="editUser.html"
                paramId="id" paramProperty="id" titleKey="user.id"/>
        <display:column property="firstName" sort="true" titleKey="user.firstName"/>
        <display:column property="lastName" sort="true" titleKey="user.lastName"/>
        <display:column titleKey="user.birthday" sort="true" sortProperty="birthday">
                <fmt:formatDate value="${userList.birthday}"  pattern="${datePattern}"/>
        </display:column>
</display:table>
```

Struts disposa de mecanismes de validacions de les entrades ingressades. Aquesta part del framework permet agregar validacions dels camps dels formularis que s'executen tan al costat del client (amb *JavaScript*) com del costat del servidor, així com definir rutines de validacions més utilitzades. Tot això es configura en el arxiu ***validation.xml***.

La maquetació de l'aplicació Web es facilitada a través de ***Struts Tiles***, que permeten composar un pàgina a partir de porcions de pàgines o plantilles. Això permet realitzar canvis en el disseny sense haver de modificar cada un dels JSP de la nostra aplicació.

Struts en principi és independent del motor de visualització encara que generalment s'escull JSP per mostrar vistes. Existeix formes per a que Struts enviï vistes per a que siguin processades per motors de plantilles com *Velocity*, transformadors d'estils de documents XSLT o d'altres bastiments com JSF.

4.5 Tapestry

Tapestry és un altre *framework* open-source que segueix el model 2 del MVC i està mantingut per la comunitat Apache. Una de les seves principals característiques és que es basa en un model de components, no utilitza JSP com a motor de plantilles, sinó que defineix un model de pàgines i components.

Tapestry proporciona una estructura consistent i assumeix responsabilitats relacionades amb la API *Servlet,* com la construcció de URL, peticions i respostes, la validació del usuari, la localització i internacionalització, proporcionant al programador una API de molt alt nivell per a que es centri en la lògica de negoci.

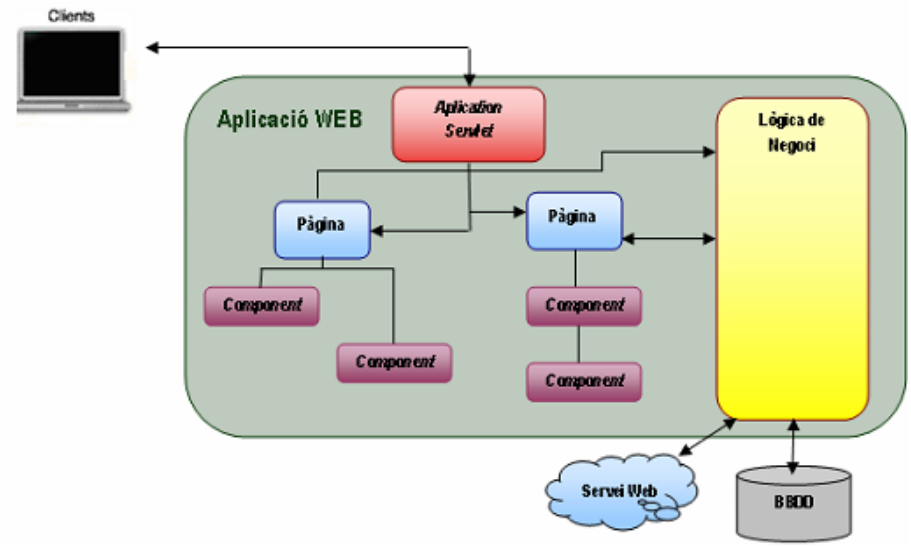


Figura 8. Funcionament de Tapestry

4.5.1 Conceptes claus

Tapestry divideix una aplicació en un conjunt de pàgines. Cada pàgina està formada per un *template* i altres components, i cada component pot està format a la vegada per altres components, sense que existeixi cap tipus de limitació en la seva profunditat.

Template: Un template pot ser per una pàgina o per un component. Conté HTML pla amb alguns Tags marcats amb un atribut especial per indicar que en aquest lloc han de situar-se els components.

Component: Un objecte reutilitzable que pot ser utilitzat com a part d'una pàgina. Els components generen html quan es mostra la pàgina i pot participar quan es selecciona un enllaç o s'envia un formulari. També es pot utilitzar per crear nous components.

Paràmetre: Els components tenen paràmetres que serveixen per enllaçar les propietats dels components amb les propietats de la pàgina. Els components generalment llegeixen els paràmetres però a vegades poden modificar-los fent que s'actualitzin les propietats de la pàgina que estaven relacionades amb aquest paràmetre.

L'especificació d'una pàgina es realitza en un fitxer XML amb extensió ***.page***. Cada pàgina té una classe que la implementa. El nom d'aquesta classe ve donada en la especificació. La plantilla d'una pàgina és un fitxer en HTML on es descriu l'aparença visual de la pàgina.

Les vistes són HTML estàndard. No existeixen biblioteques de tags ni codi java. La única diferència és la existència d'alguns atributs extres en els elements que apareixen. La inserció de l'atribut ***jwcid*** (Java Web Component ID) en una etiqueta determinada provoca que Tapestry la substitueix per un component.

```
```

En aquest exemple l'element *input* fa referència a un component *TextField*.

Per la resolució d'expressions Tapestry utilitza el llenguatge especial OGNL (*Object Graph Navigation Language*) que permet expressar l'accés a un valor d'un objecte en forma de cadena de text.

```
elCarro.getItemCarro[0].getProfucto().getPrecio()
```

En OGNL seria

```
elCarro.itemCarro[0].producto.precio
```

Hi ha utilització d'esdeveniments a partir de la subscripció de *listeners* als components que llancen els citats esdeveniments.

```
<form jwcid="@FOrm" listener="ognl:listener.enviar">
```

En Tapestry tot són components per tant, al crear pàgines, es creen components i es possible crear components que inclogui components i la facilitat amb les que es puguin crear nous components és la mateixa que per crear pàgines.

Els components que reben la entrada dels usuaris permeten la validació a través de dos paràmetres: *dysplayname* i *validators*.

4.6 JavaServer Faces

La tecnologia Java Server Faces és un framework model 2 de construcció d'interfícies d'usuari (UI) per aplicacions Web amb tecnologia Java que segueix l'especificació (JSR 127) de la *Java Community Process* (JCP).

Els components que formen part de Java Server Faces són les següents:

- Un conjunt d'API per representar components de una interfície d'usuari i administrar el seu estat, manegar esdeveniments, validar entrada, definir un esquema de navegació de les pàgines i donar suport per internacionalització i accessibilitat.
- Un conjunt per defecte de components per la interfície d'usuari.
- Dos llibreries d'etiquetes, (*JavaServer Pages (JSP) custom tag libraries*) personalitzades per a JavaServer Pages que permeten expressar una interfície JavaServer Faces dins d'una pàgina JSP.
- Un modelo d'esdeveniments al costat del servidor.
- Administració d'estats.
- Bans administrats.

Un aspecte molt important dins de les Java Server Faces és el cicle de vida de les pàgines, el qual es similar a d'una pàgina JSP: el client fa una petició HTML i el servidor respon amb la pàgina en HTML, però, donades les característiques que ofereix JSF, el cicle de vida inclou alguns passos més.

El procés d'una petició estàndard inclou sis fases, com es mostra en la següent figura, representats pels rectangles diferents al color verd.

Els rectangles verds representen qualsevol execució d'un esdeveniment produït en el cicle de vida.

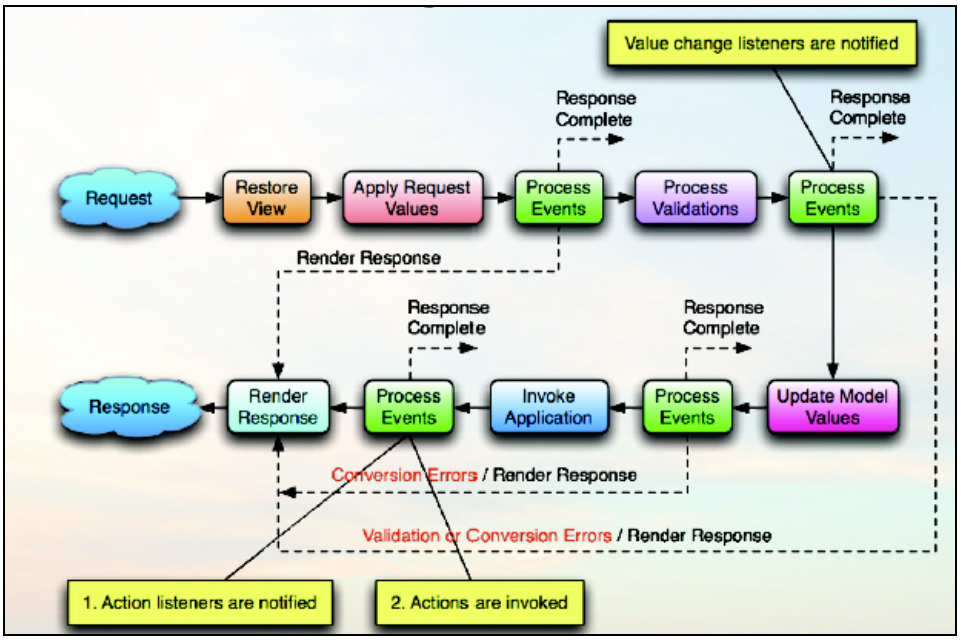


Figura 9. Cicle de vida JSF

- ***Restore View***: Crea l'arbre de components de la pàgina sol·licitada i carrega l'estat d'aquesta si ja s'havia sol·licitat prèviament.
- ***Apply Request Value***: Itera sobre l'arbre de components recuperant l'estat de cadascun assignant els valors que ve des del client.
- ***Process Validations****:* Es realitzen les validacions de cada component.
- ***Update Model Values***: S'actualitzen els valors dels *backing beans* del model.
- ***Invoke applications***: S'executa la lògica del negoci i es selecciona la pròxima vista lògica.
- ***Render Response****:* Armem la vista amb el estat actualitzat dels components i s'envia als components.

4.6.1 Funcionament bàsic

El punt d'entrada a l'aplicació és el ***FacesServlet***, que s'encarrega de controlar el cicle de vida de cada petició. JSF permet definir la lògica de navegació a través d'un o més arxius de configuració (***faces-config.xml***).

La lògica es construeix a través de regles de navegació. Cada regla s'activa segons es compleixi el patró indicat en el element ***from-view-id***.

