Anàlisi, disseny e implementació d'una aplicació de comerç electrònic en la plataforma J2EE amb Struts e Hibernate

Jordi Raya Lapuente
ETIG

Albert Grau Perisé

14/01/2008

Per a en Santi Ortega,
que la terra et sigui lleugera

Agraïments:

A la *Comunidad de Alumnos de la UOC* (CAUOC)
http://www.cauoc.com
pel seu inestimable suport acadèmic i humà
que en tantes ocasions fan possible els estudis
en una universitat com la UOC.

A tots els tutors, professors i personal administratiu de la UOC
perquè amb la seva tasca tenim aquesta universitat

I evidentment, a l'Albert, pel seu suport i paciència.

A tots i totes, moltes gràcies.

Resum

El present treball consisteix en l'anàlisi, disseny e implementació d'una aplicació de comerç electrònic equiparable en la seva forma bàsica a una botiga virtual de productes genèrics disponible a Internet mitjançant el servei web. S'inclou en l'aplicació tant la part pública o *front-end*, consistent en la part accessible al públic, on s'efectuen els actes de negoci amb usuaris externs al sistema, com la part privada o *back-end*, destinada a les tasques de manteniment per a l'administrador de la botiga.

L'aplicació es desenvolupa sota la plataforma J2EE (*Java 2 Enterprise Edition*) seguint els estàndards actuals de l'enginyeria del programari, com l'arquitectura de tres capes dividida en la capa d'accés a dades, on s'interactua amb una base de dades relacional, la capa de negoci on resideix la lògica de l'aplicació, i la capa de presentació on s'interactua amb l'usuari mitjançant un navegador web. També s'utilitzen intensivament els patrons de disseny J2EE per tal d'utilitzar les solucions comunes de la plataforma als requeriments usuals d'una aplicació J2EE.

Per a la implementació s'utilitzen els bastiments (*frameworks*) Struts per la gestió de les interaccions entre la capa de negoci i la de presentació seguint el patró MVC (*Model-View-Controler*), e Hibernate per resoldre l'accés a dades. Així, s'investigarà la implementació d'una arquitectura estandarditzada J2EE amb aquests bastiments, considerant diversos factors com els requeriments d'implementació i funcionament d'aquests, les restriccions al disseny i desenvolupament que poden imposar a una aplicació a canvi d'oferir unes funcionalitats suficientment provades, o l'efectivitat del desenvolupament ràpid que proporcionen.

Paraules clau: comerç electrònic, botiga virtual, web, Internet, J2EE, Java, Servlets, JSP, bastiments, frameworks, Struts, Hibernate, patrons, disseny d'aplicacions, arquitectura J2EE, arquitectura a tres capes, enginyeria del programari, programació orientada a l'objecte, aplicació web, aplicació distribuïda.

Àrea del TFC: J2EE.

Índex

Resum ____ ***3***

Índex ____ ***4***

Memòria ____ ***6***

Introducció ____ **6**
- Justificació del treball ____ 9
- Objectius ____ 10
- Enfocament i mètode ____ 11
- Planificació ____ 12
- Altres capítols ____ 13

Anàlisi ____ **14**
- Requeriments funcionals ____ 14
- Requeriments tecnològics ____ 15
- Rols ____ 16
- Casos d'ús ____ 17
- Interfície Gràfica d'Usuari ____ 28
- Consideracions sobre l'anàlisi ____ 37

Disseny ____ **38**
- Visió general de l'arquitectura ____ 38
- Capa de dades ____ 40
- Capa de lògica de negoci ____ 43
- Capa de presentació ____ 57

Conclusions ____ **61**

Glossari ____ ***62***

Bibliografia ____ ***64***

Annexos ____ ***66***

Annex I - notes de instal·lació i programari utilitzat ____ **66**

Annex II - Patrons de disseny en l'aplicació ____ **67**
- Data Acces Object (DAO) ____ 67
- Transfer Object (TO) ____ 72

Índex de figures

Diagrama de Gantt amb la planificació del projecte 12
Representació conceptual de l'arquitectura a tres capes 15
Diagrama de casos d'ús per Usuari i Client 17
Diagrama de casos d'ús d'Administrador sobre Categoria 20
Diagrama de casos d'ús d'Administrador sobre Producte 22
Diagrama de casos d'ús d'Administrador sobre Comanda 23
Diagrama de casos d'ús d'Administrador sobre Client 26
Diagrama de casos d'ús general per Administrador 27
Front-end: pantalla d'inici 28
Front-end: pantalla de continguts de categoria 29
Front-end: pantalla de detall de producte 30
Front-end: pantalla de continguts del carret 31
Front-end: pantalla de simulació de pagament en línia 32
Front-end: pantalla de confirmació de comanda 33
Back-end: pantalla de menú principal 34
Back-end: pantalla de llistat de productes 35
Back-end: pantalla de formulari de producte 36
Esquema de l'arquitectura 39
Diagrama E-R 40
Esquema de mapeig classe-taula amb Hibernate 41
Diagrama estàtic de la capa de negoci 43
Diagrama estàtic de Persona 45
Diagrama estàtic de Client 46
Diagrama estàtic d'Adresa 47
Diagrama estàtic de Categoria 48
Diagrama estàtic de Producte 49
Diagrama estàtic de Comanda 50
Diagrama d'estats de Comanda 51
Diagrama estàtic d'EstatComanda 51
Diagrama estàtic de ProducteComanda 52
Diagrama estàtic de Carret 53
Diagrama estàtic de ProducteCarret 54
Diagrama estàtic de ValidacioAlgorismes 55
Diagrama estàtic de les classes d'excepció 56
Esquema de mapeig classe-JSP amb Struts 57
Esquema d'exemple de les diferències de codificació entre TDLs i scripting 59
Plana del font-end que mostra la utilització de includes per implementar elements comuns 60

Memòria

Introducció

En el món actual cada cop prenen més importància les relacions comercials electròniques, tant a nivell empresarial (B2B, *Business-to-Business*) com entre empreses i consumidors (B2C, *Business-to-Client*). Podríem considerar que el comerç electrònic és una metodologia moderna per fer negocis que detecta la necessitat de les empreses, comerciants i consumidors de reduir costos, així com de millorar la qualitat de béns i serveis i millorar els temps d'entrega d'aquests béns i serveis. Així, el comerç electrònic no és una tecnologia, sinó l'ús de la tecnologia existent per a dur a terme les relacions comercials. En molts casos, només seria l'extensió electrònica del comerç tradicional, però també podríem considerar que es tracta de qualsevol transacció comercial on les parts involucrades substitueixen els intercanvis físics directes per operacions electròniques.

En una escala global, el comerç electrònic és el mitjà del que disposen les organitzacions comercials per adaptar-se als canvis dels processos d'abastiment i explotació de mercats que han aparegut com a conseqüència de l'economia global. Així, el comerç electrònic permet una millora de l'eficiència de les operacions internes, tant per interactuar de forma més propera amb els proveïdors com per acomplir les expectatives dels clients.

En el cas del comerç minorista els avantatges del comerç electrònic són innegables. Pel que fa als clients, cal destacar en primer lloc que es permet l'accés a més informació. La natura interactiva de la web i dels documents d'hipertext permeten cerques en profunditat no lineals que són iniciades i controlades pels clients mateixos, i es pot obtenir molta més informació sobre un producte o servei que la que normalment està disponible en altres formes comercials. D'altra banda, també es facilita la investigació i comparació de mercats, donat que la capacitat de Internet per acumular, analitzar i controlar grans volums de dades especialitzats permet la compra per comparació i accelera el procés de cerca de productes. Finalment, s'abarateixen els costos i preus, donat que a mesura que augmenta la capacitat dels proveïdors per competir en un mercat electrònic obert baixen els costos i preus i augmenta la qualitat i varietat dels productes i serveis com a natural conseqüència de la competència.

Paral·lelament, els avantatges per a les empreses també són considerables. Cal considerar, en primer lloc, les millores en la distribució. Internet ofereix a determinats tipus de proveïdors la possibilitat de participar en un mercat interactiu en el qual els costos de distribució i vendes tendeixen a zero. Per exemple, els productes digitals de programari es poden entregar instantàniament sense necessitat d'intermediaris. Els compradors i venedors poden contactar entre ells directament, eliminant les restriccions que es presenten en aquestes interaccions, i també es pot disminuir el temps de les transaccions comercials, incrementant l'eficiència empresarial.

Un altre avantatge important consisteix en la millora de les comunicacions. Si bé la majoria de les empreses utilitzen la web per informar als seus clients sobre l'empresa i els seus productes i serveis, cal considerar que la natura interactiva de la web facilita les relacions amb els clients fins a un punt inabastable per als procediments tradicionals. Per exemple, una botiga virtual està oberta les vint-i-quatre hores del dia, i disposa d'instruments tals com el correu electrònic o els formularis per estimular als clients a sol·licitar tanta informació con desitgin, establint una relació directa asincrònica amb l'empresa.

Finalment, cal considerar els beneficis operacionals. La utilització empresarial de la web redueix errors, temps i despeses en el tractament de la informació donat que, per la seva qualitat de procés informàtic, es facilita la integració amb els altres processos informàtics habituals en l'activitat empresarial, com la facturació i el control de costos. A més, la facilitat per

contactar amb els clients directament, ja sigui els actuals o els potencials, i la facilitat per disposar amb la mateixa facilitat de les dades públiques que de les d'ús intern de l'empresa, juntament amb la possibilitat d'aplicar els mateixos procediments informàtics per a totes dues, permet l'eliminació d'endarreriments entre les diferents etapes del procés empresarial.

Per tant, donades les evidents avantatges del comerç electrònic, s'espera que les organitzacions comercials generin la necessitat de produir aplicacions informàtiques per tal de dur-lo a terme. Ja hem assenyalat que el comerç electrònic no és una tecnologia en si mateixa, sinó l'aplicació d'aquesta en l'àmbit del comerç. Per tant, les possibilitats de dur a terme accions comercials electròniques i gaudir dels seus beneficis tant per als clients com per a les empreses són directament dependents de l'estat de l'art de la tecnologia en un moment donat. Cal pensar en les múltiples implicacions tecnològiques implicades fins i tot en les interaccions comercials més senzilles, com per exemple les qüestions relatives a identitat, seguretat, consistència de les dades, etc.

Així, les organitzacions esperen aplicacions eficients, segures i escalables, a més de valorar qüestions com la facilitat de manteniment o el temps de desenvolupament, i els desenvolupadors de programari esperen d'una plataforma de desenvolupament que sigui prou fiable i estigui prou estandarditzada en el mercat com per poder delegar-ne aquestes qüestions en productes de terceres parts amb plenes garanties d'èxit, i que permeti l'aplicació de metodologies de reconeguda eficiència.

Actualment (gener de 2008) la indústria de desenvolupament de programari està fortament polaritzada en funció de les dues principals plataformes de disseny i desenvolupament d'aplicacions empresarials orientades a Internet. La primera i més antiga és la plataforma J2EE (*Java 2 Enterprise Edition*), impulsada per Sun i una multitud de proveïdors de components, servidors d'aplicacions i sistemes de bases de dades amb més o menys pes en el mercat, com poden ser IBM, BEA, etc. que competeix directament amb la plataforma .NET, de més recent aparició en el mercat e impulsada principalment per Microsoft.

Sense entrar en una enumeració detallada, cal considerar que totes dues plataformes es basen en principis molt similars, com la programació orientada a objectes; la utilització de l'anomenada *màquina virtual* com a capa intermèdia per la interacció entre el codi i el maquinari; l'ús de servidors d'aplicacions per realitzar la lògica de negoci; l'ús de sistemes gestors de bases de dades, principalment relacionals en aquests moments, per emmagatzemar les dades; la preparació dins de les especificacions per les aplicacions distribuïdes; i finalment l'habitual utilització de navegadors web com a clients lleugers en la majoria de desenvolupaments per tal d'interactuar amb els usuaris finals. Per tant, l'elecció d'una o altra plataforma no és una qüestió d'escollir entre diferents paradigmes de desenvolupament o metodologies de programació oposades, sinó que dependria d'altres factors aliens a aquestes consideracions. En tot cas, podem afirmar amb seguretat que aquests elements sintetitzen l'estat de l'art de les aplicacions empresarials orientades a Internet, els quals han esdevingut veritables estàndards en l'enginyeria del programari.

Centrant-nos en la plataforma J2EE, en els darrers anys han aparegut diversos bastiments o *frameworks*. Un bastiment és un disseny reutilitzable per un sistema de programari que proporciona els fonaments comuns per un tipus de programari específic, amb la finalitat concreta de facilitar el desenvolupament de programari. Aquest objectiu s'aconsegueix al permetre als desenvolupadors centrar-se en l'acompliment dels requeriments de programari en comptes de tractar les operacions comunes i repetitives de baix nivell. A canvi d'aquests avantatges, cal una inversió inicial en temps per tal de conèixer el seu funcionament. Així, un bastiment seria més específic que la plataforma sobre la que funciona, i al mateix temps més genèric que, per exemple, una llibreria de codi.

Si bé és una pràctica comuna de les empreses de desenvolupament de programari la realització de bastiments per al seu ús intern, en els darrers anys han aparegut diversos bastiments d'ús lliure que han assolit gran popularitat entre la comunitat de desenvolupadors, com per exemple Struts, Hibernate, Spring, Stripes, Tapestry, etc. i podríem afirmar que han esdevingut un altre lloc comú dins del panorama tecnològic.

Així, un cop descrit l'actual estat de l'art de la tecnologia, es presenta aquest treball, el qual consisteix en l'anàlisi, disseny e implementació d'una aplicació de comerç electrònic en la plataforma J2EE, conjuntament amb els bastiments Struts e Hibernate. En aquesta aplicació es mostren les possibilitats que la esmentada plataforma ofereix per al desenvolupament d'aplicacions de comerç electrònic, utilitzant les pràctiques considerades com més recomanables per la indústria del programari en totes les etapes i aspectes possibles del seu desenvolupament.

Justificació del treball

Actualment, es pot considerar que la plataforma J2EE està plenament consolidada en el mercat i ja ha arribat a un nivell de maduresa suficientment elevat com per a ser considerada en qualsevol projecte de desenvolupament d'aplicacions de comerç electrònic, tal com mostrarien les dades sobre la seva presència en organitzacions de caire més divers. Les qüestions relacionades amb la seguretat, fiabilitat i facilitat de desenvolupament han anat millorant progressivament en cada nova versió, i per tot arreu existeixen comunitats de desenvolupadors que mostren la seva importància i el seu nivell d'implementació.

Paral·lelament, els bastiments gaudeixen d'una popularitat en creixement continuat. Si bé en uns determinats moments es van desenvolupar bastiments propietaris per a ús intern de les empreses de desenvolupament de programari, en l'actualitat els bastiments de codi obert gaudeixen d'una gran acceptació. Aquest fet no hauria de considerar-se com un arreglament de possibles deficiències de la plataforma J2EE, sinó com la resposta de la comunitat de desenvolupament a necessitats comunes als tipus de projectes més comuns.

Al mateix temps, en un àmbit més general, la indústria de l'enginyeria del programari està començant a assolir importants nivells d'estandardització. L'ús de la programació orientada a l'objecte, de UML (*Unified Modeling Language*), dels patrons de programari o de les metodologies de desenvolupament han esdevingut llocs comuns i són elements d'obligat coneixement en l'àmbit actual del desenvolupament d'aplicacions.

També cal considerar els canvis que ha comportat l'aparició de Internet. Les aplicacions distribuïdes han conegut un gran creixement gràcies a les possibilitats que ofereix la xarxa. El paradigma arquitectònic client-servidor ha pres com a configuració principal l'arquitectura de tres capes dividida en dades, lògica de negoci i presentació, on els clients ja no són clients pesats que emmagatzemen part de les dades, sinó navegadors web fàcilment utilitzables per qualsevol usuari, fàcilment disponibles i sense que requereixin instal·lacions especials.

Així, en l'estat de l'art actual de les aplicacions empresarials orientades a Internet, semblaria inútil estudiar les capacitats més que demostrades de la plataforma J2EE o dels bastiments més populars. No és aquest el nostre propòsit, sinó investigar les possibilitats d'implementació d'una arquitectura estandarditzada en la plataforma J2EE utilitzant bastiments prou coneguts i consolidats, per tal d'estudiar-ne els possibles problemes que puguin aparèixer per tal de trobar les millors solucions possibles.

Cal tenir present que la utilització d'un bastiment sempre implica acceptar una sèrie de restriccions més o menys importants, determinades tant per la mateixa arquitectura del bastiment com per les llibreries, llenguatges de *scripting* i d'altres elements auxiliars que aquests posen a la disposició dels desenvolupadors. Si bé aquests elements preveuen les operacions més usuals, existeix la possibilitat de que sorgeixi alguna necessitat no prevista i que ens veiem obligats a l'escriptura de codi aliè als paràmetres del bastiment. D'altra banda, també existeix la possibilitat que un bastiment obligui o només mostri totes les seves potencialitats si es realitzessin determinades pràctiques que entrarien en conflicte més o menys directe amb les que l'enginyeria del programari recomana.

Donat que el paradigma arquitectònic actual és l'arquitectura de tres capes, centrarem el present estudi en bastiments que actuïn sobre totes i cadascuna d'elles. Per a les capes de lògica de negoci i presentació hem triat Struts, al que podríem considerar com el bastiment més popular de la plataforma J2EE, donat que la seva implementació del patró MVC (*Model-View-Controler*) incideix tant en la lògica de negoci (el model) com en la presentació (la vista). Pel que fa a la capa de dades, l'elecció òbvia ha estat Hibernate, que acostuma a presentar-se com una alternativa de més fàcil utilització i més lleugera que l'estàndard EJB (*Enterprise Java Beans*) el qual forma part de l'especificació J2EE.

Objectius

L'objectiu principal del present projecte és l'anàlisi, disseny e implementació d'una aplicació de comerç electrònic en la plataforma J2EE utilitzant els bastiments Struts e Hibernate. Aquesta aplicació ha d'estar suficientment complerta i elaborada fins al punt en que es pugui considerar com una simulació prou convincent de les interactuacions que s'esdevindrien en l'hipotètic cas d'una posada en funcionament en un entorn de producció real.

Les interaccions que s'intenten emular són, per una banda, les que s'esdevenen entre els humans i el programari, i per altra les que es produeixen entre els diferents components del programari. Respecte al primer tipus, distingirem entre les interaccions dels clients externs a l'organització de les del personal d'administració de l'aplicació, per tal de mostrar tant els avantatges del comerç electrònic per a les dues parts que intervenen en una relació comercial com les possibilitats de la plataforma J2EE per a dur a terme eficientment la tasca de intermediació informatitzada.

Pel que fa a les interactuacions entre les diferents parts del sistema, s'implementarà una arquitectura de tres capes, com és habitual en la indústria, mantenint en tot moment una estricta separació entre dades, lògica de negoci i presentació, per tal de mostrar la capacitat de la plataforma J2EE de implementació d'aquesta arquitectura i com el rol que acompleixen els bastiments en l'esquema general de funcionament.

Així, la finalitat principal de l'aplicació consistirà en mostrar la capacitat de la plataforma J2EE per al desenvolupament d'aplicacions de comerç electrònic, i al mateix temps avaluar les possibilitats que s'ofereixen amb la utilització dels bastiments Struts e Hibernate per aquesta mateixa plataforma.

Considerem també un objectiu de la màxima prioritat el fet de mantenir en tot moment les pràctiques considerades com a més recomanables en l'enginyeria del programari, malgrat les eventuals restriccions que es puguin derivar de la utilització dels bastiments anteriorment esmentats, per tal d'avaluar i solucionar els possibles conflictes que es puguin esdevenir.

No es considera com a objectiu en el present projecte la finalització complerta de l'aplicació en tots els detalls que estarien presents en una aplicació real, com poden ser la optimització de consultes, el seguiment dels estàndards W3C en la capa de presentació, o la vistositat i usabilitat d'aquesta, donat que tals aspectes, per la seva complexitat, distraurien de l'enfocament en l'arquitectura i la interrelació d'aquesta amb els bastiments.

Enfocament i mètode

Donat que l'objectiu consisteix en investigar les interrelacions entre les pràctiques consolidades pel desenvolupament d'aplicacions empresarials i la utilització dels bastiments Struts e Hibernate, òbviament ha estat obligatori utilitzat aquestes pràctiques en totes les etapes del cicle de vida del programari sempre que ha estat possible.

Es parteix, en primer lloc, de la coneguda arquitectura a tres capes, consistent en l'estricta separació entre dades, lògica de negoci i presentació, la qual permet un fàcil escalament de les aplicacions i la reutilització de components. Cal considerar, per exemple, que si bé l'aplicació es desenvolupa amb una presentació apta per a la seva utilització amb un navegador estàndard amb capacitat per visualitzar documents HTML, en el futur es podria considerar adient ampliar la presentació cap a altres dispositius, com aplicacions residents amb interfície gràfica d'usuari, terminats WAP, o qualsevol altra tecnologia que pugui aparèixer. L'arquitectura de tres capes ens permetria efectuar els canvis adients o la substitució de la presentació sense necessitat de modificar les dades ni la lògica de negoci.

Respecte a la capa de dades, hem considerar adient la utilització d'una base de dades relacional com MySQL, donat que els sistemes gestors de bases de dades orientades a l'objecte encara no semblen prou consolidats en el mercat i en cas d'utilitzar-ne un el resultat no es podria considerar una correcta simulació d'una aplicació real. Com és habitual en el cas d'una base de dades relacional, s'ha modelitzat mitjançant diagrames E-R.

Seguint les recomanacions usuals per aquests tipus d'aplicacions, la capa de presentació s'ha codificat de la manera més lleugera possible. Així, s'han utilitzat planes JSP (*Java Server Pages*) amb tots els mecanismes disponibles per a l'alleugeriment de codi que permet l'estat actual de la tecnologia, com les llibreries de tags (TDL) proporcionades per Struts i les pròpies de J2EE, juntament amb el llenguatge de scripting EL (*Expression Language*).

En les fases d'anàlisi i disseny s'han utilitzat diagrames UML, tal com resulta habitual en la indústria de desenvolupament de programari, per tal d'estudiar les possibilitats de implementació dels models resultants en el codi de l'aplicació. D'altra banda, seguint novament les tendències actuals en la indústria, es fa un ús intensiu dels patrons de programari sempre que les circumstàncies així ho recomanen. Donat que l'aplicació es desenvolupa sota la plataforma J2EE, s'ha fet una especial indicència en la utilització dels patrons del mateix catàleg J2EE, sense obviar, però, la utilització puntual de patrons provinents d'altres catàlegs, com el conegut GoF.

En les fases de disseny e implementació s'ha fet una especial incidència en les possibilitats de reaprofitament de codi i d'utilització de les possibilitats del paradigma de la programació orientada a objectes. Per tant, s'utilitzen els mecanismes d'herència i polimorfisme sempre que sigui possible i recomanable. D'altra banda, en la implementació es segueixen escrupolosament les convencions de codi del llenguatge Java, tal com aconsella Sun, per tal de permetre un fàcil manteniment i escalabilitat de l'aplicació.

En resum, considerem que l'enfocament metodològic i tecnològic emprat és prou representatiu de l'estat de l'art actual del desenvolupament d'aplicacions empresarials en la plataforma J2EE, fet que ens permetrà l'estudi detallat de les interrelacions d'aquestes metodologies i tecnologies amb els bastiments Struts e Hibernate, tal com és l'objectiu del present projecte.

Planificació

Per a la planificació s'han seguit les etapes habituals en el cicle de vida del programari: presa de requeriments, anàlisi, disseny e implementació. Donat que la natura del projecte per la utilització de bastiments afavoreix la implementació ràpida d'un prototipus que ofereixi les funcionalitats bàsiques i a partir del qual es desenvolupi la resta de l'aplicació, s'ha optat per seguir un cicle de vida amb prototipatge.

Així, el projecte es divideix en quatre fases segons els materials entregats en cadascuna d'elles: planificació inicial, anàlisi i disseny, implementació del prototipus i entrega final.

Cada fase es subdivideix en diferents tasques, que es poden dividir al seu torn en subtasques, tal com es mostra en les capçaleres del següent diagrama de Gantt:

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
Fase 1: Planificació	**19 días**	**mié 19/09/07**	**lun 08/10/07**	
Abast del projecte	7 días	mié 19/09/07	mié 26/09/07	
Definició de requeriments	8 días	mié 26/09/07	jue 04/10/07	2
Especificacions tècniques	8 días	mié 26/09/07	jue 04/10/07	2
Elaboració del planning	3 días	jue 04/10/07	dom 07/10/07	2
Revisió final	1 día	dom 07/10/07	lun 08/10/07	5;4;3
Lliurament PAC 1	0 días	lun 08/10/07	lun 08/10/07	
Fase 2: Anàlisi i disseny	**20 días**	**mar 09/10/07**	**lun 29/10/07**	**1**
Identificació casos d'ús	5 días	mar 09/10/07	dom 14/10/07	
Diagrames estàtics	10 días	dom 14/10/07	mié 24/10/07	9
Diagrames ER	10 días	dom 14/10/07	mié 24/10/07	9
Disseny de la interfície	3 días	mié 24/10/07	sáb 27/10/07	9
Determinació del prototipatge	1 día	sáb 27/10/07	dom 28/10/07	9
Revisió final	1 día	dom 28/10/07	lun 29/10/07	13
Lliurament PAC 2	0 días	lun 29/10/07	lun 29/10/07	
Fase 3: Implementació del prototipus	**48 días**	**mar 30/10/07**	**lun 17/12/07**	**8**
Instal·lació programari	2 días	mar 30/10/07	jue 01/11/07	
Implementació BBDD	9 días	jue 01/11/07	sáb 10/11/07	17
Càrrega inicial de dades	1 día	sáb 10/11/07	dom 11/11/07	18
Implementació del prototipus	**25 días**	**dom 11/11/07**	**jue 06/12/07**	
Implementació classes d'entitat	25 días	dom 11/11/07	jue 06/12/07	19
Implementació classes DAO	25 días	dom 11/11/07	jue 06/12/07	19
Implementació presentació	25 días	dom 11/11/07	jue 06/12/07	19
Proves i qualitat	10 días	jue 06/12/07	dom 16/12/07	20
Documentació	10 días	jue 06/12/07	dom 16/12/07	20
Revisió final	1 día	dom 16/12/07	lun 17/12/07	24;25
Lliurament PAC 3	0 días	lun 17/12/07	lun 17/12/07	
Fase 4: Entrega final	**27 días**	**mar 18/12/07**	**lun 14/01/08**	
Càrrega de dades	1 día	mar 18/12/07	mié 19/12/07	
Implementació final	**10 días**	**mié 19/12/07**	**sáb 29/12/07**	
Implementació classes d'entitat	10 días	mié 19/12/07	sáb 29/12/07	29
Implementació classes DAO	10 días	mié 19/12/07	sáb 29/12/07	29
Implementació presentació	10 días	mié 19/12/07	sáb 29/12/07	29
Proves i qualitat	5 días	sáb 29/12/07	jue 03/01/08	30
Redacció memòria	10 días	jue 03/01/08	dom 13/01/08	34
Revisió final	1 día	dom 13/01/08	lun 14/01/08	35
Lliurament memòria	0 días	lun 14/01/08	lun 14/01/08	

Diagrama de Gantt amb la planificació del projecte

Altres capítols

Es detallen a continuació la resta de capítols del present treball:

Anàlisi

En aquest capítol es realitzen les etapes que al cicle de vida del programari s'identifiquen com la presa de requeriments i anàlisi de l'aplicació. Així, en primer lloc s'exposen detalladament els requeriments funcionals i les característiques tecnològiques del producte resultant. A continuació s'identifiquen els rols de l'aplicació, es detallen els casos d'ús per als rols identificats i es mostra a grans trets la interfície gràfica d'usuari que s'espera. Finalment es fan unes breus consideracions sobre les decisions preses.

Disseny

Aquest capítol exposa el disseny de l'aplicació en funció del paradigma de l'arquitectura a tres capes. Es parteix d'una descripció general de les decisions arquitectòniques, on es detalla el paper dels bastiments utilitzats i el catàleg de patrons utilitzats. A continuació es detallen els diversos aspectes de cadascuna del tres capes. En la capa de dades s'exposa el model E-R utilitzat i la seva interacció amb Hibernate. En la capa de negoci es detallen les classes d'entitat i de sessió que la conformen mitjançant descripcions textuals i diagrames UML estàtics. Finalment, en la capa de presentació s'esmenten les tecnologies utilitzades així com la utilització de Struts per la interrelació amb la capa de negoci.

Conclusions

Finalment, considerant els objectius del projecte, s'exposaran les conclusions resultants tant dels dos capítols anteriors com de la fase de implementació del model resultant de l'anàlisi i disseny prèviament exposats.

Anàlisi

Requeriments funcionals

L'aplicació resultant de la implementació del present projecte ha de complir les funcionalitats bàsiques associades al tipus de programari conegut popularment com a *botiga virtual*. Una botiga virtual consisteix en un programari resident en un servidor connectat a Internet, el qual atén les peticions de diversos clients amb la finalitat d'efectuar les transaccions comercials habituals en el comerç minorista. Aquests clients accediran al programari mitjançant un navegador web estàndard, sense que sigui necessària la instal·lació i configuració de cap programari desenvolupat específicament a efectes de comunicar-se amb el servidor a les seves màquines.

Els clients connectats d'aquesta manera podran realitzar les accions associades a la venda en detall, com cercar productes dins del el catàleg de la botiga ofereix, encarregar la compra d'aquests productes i efectuar els pagaments corresponents. La forma habitual d'efectuar aquests pagaments serà mitjançant una tarja de crèdit en un servidor d'una entitat financera aliena a l'organització propietària de la botiga virtual, per la qual cosa cal simular eficientment aquesta interacció de tal manera que l'experiència d'usuari resulti el més versemblant possible.

Per motius de seguretat, es requereix que cada client s'enregistri degudament al sistema abans de confirmar en ferm les comandes i efectuar els pagaments, per tal de disposar de les dades bàsiques de contacte en cas d'errors de funcionament. Un cop enregistrar, el client podrà modificar aquestes dades de contacte segons la seva voluntat. A més, donat que la botiga virtual estarà orientada principalment al comerç de productes físics, es demana que cada comanda s'adreci a una persona i adreça concretes, que poden coincidir o no amb les del client que encarrega la comanda.