```
<navigation-rule>
 <from-view-id>/buscador.jsp</from-view-id>
 <navigation-case>
    <from-out-come>exit</from-outcome>
    <to-view-id>/resultat.jsp</to-view-id>
 </navigation-case>
</navigation-rule>
```

El model de components permet l'enllaç de valors (*value bindings*), de mètodes (*method binding*) i de components (*component binding*).

JSF implementa un model que permet la notificació mitjançant esdeveniments i la subscripció a aquests esdeveniments mitjançant *listeners* de manera similar al model utilitzats en, per exemple, *Swing*.

La intel·ligència de la validació d'entrada resideix en els Validadors (*Validators*). Un validador s'encarrega de realitzar comprovacions sobre el valor un component durant la fase *Process Validations*. A cada component que rep la entrada de valors per part de l'usuari se li poden registrar zero o més validadors.

La codificació dels valors dels components per mostrar-los en la vista i la descodificació necessària dels valors que arriben de les peticions varia depenent del dispositiu. En JSF, aquesta codificació/decodificació es poden realitzar de dues maneres, utilitzant un model d'implementació directa o utilitzant un model d'implementació delegada.

JSF permet la creació de components propis però, la creació no es tan senzilla i es necessiten crear uns quants arxius depenent del tipus de component que es desitja crear.

4.7 Spring MVC

4.7.1 Entorn Model-Vista-Controlador

Spring és un entorn basat en peticions comparable a Struts. L'entorn defineix interfície seguint el patró d'*Strategy* (permet disposar de diversos mètodes per a resoldre un problema i escollir-ne un en temps d'execució) per totes les responsabilitats que han de ser gestionades per un entorn modern basat en peticions. Totes les interfícies estan estretament aparellades a l'API de *Servlet* per oferir totes les seves capacitats. La classe *DispacherServlet* és el controlador frontal de l'entorn i és responsable de la delegació de control a les diferents interfícies durant les fases d'execució d'una petició HTTP. Les interfícies més importants definides per Spring MVC i les seves responsabilitats són:

- ***HandlerMapping*** **:** seleccionant objectes que maneguen peticions entrants (*handlers*) basats en qualsevol atribut o condició interna o externa a les peticions.
- ***HandlerAdapter*** **:** execució d'objectes que maneguen peticions entrants
- ***Controller****:* entre el Model i la Vista per gestionar les invocacions entrants i redirigir-les a la resposta correcta
- ***View*** **:** responsable de tornar una resposta al client
- ***ViewResolver*** **:** selecciona la vista basada en un nom lògic per la vista (l'ús no és estrictament requerit)
- ***HandlerInterceptor*** **:** intercepció de peticions entrants comparable però no igual als filtres de Servlet (l'ús és opcional però no controlant per *DispatcherServlet*)
- ***LocaleResolver*** **:** resolent i gravant opcionalment amb els Paràmetres regionals. (*Locale*) d'un usuari individual
- ***MultipartResolver*** : facilita la feina de la pujada de fitxers amb el wrapping de peticions entrants

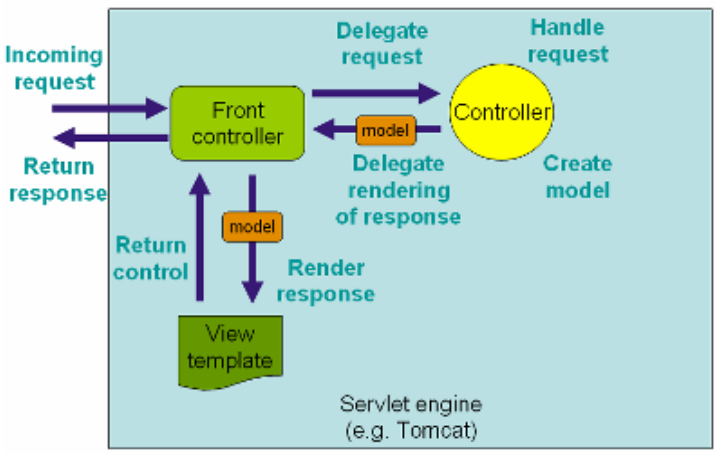


Figura 10. Spring MVC

Cada interfície de esmentada té una responsabilitat important arreu de l'entorn. Les abstraccions ofertes per aquestes interfícies són prou potents com per permetre un ampli conjunt de variacions a les seves implementacions. *Spring* MVC incorpora implementacions de totes aquestes interfícies, que juntes, ofereixen un potent conjunt de funcionalitats damunt de l'API de *Servlet*. Tanmateix, el desenvolupadors i fabricants són lliures d'escriure d'altres implementacions.

Les característiques més destacades d'aquest marc de treball són:

- Mapeig directe dels objectes de negoci: el bastiment no imposa que els nostres objectes de negoci implementin cap interfície específica.
- Separació de rols: cada component de Spring MVC té assignat un rol específic seguint els patrons de responsabilitat GRASP (veure [LAR]).
- Entorn lleuger: permet modelar l'aplicació utilitzant POJOs.
- Spring: naturalment la integració amb Spring és total.

4.8 Altres bastiments

- ***WebWork***, http://www.opensymphony.com/webwork, bastiment per crear aplicacions web proporciona validació tant a nivell d'interfície d'usuari com a nivell de dades. Utilitza *OpenSymphony Software License* (http://www.opensymphony.com/) i a partir de la versió 2.2, l'equip de desenvolupament del framework es va centrar en la integració amb Struts per la creació d'Struts 2.0.
- ***Apache Cocoon*** o *Cocoon* http://cocoon.apache.org és un entorn de programació de pàgines web dinàmiques al voltant dels conceptes de procés de transformacions en seqüència (*pipeline*), separació d'aspectes i componentització.
- **ASP.NET** http://www.asp.net/ es un conjunt de tecnologies de desenvolupament d'aplicacions web comercialitzat per Microsoft.. Es utilitzat per programadors per construir webs domèstiques, aplicacions web i serveis XML. Forma part de la plataforma .NET de Microsoft i és la tecnologia successora de la tecnologia Active Server Pages (ASP).
- **Ruby on Rails**, http://www.rubyonrails.org.es conegut com **RoR** o **Rails** està basat en el llenguatge Ruby. No només aplica la capa de presentació sinó que es possible definir des de la lògica de navegació fins l'accés a dades.
- **phpMVC** http://www.phpmvc.net és un dels frameworks que implementen MVC més complerts, sobretot perquè en la capa d'accés a dades i en la capa de presentació suporta llibreries tan conegudes com: *Pearl*, *phpLIB*, *Smarty* i moltes d'altres. És una implementació directa del model Struts que usen els Servlets de java.
- **cakePHP** http://www.cakephp.org una de les llibreries MVC perfecte per programar ràpidament una aplicació sense tenir una corba d'aprenentatge que es dilati massa en el temps.

4.9 El futur

Així com hem explicat l'evolució del model web de la capa presentació la web també té un futur pròxim. Aquesta evolució no es refereix a una actualització o evolució d'Internet, sinó, que més aviat es refereix als canvis que es fan a la plataforma. Aquesta nova percepció de la generació Web és el concepte de Web 2.0.

Tim O'Reilly, el primer en esmentar aquest concepte es refereix a aquesta revolució com: "*Web 2.0 és la revolució del negoci en la indústria dels ordinadors causat per la mobilitat de la plataforma d'Internet, i un intent d'entendre les regles per a l'èxit sobre el que és nou a la plataforma*".

Referint-se a la percepció que la segona generació de la Web es basava en comunitats i en **serveis de *hosting***, com ara els espais web de treball en xarxa, les **wikis**, i ***Folksonomy folksonomies*** que faciliten la col·laboració i en el fet de compartir entre usuaris espais per a fotografies, textos i vincles amb altres Web-sites, els tres exemples més clars són **Flick**, **Del.icio.us** i el ja famós **YouTube**. O'Reilly va titular una sèrie de conferències al voltant d'aquest concepte i des d'aleshores ha estat àmpliament adoptat.

En definitiva el que es vol dir és que hi ha una nova manera d'utilitzar la web. Mentre abans les aplicacions es basaven en millorar l'experiència de l'usuari interactuant amb ell de manera directa a través del seu portal o aplicació web, les noves aplicacions permeten interactuar de manera indirecta exposant els serveis en diferents formats per a que no només es pugui accedir de manera normal sinó que altres aplicacions puguin accedir a aquestes serveis i combinar-los amb altres aconseguint que diferents usuaris arribin a la nostra informació també.

4.10 Conclusions

Un a vegada analitzats tots els frameworks hem de veure quins es podrien adaptar millor a les nostres necessitats, per fer-ho, donarem unes directrius a seguir on compararem cadascun dels bastiments vistos. No els compararem tots sinó, que ja hem fet una selecció prèvia. Aquesta selecció s'ha fet seguint els criteris de comunitat de recolzament, anys d'utilització i experiència.

4.10.1 Criteris d'avaluació

Els criteris per seleccionar els frameworks que seguirem són:

- **Llistes i taules**: com és de fàcil ordenar o paginar una llista?
- **Validacions:** quant és de fàcil fer validacions?
- **Test:** quant és de fàcil testejar els controladors fora del contenidor?
- **Integració amb Spring:** quants frameworks es poden integrar a Spring?
- **Internacionalització:** com està de suportat la internacionalització en els controladors?
- **Decoració de pàgines:** quins mecanismes de decoració suporta el framework?
- ***Tools*:** tenim eines adients pel framework?
- **Comunitat d'usuaris:** quin és el suport real de la gent?
- **Treball:** quina és la demanda de feina amb referència al framework?
- **Aprenentatge:** quant de difícil és prendre'l?
- **Llicencia:** quines empresa li donen suport?

4.10.2 Comparativa

	Struts	JSF	Tapestry	Spring MVC
Llistes i taules ordenades	Utilització de tags	Cal aplicar lògica si volem ordenació	Té un component que ho resol	Llibreria de Tags.
Validacions	*Commons Validator*	Fàcil de configurar	Validació robusta	*Commons Validator*
Test	*StrutsTestCase*	Senzill de provar	Difícil ja que les classes són abstractes	EasyMock, jMock, Spring Mocks
Integració en Spring	Sí	Sí	Sí	Sí
Internacionalització	Sí	Sí	Sí	Sí
Decoració de la pàgina	Tiles	Tiles	*SiteMesh*	Tiles
Tools	Molts (*Beehive's*) PageFlow	Cada vegada millors.	*Spindle*	Spring té el Spring IDE – només com a validador de XML I no com a eina web
Comunitat d'usuaris	Struts és la més utilitzada	Popularitat cada vegada més ràpida	Petita	Pren popularitat sobre tot gràcies al seu bastiment general J2EE.
Corba d'aprenentatge	Mitjana	Baixa	Baixa	Mitjana
Llicencia	Apache	Especificació Sun	Apache	IBM

Figura 11. Comparativa Web frameworks

Struts és el bastiment que més temps ha demostrat que funciona, sis anys, amb una gran quantitat de desenvolupaments empresarials i un gran número de bibliografia en anglès i altres idiomes. És el més popular i amb una comunitat amb bones pràctiques conegudes. Amb una corba d'aprenentatge mitjana està recolzada per la comunitat *Apache*. En contra té la no abstracció del protocol HTTP, és difícil crear components tags propis, no és natural el mapeig dels objectes de negoci i no és una especificació. El camí està clar: l'evolució a *Struts* 2.0.

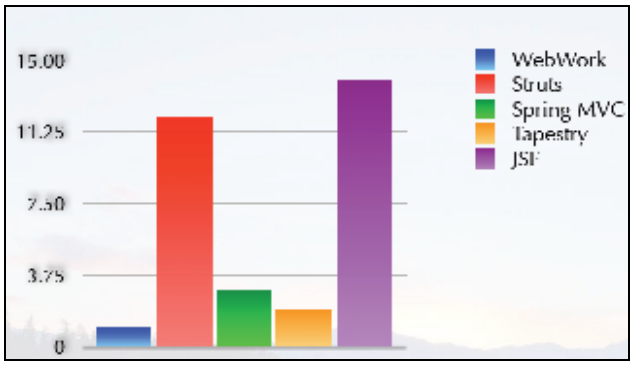
JSF permet separar clarament el contingut de la presentació de la lògica i és una especificació, això vol dir que podem trobar implementacions de codi obert i comercials. Evolució actual importantíssima amb una

gran comunitat i eines de suport. Fàcil aprenentatge. En contra, té que la creació de components és complexa i requereix *JavaScript*.

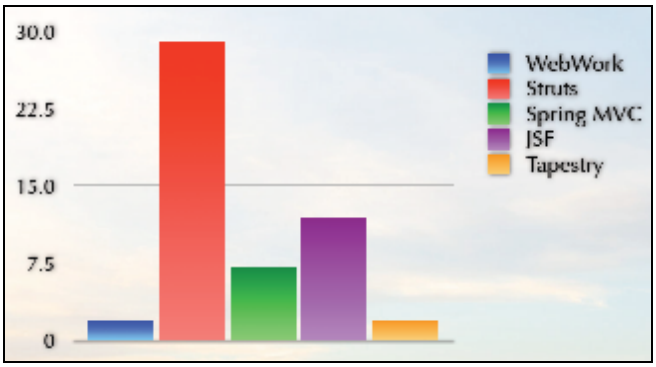
Tapestry permet el desenvolupament de components propis i la separació entre la presentació i la lògica és completa. En contra té que la comunitat és molt petita i no disposa de massa documentació.

Spring té l'avantatge de que disposa un bastiment general per l'arquitectura J2EE i dóna solucions al model MVC oferint una divisió neta entre Controladors, Models i Vistes. És molt flexible ja que implementa tota l'estructura amb interfícies. La capa Web de Spring és una petita part a dalt de la capa de negoci de Spring, que fa veure que sembla una bona pràctica de disseny. Spring ofereix un framework per a totes les capes de l'aplicació, oferint millor la integració amb tecnologies com Velocity, SXLT, FreeMaker i XL.

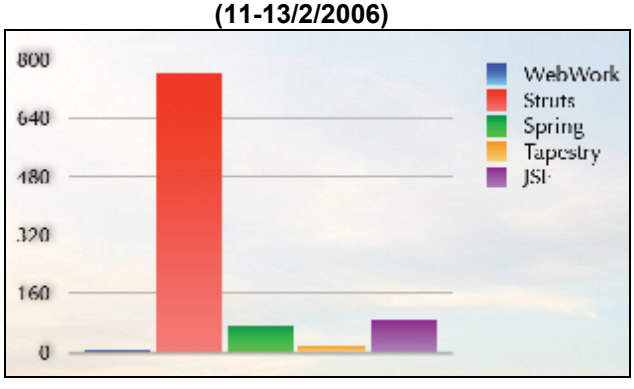
Eines conegudes



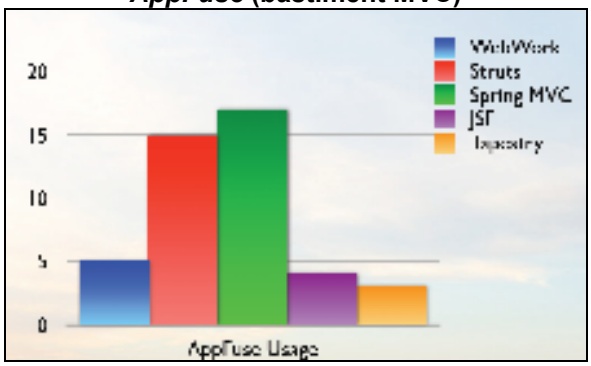
Llibres en Amazon



Demanda de feina en Monster.com (11-13/2/2006)



Utilització de bastiments en la comunitat *AppFuse* (bastiment MVC)



5 Solucions per la capa de negoci

Ja hem vist que l'arquitectura JEE segueix un estil arquitectònic client-servidor multicapa de n-nivells i que les aplicacions es poden dividir bàsicament en tres capes, en aquest apartat veurem la tecnologia per implementar la capa de negoci i d'integració.

El model d'aplicacions per a JEE promou l'ús d'un tipus de components distribuïts anomenats *Enterprise Java Beans* (EJB) per a implementar aquestes dues capes.

5.1 Arquitectura EJB

Enterprise Java Beans és una tecnologia de components de servidor que permet el desenvolupament i el desplegat d'aplicacions empresarials distribuïdes basades en components.

Tutorial Java EE 5: http://java.sun.com/javaee/5/docs/tutorial/doc/bnbls.html

L'especificació EJB és una de les moltes APIs de Java de la Plataforma JEE. EJB encapsula la lògica de negoci d'una aplicació. L'especificació EJB va ser desenvolupada inicialment el 1997 per IBM i adoptada posteriorment per Sun Microsystems (EJB 1.0 i EJB 1.1) i millorada sota la *Java Community Process* com JSR 19 (EJB 2.0), JSR 153 (EJB 2.1) i JSR 220 (EJB 3.0).

L'especificació EJB pretén aportar una manera estàndard d'implementar el codi de negoci del back-end, típicament trobat a les aplicacions d'empresa (en oposició al codi d'interfícies d'usuari anomenat front-end). Aquest codi es trobava molt sovint resolent els mateixos tipus de problemes i es va trobar que solucions a aquests problemes es programaven de forma repetida pels programadors.

Els *Enterprise JavaBeans* intentaven gestionar qüestions tan habituals com la persistència, integritat transaccional i seguretat, de manera estàndard, alliberant els programadors perquè es concentressin en el problema que els ocupava realment. D'acord amb això, l'especificació EJB detalla com un servidor d'aplicacions aporta:

- Persistència JPA.
- Processament Transaccional
- Control de la concurrència
- Esdeveniments usant Java *Message Service*
- Anomenament i serveis de directori JNDI
- Seguretat (*Java Cryptography Extension* JCE i JAAS)
- Implantació de components de programari en un servidor d'aplicacions
- RPC (crides a Procediment Remot) usant RMI-IIOP
- Oferiment de mètodes de negoci usant Serveis web
- Addicionalment, les especificacions *d'Enterpise JavaBean* defineix els rols que juga el contenidor EJB i els EJBs i també com implantar els EJB en un contenidor.

Gradualment un consens en la indústria va anar fent emergir que la virtut primera dels EJBs -habilitar integritat transaccional sobre aplicacions distribuïdes- tenia l'ús limitat per la majoria de les aplicacions empresarials. La funcionalitat oferta per entorns més senzills com Spring i Hibernate era més útil per aplicacions d'empresa.

En l'especificació EJB 3.0 (JSR 220) s'aprecia una clara influència respecte Spring, el seu ús de POJOs i el seu suport a la injecció de dependència per simplificar la configuració i la integració de sistemes heterogenis. Gavin King, el creador d'Hibernate, va participar en el procés d'EJB 3.0 i se'l considera una veu lliure en defensa d'aquesta tecnologia. Moltes funcionalitats d'Hibernate van ser incorporades a la Java Persistence API, el substitut dels Entity Beans a EJB 3.0.

L'especificació dels EJB 3.0 usa fortament les anotacions (meta dades), una funcionalitat afegida al llenguatge Java en la seva versió 5.0, per habilitar un estil de programació molt menys farragós.

D'acord amb això, en termes pràctics, EJB 3.0 és bastant una API completament nova, semblant-se molt poc a les especificacions prèvies d'EJB.

5.2 Tipus d'EJB

Un contenidor d'EJB pot representar dos categories de beans principals:

- *Session Beans* (*beans* de sessió)
  - *Stateless Session Beans* (sense estat)
  - *Stateful Session Beans* (amb estat)
- *Message Driven Beans* (MDBs o Message Beans)

Stateless Session Beans (*beans* de sessió sense estat) és un client simple dins del servisor d'aplicacions, són objectes distribuïts que no tenen estat associat a ells, de manera que s'hi pot accedir concurrentment.

Stateful Session Beans (*beans* de sessió amb estat) són objectes distribuïts que tenen estat, es mantenen al corrent de quin programa que els crida és el que estan tractant durant una sessió. Per exemple, la comprovació en una botiga web podria ser gestionada per un bean de sessió amb estat, que usaria el seu estat per estar al corrent de quin client és el que està al procés de comprovació. Per altra banda, enviar un e-mail a atenció al client podria ser gestionat per un bean sense estat, donat que és una operació puntual i no és part d'un procés de diferents passes. L'estat dels beans de sessió amb estat hauria de ser persistent, però l'accés a la instància del bean està limitat només a un client.

Message Driven Beans són objectes distribuïts que es comporten de manera asíncrona. És a dir, gestionen operacions que no requereixen resposta immediata. Per exemple, un usuari d'un lloc web que clica el botó "mantingueu-me informat de futures actualitzacions" podria disparar una crida a un Message Driven Bean per afegir l'usuari a una llista a la base de dades de l'empresa. Aquesta crida és asíncrona, ja que l'usuari no necessita esperar a ser informat del seu bon o mal funcionament).

Amb les versions anteriors s'utilitzava un altre tipus d'EJB, els EJB d'entitat. Eren objectes distribuïts amb persistència, gestionats perl contenidor de persistència CMP, o gestionada la persistència per ell mateix BMP. Els beans d'entitat van ser reemplaçats per la Java Persistent Api en EJB 3.0.

Altres tipus de beans són els *Enterprise Media Beans* (JSR 86) pensats per la integració d'objectes multimèdia en aplicacions J2EE.

5.3 Spring framework

L'**Spring framework** (abreviant, **Spring**), és un bastiment de codi obert per la plataforma Java. La primera versió va ser escrita per **Rod Johnson**, que inicialment va llençar el producte juntament amb el llibre *Expert One-on-One JavaEE Design and Development* (*Wrox Press*, octubre 2002). També hi ha un marc disponible per la plataforma .NET, Spring.NET.

El marc de treball va ser inicialment llençat al juny del 2003 sota la llicència Apache 2.0. La primera versió major 1.0 va ser distribuïda el març del 2004, amb llançaments addicionals el setembre de 2004 i març de 2005. Encara que *Spring Framework* no força cap model de programació, ha esdevingut amplament popular dintre de la comunitat Java primerament com una alternativa que desplaçaria el model *Enterprise JavaBean*. Per disseny aquest entorn ofereix una gran llibertat als desenvolupadors de Java i a més proveeix solucions fàcils i ben documentades per pràctiques habituals en la indústria.

Mentre les funcionalitats del nucli de l'entorn són usables en una aplicació Java hi ha diferents extensions i millores per construir aplicacions web damunt d'una plataforma Java EE. Gràcies a això, Spring ha aconseguit una gran popularitat i és reconegut pels fabricants com un entorn estratègicament important.

5.4 Funcionalitats clau

- Gestió de la configuració basada en *JavaBeans*, aplicant-hi principis d'Inversió de Control, més específicament usant la tècnica d'Injecció de Dependència. Això ajuda a reduir les dependències de components, en implementacions específiques, sobre altres components.
- Una factoria de Beans central, que és usada globalment.
- Capa genèrica d'abstracció per la gestió de transaccions de base de dades.

- Estratègies preincorporades per la *JTA* i un sol *DataSource* de *JDB*. Això elimina la dependència en un entorn Java EE pel suport a les transaccions.
- Integració amb entorns de persistència com *Hibernate*, *JDO*, *iBatis* i *db4o*.
- Entorn d'aplicació web MVC, construït al nucli de la funcionalitat de Spring. suportant moltes tecnologies per generar vistes, incloent-hi JSP, *FreeMaker*, *Velocity*, *Tiles*, *iText* i *POI*.
- Entorn extensiu de programació orientada a l'aspecte per proveir serveis, com ara gestió de les transaccions. Amb això es millora la modularitat dels sistemes.

Spring Framework pot ser considerat com una col·lecció d'entorns més petits, o entorns dintre d'entorns. La majoria d'aquests estan dissenyats per treballar independentment uns dels altres, tot i que se suposa que junts, milloren les funcionalitats. Aquests mòduls estan dividits en blocs de construcció típics d'aplicacions complexes:

- **Contenidor d'Inversió de Control** : configuració de components d'aplicació i gestió del cicle de vida d'objectes Java
- **Entorn de Programació Orientada a l'Aspecte** (AOP):
- **Entorn d'accés a les dades**: treballant am sistemes de gestió de bases de dades relacionals, sobre la plataforma Java usant JDBC i eines de mapeig d'objectes relacionals aportant solucions a reptes tècnics que són reusables en una multitud d'entorns basats en Java.