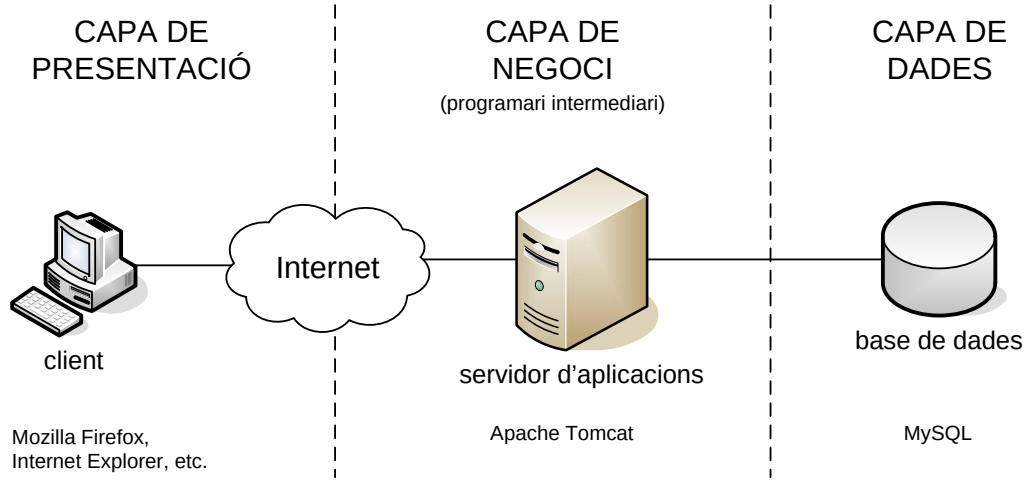
Es requereix també la utilització de les metàfores convencionals en aquest tipus de programari, com poden ser el *compte de client*, que permet que el sistema reconegui al client un cop s'ha enregistrat i li adreci salutacions personalitzades, o el *carret de la compra virtual*, que permet emmagatzemat els productes seleccionats per a la seva compra durant una sessió de treball de l'usuari.

La botiga virtual disposarà d'una part privada o *back-end*, la qual només serà accessible al personal de l'organització propietària degudament identificat com a tal. Mitjançant aquesta part del programari es realitzaran les tasques habituals de manteniment i actualització de la botiga virtual, com la creació de nous productes i categories d'aquests, la seva modificació, el seguiment de les comandes, etc. Respecte a aquest darrer punt, es considera que una comanda s'inicia en el moment de la seva creació per un client, es considera cobrada quan es rep la confirmació del pagament per l'entitat financera, i a partit d'aquet moment es pot procedir al seu enviament físic. Una comanda pot ser anul·lada en qualsevol moment del procés pels motius que es considerin convenients.

Finalment, per tal de mostrar els avantatges del comerç electrònic tant per als compradors com per als venedors, es demana que el mateix programari s'interrelacioni amb altres programaris de gestió de la organització. El cas escollit és el control dels estocs, de tal manera que a l'efectuar una comanda s'actualitzarà automàticament el inventari de productes i els clients no podran encarregar productes exhaurits.

Requeriments tecnològics

Tal com és habitual en el desenvolupament d'aplicacions empresarials orientades a Internet, es seguirà el paradigma client-servidor mitjançant una arquitectura a tres capes, que són la capa de dades, la capa de lògica de negoci i la capa de presentació. Cadascuna d'aquestes capes restarà modularitzada e independent de la resta, amb la finalitat d'escalar cadascuna d'elles de forma independent segons apareguin nous requeriments o canvis tecnològics.



Representació conceptual de l'arquitectura a tres capes

Els clients utilitzaran un navegador web estàndard amb capacitat per visualitzar documents HTML per la connexió amb el servidor d'aplicacions a través de Internet, mitjançant el protocol HTTP. Així, no caldrà dissenyar e implementar cap programari per aquesta tasca, ni instal·lar-ho a cadascun dels clients, amb els avantatges operacionals que es desprenen.

En la capa de negoci o programari intermediari (*middleware*) residirà l'aplicació pròpiament dita, que s'executarà en un servidor d'aplicacions. Segons les especificacions de la plataforma J2EE, aquest servidor disposarà d'un *contenidor web* per interactuar amb les peticions dels clients, comunicar-les a l'aplicació i retornar les dades en forma de documents HTML comprensibles pels seus navegadors.

La missió del contenidor consisteix en abstraure el desenvolupament de tasques comunes i repetitives de baix nivell, com l'obertura i tancament de *sockets*, la identificació del mateix client entre diverses peticions (recordem que HTTP és un protocol sense estat), qüestions relatives a la seguretat, etc. Els objectes de la plataforma J2EE, com els *Servlets* i JSPs, s'executen dins aquest contenidor, el qual crea automàticament i proporciona accés a altres objectes que permetran una interactuació efectiva amb el client, com *Request*, que encapsula la petició del client, *Response*, que representa el flux de dades de sortida, o *Session*, que abstrau el concepte de sessió de treball de manera transparent per al desenvolupador.

El servidor d'aplicacions escollit és Apache Tomcat, el qual no és un servidor J2EE complert donat que no disposa de contenidor EJB, però la funcionalitat del contenidor web és suficient per als requeriments de l'aplicació.

En la capa de dades s'utilitzarà el sistema gestor de bases de dades relacional MySQL, donat que és una bona implementació de l'estàndard SQL-92 i disposa de *drivers* JDBC nadius.

S'utilitzarà el bastiment Struts per resoldre les interaccions entre la lògica de negoci i la presentació en planes JSP, e Hibernate pel mapeig de les taules de la base de dades amb les classes de negoci. Tots dos bastiments actuen dins el mateix contenidor web de l'aplicació.

Rols

En funció dels requeriments funcionals es desprenen els següents rols en l'aplicació:

Administrador

Identifiquem amb aquest rol al propietari de la botiga virtual o bé un gestor pertanyent a l'organització propietària d'aquesta, amb plena capacitat per accedir a la part privada o *back-end* i realitzar les tasques pròpies d'aquesta zona.

Client

Correspon a un usuari de Internet extern a l'organització degudament enregistrat al sistema, un cop creat el seu compte de client, amb capacitat per encarregar comandes, efectuar pagaments i realitzar les tasques d'actualització del seu compte.

Usuari

Correspon a un usuari de Internet anònim per al sistema, i per tant sense les capacitats corresponents a un Client, tot i que podrà utilitzar les funcionalitats bàsiques de la botiga virtual, com visitar el catàleg de productes, efectuar recerques, consultar els productes en detall i utilitzar el carret de la compra virtual.

Casos d'ús

Es mostren a continuació els diferents casos d'ús que s'identifiquen als requeriments. Considerem convenient dividir aquests entre els casos d'ús corresponents a Usuari i Client, donat que el darrer és una especialització del primer, i els casos d'ús corresponents a l'Administrador, arran de l'estricta separació existent entre les parts pública i privada de la botiga virtual.

Casos d'ús per Usuari i Client

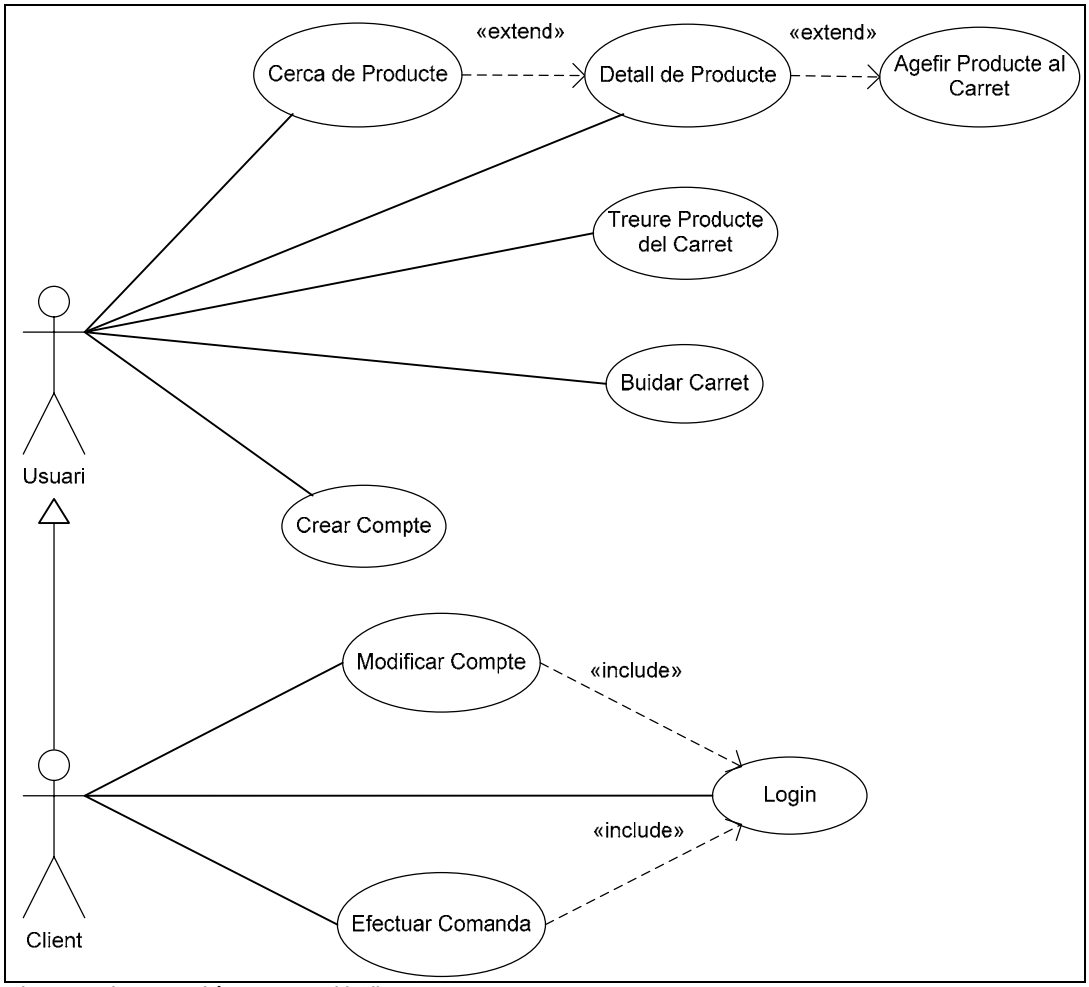


Diagrama de casos d'ús per Usuari i Client

Cerca de Producte

Resum de la funcionalitat: l'Usuari cerca un Producte seguint diversos criteris.
Paper dins del rol de l'actor: necessari per tal d'efectuar l'acció comercial sobre el producte a cercar.
Casos d'ús relacionats: Detall de Producte.
Precondició: l'Usuari ha d'estar connectat al sistema.
Postcondició: es mostra un llistat dels Productes que compleixin els criteris de cerca.
Descripció: l'Usuari cerca un producte seguint diversos criteris: rang de preus, navegació per Categories, característiques determinades, nom del fabricant o de la marca, etc. Al final de procés s'obté una llista de Productes que compleixin amb els requeriments introduïts.

Detall de Producte

Resum de la funcionalitat: es mostren totes les dades públiques d'un Producte.
Paper dins del rol de l'actor: necessari per tal de permetre la comercialització del Producte.
Casos d'ús relacionats: Cerca de Producte, Afegir Producte al Carret.
Precondició: el Producte ha d'existir a la base de dades i ha d'haver estat prèviament cercat o bé directament enllaçat des de l'exterior.
Postcondició: es mostren totes les dades públiques d'un Producte.
Descripció: un cop cercat el Producte, es mostren totes les dades d'aquest i s'ofereix l'opció d'afegir-lo al Carret de la compra. Es podrà accedir a aquesta funcionalitat bé des d'una cerca de Producte, o bé des de l'exterior de l'aplicació mitjançant un enllaç que contempli l'identificador del producte.

Afegir Producte al Carret

Resum de la funcionalitat: s'afegeix un Producte al Carret de la compra.
Paper dins del rol de l'actor: necessari per tal de permetre la comercialització del Producte.
Casos d'ús relacionats: Detall de Producte.
Precondició: el Producte ha d'existir a la base de dades, ha d'haver estat prèviament mostrat en la seva pantalla de detall, i ha d'existir estoc disponible.
Postcondició: el Producte està afegit al Carret de la compra.
Descripció: un cop mostrades les dades del Producte, l'usuari el pot afegir al carret de la compra, on es mostrarà aquest i s'afegirà el seu import al total del Carret. Es podrà seleccionar la quantitat que es demana del producte i, si s'escau, les diferents varietats existents. Els productes afegits al carret romandran actius durant la sessió de treball de l'Usuari.

Treure Producte del Carret

Resum de la funcionalitat: s'elimina un producte del Carret de la compra.
Paper dins del rol de l'actor: ús opcional per corregir errors de l'Usuari.
Casos d'ús relacionats: Cap.
Precondició: el Producte ha d'haver estat afegit al Carret de la compra.
Postcondició: el Producte està eliminat del Carret de la compra.
Descripció: des de la pantalla de consulta del Carret de la compra es podrà eliminat qualsevol dels Productes prèviament afegits abans de confirmar la comanda. Aquesta acció comportarà un nou càlcul del total de la compra.

Buidar Carret

Resum de la funcionalitat: es buida el contingut del Carret de la compra.
Paper dins del rol de l'actor: ús opcional per corregir errors de l'Usuari.
Casos d'ús relacionats: Cap.
Precondició: l'Usuari ha d'estar connectat al sistema .
Postcondició: el Carret de la compra està totalment buidat i el total del import és 0.
Descripció: des de la pantalla de consulta del Carret de la compra es podrà buidar completament aquest. Aquesta acció comportarà el buidat de tots els productes introduïts que haguessin en aquell moment i la posada a zero del total del import de la compra. No caldrà que existeixi un producte actiu en el carret en el moment d'efectuar aquesta acció.

Crear Compte

Resum de la funcionalitat: l'Usuari crea un compte de Client.
Paper dins del rol de l'actor: necessari per tal de dur a terme l'acció comercial en el rol de Client.
Casos d'ús relacionats: Cap.
Precondició: el Client no existeix a la base de dades.
Postcondició: el Client està enregistrat a la base de dades.
Descripció: l'Usuari podrà convertir-se en Client en qualsevol moment abans de confirmar una comanda. L'aplicació demanarà les dades personals i les enregistrarà permanentment a la base de dades. Es proporcionarà a l'Usuari un nom d'usuari i una contrasenya amb les quals podrà entrar com a Client en el sistema en el futur. Un cop finalitzat el procés de creació del Client, s'efectuarà un Login automàticament sense que calgui cap intervenció directa de l'usuari.

Modificar Compte

Resum de la funcionalitat: El Client modifica les dades del seu compte.
Paper dins del rol de l'actor: secundari per l'acció comercial, necessari per tal de disposar de dades actualitzades.
Casos d'ús relacionats: Login.
Precondició: el Client existeix a la base de dades i està correctament enregistrat a l'aplicació.
Postcondició: el Client ha modificat les seves dades i els canvis s'han enregistrat permanentment a la base de dades.
Descripció: un cop dins de l'aplicació, un Client que hagi estat correctament identificat com a tal podrà modificar en qualsevol moment les seves dades personals i de tramesa, les relacionades amb les formes de pagament, etc. Es mostraran els formularis corresponents amb les dades actuals i, un cop efectuades les modificacions escaients, aquestes s'enregistraran de manera permanent a la base de dades. No es permetrà al client modificar el seu identificador únic, corresponent al seu NIF.