- **Entorn de gestió de transaccions**: harmonització de diferents APIs de gestió de transaccions i orquestració de transaccions configurades per objectes Java.
- **Entorn Model-Vista-Controlador**: basat en HTTP i Servlet proveint moltes eines per la millora i la personalització.
- **Entorn d'accés remot**: importació i exportació d'objectes java a la manera de l'RPC configurable, sobre xarxes informàtiques que suporten protocols basats en HTTP, com ara RMI, CORBA, i Serveis Webs (SOAP).
- **Entorn d'autenticació i personalització**: orquestració configurativa d'autenticació i processos d'autorització que suporten molts estàndards de la indústria, protocols, eines i pràctiques via el subprojecte *Acegi security framework*.
- **Entorn de missatgeria**: registre configurable d'objectes que reben missatges per la consumpció transparent de missatges des de cues de missatges via JMS, millora d'enviament de missatges sobre APIs JMS estàndards.
- **Entorn de testeig**: suporta classes per la creació d'unitats de testeig i proves d'integració.

5.5 Exemples d'implementacions

5.5.1 Arquitectura no distribuïda

Es pot considerar una aplicació no distribuïda quan la capa client i la capa EIS estan en màquines separades, però la capa intermèdia i tots els components que la formen es troben desplegats a la mateixa màquina, de fet, al mateix servidor d'aplicacions.

Aquesta és l'arquitectura més simple. Els components web i els components de negoci s'executen a la mateixa màquina virtual i els components de negoci no són EJB, només es necessita un contenidor web. Amb tot, hi ha una clara separació entre els components de negoci i els components de presentació.

1. **Arquitectura no distribuïda sense fer servir EJB**

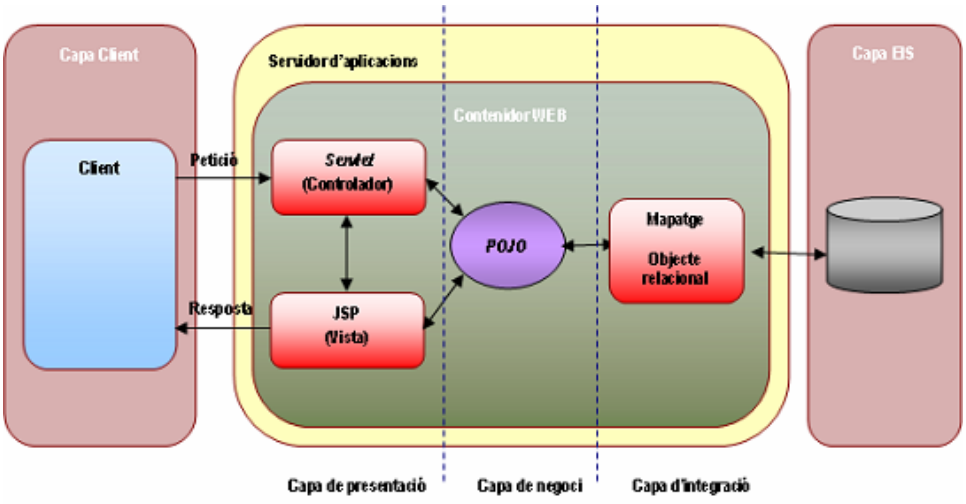


Figura 12. Aplicació Web sense fer servir EJB per la capa de negoci

En aquesta arquitectura tenim:

- Capa de presentació amb *Servlets* i JSP seguint un patró de disseny MVC.
- Accés local de la capa de presentació a la capa de negoci.
- Capa de negoci implementada amb POJO.
- Capa d'integració normalment desenvolupada amb algun *framework* de mapeig objecte/relacional o bé directament amb classes Java seguint el patró DAO (*Data Access Object*).

2. **Arquitectura no distribuïda fent servir EJB**

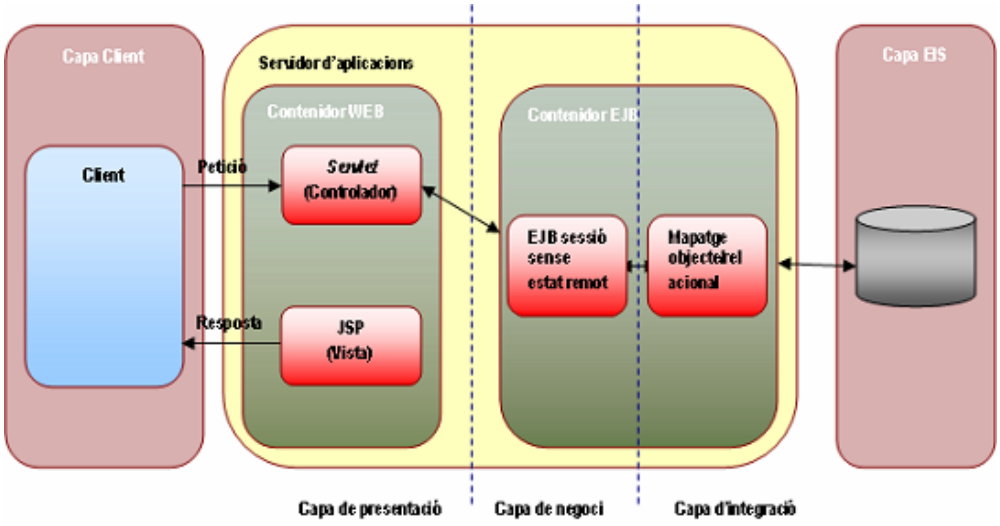


Figura 13. Aplicació Web amb EJB per la capa de negoci.

Aquesta arquitectura és similar a l'anterior, però ara els components de negoci són EJB que s'accedeixen amb interfícies locals des dels components de presentació en lloc de ser POJO. Els components de negoci i els components web s'executen a la mateixa màquina virtual.

En aquesta arquitectura tenim:

- Capa de presentació amb *Servlets* i JSP seguint un patró de disseny MVC.
- Accés local de la capa de presentació a la capa de negoci.
- Façana d'accés a la capa de negoci amb EJB de sessió sense estat locals.
- Capa d'integració normalment desenvolupada sense fer servir EJB d'entitat sinó amb opcions més simples com, per exemple, algun *framework* de mapeig objecte/relacional.

5.5.2 Arquitectura distribuïda

Tenim dues opcions: fer servir EJB per la capa de negoci o bé serveis web.

3. Arquitectura distribuïda amb EJB

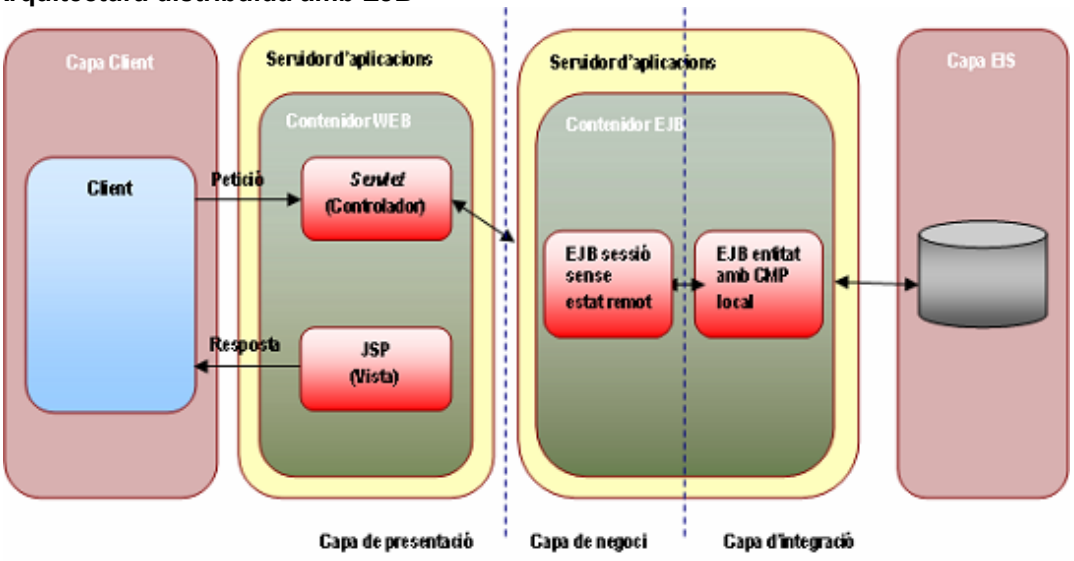


Figura 14. Aplicació Web amb EJB per la capa de negoci, esp: 2.X

Arquitectura força més complexa que permet als clients fer crides remotes als mètodes de negoci. La comunicació entre la capa de presentació i la capa de negoci es distribuïda i, fins i tot, la comunicació entre diferents components de la capa de negoci pot ser distribuïda.

Evolució a EJB 3.0: Enrere queden les tasques farragoses, ja és fàcil escriure EJBs. Simplement cal entendre bé les anotacions. No hi ha més interfície Home, interfície Remota i fitxer ejb-jar.xml. Ja només cal la interfície de negoci i un bean que implementa aquesta interfícies.

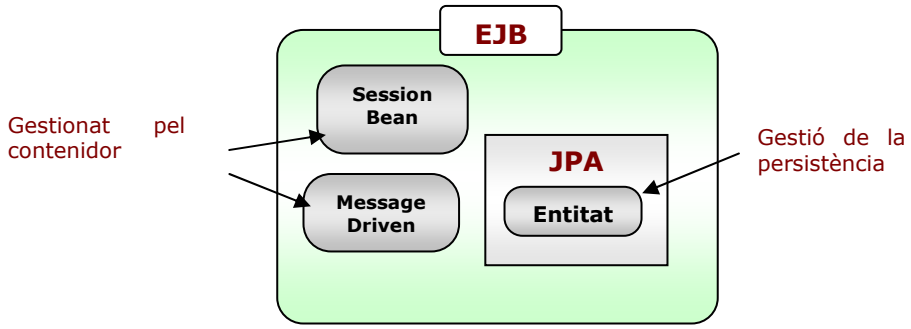


Figura 15. Aplicació EJB 3.0.

En aquesta arquitectura, que és la que proposa Sun als *Blueprints*, tenim:

- Capa de presentació amb *Servlets* i JSP seguint un patró de disseny MVC.
- Accés remot de la capa de presentació a la capa de negoci mitjançant RMI.
- Façana d'accés a la capa de negoci amb EJB de sessió sense estat remots.
- Capa d'integració JPA que reemplacen als CMP d'entitat.

En aquesta arquitectura tenim components de negoci distribuïts.

En molts casos aquesta arquitectura és més robusta i escalable, però també és força més complexa i pot arribar a proporcionar un rendiment pitjor.

4. **Arquitectura distribuïda amb serveis web**

Arquitectura molt similar a l'anterior, però que permet a la capa de negoci interactuar amb clients que no suportin la tecnologia J2EE mitjançant serveis web.

Un servei web (també conegut com *Web Service* en anglès) és una col·lecció de protocols i estàndards que serveix per intercanviar dades entre aplicacions. Diferents aplicacions de programari desenvolupades en llenguatges de programació diferents i executades sobre qualsevol plataforma poden utilitzar els serveis web per l'intercanviï de dades en una xarxa com Internet. Aquesta gran interoperatibilitat s'aconsegueix gràcies a l'adopció d'estàndards oberts. Les organitzacions OASIS i W3C són les responsables de l'arquitectura i reglamentació dels serveis web. Per garantir la interoperatibilitat entre les diferents implementacions existeix un organisme, el WS-I, que és l'encarregat d'especificar de forma exhaustiva tots els aspectes d'aquests estàndards.

5.6 Resum

Molts programadors han utilitzat els contenidors lleugers com Spring amb gestió a la persistència amb bastiments ORM com Hibernate o iBatis per què els EJB eren considerats massa farragosos i complicats de programar, i per tant, només s'utiltzaven en servidors d'aplicacions distribuïdes Java EE completes.

Aquest sentiment poc a poc està canviant donat que moltes d'aquestes característiques de persistència han estat incorporades dins de la API JPA, i projectes com Hibernate i TopLink ara formen part de la implementació d'aquesta API.

Altrament la utilització d'Spring està molt popularitzat i els seus grans defensors són a la vegada grans detractors dels EJB, com a quadre resum podríem dir que:

- Utilitzem EJB si:
  - Preferim anotacions a XML.
  - Busquem una solució estretament lligada que pren decisions per nosaltres per defecte i minimitza la configuració.
  - Tenim una aplicació amb molta dependència d'estats.
  - Volem un estàndard global i una especificació.
  - Preferim JSF i d'altres bastiments com Oracle ADF, JBoss Seam.

- Utilitzem Spring si:
  - Volem que la nostra aplicació sigui portable a qualsevol plataforma que no suporti EJB 3.0.
  - Volem integrar iBatis, Quartz o –Acegi.
  - Volem les avantatges de la AOP.
  - La nostra aplicació necessita molta configuració.
  - Volem flexibilitat.

5.7 Comparativa EJB 3.0 i Spring

Persistència

Característica	Spring	EJB 3.0
Persistència utilitzant un ORM	√	√
Implementació	Hibernate, JPA, JDO, TopLink,	JPA (inclou Hibernate, Kodo i
Suport JDBC	√	--
Mapeig	XML, Annotations	Annotations, XML
Estàndard	√ (If using JPA)	√

Gestió de les transaccions

Característica	Spring	EJB 3.0
Transaccions declaratives	√	√
Transaccions Programades	√	√
Demarcació	AOP	Metodologia amb Session Beans
Tipologia de suport	JDBC, Hibernate, JTA	JTA
Suport per les transaccions distribuïdes	√ (amb JTA)	√
Configuració	XML	Per defecte transaccional, altrament per XML
Estàndard	√ (With JTA)	√

Manipulació d'estats

Característica	Spring	EJB 3.0
Constructor	Prototype (instance) beans	Beans de sessió sense estat
Gestió de la instància	Singleton, prototype, request, session, global	Instance per lookup
Gestió del cicle de vida	√ (Inicialització/Destrucció)	√ (Creació/Destrucció, Activació/Passivació)
Coincidència transaccional	--	√ (SessionSynchronization interface)
Standard	N/A	√

Suport remot

Característica	Spring	EJB 3.0
Protocols suportats	RMI, Web Services, Hessian, Burlap, HTTP	RMI, Web Services
Objectes remots	√ (Qualsevol POJO)	√ (només EJB)
Propagació transaccional	--	√
Propagació de la seguretat	--	√

Font: Comparative Analysis of EJB3 and Spring Framework", Janis Graudins, Larissa Zaitseva, Document original publicat en :http://ecet.ecs.ru.acad.bg/cst06/Docs/cp/SIII/IIIA.18.pdf

Els resultats mostren que Spring Framework es preferible per utilitzar-se en companyies petites, amb productes Open Source. És un bastiment molt simple, convenient i flexible però, molt poderós. Es recomana el seu ús d'Spring en els casos on un contenidor d'aplicacions pessat no sigui necessari.

EJB 3.0, per contra, pot ser utilitzat per empreses on es vulgui una arquitectura bassada en EJB amb un compromía a llarg termini i esperant que les noves versions siguin compatibles amb les anteriors. La integració d'EJB amb el servidor d'aplicacions ofereix oportunitats d'escalat i optimitzacions en les aplicacions desenvolupades.

5.8 Altres bastiments

HiveMind, (http://hivemind.apache.org/)

HiveMind és un entorn de serveis i configuracions aplicant-hi principis d'Inversió de Control (IoC). La utilització de *HiveMind* garantitza l'ús de certs principis de disseny, millora la encapsulació i la modularització, així com és més senzill testejar l'aplicació o reutilitzar parts comuns en altres projectes.

- **Serveis:** en HiveMind els serveis s'encapsulen utilitzant POJOs de fàcil accés i combinació. Aquests serveis defineixen una interfície d'implementació en Java. HiveMind gestiona i es responsabilitza del cicle de vida dels serveis i els permet col·laborar uns amb els altres a través de la injecció de dependències.

- **Configuració**: HiveMind ens permet proporcionar una configuració complexa pels nostres serveis de dades en un format definit per l'usuari. HiveMind integrarà les contribucions d'aquestes dades en múltiples mòduls i els convertirà en els nostres objectes de dades.

Excalibur, (http://excalibur.apache.org/)

Excalibur és un altre projecte de codi obert desenvolupat per la Fundació Apache, proporciona un marc de treball al costat del servidor a mode de contenidor lleuger en Java anomenat *Fortress* aplicant el principi d'Inversió de Control.

Fortress sap manegar components que han estat construïts utilitzant un cicle de vida rígid del bastiment *Avalon*, que com a projecte de treball per desenvolupar un *Server Framework* va finalitzar l'any 2002 i es va dividir en quatre subprojectes més: *Excalibur*, *Loom*, *Metro* i *Castle*.

6 Frameworks de persistència

Generalment en les aplicacions es fa necessari guardar i recuperar la informació en un mecanisme d'emmagatzemament persistent, una base de dades relacional per exemple. Els objectes persistents són aquells que requereixen emmagatzematge persistent. Aquest capítol es basa sobre el disseny orientat a objectes d'un bastiment per a objectes persistents que han de guardar-se en un magatzem persistent.

6.1 Mecanismes d'emmagatzemament i els objectes persistents

- **Bases de dades orientades a objectes.** Si amb una d'elles s'emmagatzemen i es recuperen objectes, no es necessiten més serveis de persistència de tercers.

- **Bases de dades relacionals.** Donat el seu predomini, usualment necessitem la seva utilització que les orientades a objectes. En aquest context surten diferents problemes per la seva desigualtat en les representacions de dades orientades a registres i les orientades a objectés.

- **Altres**. A més de bases de dades relacionals de dades, a vegades convé guardar els objectes en altres mecanismes d'emmagatzemament: arxius plans, bases de dades jeràrquiques, etc.

6.2 La solució: un bastiment de persistència

Un bastiment o estructura de persistència és un conjunt reutilitzable, i generalment expansible de classes que ofereixen serveis als objectes persistents. Normalment, un bastiment de persistència ha de traduir els objectes a registres i guardar-los en un a base de dades; després tradueix els registres a objectes quan els recupera de la base de dades.

La gestió de dades en la programació orientada a objectes es fa a través de la manipulació d'objectes , que quasi sempre són valors no escalables. Considerant l'exemple d'una entrada d'una llibreta d'adreces, que representa una persona amb zero o més adreces o zero o més telèfons, en orientació a objectes això es representaria com un objecte "persona", de la qual penjarien les seves dades personals, les adreces i els telèfons, aquestes dues darreres dades, també objectes.

Les bases de dades relacionals, només poden emmagatzemar valors escalars, com cadenes de caràcters o enters i organitzar-los en taules. El programador ha de convertir els valors representats en forma d'objectes en valors simples o agrupats per l'emmagatzematge a la base de dades i implementar el procés invers. Alternativament podria treballar només amb valors escalars, perdent els avantatges de la programació orientada a objectes. Per poder gaudir de la potència de la Programació a Objectes i estalviar els processos de conversió d'objecte a registre de la base de dades i el seu invers, s'ha creat l'automatització d'aquest procés, mitjançant el mapeig d'objecte relacional.

6.3 Requeriments del bastiment

Desitgem disposar amb un esquema de persistència que ofereixi serveis als objectes persistents. En particular l'esquema hauria d'oferir les següents funcions:

- Emmagatzemar i recuperar objectes en un mecanisme d'emmagatzemament persistent.
- Transaccions del tipus *commit* i *rollback*.

El disseny haurà de suportar les següents qualitats:

- Poder estendre's per suportar qualsevol mecanisme d'emmagatzemament.
- Requerir un mínim de modificació del codi actual
- Facilitat d'ús.
- Ser molt transparent.

6.4 Implementació

Les bases de dades relacionals són les més típiques entre les bases de dades, utilitzen taules per organitzar les dades. Aquestes taules estan associades a través de restriccions declaratives, en lloc d'apuntadors o enllaços, per tant, les mateixes dades que, en programació, poden ser emmagatzemades en un simple valor d'objecte, en una base de dades relacional necessiten ser emmagatzemades en diferents taules.

Una implementació d'objecte relacional necessita poder triar quines taules ha d'usar per poder generar l'*SQL* necessari. La diferència d'aquest tractament de dades entre la solució de l'aplicació orientada a l'objecte, en Java, i el com s'emmagatzemen les dades en un sistema relacional de bases de dades, com Oracle o DB2, implica un complex conjunt de reptes a resoldre, amb l'objectiu de satisfer tasques com la millora del rendiment, l'escalabilitat lineal o la gestió de les operacions CRUD (*create*, *read*, *update* i *delete*).

Els objectius reals de fer servir una eina ORM, és el de guanyar temps, simplificar el desenvolupament (és a dir, l'eina ORM absorbeix la part complexa que tenia el desenvolupador), incrementar el rendiment i l'escalabilitat.

6.5 Exemples de bastiments

Donada la característica d'aquest projecte ens centrarem en bastiments de codi obert per java.

6.5.1 Hibernate

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

La funcionalitat bàsica d'Hibernate és la del mapeig de classes Java en taules de Base de Dades i tipus de dades Java sobre tipus de dades d'SQL. Hibernate també proveeix un llenguatge de *query* (HQL) i facilitats per la recuperació de dades. Hibernate genera les crides SQL i delega al desenvolupador la gestió manual del resultat de la *query* i la conversió a objectes. Gràcies a això una aplicació pot ser portable a la majoria de bases de dades SQL, amb mínima càrrega addicional. El requeriment necessari perquè una aplicació basada en Hibernate pugui usar una base dades SQL és que existeixi la peça de software encarregada de la traducció d'HQL al dialecte SQL del SGBD sobre el que ha de córrer l'aplicació.

Hibernate aporta persistència per POJOs; l'únic requeriment estricte per una classe persistent és el d'oferir un constructor públic i sense arguments. Hibernate pot ser usat tant en aplicacions *standalone* com en aplicacions JEE que usen Servlets o EJB *session beans*.

6.5.2 OJB

ObjectRelationalBridge (OJB) http://db.apache.org/ojb/, és una solució pel mapeig objecte relacional que fa transparent la persistència pels objectes Java sobre una base de dades relacional. OJB ha estat dissenyat per a un gran ventall d'aplicacions, des de sistemes integrats fins a aplicacions riques multicapes basades en l'arquitectura J2EE. OJB s'integra d'una manera elegant dins de servidors d'aplicacions J2EE, suporta JNDI de cerca en bases de dades. OJB proporciona un suport especial pels EJB BMP.

6.5.3 Castor

Castor, http://castor.exolab.org/, és una bastiment pel mapeig d'objectes relacional de codi obert. Proporciona el camí més curt entre els objectes Java, documents XML i taules relacionals. Castor proporciona un entorn de Java a XML, persistència de Java a SQL i molt més.

6.5.4 Torque

Apache Torque, http://db.apache.org/torque/, és un bastiment pel mapeig objecte relacional per aplicacions en java. Torque facilita l'accés i manipulació de les dades dins d'una base de dades relacionals utilitzant objectes java. Torque genera tots els recursos de la base de dades necessaris per la nostra aplicació i inclou un entorn d'execució per generar les classes necessàries. Torque ha estat desenvolupat dins del projecte Jakarta Turbine Framework.

6.5.5 Ibatis SQL Maps

iBATIS SQL Maps, http://ibatis.com/common/sqlmaps.html, proporciona una solució simple i flexible pel transport de dades entre els objectes java i una base de dades relacions.

El seu funcionament es basa en el mapeig de sentencies SQL que s'inclouen en fitxers XML. Això significa que requereix coneixements de SQL per part del programador. Altrament, permet la optimització de les consultes, ja sigui amb llenguatge estàndard o amb SQL propietari del motor de la base de dades utilitzat. Amb iBATIS, sempre es sap el que s'està executant en la base de dades, i tenim eines per evitar el problema de les "N + 1 consultes" i per generar consultes dinàmiques molt potents.

Quan el model de dades és molt canviant o es preexistent al desenvolupament de l'aplicació (i compartit amb altres), iBATIS és un clar exemple d'ús. També ho és quan les relacions entre les entitats del modelo són molt complicades, perquè amb una mica de feina es poden aconseguir que el número de consultes que es passen a la base de dades no sigui excessiu.

iBATIS ha guanyat pes en la comunitat, fins arribar a incorporar-se al projecte Apache.

6.5.6 TJDO

TriActive JDO (TJDO), http://tjdo.sourceforge.net/, és una implementació de codi obert a partir de les especificacions del JDO de Sun (JSR 12), dissenyat per donar suport transparent a la persistència utilitzant JDBC.

6.5.7 JD0 2.0 (JSR 243)

La API JDO, consta només d'unes poques interfícies, és un dels APIs més fàcils d'aprendre de tota la tecnologia existent actualment per la persistència d'objectes estandarditzada. Hi ha moltes implementacions de JDO entre les que podem escollir. La implementació de referència de JDO l'ha posat Sun a la nostra disposició.

JDO permet treballar amb objectes normals de Java (POJOs *plain old Java objects*) en lloc de APIs propietàries. JDO proporciona una capa d'abstracció entre el codi de la aplicació i el motor de persistència.

6.5.8 EJB 3.0 (JSR 220)

Gradualment un consens en la indústria va anar fent emergir que la virtut primera dels EJBs, habilitar integritat transaccional sobre aplicacions distribuïdes, tenia l'ús limitat per la majoria de les aplicacions empresarials. La funcionalitat oferta per entorns més senzills com *Spring* i *Hibernate* era més útil per aplicacions d'empresa. D'acord amb això, l'especificació EJB 3.0 (JSR 220) era un punt de partida radical respecte els seus predecessors, seguint aquest nou paradigma. Aquí s'aprecia una clara influència respecte Spring, el seu ús de POJOs i el seu suport a la injecció de dependències per simplificar la configuració i la integració de sistemes heterogenis. Gavin King, el creador d'Hibernate, va participar en el procés d'EJB 3.0 i se'l considera una veu lliure en defensa d'aquesta tecnologia. Moltes funcionalitats d'Hibernate van ser incorporades a la *Java Persistence API*, **JPA**, el substitut dels *Entity Beans* a EJB 3.0. L'especificació dels EJB 3.0 usa fortament les anotacions (meta dades), una funcionalitat afegida al llenguatge Java en la seva versió 5.0, per habilitar un estil de programació molt menys farragós.