Efectuar Comanda

Resum de la funcionalitat: El Client confirma la Comanda i abona el import.
Paper dins del rol de l'actor: necessari per tal d'efectuar l'acció comercial.
Casos d'ús relacionats: Login.
Precondició: el Client existeix a la base de dades, està correctament enregistrat a l'aplicació i hi té com a mínim un Producte introduït en el Carret de la compra.
Postcondició: el Client ha efectuat la comanda en línia i el carret de la compra s'ha buidat.
Descripció: un cop examinat el contingut del Carret de la compra, el Client pot introduir les dades de tramesa, confirmar-la, i finalment efectual el pagament en línia mitjançant la seva tarja de crèdit en una entitat financera aliena a la botiga virtual. L'aplicació emmagatzemarà les dades de la Comanda en funció de les dades del Carret i del Client per tal de permetre el seu seguiment i comunicarà a l'administrador l'existència de la nova comanda per tal d'efectuar la tramesa física. A més, es reduiran els estocs dels productes comprats en les quantitats seleccionades de forma automatitzada. El Carret de la compra es buidarà i es retornarà a l'aplicació un cop s'hagi efectuat el pagament

Login

Resum de la funcionalitat: el Client s'enregistra al sistema per tal que aquest el reconegui com a tal i pugui realitzar les accions només reservades a aquest.
Paper dins del rol de l'actor: necessari per tal d'identificar al Client i, per tant, efectuar l'acció comercial.
Casos d'ús relacionats: Modificar Compte, Efectuar Comanda.
Precondició: el Client ha de disposar de les seves dades de connexió.
Postcondició: el Client està correctament enregistrat al sistema.
Descripció: mitjançant l'habitual ús d'un nom d'usuari i d'una contrasenya, el Client s'enregistrarà al sistema per tal que aquest el reconegui com a tal. D'aquesta manera, l'aplicació ja coneixerà les dades de pagament en cas de que s'efectuï alguna comanda, podrà rebre recomanacions, modificar les seves dades, etc. El procés serà automàtic si un Usuari no estava enregistrat prèviament i crea un compte.

Casos d'ús per Administrador

Els casos d'ús per Administrador es corresponen amb les típiques accions CRUD (*Create-Read-Update-Delete*) de les entitats prèviament esmentades sobre les que s'esperen aquest tipus d'accions de manteniment, que són Categoria, Producte, Comanda i Client. Per tant, amb la finalitat de facilitar la legibilitat dels diagrames, dividirem aquests casos d'ús en funció de l'entitat sobre la que actuen, a més de considerar els casos d'ús general comuns a aquesta divisió.

Casos d'ús sobre Categoria

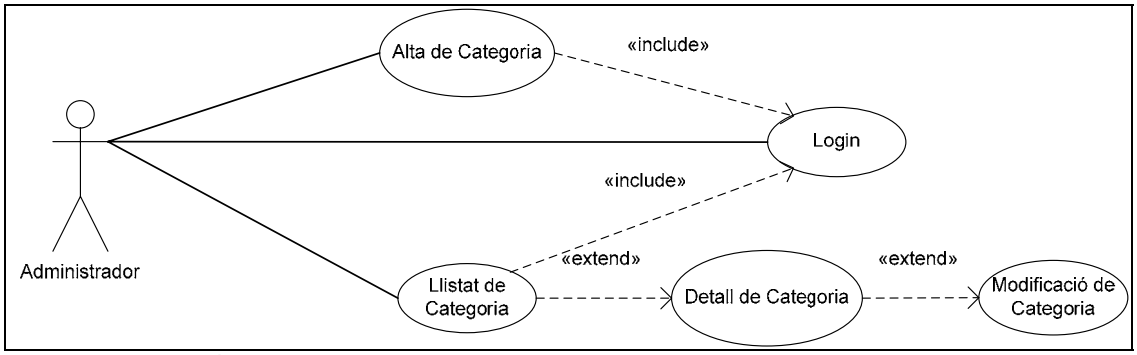


Diagrama de casos d'ús d'Administrador sobre Categoria

Alta de Categoria

Resum de la funcionalitat: l'Administrador dona d'alta una nova Categoria a la base de dades.
Paper dins del rol de l'actor: necessari per tal de disposar de Categories per assignar als Productes a l'aplicació i, per tant, dur a terme el procés de negoci.
Casos d'ús relacionats: Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema.
Postcondició: la Categoria s'ha enregistrat correctament a la base de dades.
Descripció: l'Administrador interactua amb un formulari per tal d'informar a l'aplicació dels diferents valors d'una Categoria. Un cop acabat, aquests valors s'emmagatzemaran a la base de dades, o bé s'informarà de l'error si no ha estat possible.

Llistat de Categoria

Resum de la funcionalitat: l'Administrador llista totes les Categories existents a la base de dades.
Paper dins del rol de l'actor: necessari per tal de seleccionar una Categoria per interactuar posteriorment amb aquesta, o bé seleccionar els productes adscrits amb aquesta Categoria.
Casos d'ús relacionats: Detall de Categoria, Modificació de Categoria, Llistat de Producte, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema i ha d'existir com a mínim una Categoria a la base de dades.
Postcondició: es mostren totes les Categories existents a la base de dades i es poden seleccionar per efectuar altres operacions.
Descripció: l'Administrador llista les Categories enregistrades a la base de dades, sense que calgui efectuar cap recerca donat que es suposa que el nombre de Categories és reduït i fàcilment tractable. Al final de la interactuació les Categories estaran disponibles per tal d'efectuar altres operacions sobre aquestes o bé per llistar els seus productes associats.

Detall de Categoria

Resum de la funcionalitat: l'Administrador consulta una Categoria preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal d'interactuar posteriorment amb la Categoria consultada.
Casos d'ús relacionats: Llistat de Categoria, Modificació de Categoria, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Categoria a consultar ha d'existir a la base de dades i ha estat prèviament localitzada mitjançant la seva aparició a un llistat.
Postcondició: es mostren totes les dades disponibles de la Categoria.
Descripció: un cop localitzada la Categoria segons els criteris disponibles, es mostren totes les dades d'aquesta per tal de permetre la seva consulta en detall i, si s'escau, modificar-ne les que estiguin disponibles per l'edició.

Modificació de Categoria

Resum de la funcionalitat: l'Administrador modifica les dades d'una Categoria preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal de mantenir les dades de les Categories actualitzades.
Casos d'ús relacionats: Detall de Categoria, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Categoria a modificar ha d'existir a la base de dades, ha estat prèviament localitzada mitjançant la seva aparició a un llistat i s'han consultat els seus detalls .
Postcondició: les dades de la Categoria es modifiquen permanentment a la base de dades.
Descripció: es mostrarà un formulari amb els mateixos camps que al Detall de Categoria, amb la possibilitat d'editar-ne els seus valors exceptuant el seu identificador únic. Un cop efectuats els canvis corresponents, aquests s'enregistraran de forma permanent a la base de dades.

Casos d'ús sobre Producte

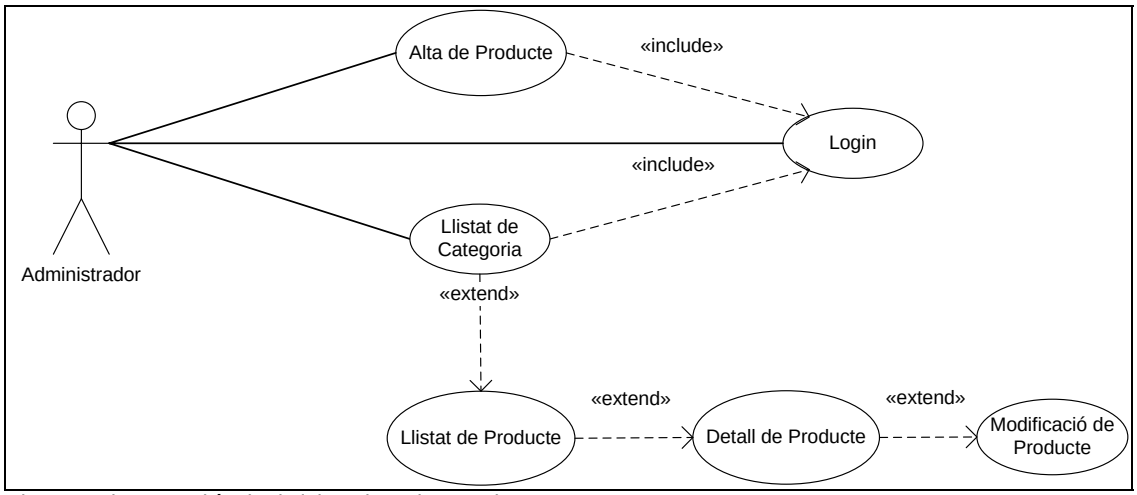


Diagrama de casos d'ús d'Administrador sobre Producte

Alta de Producte

Resum de la funcionalitat: l'Administrador dona d'alta un nou Producte a la base de dades.
Paper dins del rol de l'actor: necessari per tal de disposar de Productes a la base de dades i, per tant, dur a terme el procés de negoci.
Casos d'ús relacionats: Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema.
Postcondició: el Producte s'ha enregistrat correctament a la base de dades.
Descripció: l'Administrador interactua amb un formulari per tal d'informar a l'aplicació dels diferents valors d'un Producte. Un cop acabat, aquests valors s'emmagatzemaran a la base de dades, o bé s'informarà de l'error si no ha estat possible.

Llistat de Producte

Resum de la funcionalitat: l'Administrador llista tots els Productes existents a la base de dades pertanyents a una Categoria determinada.
Paper dins del rol de l'actor: necessari per tal de seleccionar un Producte per interactuar posteriorment amb aquest.
Casos d'ús relacionats: Llistat de Categoria, Detall de Producte, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema i ha d'existir com a mínim un Producte a la base de dades.
Postcondició: es mostren tots els Productes existents a la base de dades que pertanyin a una Categoria prèviament escollida i es poden seleccionar per efectuar altres operacions.
Descripció: l'Administrador llista els Productes enregistrats a la base de dades en funció de la Categoria a la que pertanyen, que ha estat prèviament escollida mitjançant el llistat corresponent. Al final de la interactuació els Productes estaran disponibles per tal d'efectuar altres operacions sobre aquests.

Detall de Producte

Resum de la funcionalitat: l'Administrador consulta un Producte preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal d'interactuar posteriorment amb el Producte consultat.
Casos d'ús relacionats: Llistat de Producte, Modificació de Producte, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, el Producte a consultar ha d'existir a la base de dades i ha estat prèviament localitzat mitjançant la seva aparició a un llistat.
Postcondició: es mostren totes les dades disponibles del Producte.
Descripció: un cop localitzat el Producte segons els criteris disponibles, es mostren totes les dades d'aquest per tal de permetre la seva consulta en detall i, si s'escau, modificar-ne les que estiguin disponibles per l'edició.

Modificació de Producte

Resum de la funcionalitat: l'Administrador modifica les dades d'un Producte preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal de mantenir les dades dels Producte actualitzades.
Casos d'ús relacionats: Detall de Producte, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, el Producte a modificar ha d'existir a la base de dades, ha estat prèviament localitzat mitjançant la seva aparició a un llistat i s'han consultat els seus detalls .
Postcondició: les dades del Producte es modifiquen permanentment a la base de dades.
Descripció: es mostrarà un formulari amb els mateixos camps que al Detall de Producte, amb la possibilitat d'editar-ne els seus valors exceptuant el seu identificador únic. Un cop efectuats els canvis corresponents, aquests s'enregistraran de forma permanent a la base de dades.

Casos d'ús sobre Comanda

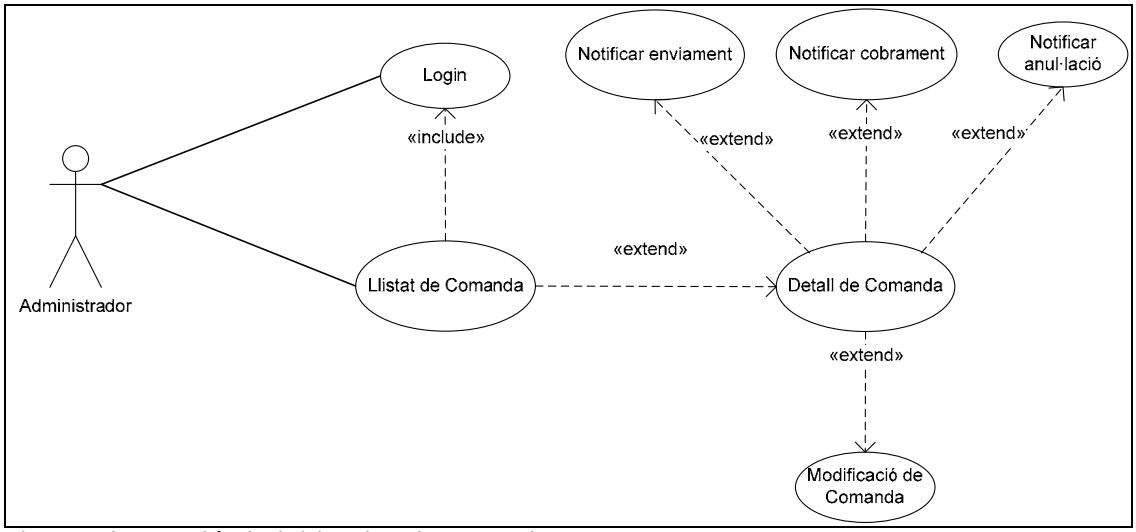


Diagrama de casos d'ús d'Administrador sobre Comanda

Llistat de Comanda

Resum de la funcionalitat: l'Administrador llista totes les Comandes existents a la base de dades que estiguin en un estat determinat.
Paper dins del rol de l'actor: necessari per tal de seleccionar una Comanda per interactuar posteriorment amb aquesta.
Casos d'ús relacionats: Detall de Comanda, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema i ha d'existir com a mínim una Comanda a la base de dades.
Postcondició: es mostren totes les Comandes Productes existents a la base de dades que estiguin en un estat determinat prèviament escollit i es poden seleccionar per efectuar altres operacions.
Descripció: l'Administrador llista les Comandes enregistrades a la base de dades en funció de l'estat en el que es trobin, que ha estat prèviament escollit. Al final de la interactuació les Comandes estaran disponibles per tal d'efectuar altres operacions sobre aquestes.

Detall de Comanda

Resum de la funcionalitat: l'Administrador consulta una Comanda preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal d'interactuar posteriorment amb la Comanda consultada.
Casos d'ús relacionats: Llistat de Comanda, Notificar enviament, Notificar cobrament, Notificar anul·lació, Modificació de Comanda, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Comanda a consultar ha d'existir a la base de dades i ha estat prèviament localitzada mitjançant la seva aparició a un llistat.
Postcondició: es mostren totes les dades disponibles de la Comanda.
Descripció: un cop localitzada la Comanda segons els criteris disponibles, es mostren totes les dades d'aquesta per tal de permetre la seva consulta en detall i, si s'escau, modificar-ne les que estiguin disponibles per l'edició.

Modificació de Comanda

Resum de la funcionalitat: l'Administrador modifica les dades d'una Comanda preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal de mantenir les dades de les Comandes actualitzades.
Casos d'ús relacionats: Detall de Comanda, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Comanda a modificar ha d'existir a la base de dades, ha estat prèviament localitzada mitjançant la seva aparició a un llistat i s'han consultat els seus detalls .
Postcondició: les dades de la Comanda es modifiquen permanentment a la base de dades.
Descripció: es mostrarà un formulari amb els mateixos camps que al Detall de Comanda, amb la possibilitat d'editar-ne els seus valors exceptuant el seu identificador únic. Un cop efectuats els canvis corresponents, aquests s'enregistraran de forma permanent a la base de dades.