D'acord amb això, en termes pràctics, EJB 3.0 és bastant una API completament nova, semblant-se molt poc a les especificacions prèvies d'EJB.

6.6 Resum

Un nombre de sistemes ORM han estat creats al llarg dels anys però el seu efecte al mercat sembla prou dispar. **NeXT** n'ha fet un dels considerats millors, *l'Enterprise Objectes Framework* (EOF), però ha fallat a l'hora de tenir un impacte perllongat al mercat, principalment per estar massa lligat al *toolkit* de NeXT, *OpenStep*. Posteriorment va ser integrat a **WebObjects**, també de NeXT i primer servidor d'aplicacions orientat a objectes. A partir de la compra de NeXT per part d'**Apple**, el 1997, EOF aporta la darrera tecnologia del lloc web de comerç electrònic de la companyia, els serveis **.Mac** i l'***iTunes Music Store***. **Apache Cayenne** va aparèixer a l'estiu de 2002 i es proposa complir l'estàndard JPA.

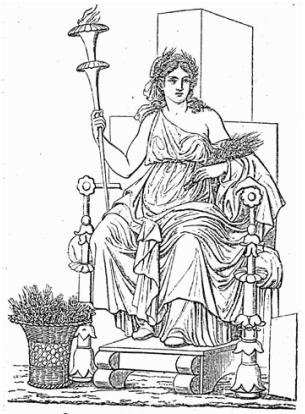
Una utilitat alternativa està sent presa amb tecnologies com **RDF**, **SPARL** o el concepte del "*triplestore*". RDF és una serialització del concepte subjecte-predicat-objecte, **RDF/XML** és una representació del mateix

en XML, SPARQL és un llenguatge de *queries* tipus SQL i *triplestore* és una descripció general de qualsevol base de dades que treballa amb un tercer component.

Més recentment, un sistema semblant ha començat a evolucionar dintre del món Java, conegut com a JDO. A diferència d'EOF, JDO és una especificació, de manera que hi ha diferents implementacions dels diversos fabricants. L'*Enterprise Java Bean 3.0* (EJB3) també cobreix la mateixa àrea. Hi ha hagut conflicte d'estàndard entre ambdues especificacions. JDO disposa de bastants implementacions comercials, mentre EJB 3.0 encara està en fase de desenvolupament. Tanmateix, la JCP ha anunciat nous estàndards per unir els que hi ha en conflicte i aconseguir que l'estàndard futur funcioni arreu de les arquitectures basades en Java. L'altre exemple a esmentar és **Hibernate**, l'entorn ORM més usat al món Java i que ha inspirat l'especificació EJB3.

A l'entorn web, **Ruby on Rails** juga un rol central i és gestionat per l'eina de *wrapping ActiveRecord. DBIx::Class* juga un rol similar per l'entorn basat en **Perl** i que s'anomena ***Catalayst***. En **Python** també hi ha múltiples ORMs, com ara **SQLObject**, **SQLAlchemy**, **Dejavu**,**PyDAO**, **Strom**, etc. Molts entorns python suporten un o més ORMs.

Disseny i aplicació d'una solució



Demèter *o* ***Demetra*** *daessa de la mitologia grega de l'agricultura.*

"Don't talk with strangers".

"Llei de Demèter", (LoD). Lieberherr, Holland i Riel 1988.
'LoD' i la seva analogia de mètodes i classes amb les joguines:
Només pots jugar amb tú mateix, amb els juguets que t'han donat o amb els juguets que has construït.

En els capítols anteriors hem cobert totes les funcionalitats arquitectòniques que formen la plataforma J2EE per donar suport a una aplicació empresarial. Hem parlat de les diferents solucions que trobem dins d'aquest mercat canviant. Els bastiments estudiats cobreixen un conjunt llarg de tecnologies que moltes aplicacions utilitzen o bé com a part de la seva solució arquitectònica, o com a solució global. En resum, l'estudi ens ha servit per definir l'escenari tecnològic d'aquesta arquitectura segons els paradigma multicapa de presentació, negoci i persistència.

És hora de posar en comú aquests coneixements dins d'un exemple complet d'aplicació. Aquesta solució vindrà proporcionada pel bastiment d'Spring, que cobrirà totes les capes de l'arquitectura, des del model conversacional de la capa web, fins la solució de mapatge objecte-relacional en la persistència, sense oblidar les possibilitats que trobem en la capa de negoci.

Spring cobreix un ventall molt gran de possibilitats tecnològiques aplicades per donar solucions a aplicacions web, només veurem una part d'aquest bastiment en l'aplicació exemple. Hem escollit limitar l'aplicació a les tecnologies més comuns d'utilització donades les característiques d'aquest projecte però, això no vol dir que l'aplicació estigui limitada, ja que la utilització d'Spring permet anar introduint noves solucions a mida de les necessitats que vagin sortint a l'aplicació.

Per tant, hem avaluat la possibilitat que l'aplicació proporcioni la oportunitat de veure un exemple complet amb tots els arxius de configuració i el codi Java corresponent, amb la part de desplegament inclosa. D'aquesta manera es veuran les relacions existents entre els arxius de configuració i la seva funcionalitat.

En la primera part d'aquest document explicarem com l'aplicació està dissenyada i el seu diagrama de components amb els patrons utilitzats. A continuació, descriurem els requeriments que la nostra aplicació ha de reunir i les tecnologies adequades per poder-la construir. Per últim explicarem la seva implementació seguint la planificació en fases: programació de la interfície web seguint el model WebFlow, implementació de la capa de negoci utilitzant els patrons arquitectònics adequats i construcció del mapeig objecte-relació de la capa de persistència.

Per tant, la intenció és presentar una cobertura prou important dins de les parts de l'arquitectura i les raons per les quals hem fet cada elecció, en aquest capítol no pretenem cobrir totes les parts tècniques de la implementació, però sí, les més importants i les que fan d'aquest projecte el caràcter diferenciador necessari. Tot i això, es subministra un arxiu de desplegament amb tota l'aplicació per poder-la estudiar.

7 Anàlisi d'una solució arquitectònica

En aquest apartat es donarà una visió general de l'arquitectura adequada que hom pretén desenvolupar com estàndard dins d'un projecte empresarial sota la plataforma JEE.

Veurem l'esquema general que ja coneixem de l'arquitectura estudiada i quines són les parts a definir amb bastiments adequats. Aprofundirem en quins requeriments hauria de donar suport la nostra solució amb garanties, i es definirà quin serà el seu enfocament.

7.1 Objectius arquitectònics

Es pretén donar una orientació de quines parts serien les adequades per desenvolupar un entorn distribuït empresarial que permeti donar servei als següents requeriments:

- Orientar a desenvolupar una capa de presentació de qualitat seguint uns estàndards.
- Estructurar la capa de negoci.
- Proporcionar una capa de persistència de qualitat.
- Gestió de seguretat i autentificació.
- Gestió d'autoritzacions.
- Monitorització.
- Accés a serveis externs.
- Utilitzar eines existents del mercat.

7.2 Objectius empresarials

El projecte també ha de donar servei a qualssevol perspectiva empresarial orientada a obtenir productes de qualitat en desenvolupament de projectes JEE. Per tant, es considera que el model arquitectònic ha de cobrir:

- Reduir costos en la creació i manteniment d'aplicacions.
- Minimitzar el temps de desenvolupament.
- Eliminar tasques repetitives de programació.
- Optimitzar processos de construcció de programari.
- Permetre el desenvolupament en factories externes (*outsourcing*).
- Preparar equips especialitzats en diferents responsabilitats.
- Facilitar el control i la gestió de la configuració.
- Augmentar la qualitat del producte (cap a una certificació CMMI®, SPICE).
- Evitar interferències per canvis en la tecnologia.

Aquest model d'estructuració permetrà orientar a un equip a desenvolupar projectes de qualitat sota l'arquitectura JEE en qualsevol empresa, en el nostre cas, orientarem el model de la solució a donar suport de projectes sota una arquitectura orientada a **aplicacions**, que és la divisió lògica de grans companyies com el sector bancari i les assegurances.

Normalment les grans empreses financeres i del sector assegurador es divideixen en grans àrees de negoci que donen suport a la vegada a diferents subdivisions, aquestes darreres, en ocasions, també estan dividides en àrees més especialitzades que donen suport a una part concreta del negoci, a aquests elements els anomenem aplicacions. Una aplicació per exemple, podria ser la part de valors de l'àrea d'actiu d'un gran banc, o la part alfabètica (dades de clients), de l'àrea d'assegurances llar d'una asseguradora. Com veiem aquesta divisió es pot portar a altres exemple, com pot ser l'àrea de defensa militar d'un país, la divisió de sistemes d'informació d'una empresa automobilística, etc.

8 El disseny de l'arquitectura model

8.1 Model de l'arquitectura

L'arquitectura que es proposarà és una arquitectura d'aplicacions distribuïda empresarial basada en la plataforma *Java™ Enterprise Edition (JEE)* i es considera la utilització de bastiments d'aplicació per a cada capa de l'arquitectura, on cada bastiment superior utilitza els serveis de les capes inferiors.

Amb l'objectiu de separar clarament les diferents funcionalitats que es requereix en una aplicació, aquesta es divideix en capes seguint el model de les *blueprints* de Sun. Per tant, tenim una consideració especial en els següents aspectes:

- Aplicació del model MVC en la capa presentació.
- Estructuració de components de negoci seguint una solució coneguda.
- Capa de persistència de qualitat.
- Optimització i gestió de recursos tècnics.

8.1.1 Visió de la solució

Per poder donar una arquitectura sòlida, els objectius que cal assolir són:

- Estudi dels diferents models de desenvolupament existents i el seu encaix dins dels requeriments.
- Anàlisi de les arquitectures que el puguin suportar.
- Es desenvolupa un nou model que:
    - Permeti definir, aprofitar i niar fluxos de navegació.
    - Permeti separar la lògica de navegació de la lògica de negoci i la d'integració..
    - Definir fluxos simples.
    - Multiidioma.
    - Multicanal.

Respecte a la part de programari comercial escollit farem especial èmfasi en l'elecció d'Spring com a bastiment de negoci, EJB segueix sense tenir opció, encara que disposem de servidors d'aplicacions *open source* com *JBoss*, *Gerònimo* i *GlassFish*.

La part de presentació seguirà el Spring MVC i WebFlow amb una biblioteca de *custom tags,* tot i que Struts és líder del mercat hem pensat en seguir tota la construcció amb Spring però, en futurs pròxims podríem tenir en compte WebWork, JSF i les noves opcions Struts 2.0/WebWork 2, *Shale*, etc. **Utilitzar Spring ens deixa moltes portes obertes per si hi ha algun canvi.**

En la part de persistència hem escollit Hibernate, encara que iBatis avança amb claredat sobre tot des de que està en el projecte Apache.

Després està l'automatització... *Ant* o *Maven*. En principi i per l'experiència dels equips hem escollit Ant però, podem considerar-lo com un producte ja finalitzat.

En l'apartat de versionat, *Subversion* ja es pot considerar com el successor de CVS des de que Apache ha posat els seus repositoris en ell.

Per últim, el sistema de *logs* vindrà definit per la llibreria estàndard Log4j.

La solució de l'arquitectura presentada aportarà una estructura d'inici per a realitzar desenvolupaments empresarials en JEE, com força una divisió de l'aplicació en capes arquitectòniques, els components a desenvolupar per l'equip desenvolupador d'una aplicació estarà inclosa en alguna de les capes en funció de la seva responsabilitat.

8.1.2 Arquitectura orientada a aplicacions

Les grans companyies, i dins dels sistemes d'informació, estan formades per grans àrees de negoci dividides en diferents aplicacions. Així per exemple, poden tenir sistemes d'informació multicanal, sistemes d'informació d'infraestructura, sistemes d'informació de passiu...

El sistema financer està orientat en conjunts d'institucions, organismes i entitats que configuren el mercat on els ofertors i demandants de fluxos de diners troben un lloc comú pels seus intercanvis, sense necessitat a priori d'un coneixement previ d'ambdós.

Així trobem que la configuració d'una entitat financera estarà formada pel Passiu, l'Actiu i altres serveis bancaris (domiciliacions, xecs bancaris...). La divisió feta dins de la nostra arquitectura serà per aplicació interbancària: estalvi vista, estalvi en divisa, alfabètic de persones.

En cada aplicació es crearà un catàleg de serveis, que podrà ser utilitzat per altres aplicacions. Aquests serveis seran reutilitzables.

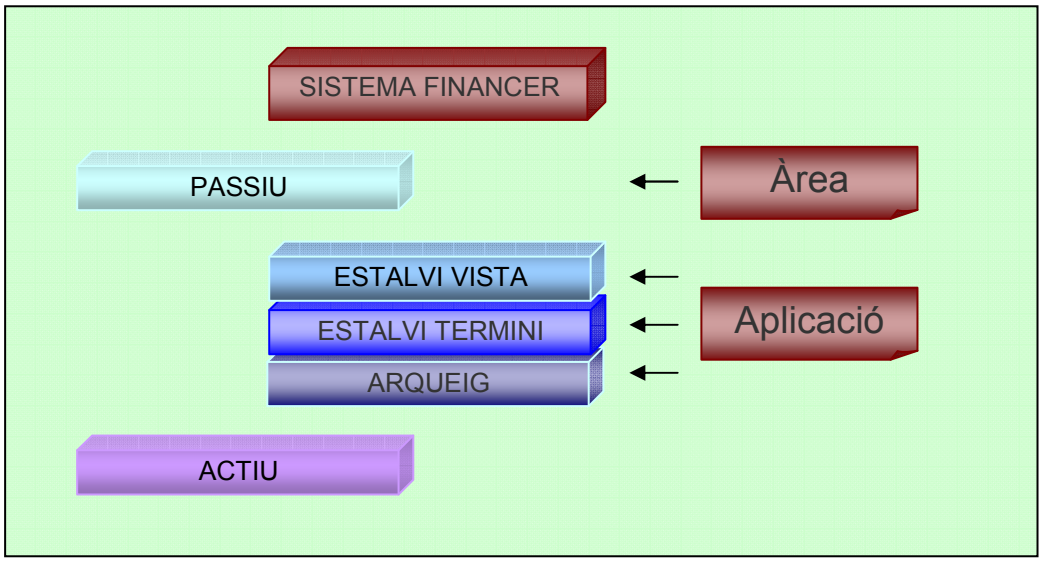


Figura 16. Model de negoci d'una entitat bancària

Dins de cada servei tindrem les operacions a fer per aquell servei concret. Per exemple, l'aplicació Estalvi a la Vista, disposa de comptes corrents i llibretes. Un servei d'aquesta aplicació serà el cas d'ús d'una llibreta o compte. Aquest cas d'ús definirà n-operacions sobre el compte/llibreta com podria ser la obtenció de saldo.

Serveis d'aplicació
Ja hem comentat que un servei d'aplicació descriu un cas d'ús concret d'una aplicació determinada, aquest servei tindrà un únic propietari (l'aplicació) però es mantindrà dins del catàleg disponible de l'arquitectura de la entitat bancària.

8.2 Disseny de la solució

8.2.1 Capa presentació

Es composa dels components o agrupacions de components que en el seu conjunt són responsables de la interacció amb els usuaris (capa client), que amb l'objectiu de separar les diferents funcionalitats requerides en una aplicació es divideix en capes pròpies del model MVC.

Implementació de la capa amb Spring MVC que aportarà:

- Gestió d'errors.
- Configuració del control de forma autoritzada.
- Interrelació entre el model i les vistes.
- Contenidor d'aplicació lleuger.

Possibilitat d'integració amb AJAX i utilització d'un llibreria pròpia de Tags. És molt important limitar als desenvolupadors la creació de pàgines sense seguir uns criteris d'usabilitat, és per aquest motiu, que abans de la construcció d'una pantalla ha de passar per un control. Per fer-ho es subministra una llibreria de *customs tags*, i mai es pot sortir de les seves directrius.

8.2.2 Capa de la lògica de negoci

La nostra proposta d'utilitzar Spring en la capa de la lògica de negoci és perquè permet la reusabilitat dels objectes de negoci i d'accés a dades, independitzant d'aquesta forma els objectes de Java dels serveis específics de JEE i d'entorns administrats.

Aquest bastiment que s'utilitza per la integració de capes proveeix solucions per diversos reptes tècnics encarats per desenvolupadors Java i organitzacions que volen crear aplicacions basades en la Plataforma Java. Donada la clara amplitud de la funcionalitat que s'hi ofereix, pot fer-se complicat distingir entre els blocs principals que composen el marc de treball. Spring Framework no està lligat exclusivament a la Java EE, encara que la seva prou aconseguida integració en aquesta àrea és una raó important per la seva elecció.

8.2.3 ORM

El bastiment per mapejar les entitats d'objectes a dades de taules relacionals escollit serà Hibernate que és menys invasiu que altres marcs de treball de mapeig O/R.

Hibernate ofereix facilitats per la recuperació i actualització de dades, control de transaccions, repositoris de connexió a bases de dades, consultes programàtiques i declaratives, i un control de relacions d'entitats declaratives.

8.2.4 Eines pel desenvolupament

Les eines de desenvolupament han de estar pensades per una banda per agilitar i automatitzar els processos de definició i desenvolupament d'aplicacions amb el model explicat, però per l'altra banda ens permetran definir processos de forma rígida sense que la espontaneïtat tingui cabuda.

- **Entorn d'integració de desenvolupament d'aplicacions de negoci:** Eclipse + *Plugin subversion* + Spring IDE 1.x *Plugin* + Hibernate *Syncronizwer Plugin*
- **Edició de JSPs:** utilitzant *WebTools* amb *code assistance* i *syntax higlight*.
- **Definició de beans** *Free Browsing* amb l'Spring IDE.
- **Edició gràfica de Webflows**: *Spring IDE WebFlow support.*

8.3 Model conceptual

En el model conceptual de la nostra aplicació mostrem cada capa segons les especificacions J2EE, tal i com es veu en la figura l'esquema que segueix és el d'una arquitectura segons el disseny multicapa estudiat en els anteriors apartats.

Des de la capa client es podrà accedir a la nostra aplicació utilitzant diferents clients (navegador tradicional). Per a homogeneïtzar l'aspecte de les aplicacions es desenvolupa una llibreria de components visuals. Aquesta llibreria a part de proporcionar l'estil de tots els components també proporciona totes les funcionalitats necessàries que necessita l'aplicació en la capa presentació, per tant evitem l'ús per part de les aplicacions de llenguatges HTML o *JavaScript*. El desenvolupador només ha d'usar els *Tags* dels components definits per un equip d'usabilitat, eficiència i auditoria. Els serveis que proporciona són l'accés dels components als literals definits per l'aplicació propietària, accés a fitxers *properties*, i l'ajuda guiada al usuari i la construcció de *URLs* emmascarades.

URLs emmascarades dins de components predefinits. Aquest concepte és habitual quan dins de la construcció de vistes ens trobem amb la necessitat de la reusabilitat de components, com per exemple, el *Tag empleat*, on per defecte es pot accedir al Cas d'Ús de consulta d'empleat de forma intrínseca en l'enllaç predefinit del component.

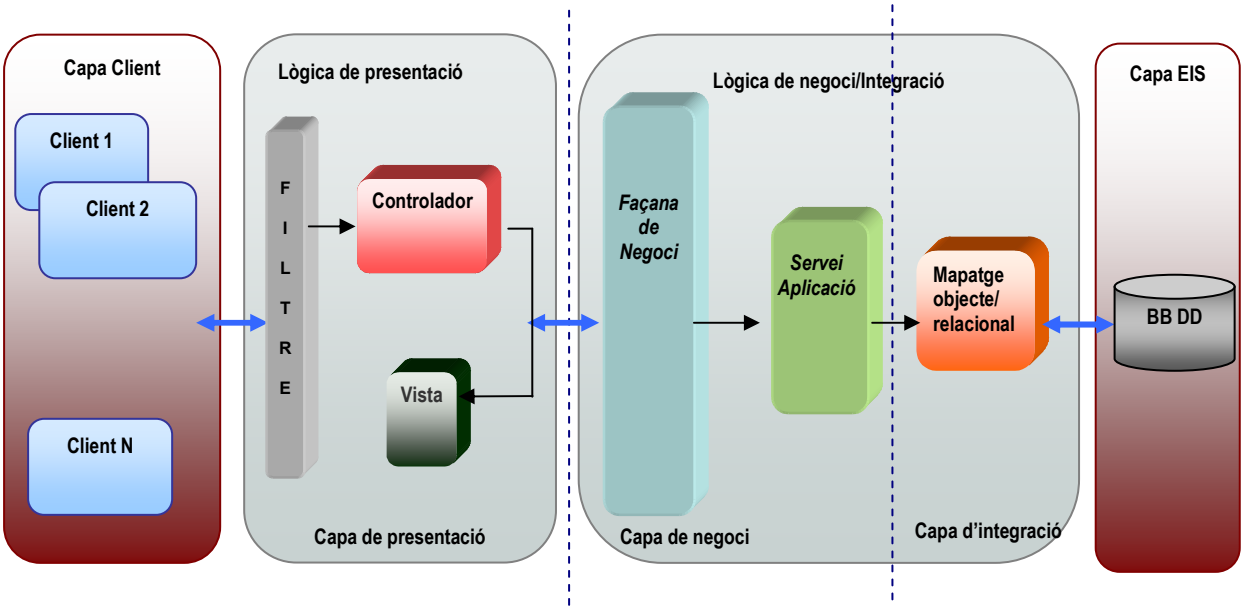


Figura 17. Arquitectura a seguir (components)

La navegació es fa utilitzant dos tipus diferents de controladors: el FlowBrowsing i el FreeBrowsing. En general el FlowBrowsing s'utilitza quan la informació que es vol mostrar necessita més d'una pantalla o hi ha navegació intel·ligent entre pantalles, és a dir, la informació de la primera pantalla es necessita per la segona. En canvi, el FreeBrowsing s'utilitza en navegacions senzilles, una petició, una pantalla.

En la resta de capes podem distingir altres components que segueixen el model arquitectònic de la especificació dels patrons de Sun. Per exemple, trobem un filtre principal que s'encarregarà de la validació d'usuaris, és el punt d'entrada a l'aplicació.

Per l'accés a la lògica de negoci ho centralitzarem amb un patró façana que serà la classe que unifiqui les interfícies i assigni la responsabilitat de col·laborar amb tot el sistema. Per poder desacoblar les seves funcions, optem per utilitzar el patró *Aplication Service*, que englobarà els casos d'ús reutilitzables d'una aplicació. Per exemple, tots els mètodes que es poden fer en un compte corrent d'un client.

Per últim, es veiem també de forma general que s'utilitzarà un marc de persistència per poder resoldre el mapatge dels objectes Java a taules DB2 de persistència.

A continuació, veiem com representaríem aquests components d'arquitectura utilitzant una implementació com podria ser *Spring* i *Hibernate*.

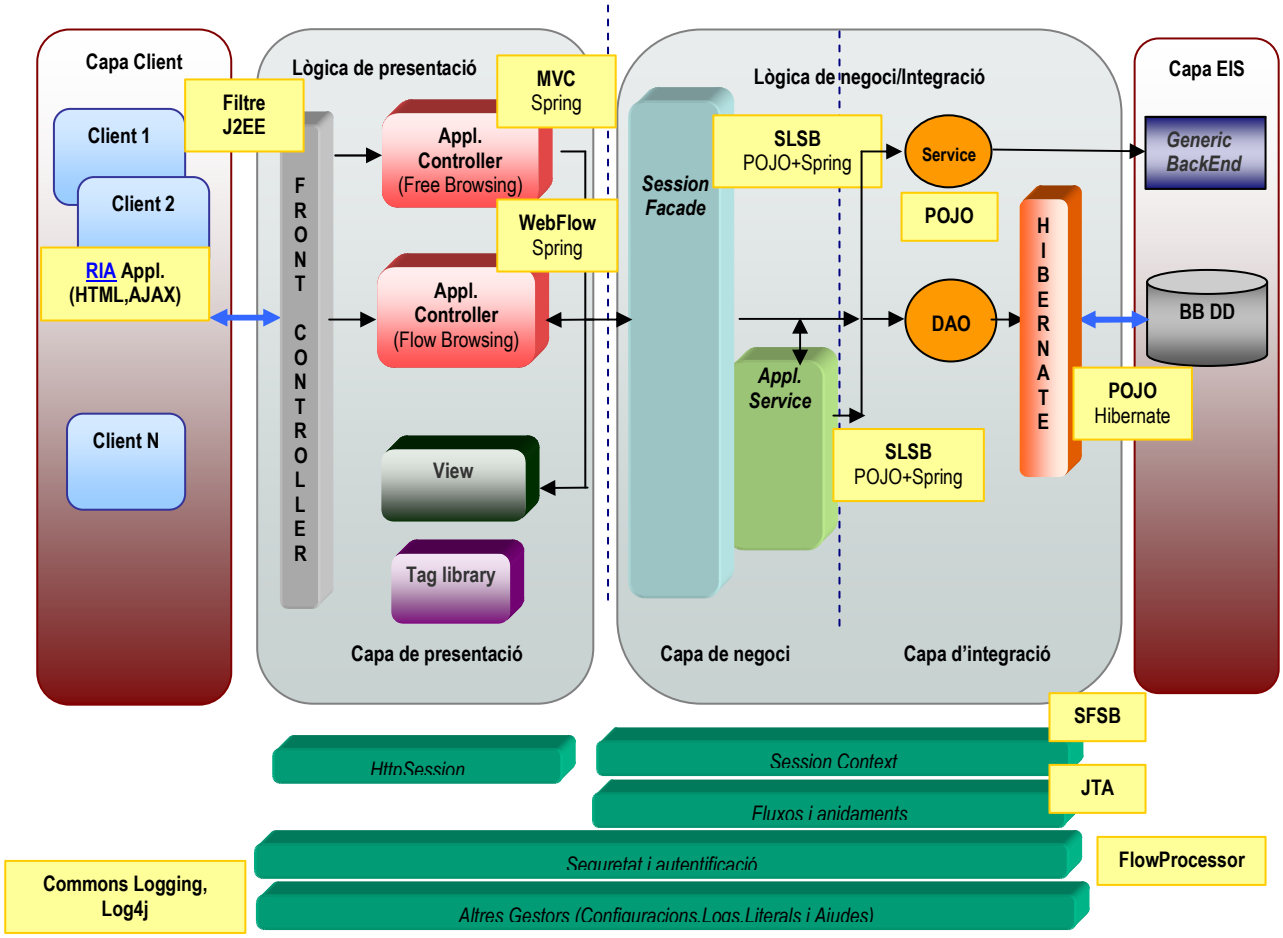


Figura 18. Arquitectura a seguir (implementació)

El funcionament de l'aplicació haurà de seguir el següent flux d'esdeveniments, quan un usuari intenta accedir a qualsevol operativa de l'entorn, i per primera vegada, es mostrarà la pantalla de login, on l'usuari haurà d'introduir una sèrie de dades per autenticar-se en el sistema. El formulari d'aquesta pantalla enviarà les dades introduïdes al servidor i desencadenarà el procés d'autentificació que es tingui configurat. Una vegada l'usuari s'hagi autenticat correctament serà redirigit al recurs sol·licitat. En els mòduls de seguretat s'haurà emmagatzemat les dades necessaris de context de l'usuari. En cas que l'usuari no introdueixi correctament les dades de login es capturarà l'excepció i es mostrarà una pàgina d'error configurada.

Una vegada l'usuari estigui en el sistema podrà accedir a les diferents opcions que l'entorn li faciliti, i seguint el model de negoci predefinit. Hi haurà un mòdul per una àrea de negoci (casos d'ús possibles), i dins d'aquests casos d'ús hi haurà d'altres corresponents a serveis d'aplicació que facilitarà les funcionalitats d'accés a la lògica de negoci.

A continuació mostrarem el disseny intern de cada un dels components més importants descrits en el model conceptual de l'arquitectura escollida. Per fer-ho utilitzarem les plantilles de patrons que ja coneixem i aplicarem alguna implementació possible utilitzant l'exemple d'algun bastiment *Spring*.

Properament s'explicarà una aplicació *demo* que segueixi el model d'arquitectura plantejat i que sigui exemple d'un desenvolupament ben construït.

8.3.1 Front Controller

Front Controller: punt únic d'entrada de peticions per aplicació.

Funcionalitat:

- Credencials: valida si l'usuari disposa de permisos necessaris per resoldre la petició.