Notificar cobrament

Resum de la funcionalitat: l'Administrador modifica les dades d'una Comanda preexistent a la base de dades per tal de canviar-ne el seu estat a "cobrada".
Paper dins del rol de l'actor: necessari per tal d'efectuar el seguiment de les Comandes.
Casos d'ús relacionats: Detall de Comanda, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Comanda a modificar ha d'existir a la base de dades, ha estat prèviament localitzada mitjançant la seva aparició a un llistat i ha d'estar en estat "iniciada" .
Postcondició: la Comanda passa a estar en estat "cobrada" de forma permanent a la base de dades.
Descripció: es mostrarà un llistat de les comandes en estat "iniciada", del qual es podrà seleccionar una Comanda per tal de notificar el seu cobrament, el qual ha estat comunicat a l'Administrador de forma aliena al sistema. Un cop efectuat el canvi d'estat, la Comanda passarà a tenir l'estat "cobrada" de forma permanent a la base de dades.

Notificar enviament

Resum de la funcionalitat: l'Administrador modifica les dades d'una Comanda preexistent a la base de dades per tal de canviar-ne el seu estat a "enviada".
Paper dins del rol de l'actor: necessari per tal d'efectuar el seguiment de les Comandes.
Casos d'ús relacionats: Detall de Comanda, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Comanda a modificar ha d'existir a la base de dades, ha estat prèviament localitzada mitjançant la seva aparició a un llistat i ha d'estar en estat "cobrada" .
Postcondició: la Comanda passa a estar en estat "enviada" de forma permanent a la base de dades.
Descripció: es mostrarà un llistat de les comandes en estat "cobrada", del qual es podrà seleccionar una Comanda per tal de notificar el seu enviament, el qual ha estat comunicat a l'Administrador de forma aliena al sistema. Un cop efectuat el canvi d'estat, la Comanda passarà a tenir l'estat "enviada" de forma permanent a la base de dades.

Notificar anul·lació

Resum de la funcionalitat: l'Administrador modifica les dades d'una Comanda preexistent a la base de dades per tal de canviar-ne el seu estat a "anul·lada".
Paper dins del rol de l'actor: necessari per tal d'efectuar el seguiment de les Comandes.
Casos d'ús relacionats: Detall de Comanda, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, la Comanda a modificar ha d'existir a la base de dades, ha estat prèviament localitzada mitjançant la seva aparició a un llistat i ha d'estar en qualsevol estat a excepció d'"anul·lada" .
Postcondició: la Comanda passa a estar en estat "anul·lada" de forma permanent a la base de dades.
Descripció: es mostrarà un llistat de les comandes en qualsevol estat a excepció d'"anul·lada", del qual es podrà seleccionar una Comanda per tal de notificar la seva anul·lació, la qual ha estat comunicat a l'Administrador de forma aliena al sistema. Un cop efectuat el canvi d'estat, la Comanda passarà a tenir l'estat "anul·lada" de forma permanent a la base de dades.

Casos d'ús sobre Client

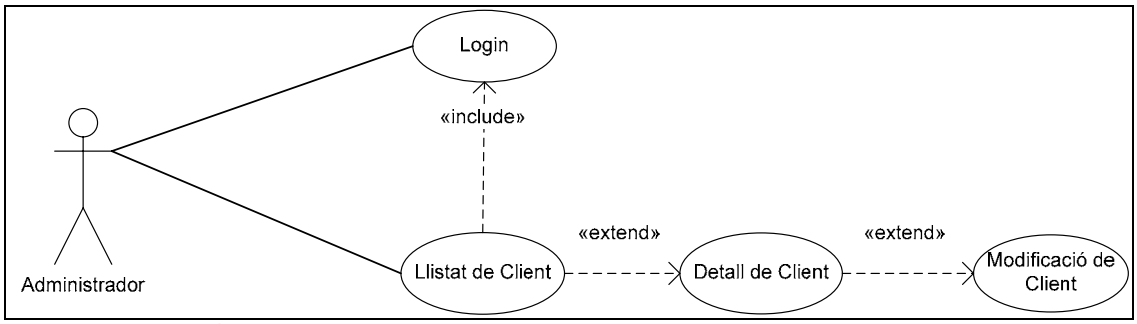


Diagrama de casos d'ús d'Administrador sobre Client

Llistat de Client

Resum de la funcionalitat: l'Administrador llista tots els Clients existents a la base de dades.
Paper dins del rol de l'actor: necessari per tal de seleccionar un Client per interactuar posteriorment amb aquest.
Casos d'ús relacionats: Detall de Client, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema i ha d'existir com a mínim un Client a la base de dades.
Postcondició: es mostren tot els Clients existents a la base de dades i es poden seleccionar per efectuar altres operacions.
Descripció: l'Administrador llista els Clients enregistrada a la base de dades, sense que calgui efectuar cap recerca donat que es suposa que el nombre de Clients és reduït i fàcilment tractable. Al final de la interactuació els Clients estaran disponibles per tal d'efectuar altres operacions sobre aquests.

Detall de Client

Resum de la funcionalitat: l'Administrador consulta un Client preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal d'interactuar posteriorment amb el Client consultat.
Casos d'ús relacionats: Llistat de Client, Modificació de Client, Login.
Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, el Client a consultar ha d'existir a la base de dades i ha estat prèviament localitzat mitjançant la seva aparició en un llistat.
Postcondició: es mostren totes les dades disponibles del Client.
Descripció: un cop localitzat el Client segons els criteris disponibles, es mostren totes les dades d'aquest per tal de permetre la seva consulta en detall i, si s'escau, modificar-ne les que estiguin disponibles per l'edició.

Modificació de Client

Resum de la funcionalitat: l'Administrador modifica les dades d'un Client preexistent a la base de dades.
Paper dins del rol de l'actor: necessari per tal de mantenir les dades dels Clients actualitzades, secundari en el sentit que la mateixa acció pot ser realitzada pels mateixos Clients a la part pública (veieu el cas d'ús Modificar compte).
Casos d'ús relacionats: Detall de Client, Login.

Precondició: l'Administrador ha d'estar correctament enregistrat al sistema, el Client a modificar ha d'existir a la base de dades, ha estat prèviament localitzat mitjançant la seva aparició a un llistat i s'han consultat els seus detalls .
Postcondició: les dades del Client es modifiquen permanentment a la base de dades.
Descripció: es mostrarà un formulari amb els mateixos camps que al Detall de Client, amb la possibilitat d'editar-ne els seus valors exceptuant el seu identificador únic. Un cop efectuats els canvis corresponents, aquests s'enregistraran de forma permanent a la base de dades.

Casos d'ús general d'Administrador

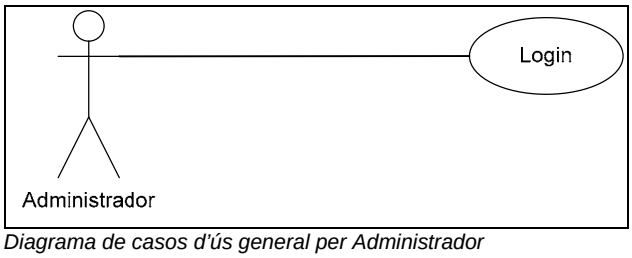


Diagrama de casos d'ús general per Administrador

Login

Resum de la funcionalitat: l'Administrador s'enregistra al sistema per tal d'iniciar una sessió de treball en la part privada o *back-end* de la botiga virtual.
Paper dins del rol de l'actor: imprescindible per dur a terme qualsevol operació.
Casos d'ús relacionats: tots els altres casos d'ús d'Administrador.
Precondició: l'Administrador ha de disposar de les seves dades de connexió.
Postcondició: l'Administrador està correctament enregistrat al sistema.

Interfície Gràfica d'Usuari

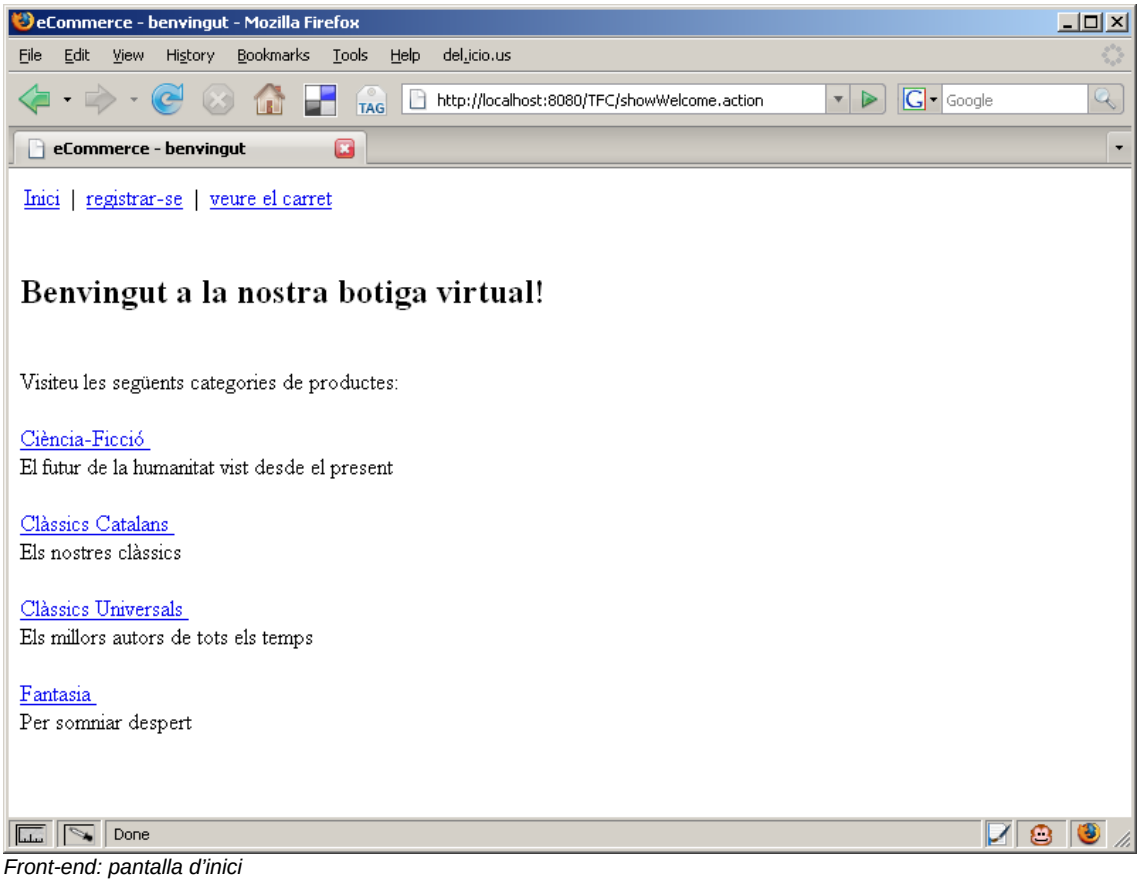
Tal com s'ha comentat en la definició d'objectius del projecte, no es contempla la elaboració d'una interfície d'usuari complerta que mostri l'estat de l'art del desenvolupament d'aplicacions web en aquest aspecte. Cal considerar que aquest camp ha experimentat una sèrie de innovacions en els darrers anys, com l'acceptació universal dels processos d'estandardització impulsats per l'W3C, l'aparició de la web semàntica o la gran importància que han assolit les qüestions relatives a accessibilitat i usabilitat. Així, donat que aquests aspectes constituirien en si mateixos un àmbit d'estudi força ampli, excedirien l'abast del present treball.

Per tant, la interfície gràfica d'usuari serà prou bàsica i només disposarà dels elements imprescindibles per acomplir les funcionalitats requerides. Aquesta interfície gràfica es generarà en planes JSP, mitjançant els elements que Struts posa a la disposició dels desenvolupadors per generar els elements d'aquesta i així avaluar les seves possibilitats. El servidor d'aplicacions traduirà el codi de les planes JSP en documents d'hipertext comprensibles per al navegador de l'usuari. No es contempla la inclusió de plantilles temàtiques, CSS (*Cascade Style Sheets*) ni cap element destinat a la estètica dels documents.

Es mostren a continuació algunes de les pantalles més representatives de les funcionalitats de l'aplicació. Per raons d'espai no es mostren totes i cadascuna de les possibles pantalles, però s'incideix en els aspectes de la interfície gràfica d'usuari que cal destacar.

Part pública o front-end

Pantalla d'inici

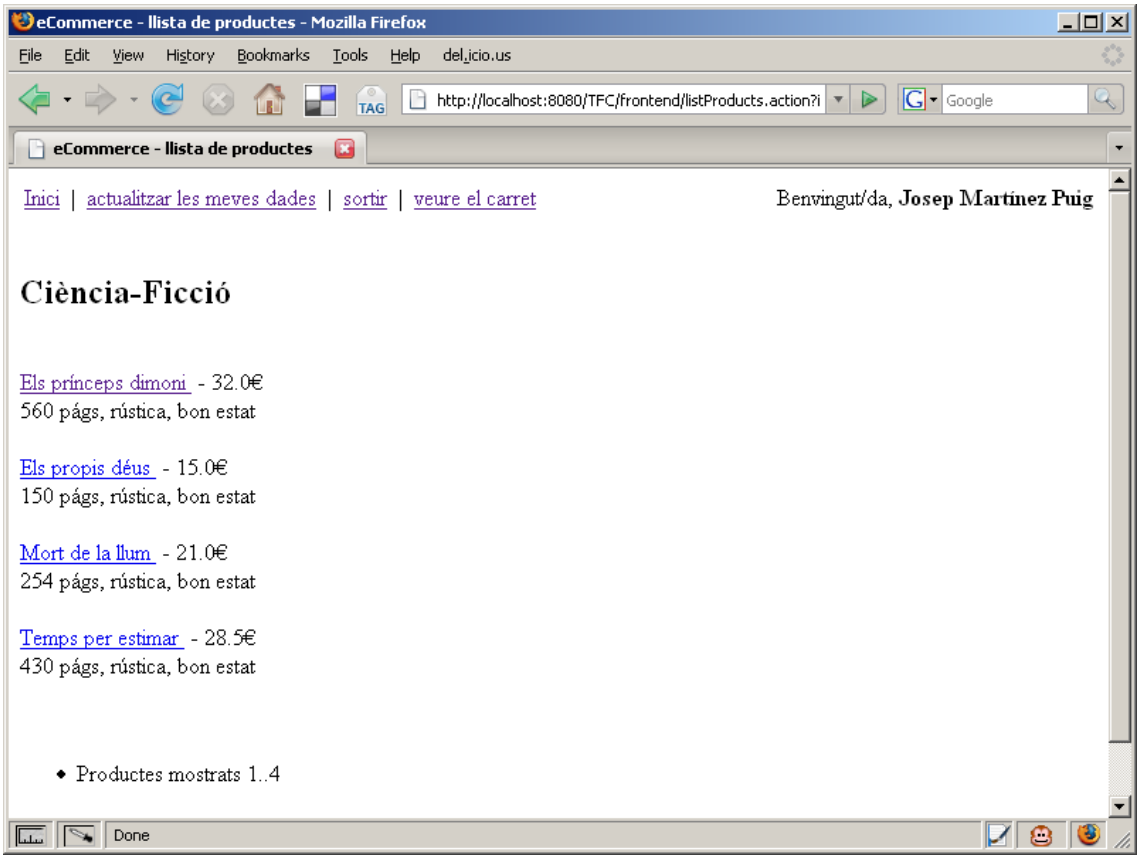


Front-end: pantalla d'inici

S'observa a la pantalla d'inici una barra de navegació superior, com és habitual en aquest tipus de programari, i que per tant cal suposar que l'usuari ja està acostumat al seu ús. En aquesta barra superior es contenen els enllaços cap a la mateixa plana inicial, al formulari de creació de compte de client i a la revisió del carret de la compra.

La resta de la pantalla ofereix un llistat de les categories de productes existents per tal de permetre el examinar els productes associats.

Continguts de categoria

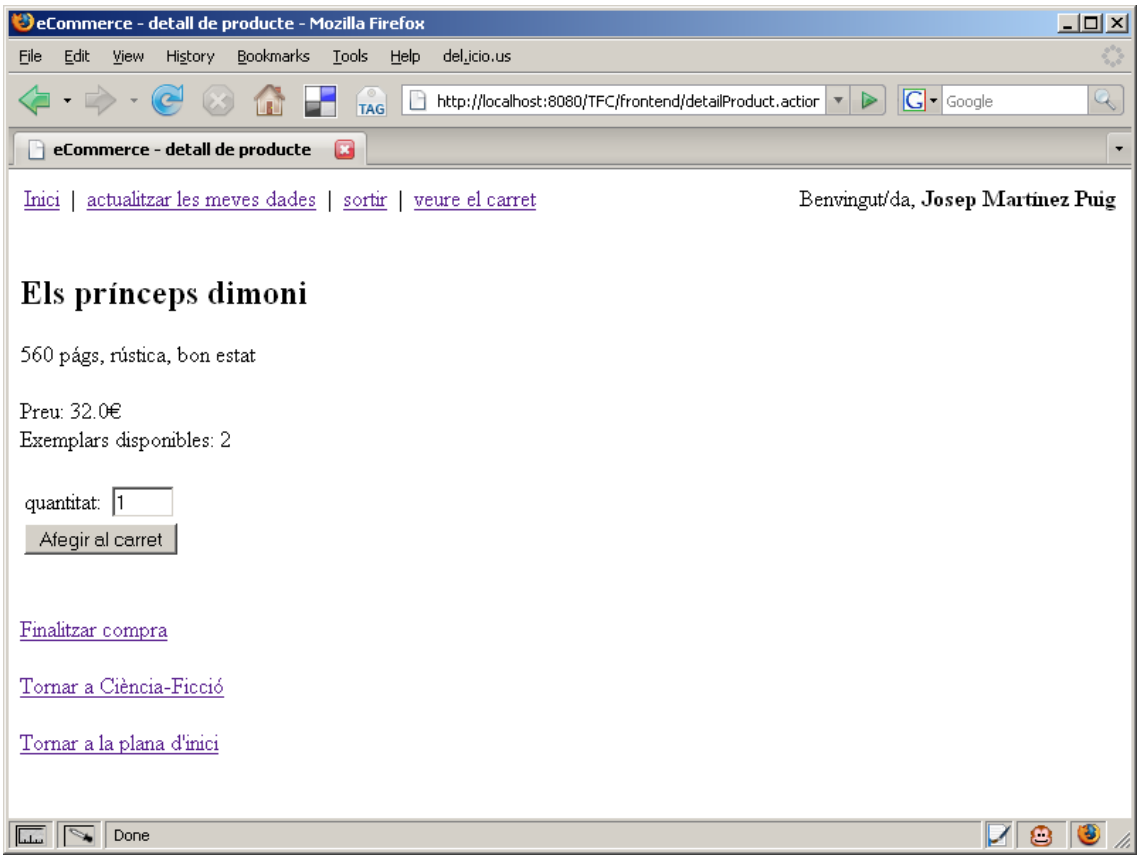


Front-end: pantalla de continguts de categoria

En aquesta pantalla es mostren els productes que pertanyen a una categoria seleccionada prèviament.

S'observa que l'usuari s'ha enregistrat correctament al sistema, i la barra de navegació superior ha modificat els seus continguts per tal d'adaptar-se a aquest fet. Així, en comptes de registrar-se, a l'usuari se li ofereix la possibilitat d'actualitzar les seves dades i de desconnectar-se del sistema. A més, a la part dreta apareix una salutació personalitzada.

Detall de producte

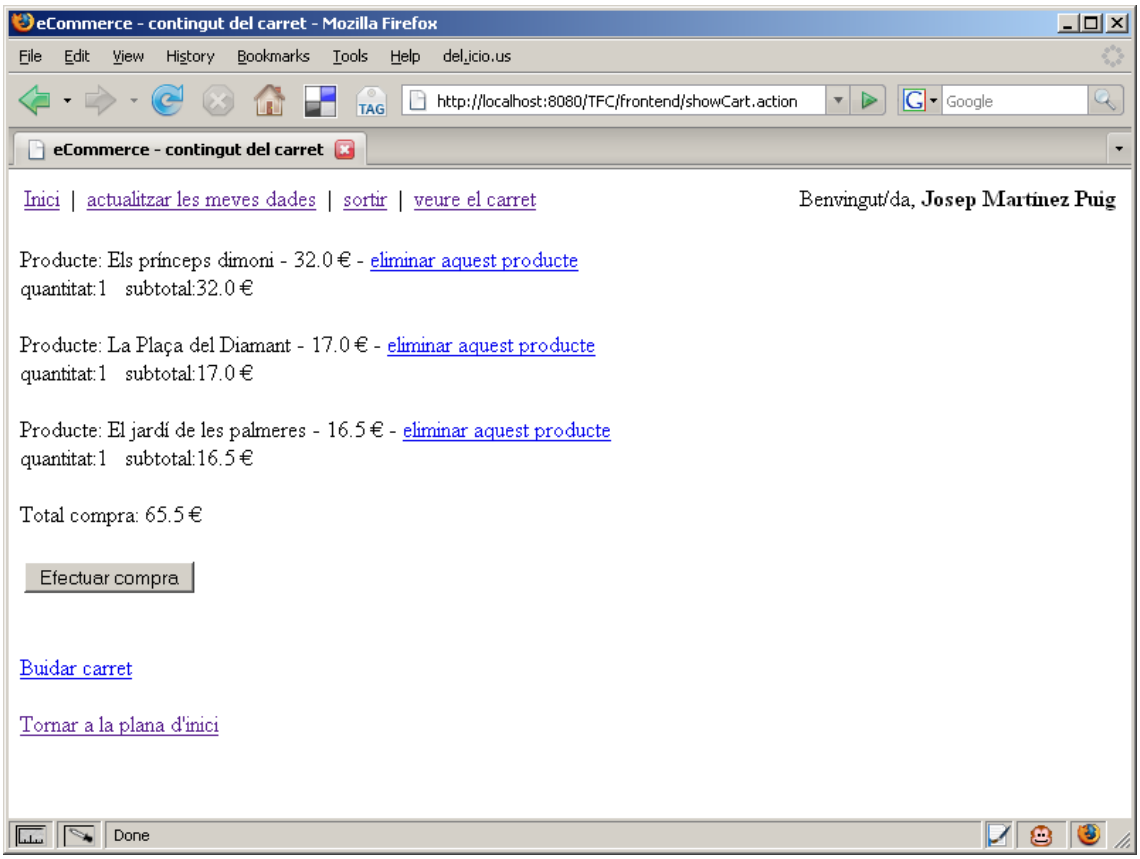


Front-end: pantalla de detall de producte

En aquesta pantalla es detalla el producte prèviament seleccionat i es mostren totes les dades disponibles d'aquest, com la seva descripció detallada, el preu i la quantitat disponible d'estoc en el moment actual.

S'ofereix a l'usuari la possibilitat d'afegir aquest producte al carret de la compra virtual, indicant la quantitat demanada. També apareixen els enllaços corresponents per tornar a la categoria de productes prèviament visitada i a la plana d'inici.

Continguts del carret

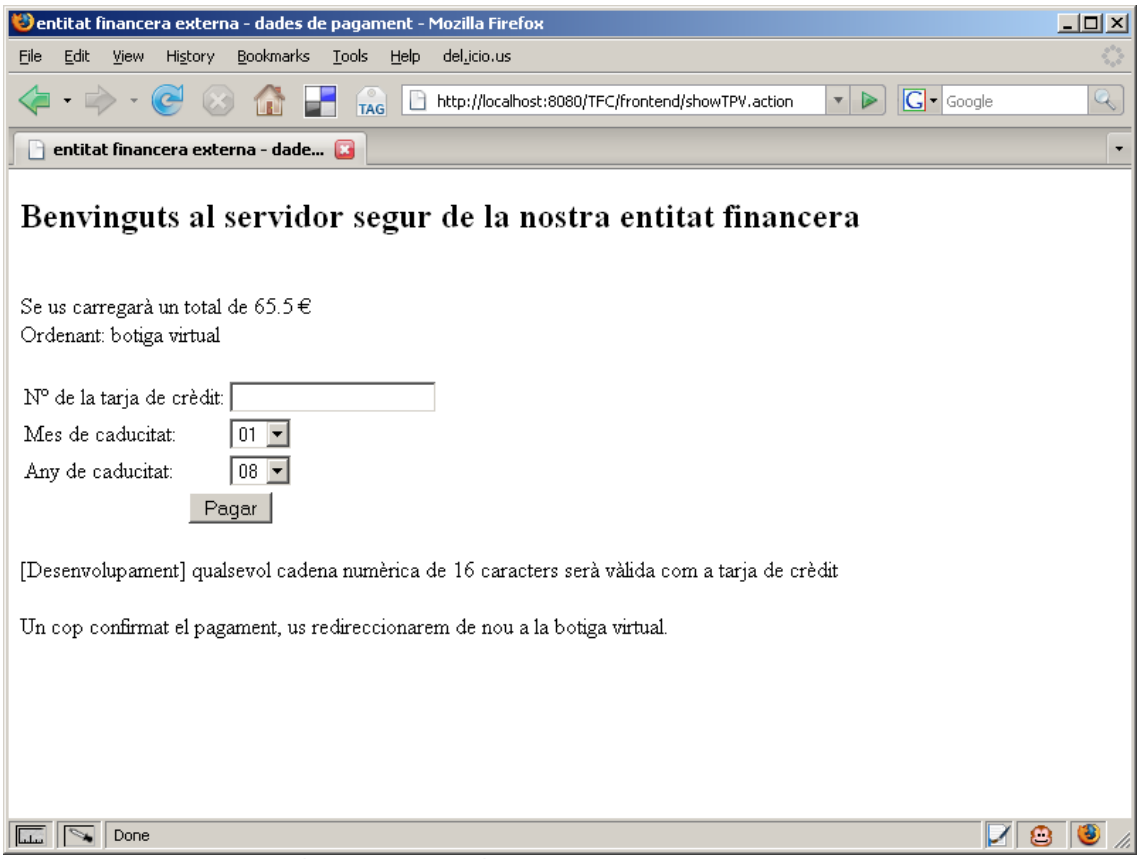


Front-end: pantalla de continguts del carret

En aquesta pantalla es detallen els productes introduïts al carret de la compra virtual fins al moment present, juntament amb les seves quantitats i els subtotals corresponents. Per cada producte existeix la possibilitat de treure'l del carret. Al final del llistat es mostra el total del import de la compra que hauria d'abonar el client en cas de continuar el procés comercial.

El procés de la compra seguiria amb l'accionament del botó "efectuar compra", el qual portaria a les pantalles de introducció de dades de tramesa, a la confirmació d'aquesta i finalment a la simulació del pagament virtual.

Simulació de pagament en línia



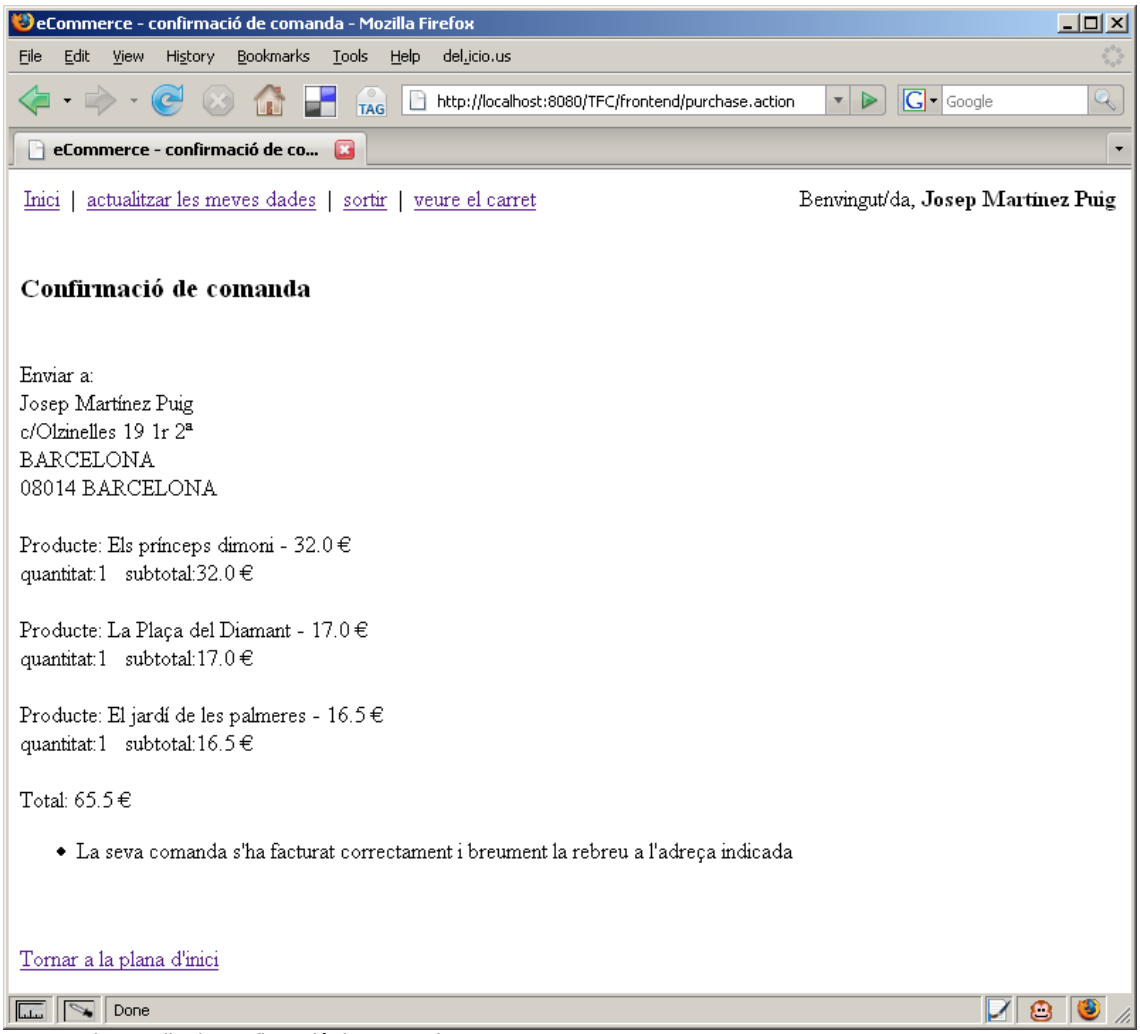
Front-end: pantalla de simulació de pagament en línia

En aquesta pantalla es simula el pagament en línia dels productes seleccionats prèviament i dipositats al carret de la compra virtual.

Usualment, en aquest tipus de programari el pagament s'efectua mitjançant la connexió per HTTPS a un servidor d'una entitat financera aliena a l'organització propietària de la botiga virtual. Per tal de simular eficientment aquesta circumstància, aquesta pantalla està desproveïda dels elements gràfics característics de la botiga virtual, com la barra de navegació, i s'indica a la capçalera que la plana pertany a una altra organització.