- Gestió del *login* als diferents entorns necessaris i informa de la seva funcionalitat al usuari.
- Administració i monitorització.

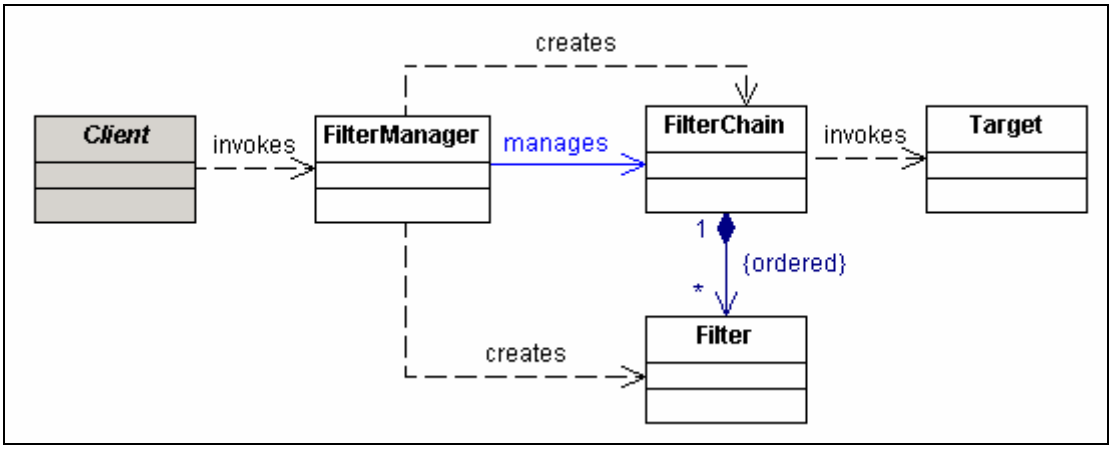
Abstracció:

- Patró *Intercepting Filter* de J2EE (*Core J2EE Patterns*).
- Patró *Front Controller* de J2EE (*Core J2EE Patterns*).

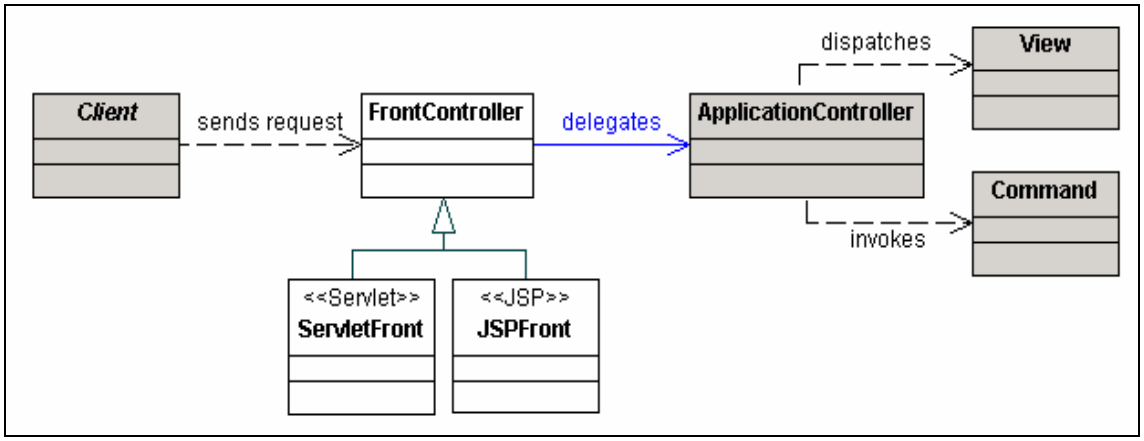
Implementació:

- *Spring* MVC.
- *Struts* MVC.

Disseny del patró:



http://www.corej2eepatterns.com/Patterns2ndEd/InterceptingFilter.htm



http://www.corej2eepatterns.com/Patterns2ndEd/FrontController.htm

8.3.2 *Application* Controller

Application Controller: patró J2EE que permet centralitzar la recuperació de peticions i la invocació de la lògica de negoci.

Funcionalitat:

- *DataBinding* i validació: vincular les dades rebudes de la petició a l'objecte que utilitzarem per invocar la lògica de negoci i validació d'aquests.
- Invocació de la lògica de negoci.
- Selecció de la vista a presentar en funció del resultat obtingut de la crida a la lògica de negoci..

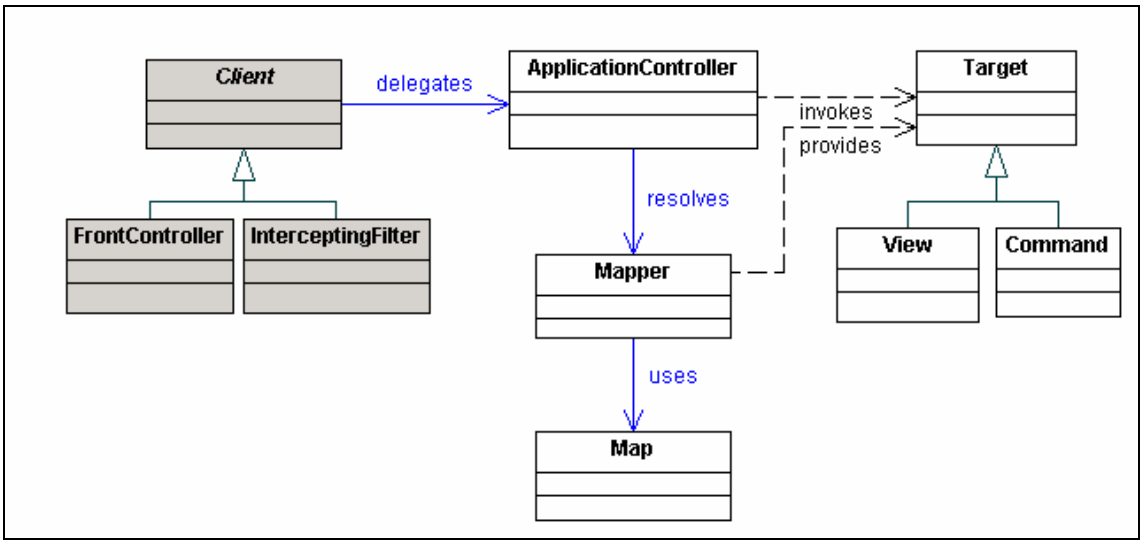
Abstracció:

- Patró *Aplication Controller* de J2EE (*Core J2EE Patterns*).

Implementació:

- Navegació lliure: *Spring* MVC.
- Flux de navegació: *Spring WebFlow*.

Disseny del patró:



http://www.corej2eepatterns.com/Patterns2ndEd/ApplicationController.htm

8.3.3 Façana de negoci

Session Facade: patró J2EE que permet desacoblar al client (*Aplication Controller, Business Delegate*) dels objectes de negoci, serveis, fluxos reutilitzables, etc.

Funcionalitat:

- La funció principal d'aquest component és agregar la lògica de negoci tal i com ho necessita l'aplicació client. Per lo general, no és un element reusable entre aplicacions ja que particularitza un determinat cas d'ús.

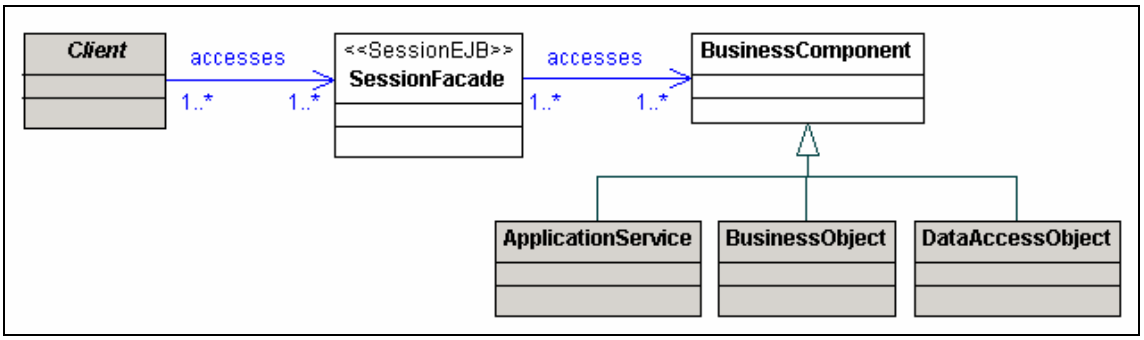
Abstracció:

- Patró *Aplication Controller* de J2EE (*Core J2EE Patterns*).

Implementació:

- EJB de sessió.
- POJO en EJB utilitzant Spring.
- Configuració de fluxos utilitzant Spring (*flowprocessor*)

Disseny del patró:



http://www.corej2eepatterns.com/Patterns2ndEd/SessionFacade.htm

8.3.4 Servei d'aplicació

Aplication Service: patró J2EE que representa un flux de serveis, DAOs i *Business Objects* que és reutilitzable en múltiples parts de la mateixa aplicació o entre altres aplicacions.

Funcionalitat:

- Aquest component s'ha d'utilitzar en aquells casos en els que es detecta una lògica d'integració reusable. Per exemple, quan dos o més serveis tenen una agrupació que es sol repetir amb freqüència en el codi. Per tant, permet reutilitzar casos d'ús i evitar "copy&paste" entre *Session Facades*.

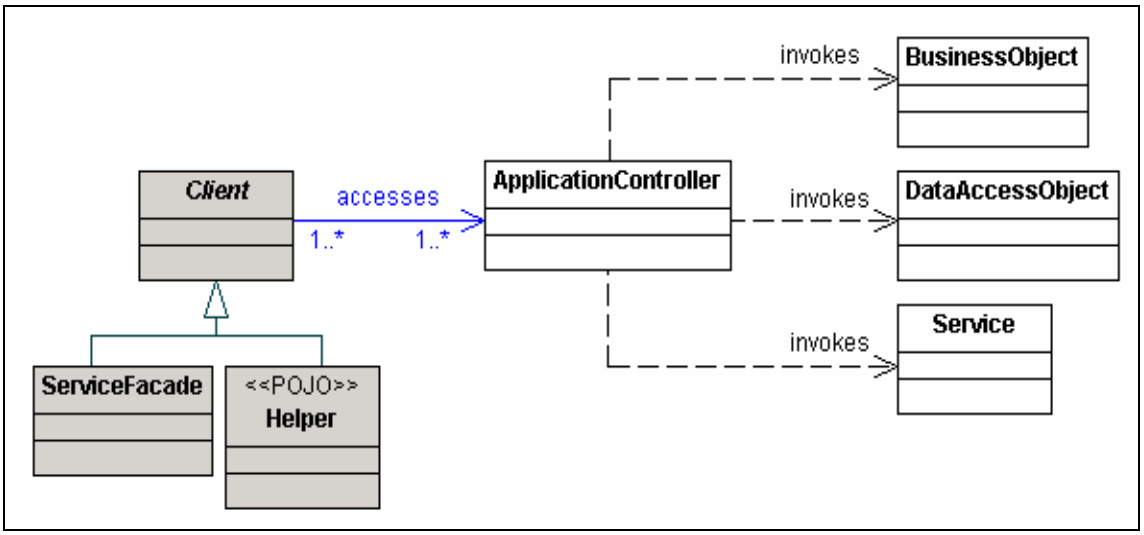
Abstracció:

- Patró *Aplication Service* de J2EE (*Core J2EE Patterns*).

Implementació:

- EJB de sessió.
- POJO en EJB utilitzant Spring.
- Configuració de fluxos utilitzant Spring (flowprocessor)

Disseny del patró:



http://www.corej2eepatterns.com/Patterns2ndEd/ApplicationService.htm

8.3.5 Servei

Service: servei que ens ofereix qualsevol recurs extern..

Funcionalitat:

- El client s'abstrau dels "*internals* necessaris" per invocar al servei en el recurs intern.

Implementació:

- POJO (*Plain Old Java Object)*

Data Access Object (DAO): patró J2EE que ens permet abstraurà i encapsular l'accés a un data source (data model) tant en lectura com per escriptura.

Funcionalitat:

- Abstreure i encapsular l'accés a Base de Dades.

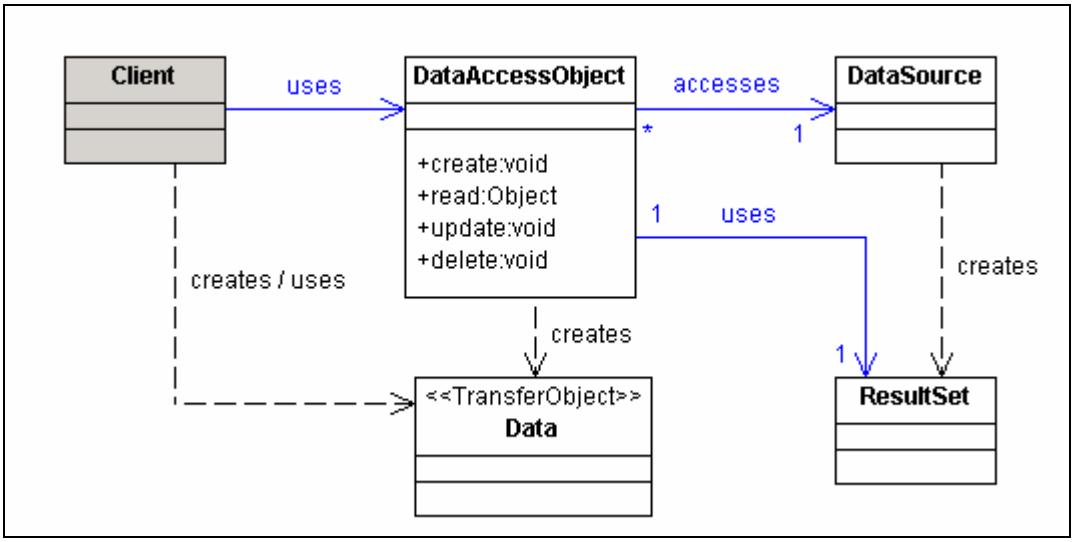
Abstracció:

- Patró *DAO* de J2EE (*Core J2EE Patterns*).

Implementació:

- POJO (Plain Old Java Object).
- Spring ORM.
- Hibernate, JDO, iBAtis.

Disseny del patró:



http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm

9 Aplicació exemple

9.1 L'elecció tecnològica

L'aplicació demo està bassada en una aplicació web, llavors, la primera elecció que cal fer és quin servidor d'aplicacions seleccionem. Tanmateix molts servidors J2EE proporcionen implementacions compatibles, els passos de desplegament i configuracions són molts semblants entre ells. La implementació original es podria fer utilitzant un contenidor d'EJBs però, ara per ara, no ho considerem necessari per aquest exemple. Al 2001 EJBs eren els únics que proporcionaven la solució de la problemàtica de les transaccions en aplicacions J2EE. Les transaccions del bastiment d'Spring ara per ara proporcionen una alternativa atractiva, és per això que no utilitzarem EJBs.

El servidor **Apache Tomcat** ens proporcionarà l'entorn adequat pel nostre exemple, és molt conegut i familiar dins de la comunitat de desenvolupadors d'aplicacions web.

La següent elecció és el servidor de base de dades. Per testejar l'aplicació hem utilitzat **HSQL** i **MySQL**. Tot i això l'exemple podria treballar en altres servidors SQL, només caldria modificar els arxius de configuració. En el nostre exemple, l'aplicació es subministra amb uns *scripts* per provar-la amb **MySQL** amb la versió mínima **4.1**.

Com a eina per compilar, crear directoris de treball, desplegar, arrencar i parar l'aplicació utilitzem el **Apache Ant**.

Per la escriptura de codi i l'organització de les llibreries utilitzem l'eina **Eclipse 3.X**, i els corresponents **pluggins**. Com a norma de programació recomanem seguir el *Manual de Estilo de Programación*: http://www.ahristov.com/tutoriales/Blog/Estilo_de_Programacion.html.

9.2 Requeriments de l'aplicació

L'aplicació que pretenem implementar ha de reflectir un model de construcció seguint les pautes estudiades dels bastiments i les bones maneres de programació de l'estudi plantejat. D'aquesta manera pretenem reflectir els coneixements adquirits en un exemple d'implementació.

El requeriment de la nostra aplicació és la construcció d'una prestació *online* de bloqueig de targetes per a donar suport a una empresa proveïdora de serveis *Call Center* d'una entitat financera. Donat que és una aplicació pilot és molt important que l'aplicació pugui canviar l'aspecte de forma senzilla.
L'empresa proveïdora està situada en qualsevol província o país que millor oferta faci a la entitat bancària, per aquesta raó, cal que hi hagi un suport a la internacionalització de l'aplicació. No és un requeriment immediat però, cal tenir-lo en compte.

Per últim, donat que aquesta aplicació és utilitzada dins de la xarxa segura de la entitat bancària és molt important, que amb petits canvis la mateixa aplicació pugui funcionar a oficines, si més no, la part de negoci.

Objectius que ha de cobrir la nostra aplicació:

1 **Navegació:**
- Permetre transaccions simples i complexes
- Aprofitament de fluxos.
- Navegació lliure
- Suport a multiidioma.
- Multicanal (diferents accessos des de la capa client).
- Seguretat, autentificació, autorització i monitorització

2 **Presentació:**
- Homogènia.
- Facilitat en canviar la imatge i l'estil.

- Mecanismes de diàleg.

3 **Lògica d'integració.**
- Centralització d'una façana.
- Gestió global transaccional.

4 **Lògica de negoci.**
- Aprofitament de la lògica de negoci.
- Catàleg de serveis: Application Service.
- Gestió del context.

Nota: Els següents requeriments de negoci no descriuen tots els requeriments que una aplicació empresarial de bloqueig de targetes necessitaria. En qualsevol cas ens servirà per poder ensenyar unes pautes arquitectòniques de construcció utilitzant el bastiment Spring.

9.3 Clients

L'aplicació serà utilitzada per dos tipus d'usuaris:

1 Usuaris de la empresa proveïdora.
2 Usuaris de la entitat bancària (empleat o terminalista).

L'aplicació per tant, defineix un sistema de rols d'utilització que per una banda donaran servei a les empreses externes amb, és clar, una limitació tècnica (autentificació) i una altra administrativa (LOPD), i als usuaris de la entitat que tindran una limitació administrativa (LOPD) segons el seu nivell de seguretat.

Els usuaris utilitzaran un navegador prim per l'accés a la capa presentació.

Nota: En l'aplicació exemple no s'implementarà el mecanisme d'usuaris, autentificació i autorització.

9.4 Planificació de l'aplicació

L'aplicació de suport al Call Center determina una solució *online* de suport als clients d'una entitat bancària fora dels canals operatius humans tradicionals (oficina) en situacions problemàtiques (incidències, robatoris). Aquestes situacions acostumen a produir-se quan el client vol rebre atenció de forma immediata i per diferents raons com mobilitat o calendaris festius no es pot desplaçar a la oficina.

Cal determinar una planificació per donar suport al *Call Center* tan bon punt sigui possible, substituint l'actual sistema pel dissenyat. El cas d'ús seleccionat en primer lloc és el de bloqueig de targetes d'un client per robatori o pèrdua.

Planificació:
- Fase 1: Construcció del nucli de la interfície d'Internet.
- Fase 2: Construcció de la part de negoci: clients i targetes.
- Fase 3: Suport a la persistència.
- Fase 4: La interfície d'administració i control d'usuaris.
- Fase 5: Més sofisticació a la interfície web, oferint altres serveis i noves prestacions amb suport a la internacionalització.

La nostra aplicació exemple estendrà només les fase 1, 2 i 3. El nucli complet de la interfície web i la part de negoci i persistència. En els propers capítols detallem les parts de cada fase i la seva relació amb l'estudi del projecte.

9.5 Disseny del cas d'ús bloqueig de targeta

Per millorar la comprensió dels requeriments, descriurem a continuació els casos d'ús, és a dir, les descripcions narratives dels processos del domini. Els casos d'ús descriuen la seqüència d'esdeveniments d'un actor (agent extern) al interactuar amb el sistema per completar un procés.

En el nostre sistema trobem dos actors diferents. Per una banda trobem l'usuari de l'aplicació situat al *Call Center*, és adir, l'usuari del subsistema del *FrontOffice*. Accedeix al sistema (des de web) per operar per aquest canal (bloqueig de targetes). L'altre actor és el terminalista o empleat bancari. És l'usuari del *FrontOffice* però, ara situat en una oficina bancària i utilitzar un terminal financer adaptat a la operativa i no un navegador prim.

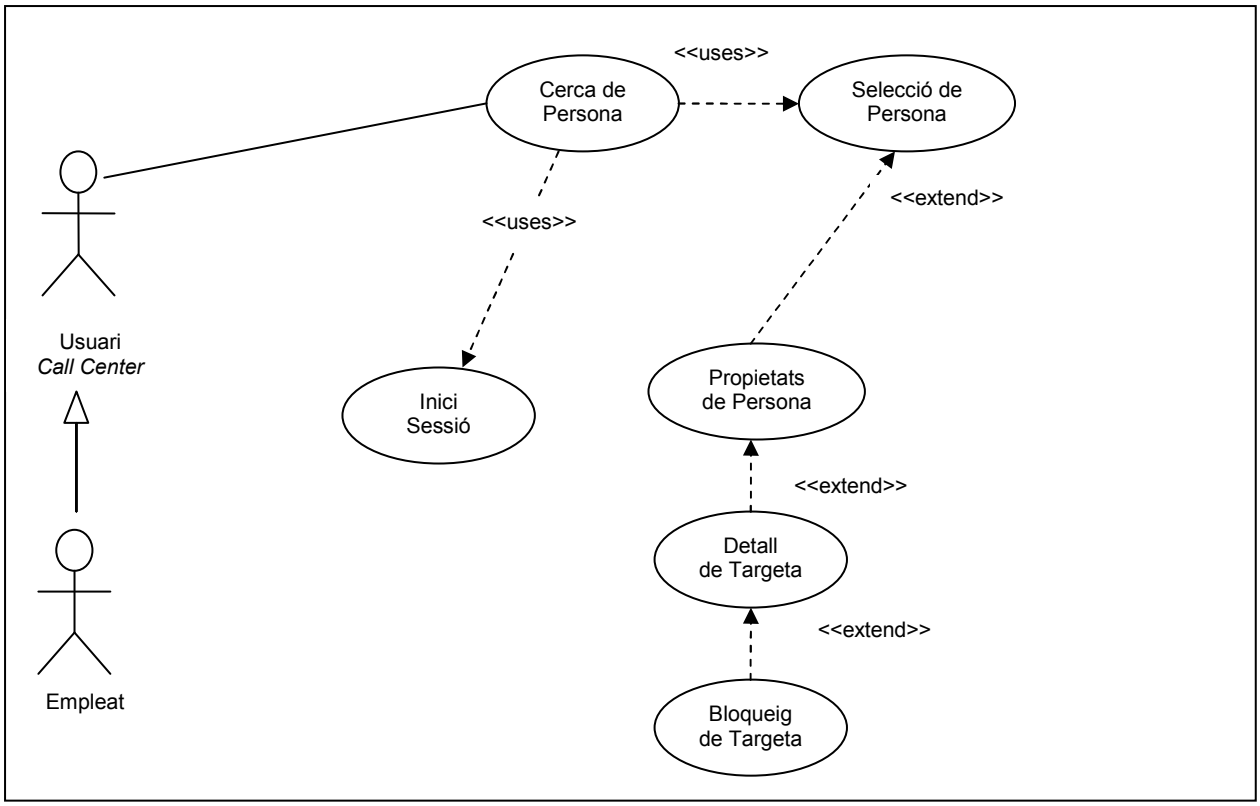


Figura 19. Disseny de casos d'ús de l'aplicació exemple

9.6 Les capes de l'aplicació en Spring

L'aplicació que pretenem implementar serà una aplicació web per a donar suport a una empresa proveïdora de serveis *Call Center* d'una entitat financera. Estarà formada en diferents capes per limitar la dependència entre elles i fer-les lo més senzilles possibles.

És una pràctica molt acceptada i recomanada pel desenvolupament d'aplicacions J2EE. La següent figura mostra les capes de l'aplicació i els arxius de configuració d'Spring i no-Spring associats a cada capa. Tal i com podem veure la utilització del bastiment Spring ens permet codificar molt menys codi en les classes de negoci. Molt codi que tradicionalment era implementat per configurar l'aplicació, pel control de flux de les pantalles o per la crida a les diferents classes de negoci ara és innecessari per què esdevé especificada dins del propi bastiment. A més, podem utilitzar conceptes de AOP per la seguretat per exemple.

Altrament, la dificultat afegida és la utilització de molts més arxius de configuració XML. Tal vegada, és un dels primers problemes quan comencem a treballar amb Spring però, permet disposar d'aplicacions ben construïdes, fàcils de mantenir, ben configurades i robustes.

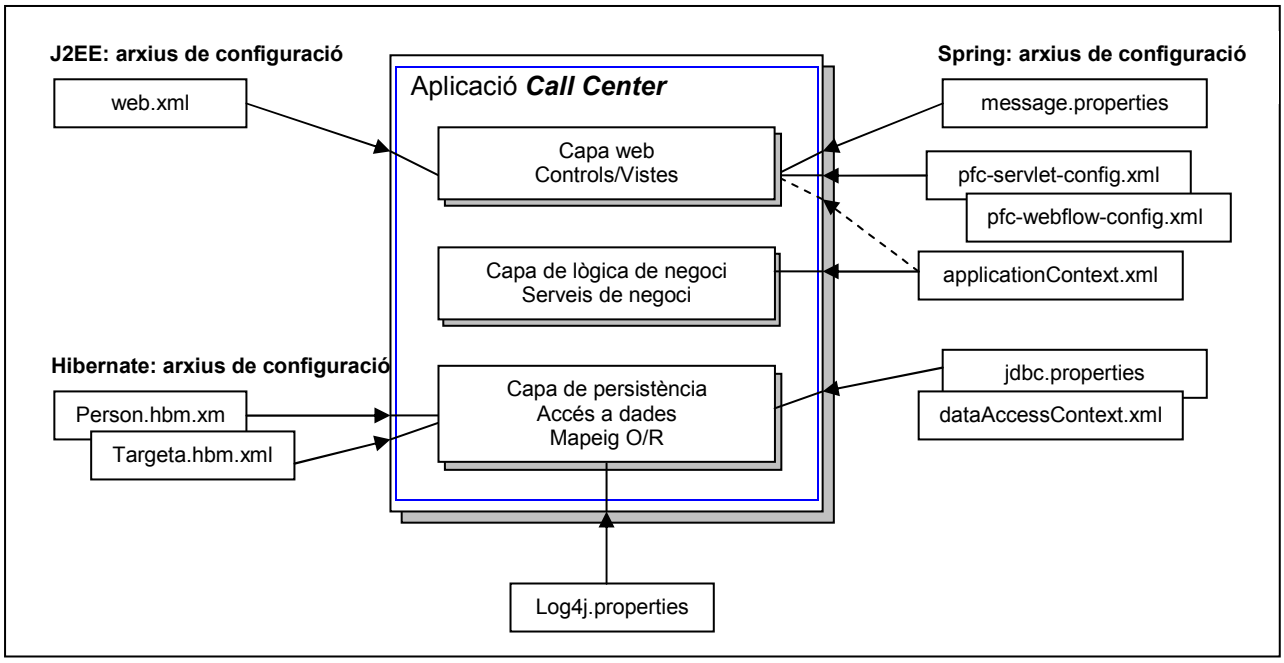


Figura 20. Les diferents capes de l'aplicació exemple

9.7 Interfície Web

L'usuari comença la interacció a partir de la pantalla d'inici o benvinguda. Aquesta pantalla introdueix al sistema la prestació de bloqueig de targetes. L'usuari pot llavors accedir a la cerca de persones a partir d'uns paràmetres de selecció. Si la persona no és client de l'entitat bancària es mostra un missatge d'error i permet tornar a executar la cerca.

Quan cerquem un client pot ser que en trobem més d'un donada la coincidència de nom i cognoms per exemple, llavors, arribem a una pantalla on es permet seleccionar a una persona de la llista. La selecció d'una persona permet mostrar les dades detallades de les persones i les targetes contractades. Una vegada seleccionada la targeta permetrà accedir a la pantalla de detall de targeta i acció de bloqueig. Acceptat el bloqueig tornem a veure ara el detall de la targeta amb el nou estat i la confirmació de que ara la targeta està bloquejada.

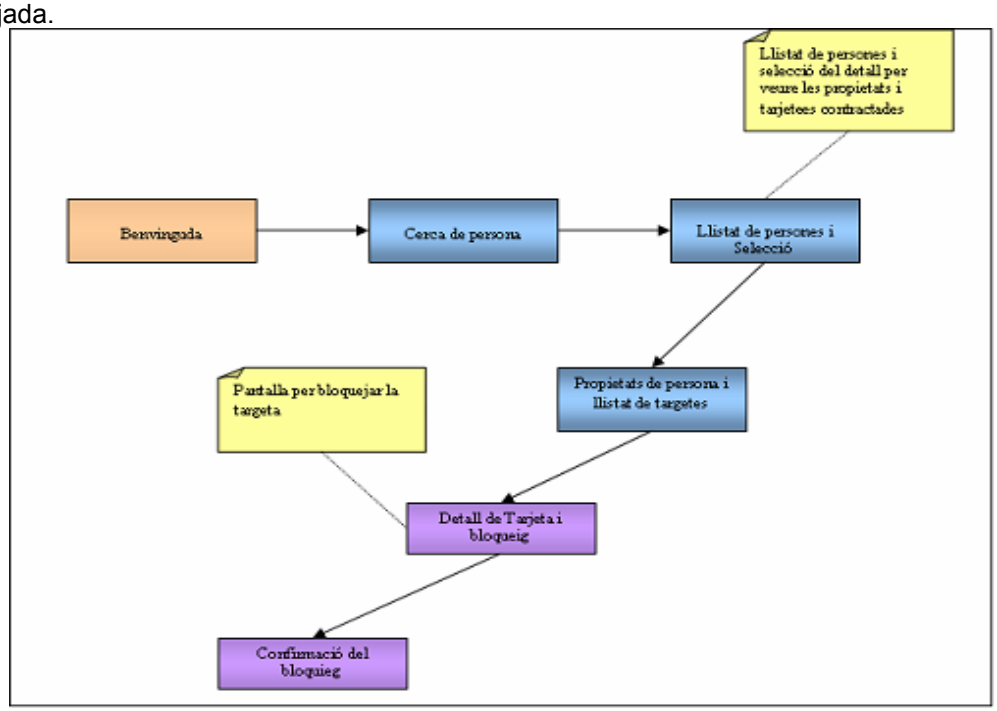


Figura 21. Disseny de la interfície gràfica de l'aplicació exemple

9.8 Fase 1: Construcció de la capa presentació

En el capítol de disseny analitzàvem els motius de la utilització d'un o altre patró per la construcció de la capa de presentació i varem arribar a la conclusió de la utilització del patró ***Application Controller*** per a la validació i verificació dels formularis, vincular les dades de petició i recepció, invocar a la lògica de negoci i per la selecció de la vista.

Fins aquí no hi ha res de nou. En la construcció d'un patró com el descrit s'utilitza normalment l'especificació d'un S*ervlet*. Actualment, l'especificació (http://www.jcp.org/aboutJava/comunityprocess/final/jsr154) proporciona desenvolupaments estàndards per processar peticions dins d'una aplicació web.

Moltes aplicacions utilitzen casos d'ús que no tenen una naturalitat definida per les possibilitats que ofereix l'especificació *Servlet*. Aquests casos d'ús necessiten més d'una pàgina sense necessitar la longevitat de la sessió. L'usuari potser que necessiti diferents casos d'ús per sessió de navegador o potser necessiti executar el mateix cas d'ús una i una altra vegada.

En aquest escenari les dades associades a un cas d'ús necessiten ser eliminades en cada execució nova, on habitualment la sessió no ho permet. És a dir, l'especificació *Servlet* s'oblida del concepte de la oportunitat de conversació per donar suport a l'execució de casos d'ús amb múltiples pàgines.

Per les navegacions complexes com la descrita utilitzarem la solució *Spring Web Flow* proporcionada pel bastiment. L'aplicació de *Call Center* treballa seguint aquest model conversacional.

9.8.1 Spring Web Flow

En primer lloc direm que per *l'Spring Web Flow* els fluxos de conversació són el seu principal propòsit. **El component més important és el *flow* (conversació)**. Les conversacions es poden executar en paral·lel sense interferir unes amb les altres, i quan la conversació està finalitzada, tots els recursos carregats automàticament es destrueixen.

En segon lloc, la definició de *l'Spring Web Flow* està fora de forma deliberada de l'especificació Servlet. Els components principals dins de *l'Spring Web Flow* són els flows, els estats, les accions i les vistes (*flow*, *action*, *state*, *views*). Com es pot veure, no hi ha una definició web específica sobre el significat d'un *flow*, el *flow* es pot descriure en sí mateix com un conjunt de pàgines web, un conjunt de pantalles, o qualsevol comunicació entre sistemes. En cap moment dins de *l'Spring Web Flow* es veurà el *HttpServletRequest* o el *HttpServletResponse*.

En tercer lloc, es pot trobar l'*Spring Web Flow* com una forma bastant simple de treballar, amb una terminologia senzilla i amb una corba d'aprenentatge mitja. També disposa d'eines que facilita la construcció de flows com l'editor Spring IDEWebFlow per l'Eclipse http://www.springide.org/project).

9.8.2 Implementació Bloqueig de targeta

La prestació implementada "bloqueig de targeta" està sota la particularitat d'un flow seguint el model web descrit en l'apartat 2.5. Per la seva definició hem creat un flow amb diferents estats, on cada estat, representa unes vegades el moment d'interacció del sistema amb l'usuari i una pantalla de presentació (view-state) o el moment d'interacció del sistema amb la capa de negoci (action-state).

Donada la característica del model de conversació definit, el flow principal es divideix en sub-flows o sub-conversacions internes a la conversació principal, en el nostre cas, la representació del detall d'una persona i el detall d'una targeta el representarem com sub-flows del flow principal "Bloqueig de targeta". Un sub-flow té les mateixes característiques que qualsevol flow.

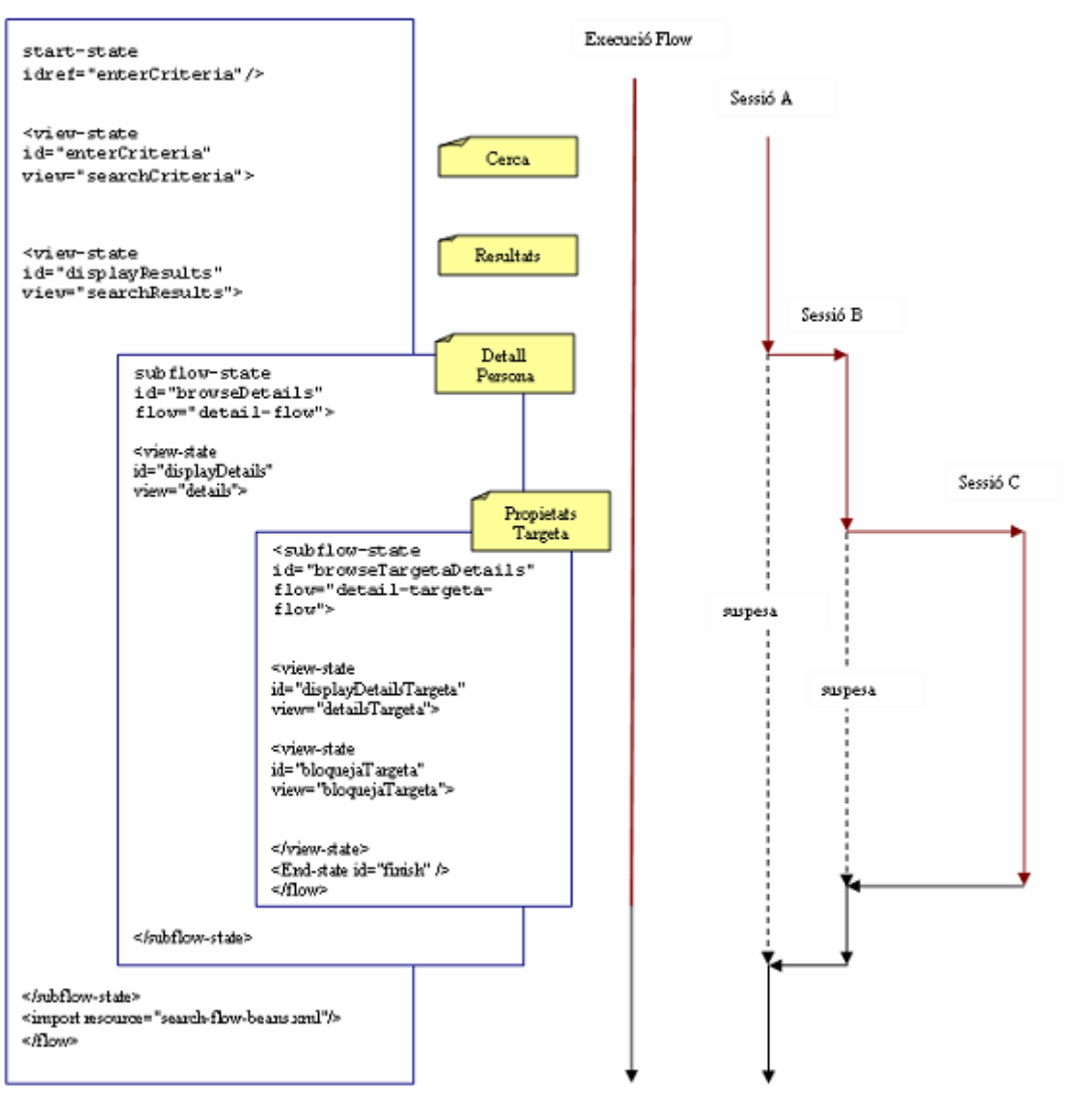


Figura 22. Diagrama conversacional de l'aplicació exemple

9.8.3 Configurar el web.xml

L'aplicació quan s'inicia utilitza la informació per configurar que trobem en l'arxiu estàndard J2EE **web.xml**. En aquest arxiu especifiquem el context local per la utilització dels continguts webs que utilitzar la nostra aplicació.

La classe que utilitzem per carregar el context de l'aplicació és la classe ***ContextLoaderListener***. Utilitzem la classe ***log4j*** per als missatges de log.

Una altra important configuració és la declaració del ***Servlet*** utilitzat. Utilitzem un classe estàndard d'Spring, el *DispatcherServlet*. El codi de l'arxiu es proporciona en una entrega a part.

Amb aquests punts es defineix el contenidor d'Spring, i ens assegurem que no cal cridar cap altra objecte de l'aplicació requerit pel context d'Spring de forma específica. La injecció de dependència treballa en tot el codi de l'aplicació.

9.8.4 Els controladors

Els arxius **pfc-servlet-config.xml** i **pfc-webflow-config.xml**, disposen de les configuracions dels servlets que necessitem, ja que com utilitzem WebFlow s'especifica la utilització del ***FlowController***, un controlador de la especificació MVC d'Spring.

L'arxiu pfc-webflow-config.xml, configura el bean **flowRegistry**, un mena de diccionari de definició de flows que poden executar-se en l'aplicació. El codi dels arxius es proporcionen en una entrega a part.

9.8.5 La tecnologia per les vistes

Hem escollit utilitzar la tecnologia JSP per implementar les pantalles de l'aplicació donada la seva gran popularitat. És molt senzill implementar les vistes utilitzant solucions o plantilles disponibles en el mercat com **Velocity** o **FreeMarker**. Particularment, i donat que un dels objectius del projecte és donar pautes per al disseny i construcció d'aplicacions empresarials el nostre consell és no utilitzar productes que embrutin el codi font de les pàgines web i sempre que puguem utilitzar JSP, JSLT i Custom Tags per limitar el "*libre albedrio*" de la programació.

No veurem totes les pantalles de l'aplicació de forma individual però, en mostrarem l'exemple d'una d'elles per veure totes els seus components i característiques. La pantalla escollida és detall.jsp (DetallClientTargeta). Aquesta pantalla figura 3.5, té una descomposició molt simple. Hi ha un contenidor a dalt i a baix del cos principal. Dins del cos de la pàgina trobem tres parts diferenciades: les dades del client, el llistat de targetes contractades dins d'una taula i el guió de l'operador. Com a components més importants podem distingir etiquetes simples d'informació, taula de dades, el custom tag Pan, imatges, un link per accedir al detall de la targeta i un botó per tornar enrera.

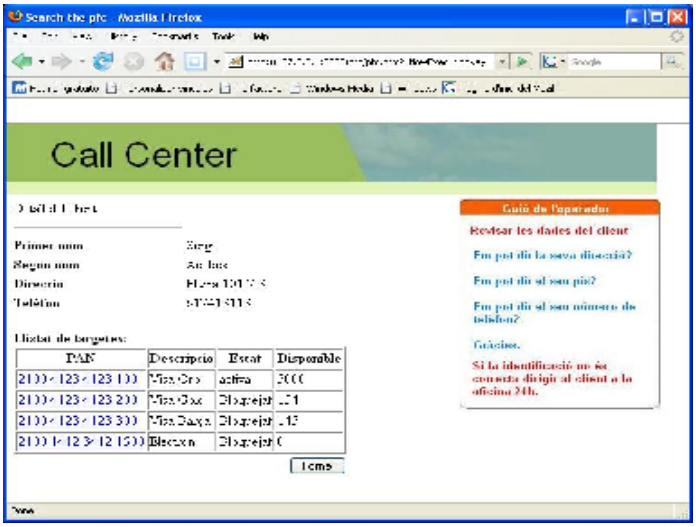


Figura 23. JSP de detall del client i les targetes contractades

El codi que genera aquesta pàgina només utilitza codi *JSP*, *JSLT*, *Customs Tags* i HTML (caldria reconduir a *Custom Tags*). El codi de l'arxiu es proporciona en una entrega a part.

9.8.6 Llibreria de Custom Tags

Una de les idees que hem posat més èmfasi durant el procés d'aquest projecte és la de limitar als dissenyadors de pàgines WEB la utilització de codi que no passes un procés d'usabilitat i auditoria. Per aquesta raó definirem al projecte la utilització de *custom tags* per a definir els components visuals de les nostres pantalles.

JSP permet definir TAGs propis de l'aplicació, si bé és una tasca molt tediosa, permet simplificar el codi de les JSP i com hem dit anteriorment permetre de forma homogènia la creació de components visuals. L'objectiu final de la utilització de llibreries pròpies és la disposar de components reutilitzables i definits per un equip d'usabilitat amb un aspecte comú per a tots els aplicatius que permeti en un futur modificar els formats sense esforç important.

Per definir un custom tag necessitem crear un XML amb una extensió **.TLD** (tag library descriptor). Per indicar a les pàgines JSP que utilitzarem tags definits en una TLD específica s'utilitza la directiva ***taglib***, en la que s'indica un prefix que es posarà davant cada vegada que invoquem tags de la nostra llibreria. En l'aplicació exemple és "pfc".

```
<taglib>
	<tlibversion>1.0</tlibversion>
	<jspversion>1.0</jspversion>
	<shortname>Exemples de Custom Tags</shortname>
	<!--  definicio del TAG PAN -->
	<tag>
		<name>pan</name>
		<tagclass>main.java.org.springframework.webflow.samples.pfc.tags.panTAG</tagclass>
		<bodycontent>JSP</bodycontent>
		<info>Pan una targeta</info>
	</tag>
</taglib>
```

El tag escollit és el tag PAN, que dibuixa l'estructura d'una targeta en la pàgina JSP. És a dir, en la base de dades s'emmagatzema un identificador de "plàstic" o targeta. Aquest identificador segons el país, l'entitat bancària i la tipologia de la targeta, defineix un identificador de 16 posicions separats 4 a 4. Ex: 4018 3456 1234 0018. Per cridar al component PAN només cal definir la llibreria de tags pròpia i cridar-lo en el JSP.

```
<%@ include file="includeTop.jsp" %>
<%@ taglib uri="/WEB-INF/demo.tld" prefix="pfc" %>

…
                        <td><b>Pan</B></td>
                        <td><pfc:pan>${targeta.id}</pfc:pan></td>
                <tr>
                        <td><b>Descripcio</b></td>
                        <td>${targeta.descripcio}</td>
                </tr>
```

El component tindrà una classe java <<***public class*** *panTAG* ***extends*** *BodyTagSuppor*>>, que s'encarregarà dels càlculs del PAN, on la imatge final del component serà:

PAN
2100 4123 4123 100

9.9 Fase 2: Construcció de la part de negoci

La capa de lògica de negoci és potser la part més important de l'aplicació. La lògica de l'aplicació normalment resideix aquí, i és important abordar qualsevol dependència amb la resta de capes i les tecnologies a escollir per la implementació, per poder seguir el criteri de reusabilitat i manteniment. En un disseny orientat a objectes la capa de la lògica d'aplicacions es divideix en altres menys denses: objectes del domini i serveis.

Domini

- Gestió de persones: Conjunt de funcions que controlen les peticions de l'usuari.
  - ***PersonManager:*** permet enregistrar en el sistema una persona i gestionar-la.
  - ***TargetaManager:*** permet enregistrar en el sistema una targeta i gestionar-la.

Serveis

- Administració: Conjunt de funcions per gestionar l'aplicació.
  - iAdmin: Gestió i administració de l'aplicació.
- Seguretat (servei d'autoritzacions i autentificacions de la pròpia aplicació)

9.9.1 Diagrama de classes

En la nostra aplicació ens centrarem en el domini de Persona i de Targeta. A continuació anem a definir el diagrama de classes de la nostra aplicació. Hem de tenir en compte només **Person** i **Targeta** però, descrivim el diagrama complet per podar-nos en context.

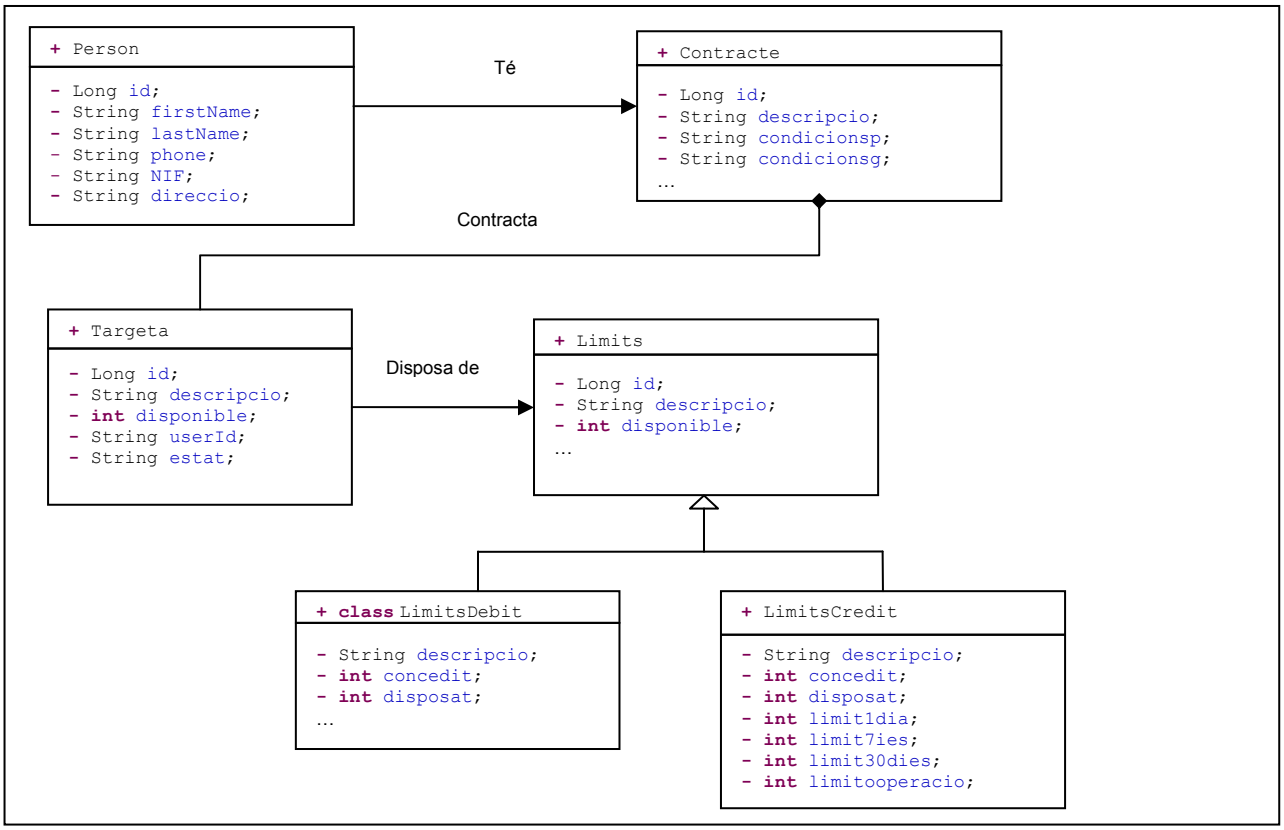


Figura 24. Diagrama de classes de l'aplicació exemple

Tota la lògica de negoci està en dos classes, i la capa Web accedirà a aquesta lògica a través de dos interfícies. Comentarem la primera, que és **PersonaManager**, i proporciona informació d'un client de la entitat bancària, on els mètodes primaris més importants són:

```
public List<Person> search(SearchCriteria query);
public Person getPerson(Long Id);
```

```
public List<Person> search(SearchCriteria query) {
                List<Person> res = new ArrayList<Person>();
                List<Person> persons = dao.getPersons();

                for (Person person : persons) {
                        if ((person.getFirstName().indexOf(query.getFirstName()) != -1)
                             && (person.getLastName().indexOf(query.getLastName()) != -1)
                               && (person.getNIF().indexOf(query.getNIF()) != -1))

                          {
                                res.add(person);
                        }
                }
                return res;
        }
```

9.9.2 El contenidor IoC

El contenidor de beans de l'Spring configura la capa de negoci. La instanciació del contenidor IoC es fa utilitzant els arxius de configuració XML, entre ells i el més important és el **applicationContext.xml**. Un dels aspectes més importants de la capa de serveis de negoci és la definició semàntica de les transaccions. Dins d'aquest arxiu es defineixen els beans i les seves propietats configurant el contenidor. Spring proporciona unes classes que permeten definir la capa de negoci utilitzant la Programació Orientada a Aspectes (AOP). El *transactional proxy* és el nom que rep l'objecte, i la capa web no ha de ser conscient de la gestió transaccional que hi ha darrera del bastiment.

Les classes com **ProxyFactoryBean** o **TransactionProxyFactoryBeab** d'Spring permeten encapsular diferents implementacions d'una classe interceptant les peticions i crides dels mètodes escollint la classe resultant.

Escollim la classe **TransactionProxyFactoryBean** que és senzilla d'utilitzar i proporciona totes les funcions que necessitem per la nostra aplicació. Els objectes **personManager** i **targetaManager** són el resultat d'aplicar el ***Consell*** (***Advice***) a l'Objecte Destinatari. Per tant, la resta de l'aplicació només suportarà l'objecte destinatari: **personManagerTarget** i **targetaManagerTarget** respectívament.