En aquesta pantalla s'introdueix la tarja de crèdit, la data de caducitat d'aquesta i altres dades que puguin requerir les entitats financeres per tal de confirmar la validesa del pagament. Un cop introduïdes i validades aquestes dades, es redireccionarà de nou a la botiga virtual on es mostrarà la confirmació del procés de compra.

Confirmació de comanda

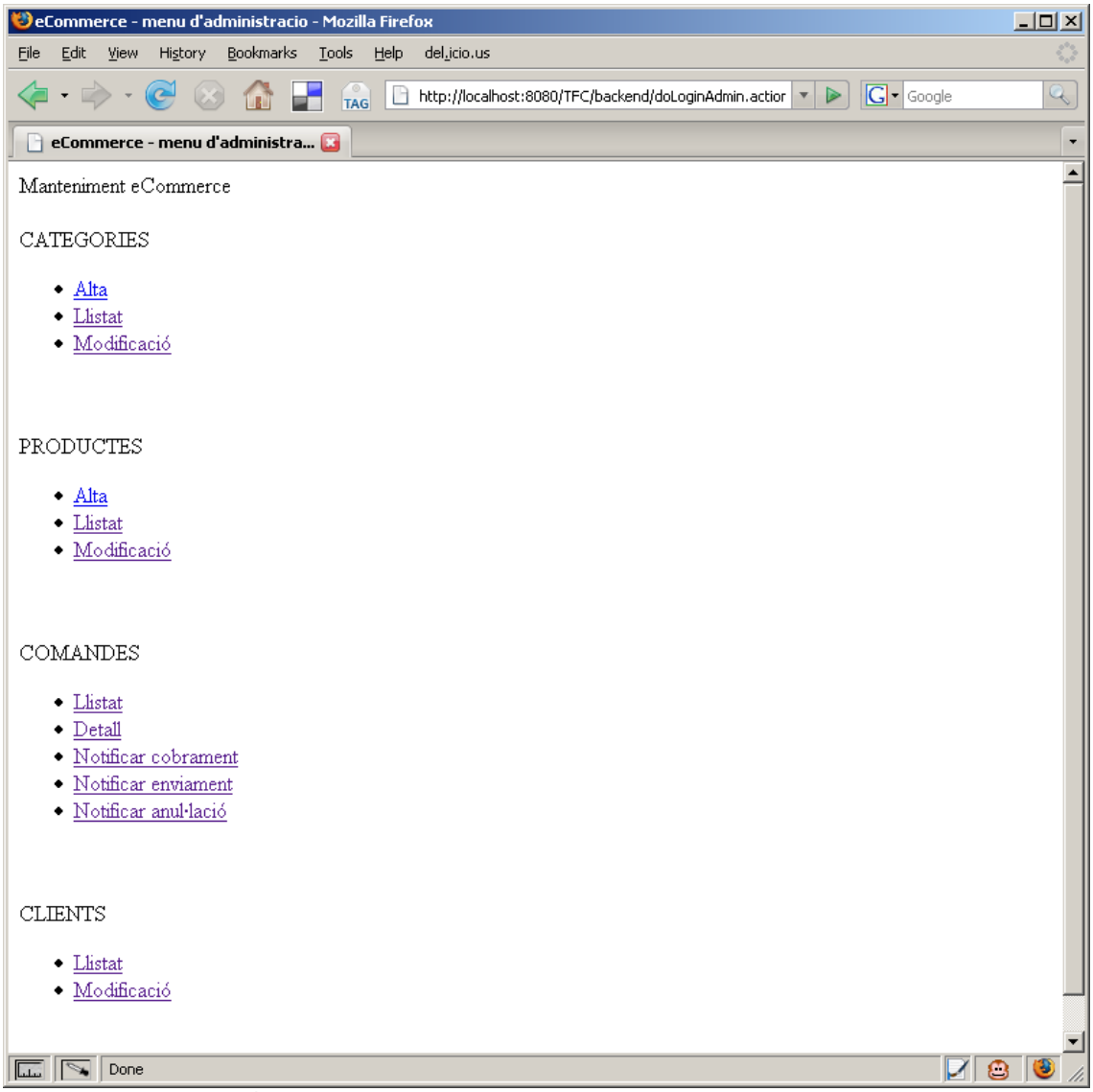


Front-end: pantalla de confirmació de comanda

Un cop confirmat el pagament, es mostra a l'usuari les dades de la nova comanda que s'ha creat mitjançant el procés de compra per tal de confirmar el correcte funcionament d'aquest procés i la finalització en l'estat esperat.

Part privada o back-end

Menú principal

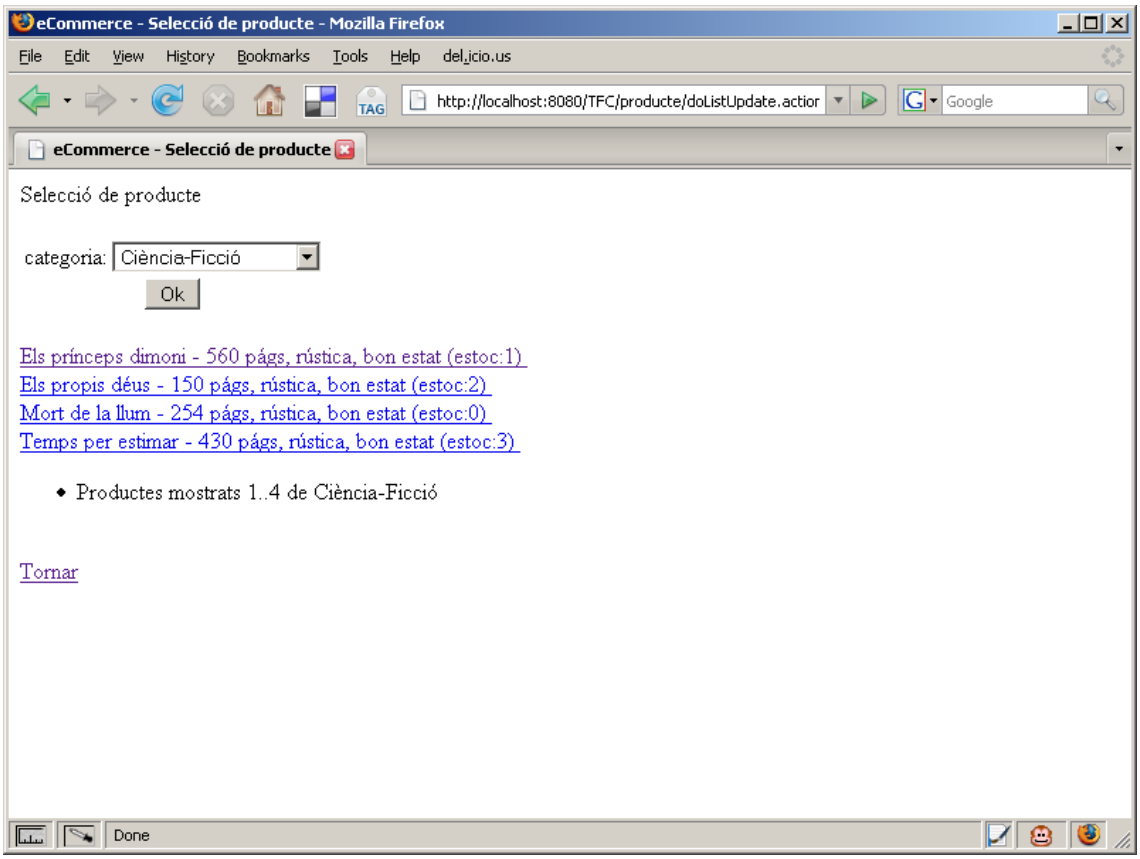


Back-end: pantalla de menú principal

El menú principal de la part privada o *back-end* conté els enllaços a les accions de manteniment de les diferents entitats del programari, com les categories, els productes, les comandes i els clients.

Aquesta pantalla i la resta de pantalles de la part privada només són accessibles un cop l'administrador s'hagi registrat degudament al sistema.

Llistat de productes

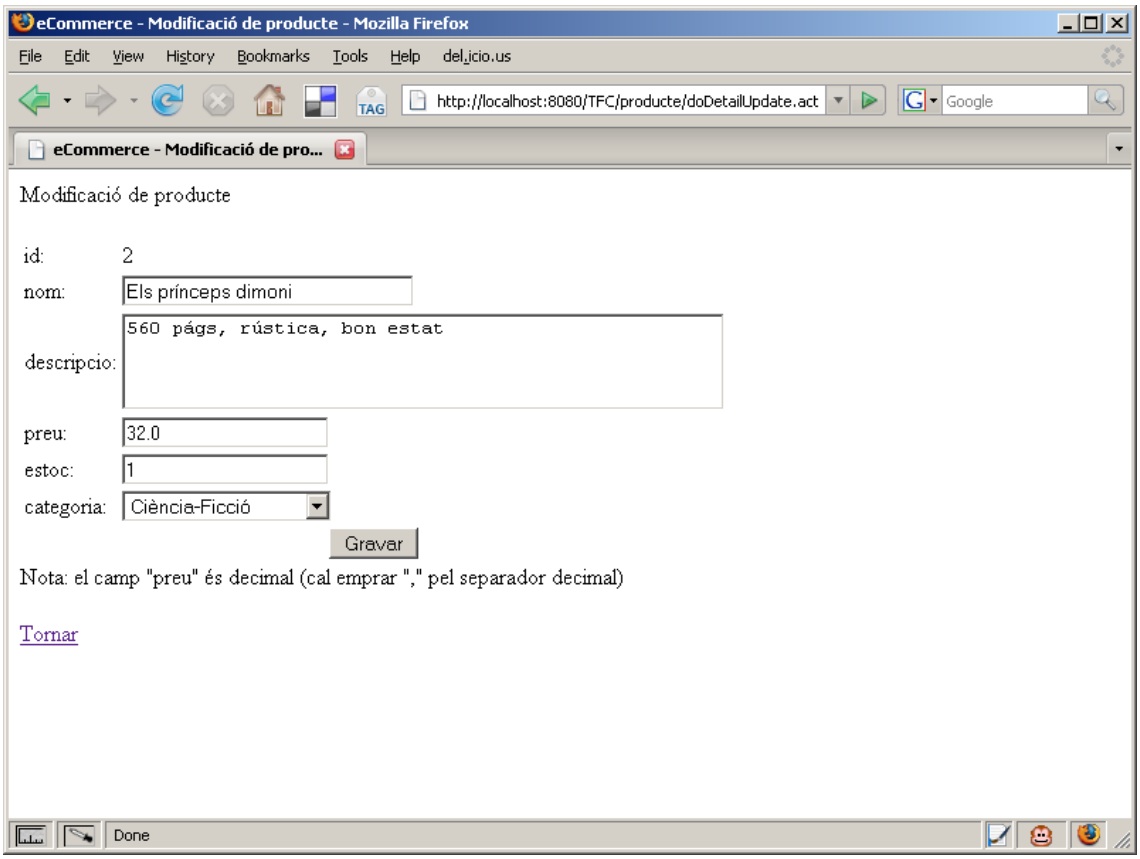


Back-end: pantalla de llistat de productes

Prenent com a exemple d'entitat de negoci els productes, es mostra una típica pantalla de llistat. En aquest cas concret es llisten els productes que pertanyen a la categoria prèviament escollida per l'administrador mitjançant la llista desplegable de la part superior.

Els productes llistats contindran enllaços d'hipertext per tal de permetre la seva selecció per a posteriors consultes i modificacions, o bé poden estar desproveïts d'aquests enllaços en cas de que només es vulgui consultar el llistat.

Formulari de producte



Back-end: pantalla de formulari de producte

Un cop seleccionat el producte, es mostrarà el formulari per la modificació de les seves dades. En cas de que es realitzi una alta de producte, el formulari serà el mateix amb la particularitat òbvia que els controls d'entrada de dades apareixeran buits. En el cas de les consultes, el contingut dels camps apareixerà com a text simple.

En aquesta pantalla s'observa la utilització de llistes desplegables amb la finalitat d'evitar errors d'introducció de dades a l'administrador, i com el sistema no permet la modificació del camp corresponent a l'identificador únic de l'entitat.

Consideracions sobre l'anàlisi

Tal com es pot observar en el model de casos d'ús proposat, la separació entre les accions dels Usuaris i Clients per una banda, i per una altra les de l'Administrador és suficientment entenedora per tal de copsar la diferència funcional entre els rols assignats.

Donat el limitat abast del present treball, no hem inclòs entre els casos d'ús per usuari i client les funcionalitats més avançades que es trobarien en un programari d'aquest tipus, com les recerques per múltiples criteris o les recomanacions personalitzades. Tot i així, esperem cobrir les funcionalitats bàsiques que permetin que el programari assoleixi el nivell de versemblança requerit.

Pel que fa als casos d'ús per Administrador, aquests consisteixen bàsicament en el CRUD de les diferents entitats que conformen l'aplicació, resultant en una estructura força repetitiva prou adient per aplicar les tècniques actuals de la indústria del programari enfocades al reaprofitament de codi, amb la finalitat d'alleugerir aquestes repeticions.

En aquests casos d'ús s'ha considerat convenient separar la modificació de Comanda dels seus canvis d'estat, de manera que resulti més entenedor el cicle de vida d'aquest objecte.

S'han obviat determinades accions d'Administrador, com la creació de Comandes i Clients, tant per simplificar les funcionalitats resultants com per considerar que aquestes tasques ja estan cobertes entre les funcionalitats que s'ofereixen als Clients i Usuaris, i que malgrat la separació de rols no s'impedeix que la mateixa persona pugui interactuar amb l'aplicació en rols diferents. També s'han obviat les operacions de donar de baixa o esborrament de les diferents entitats, donat que resultarien innecessàries per al nivell actual de desenvolupament de l'aplicació.

Disseny

Visió general de l'arquitectura

De la documentació de l'anàlisi es desprèn l'existència d'una sèrie d'entitats, com client, producte, comanda, etc. que hauran d'emmagatzemar el seu estat a la base de dades. Aquestes seran les *classes d'entitat*, que constituiran el nucli de la capa de negoci. D'altra banda, també existeixen elements no persistents, vinculats a la sessió d'usuari, com el carret de la compra. Per distingir-les, anomenarem aquestes classes com *classes de sessió*. En l'àmbit de l'especificació EJB, serien els equivalents aproximats dels *Entity Beans* amb persistència gestionada per la classe i del *Sessions Beans*.

Donat que cada classe d'entitat necessita un accés propi a la base de dades a més de implementar la lògica de negoci, es pot preveure la circumstància de que la seva codificació arribi a tenir una extensió considerable. A més, donat que s'utilitza el bastiment Hibernate per l'accés a les dades, en cas de modificar els orígens d'aquestes s'haurien de modificar també aquestes classes, fet que dificultaria el manteniment.

Ens trobem, per tant, en un escenari adient per aplicar el patró de disseny *Data Access Object* (DAO) (J2EE), que consisteix en la delegació de l'accés a les dades en una altra classe independent de la classe d'entitat. D'aquesta manera, la classe d'entitat pot concentrar-se en la implementació de les regles de negoci, i en cas de modificar l'origen de les dades, per exemple, utilitzant l'API JDBC directament, aquesta no hauria de modificar-se. Així, cada classe d'entitat disposarà d'una classe DAO per aquesta finalitat.