```
<!— Objecte Proxy personManager -->

<bean id="personManager"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"><ref local="transactionManager"/></property>
    <property name="target"><ref local="personManagerTarget"/></property>
    <property name="transactionAttributes">
      <props>
        <prop key="save*">PROPAGATION_REQUIRED</prop>
        <prop key="remove*">PROPAGATION_REQUIRED</prop>
        <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
      </props>
    </property>
</bean>
```

La classe ***Target*** (Destinatari), és la classe aconsellada, en el nostre cas són els serveis implementats:

```
<!— Objecte Target  personManager -->
<bean id="personManagerTarget" class="main.java.org.springframework.webflow.samples.pfc.service.impl.PersonManagerImpl">
    <property name="personDAO"><ref local="personDAO"/></property>
</bean>
```

Per a més informació veieu la documentació de referència d'Spring:
http://static.springframework.org/spring/docs/2.5.x/reference/index.html

9.10 Fase 3: Suport per la persistència

La part de persistència és molt important dins de l'aplicació. És on definim com les dades seran emmagatzemades, potencialment en un temps indefinit. Tal i com hem descrit en l'apartat de disseny definim una relació entre la bases de dades relacional i el nostre model conceptual del món real. En aquest cas hem seleccionat **Hibernate** com a bastiment de mapatge entre un món i un altre, tal vegada, podíem haver seleccionat un altre però, hem de remarcar que les eines d'Spring ens permeten poder escollir qualsevol altra solució, i només amb un canvi en els arxius de configuració podem canviar una per l'altra.

9.10.1 El model entitat relació

En la nostra aplicació per simplificar la implementació ens centrarem en dues entitats: les persones i les targetes. Les dues estaran dins d'una base de dades i estaran relacionades entre elles.

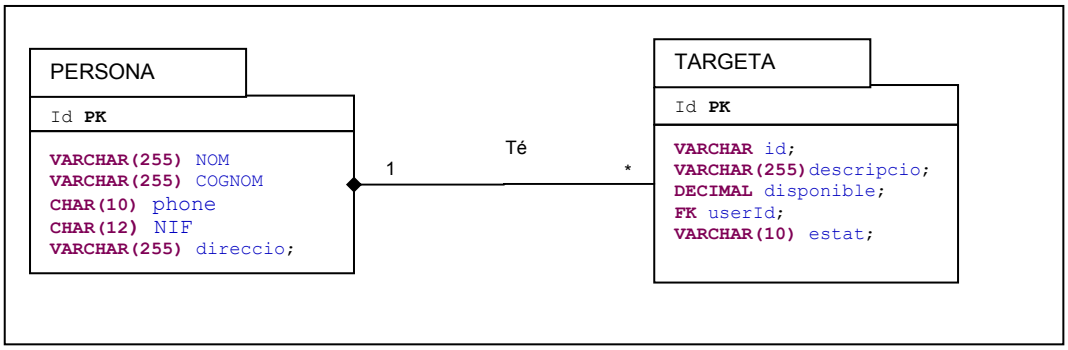


Figura 25. Taules de l'aplicació exemple

9.10.2 Mapeig objecte relació

Com hem comentat utilitzem Hibernate com a solució de mapeig Objecte/Relació. L'elecció d'un ORM pot ser una tasca molt difícil especialment considerant les recents especificacions **EJB 3.0**. Spring suporta tots els ORM més populars i també permet treballar amb EJB 3.0. Hibernate va ser el primer ORM que va incorporar Spring i continua sent molt popular. Per tant, és l'elecció més lògica per al nostre exemple d'aplicació ja que reuneix tots els seus requeriments.

En aquest apartat veurem el mapeig de les dades de l'objecte a la entitat relacional i les seves definicions dins dels arxius **xxx.hbm.xml** per a cada classe. Aquestes definicions de mapeig comencen en la definició del domini d'objectes i les seves taules.

Els XML de mapeig d'Hibernate són molt intuïtius i de fàcil comprensió encara que no tinguem familiaritat amb Hibernate.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
          "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
          "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<!--
Arxiu que mapeja el POJO a classe Hibernate de persistència
 -->
<hibernate-mapping package="main.java.org.springframework.webflow.samples.pfc">
          <class name="Person" table="person">
                    <id name="id" column="ID" unsaved-value="null">
                              <generator class="assigned"/>
                    </id>
                    <property name="firstName" column="NOM"
                                        not-null="true"/>
                     <property name="lastName" column="COGNOM"
                                        not-null="true"/>
                    <property name="phone" column="PHONE"/>
                    <property name="NIF" column="NIF"/>
                    <property name="direccio"  column="direccio"/>
          </class>
</hibernate-mapping>
```

Per la creació de la implementació Hibernate de les classes DAO de l'aplicació ho farem dins del paquet **dao/hibernate** de la següent forma:

```
public class PersonDAOHibernate extends HibernateDaoSupport implements PersonDAO {
    private static Log log = LogFactory.getLog(PersonDAOHibernate.class);

    public List getPersons() {
        return getHibernateTemplate().find("from Person");
    }

    public Person getPerson(Long id) {
        return (Person) getHibernateTemplate().get(Person.class, id);
    }

    public void savePerson(Person person) {
        getHibernateTemplate().saveOrUpdate(person);

        if (log.isDebugEnabled()) {
            log.debug("userId set to: " + person.getId());
        }
    }

    public void removePerson(Long id) {
        Object person = getHibernateTemplate().load(Person.class, id);
        getHibernateTemplate().delete(person);
    }
}
```

9.10.3 Implementació del patró DAO

La utilització d'un bastiment per l'accés a les dades físiques de les classes ens permet que la lògica d'accés no sigui complicada. Tota la complexitat està en l'arxiu mapeig. Utilitzem un *Data Acces Object* (DAO) per separar el codi d'accés específic de la resta de l'aplicació. Totes els mètodes requerits els trobarem en les

interfícies **personDAO** i **targetaDAO** i qualsevol classe implementada cal que implementin aquestes interfícies.

En l'arxiu de configuració ApplicationContext.xml es defineixen els beans necessaris per les classes DAO que trobem al sistema transaccional proporcionat per la classe ***transactionManager*** d'accés a dades segons la tecnologia utilitzada, és clar, un ***HibernateTrnasactionManager*** en el nostre cas però, que si utilitzem **JDO** o **iBatis**, llavors la substituirem per un *transaction manager* per a ***JDOTransactionManager***.

A continuació mostrem l'estructura bàsica d'aquest fitxer de configuració:

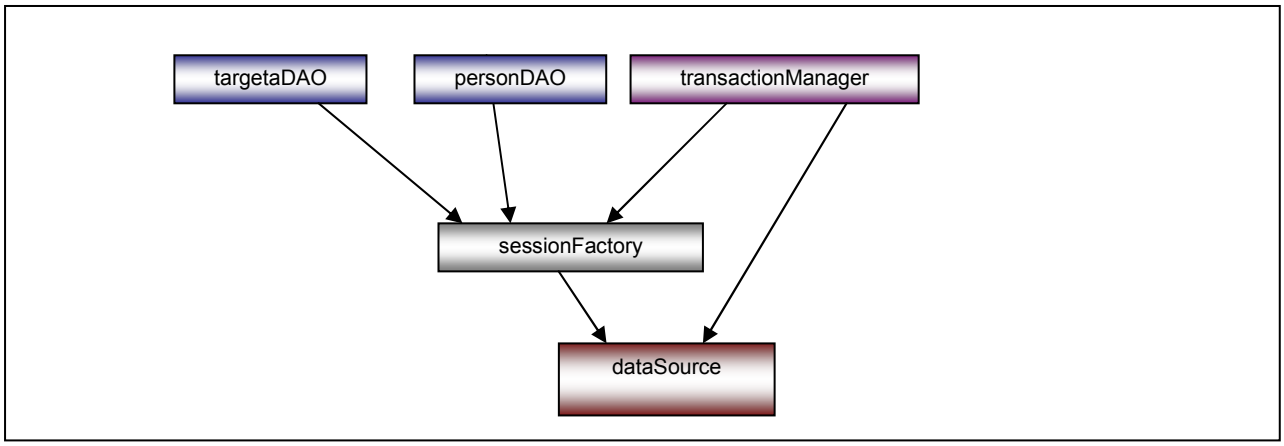


Figura 26. ApplicationContext

9.11 Configuració de l'aplicació

Necessitem dos servidors: un servidor de base de dades, i un servidor d'aplicacions web. Tal i com hem comentat pel testeig de l'aplicació hem utilitzat MySQL versió 4.1 i Apache Tomcat versió 6.0. Baixeu-vos també el programa Apache Ant com a desplegador de l'aplicació.

9.11.1 MySQL

Aquest servidor de bases de dades és programari lliure per tant, podem baixar-nos tot els seus components des de l'adreça www.mysql.com. També haurem de baixar-nos el *driver* JDBC MySQL Connector/J versió mínima 3.1.

Una vegada instal·lat el MySQL es pot cridar a l'arxiu ***build.xml*** per inicialitzar la base de dades creant les taules i carregar-les amb les dades de l'aplicació. Per fer-ho descarregar l'arxiu **.zip** que ve amb el document i executar ant **initDB**, **int populate**, respectivament. Sempre podem executar **ant**, per veure el menú general i les opcions de l'arxiu build.xml.

9.11.2 Tomcat

Podem baixa-nos el servidor d'aplicacions Jakarta Tomcat i instal·lar-lo des de:
http://jakarta.apache.org/tomcat/index.html.

La seva instal·lació és molt simple, només cal descomprimir la distribució i instal·lar-lo al directori corresponent de Windows. En el nostre cas ho hem fet a :
C:\Archivos de programa\Apache Software Foundation

Tenir-ho en compte dins dels arxiu de propietats del projecte. La millor opció és configurar les variables d'entorn CATALINA_HOME, JAVA_HOME, CLASSPATH de forma adient (queda fora del projecte la seva configuració).

9.11.3 Compilació i desplegament

Per compilar i desplegar l'aplicació és molt senzill, només cal executar **ant deploywar**.

Hi ha altres maneres de fer-ho, per exemple, desplegant l'aplicació diréctament utilitzant l'arxiu **.war** creat fent ant war, dins de l'administrador del Tomcat. O, posant l'arxiu .war creat (**pfc.war**) en el directori **weapps** del Tomcat i arrencar el servidor.

Per executar només cal posar: http://localhost:8080/pfc.

9.12 Resum

En aquest darrer capítol del projecte hem vist com en una aplicació exemple podem utilitzar ***Spring*** com a base per desenvolupar una aplicació ***J2EE***. Hem examinat les diferents capes de l'arquitectura i els arxius de configuració associats. També hem pogut estudiar i discutir com es codifiquen les opcions escollides d'aquesta tecnologia i que cal fer per poder desenvolupar una aplicació amb Spring, ***Webflow*** i ***Hibernate***.

Hem cobert tots els passos necessaris per preparar el servidor de base de dades ***MySQL***, la seva configuració i la relació amb els arxius de configuració. En aquests arxius també es defineixen la relació de les entitats de persistència amb les transaccions Hibernate així com la programació orientada a aspectes i la inversió de control caracteritzada per Spring.

Amb la configuració de la base de dades hem vist també com construir i desplegar una aplicació dins del servidor ***Tomcat***. La utilització d'eines com ***Apache Ant***, en ha servit per automatitzar tots els passos fins arribar a provar l'exemple: construcció de la BD, càrrega de dades, compilació i desplegament de l'aplicació, càrrega en el servidor i gestió del servidor (aturar, arrencar i replegar l'aplicació).

La utilització de la tecnologia ***WebFlow*** permet construir aplicacions amb navegació complexa complementant les especificacions Servlet.

La construcció de les pàgines amb ***JSP*** i llibreries de ***Tags*** pròpies ens permeten posar ordre a la construcció de pantalles d'una forma homogènia. El dissenyador de pàgines web només cal que utilitzi les eines que disposa sense deixar-se influir per la seva opinió.

Hibernate facilita l'accés a les dades en la capa de persistència per tant no hi ha necessitat de construir classes i lògica complexa d'accés a bases de dades.

Utilitzar patrons com ***Facades***, ***DAOs***, ***Application Services*** i ***Service*** ens permet crear un catàleg de prestacions, avui és la de bloqueig de targeta, gestió d'usuaris o gestió de targetes, demà pot ser la de consulta de moviments o realitzar un ingrés a un compte.

Donat que el marc de treball ofereix gestió de transaccions declaratives ens fa descartar la utilització de EJBs. Per tant, l'aplicació es desplega en un arxiu WAR en qualsevol contenidor web compatible amb la versió 2.3 *Servlet*. Això simplifica la fase de desplegament i permet la transportabilitat entre servidors.

Per últim, la utilització de programari lliure permet reduir costos. Hem utilitzat *MySQL* y *Tomcat*, dues solucions senzilles però, per aplicacions empresarials hauríem d'evolucionar cap a tecnologies com ***Oracle*** o ***Weblogic***.

10 Conclusions i línies obertes

10.1 Conclusions

La finalitat del projecte era l'estudi arquitectònic de la plataforma J2EE sota l'estructuració del seu model en capes i dins de l'escenari concret dels bastiments que trobem al mercat per tal de poder desenvolupar una aplicació empresarial.

10.1.1 L'estudi

Respecte a l'evolució dels frameworks web podem afirmar que cada vegada més van cap a una abstracció del protocol HTTP per a beneficiar-se dels models basats en controls, components i esdeveniments que tan èxit van tenir en l'àmbit d'aplicacions escriptori. La majoria de bastiments estudiats s'enfronten als mateixos problemes: navegació, validació, internacionalització, errors, escalabilitat. Els més antics com Struts permeten una utilització més completa de les dades processades mentre que els més recents com *Tapestry*, *JSF* o *Spring* cerquen l'abstracció casi total del protocol.

Respecte a la capa de negoci i integració hem constatat que hi ha dos grans grups crítics, aquells que defensen el Spring framework i la comunitat Sun amb la seva darrera especificació EJB 3.0. La diferència més gran en utilitzar o no un bastiment com Spring és la facilitat que ens proporciona per utilitzar la tecnologia J2EE d'una manera més simple, en canvi, el suport de la comunitat Sun i l'estandardització EJB proporcionen raons importants pel seu ús.

Respecte als sistemes d'emmagatzemament persistent, molts paquets han estat desenvolupats per reduir la farragosa tasca de desenvolupar sistemes de mapeig proveint llibreries o classes que poden fer-ho automàticament. Donada una llista de taules de la base de dades i objectes del programa, automàticament mapejaran peticions entre un i l'altre. Demanar a un objecte persona els seus telèfons, resultarà en la creació i enviament d'una *query* correcta i adequada i els resultats seran traduïts directament en objectes telèfon dintre del programa.

A la pràctica, tanmateix, les coses mai són tan senzilles. Tots els sistemes ORM tendeixen a fer-se visibles de diverses maneres, reduint alguns graus la pròpia capacitat d'ignorar la base de dades. Pitjor, la capa de traducció pot ser lenta i ineficient (notablement en termes de l'SQL que escriu), resultant programes més lents i utilitzant més memòria que el codi escrit "a mà".

No hi ha un *framework* millor per a tot, cada un d'ells dóna solucions determinades a diferents problemes i escenaris. La millor elecció és conèixer bé quina aplicació s'ha de desenvolupar, l'equip humà que hi ha al darrera i el suport que hom disposa pel desenvolupament del projecte. L'elecció final dependrà també d'altres factors com el suport d'una comunitat extensa i una bona documentació que recolzi el projecte.

10.1.2 La solució

En el disseny de la solució final aportem un estudi arquitectònic i un model base que han de cobrir qualssevol aplicació creada sota aquest entorn i establim uns models de disseny que han de ser seguits per tots els desenvolupaments. Aquestes característiques facilitaran la construcció d'aplicacions de qualitat.

Promovem la construcció segons la divisió en capes de la plataforma, seguint el model recomanat per Sun, i recomanem la integració tecnològica utilitzant bastiments i l'ús d'estàndards de construcció i disseny com els patrons.

Potenciem la utilització de nous paradigmes de la enginyeria del progtamari com la programació orientada a aspectes AOP i les bones pràctiques de la codificació amb l'ús de les noves tendencies en la programació com la inversió de control IoC.

També fem especial èmfasi en la reutilització de components, exemple d'això és el catàleg de serveis: Application Services i Services. Aquest catàleg són funcionalitats genèriques candidates a ser utilitzades per la resta d'aplicacions, reduïnt costos i optimitzant els processos.

La realització del projecte ha permès obtenir una visió pròpia de tots els punts que intervenen en la elaboració d'un projecte. Des de la direcció del mateix, amb un anàlisi previ, passant pel disseny i implementació, donant-li sentit comercial al producte final. En resum, creiem que s'ha aconseguit cobrir els objectius fixats al principi del projecte.

10.2 Línies obertes

El projecte compleix una primera part d'un projecte molt més ambiciós: la creació d'una arquitectura concreta per desenvolupar projectes empresarials sota la perspectiva del món de les assegurances i de les grans entitats bancàries, és a dir, la construcció del framework propi sota les solucions del mercat escollides: Spring, Hibernate, JSP, SXLT, WebFlow i custom tags.

Hem fet una primera feina, la més important, d'estudi i anàlisi però, queda la darrera feina: la creació d'una arquitectura i d'una metodologia de treball.

La realització de projectes en entitats bancàries, per posar un exemple concret, també hauria de definir un procés d'enginyeria de programari global a nivell organitzatiu, que tingui en compte la gestió de la configuració i la gestió del canvi. Aquests darrers temes juntament amb la definició final del model arquitectònic permetria trobar un producte final robust i de qualitat.

La utilització d'aquesta memòria podria enriquir alguns d'aquests conceptes com a base per construir paquets de programari empresarial sota la plataforma J2EE que utilitzi solucions estàndard del mercat.

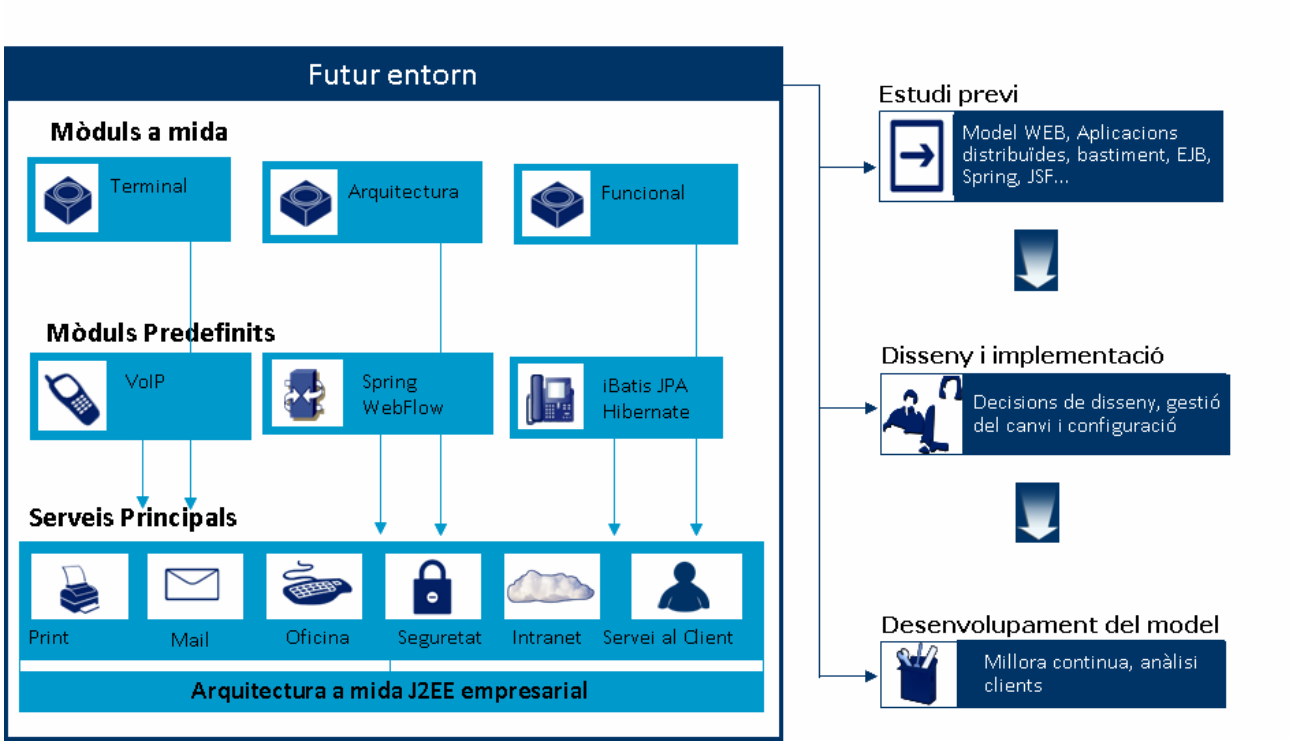


Figura 27. Proposta del model futur

11 Bibliografia i referències

11.1 Internet

Lloc oficial d'Spring
http://www.springframework.org

The Spring Framework v-2.5- Reference Documentation
http://static.springframework.org/spring/docs/2.5.x/reference/index.html

Spring Web Flow - Reference Documentation
http://static.springframework.org/spring-webflow/docs/1.0.x/reference/

Wikipedia
http://www.wikipedia.org/

JavaHispano
http://www.javahispano.org/

Projecte Apache
http://jakarta.apache.org/tomcat/index.html

MySQL
www.mysql.com

Sun i Blueprints
http://java.sun.com/reference/blueprints/

Hibernate
http://www.hibernate.org/

Tutorial Java EE 5
http://java.sun.com/javaee/5/docs/tutorial/doc/bnbls.html

11.2 Llibres

Ingeniería del Software – Un enfoque práctico.
Pressman, Roger S. Mc Graw Hill, 1993.

Appling UML and Patterns, 3rd ed.
Larman, Craig. Addison Wesley International, 2004.

Professional Java Development with the Spring Framework.
Rod Johnsonet al.
John Wiley & Sons 2005.

Pro Spring.
2005 by **Rob Harrop and Jan Machacek.**

Expert Spring MVC and Web Flow.
2006 by **Seth Ladd, Darren Davison, Steven Devijver, and Colin Yates.**

11.3 Tutorials

Spring Framework Tutorial.
Isabelle Muszynski.
15 April 2003.

The Spring 1.x Primer.
Matt Raible.
2004-2007.

Developing a Spring Framework MVC application step-by-step.
Thomas Risberg.
July, 2003 .