Una altra qüestió que cal considerar és el tipus d'objectes que es transmetran, per una banda, entre les classes DAO i les d'entitat, i per un altra, entre aquestes i els JSPs de la capa de presentació. Es podrien utilitzar classes de l'API *java.util.Collections* per representar els registres de la base de dades, o bé les mateixes classes d'entitat. En el primer cas ens trobaríem amb estructures força genèriques i dependents del coneixement de les operacions que es realitzen a les classes DAO i dels resultats esperats, mentre que en el segon s'incrementaria el tràfic de xarxa considerablement si l'aplicació fos distribuïda i, per exemple, les classes d'entitat operessin en una màquina diferent del contenidor web i calgués transmetre-les mitjançant RMI, Corba, o qualsevol tecnologia de distribució.

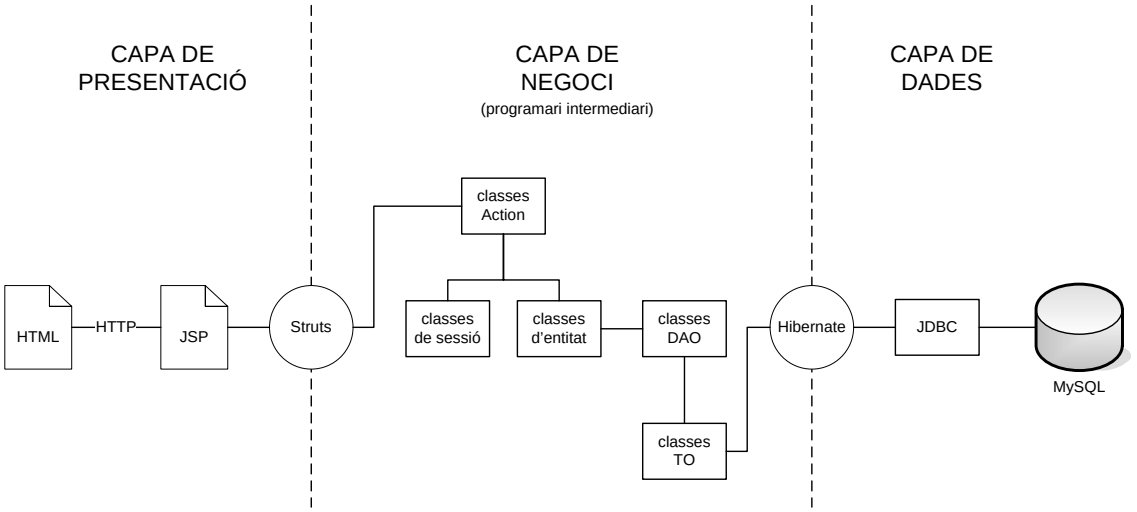
La solució òptima consisteix en utilitzar el patró de disseny *Transfer Object* (TO) (J2EE). Una classe TO conté una estructura bàsica de les dades, que poden ser atributs públics o bé atributs privats acompanyats dels seus mètodes accesors de lectura i escriptura (*getters* i *setters*) i res més, sense cap mena de lògica de negoci. El resultat és una estructura lleugera, apta per la seva serialització i deserialització en el context de les aplicacions distribuïdes. Així, aquestes classes constituiran l'objecte de intercanvi entre les classes DAO i les d'entitat per una banda, i entre les d'entitat i sessió amb les classes gestores de presentació per un altra, i finalment, entre aquestes i les planes JSP.

D'aquesta manera, obtenim el benefici addicional de servir als JSPs classes molt simples, que només contenen les dades necessàries aptes per la seva visualització immediata, i així s'amaga la complexitat de la lògica de negoci a la capa de presentació.

En el cas de que es necessitin presentar camps calculats que no apareguin a l'estructura de la classe TO, o bé calgui utilitzar objectes compostos, s'utilitzarà el patró de disseny *Composite Object* (J2EE) que bàsicament consisteix en la creació d'un nou objecte agregant altres objectes que el componen. Així, es crearà un nou *Transfer Object* que contindrà els altres *Transfer Object* necessaris per tal de formar l'objecte compost.

En resum, generalment, cada classe d'entitat disposarà d'una classe DAO i d'una altra TO, i cada classe de sessió disposarà d'una classe TO. En tots dos casos es poden emprar classes TO composades si la necessitat ho requereix.

Un cop exposades les consideracions de disseny de la lògica de negoci, cal considerar el paper dels bastiments Struts e Hibernate en aquesta arquitectura. Struts actua en la frontera entre la capa de presentació i la lògica de negoci, mentre que Hibernate ho fa entre aquesta i la capa de dades, tal com es mostra en el següent esquema:



Esquema de l'arquitectura

Struts consisteix en una implementació del patró *Model-View-Controler* per la plataforma J2EE. Aquest patró de disseny respon a la necessitat de separar la presentació de les dades (la vista) de la lògica de negoci (el model) introduint un tercer element, el controlador. L'esquema pot semblar similar a l'arquitectura de tres capes, però existeixen diferències a nivell topològic. L'arquitectura a tres capes segueix un model lineal, en el que el client mai es comunica directament amb les dades ni a l'inrevés, i tot el flux de informació passa a través del programari intermediari. En canvi, el model MVC és triangular: la vista envia actualitzacions al controlador, el controlador actualitza el model, i la vista és actualitzada directament pel model.

En el cas de Struts, la vista son les planes JSP, el model són les classes gestores de presentació (classes *Action* en l'esquema), i el controlador és un *Servlet*, juntament amb classes auxiliars, filtres, etc. que resulta completament transparent per al desenvolupador.

Així, la funció de Struts serà la de mapejar les planes JSP de la presentació amb les classes Action, les quals s'encarregaran de cridar i utilitzar les classes d'entitat i de sessió, i disposaran les dades obtingudes com a resultat en forma de classes TO a disposició dels JSPs.

En l'altre extrem de la capa de negoci, Hibernate s'encarrega de mapejar les taules de la base de dades amb classes que continguin els atributs adients per encapsular els registres, estalviant les tasques típiques d'obrir i tancar connexions, llegir objectes *ResultSet*, convertir tipus de dades, etc. Tal com s'ha exposat anteriorment, s'utilitzarà exclusivament dins les classes DAO. Les classes mapejades seran les classes TO, donat que representen les dades d'una entitat, i donat que no contenen cap lògica de negoci, el seu ús permet visualitzar fàcilment l'estructura de les dades.

D'aquesta manera, les classes d'entitat esdevenen *Decorators* (GoF) de les classes TO, donat que estenen la funcionalitat d'aquestes a l'afegir les regles de negoci i les comunicacions amb les classes DAO. La funció de servir les dades es delega en la classe TO continguda.

Capa de dades

Disseny de la persistència

Un cop identificades les entitats que intervenen en l'aplicació mitjançant la documentació de l'anàlisi, es dissenya la persistència d'aquelles entitats que així ho requereixen. Aquest disseny es realitza considerant que s'emprarà el sistema gestor de bases relacional MySQL. Tot i que semblaria més adient emprar un sistema gestor de bases de dades orientat a l'objecte, considerem que aquest tipus de programari encara no està prou consolidat en el mercat, fet que alteraria la versemblança del producte final.

Així, la persistència s'implementarà mitjançant el model E-R que es detalla a continuació:

Diagrama E-R

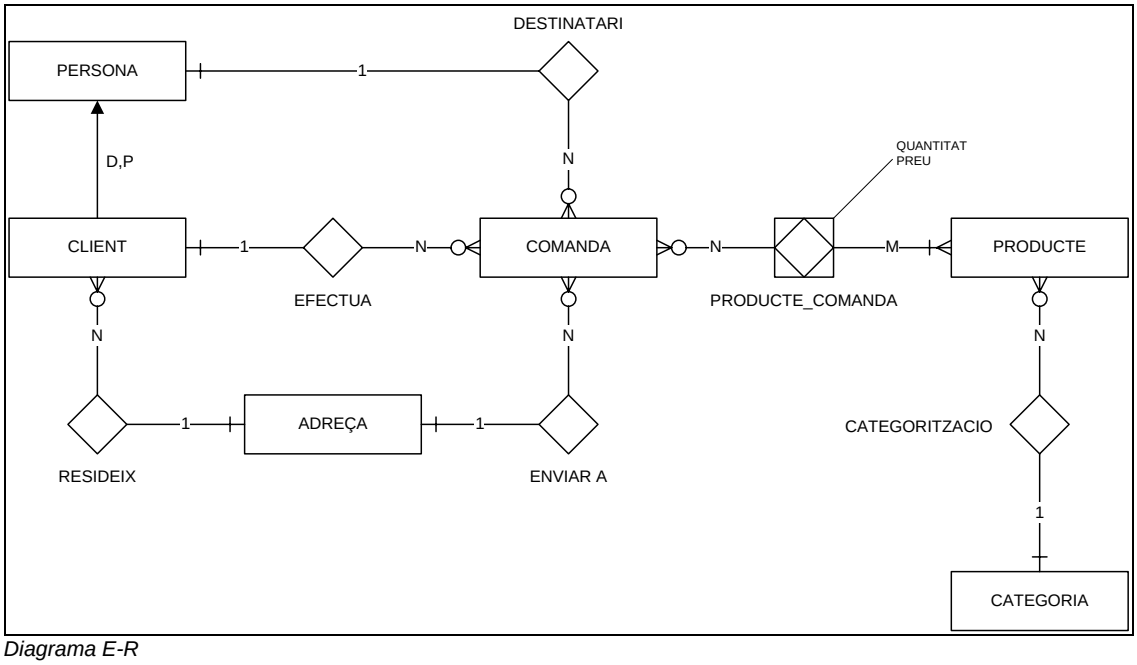


Diagrama E-R

Llistat d'entitats i relacions resultants de la transformació al model Relacional

Entitats:

PERSONA (id, nom, cognom1, cognom2)

ADRESA (id, adresa, poblacio, provincia, codiPostal, telefon)

CLIENT (id, NIF, password, email, telefon, id_adresa)
{id} clau forana cap a PERSONA (id)
{id_adresa} clau forana cap a ADRESA(id)

CATEGORIA (id, nom)

PRODUCTE (id, nom, descripcio, preu, estoc, id_categoria)
{id_categoria} clau forana cap a CATEGORIA(id)

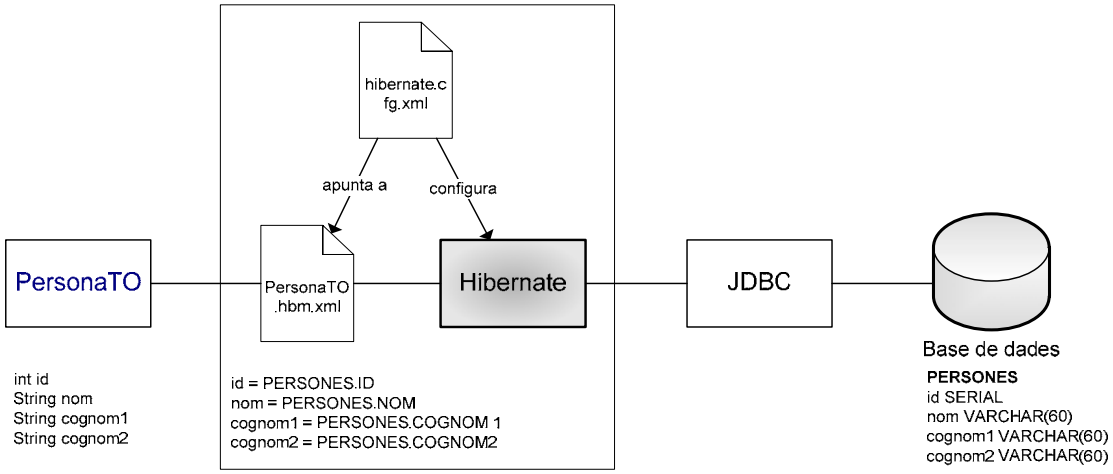
COMANDA (id, estat, data_creacio, data_cobrament, data_enviament, data_anulacio, id_client, id_persona_desti, id_adresa_desti)
{id_client} clau forana cap a CLIENT(id)
{id_persona_desti } clau forana cap a PERSONA(id)
{id_adresa_desti } clau forana cap a ADRESA(id)

Relacions:

PRODUCTE_COMANDA (id, id_comanda, id_producte, quantitat, preu)
{id_comanda} clau forana cap a COMANDA(id)
{id_producte} clau forana cap a PRODUCTE(id)

Accés a dades amb Hibernate

De les entitats i relacions del model relacional es desprenen les classes de persistència, les quals seran l'objecte de mapeig amb les taules de la base de dades mitjançant els sistemes de configuració del bastiment Hibernate. Aquest mapeig es realitza mitjançant arxius XML, dels quals existeix un de configuració general de l'eina (hibernate.cfg.xml) que enllaça amb l'arxiu de configuració de cada entitat (per exemple, PersonaTO.hbm.xml). El següent esquema conceptual mostra un exemple de mapeig d'una entitat:



Esquema de mapeig classe-taula amb Hibernate

D'aquesta manera, el desenvolupament s'allibera de la tasca d'escriure les sentències SQL, realitzar les lectures i actualitzacions, carregar les dades a partir d'objectes *ResultSet*, etc. com és habitual quan es treballa amb sistemes gestors de bases de dades relacionals. Així, per posar un exemple, gràcies al mapeig entre taules i classes, l'actualització d'una propietat d'una classe de persistència mitjançant un mètode accesor d'escriptura o *getter* es reflectirà immediatament en la taula corresponent de la base de dades, sempre i quan aquesta operació s'efectuï dins una sessió de Hibernate, sense que calgui obtenir manualment una connexió a la base de dades, escriure les sentències SQL, etc.

Les classes de persistència segueixen el patró de disseny *Transfer Object* (J2EE). Aquest patró neix de la necessitat de disposar d'objectes lleugers que només continguin les dades d'una entitat sense altres elements com regles de negoci, etc. amb la finalitat de permetre el seu tràfic eficient per la xarxa en els casos en els que només calgui mostrar aquestes dades.

Hibernate no obliga a implementar aquest patró per les classes mapejades, només a l'existència de mètodes accesors de lectura i escriptura per als camps directament mapejats i a que les classes tinguin un constructor buit, per tal de complir l'especificació *Java Beans*. Per tant, res impediria que aquestes classes continguessin també la lògica de negoci, però considerem que amb l'estratègia seguida obtenim els beneficis del patró *Transfer Object*, i addicionalment el codi resulta més entenedor al distingir les classes que reflecteixen directament un registre de la base de dades corresponent a una entitat, de les classes que modelen l'entitat mateixa, les quals actuen com un *Decorator* (GoF) del *Transfer Object*.

En tot cas, s'ha de considerar que existeixen diverses estratègies per a la implementació del patró *Transfer Object*. Per exemple, es pot realitzar amb classes que només disposin dels atributs públics necessaris sense mètodes accesors de lectura i escriptura (*getters* i *setters*). Si bé, com hem assenyalat prèviament, Hibernate no obliga a la implementació d'aquest patró per a les classes mapejades, si que obliga a que aquestes classes disposin dels mètodes accesors corresponents. Per tant, la implementació del *Transfer Object* ha de seguir una estratègia determinada, consistent en utilitzar classes que només disposin dels atributs corresponents juntament amb els seus mètodes accesors.

S'indiquen les classes que segueixen aquest patró, ja siguin persistents o no, mitjançant el sufix -TO (per exemple, PersonaTO).

Respecte a les relacions d'herència i associació, Hibernate ja proporciona els mecanismes adients per treballar amb aquestes característiques compartides, encara que amb les seves lògiques diferències, entre el model de dades relacional i la programació orientada a l'objecte. Així, en els arxius de mapeig es poden declarar les subclasses d'una entitat e indicar en quines taules es realitza l'especialització, al mateix temps que es declara la relació d'herència entre les dues classes mapejades. En el projecte s'utilitza aquest mecanisme per modelar la herència entre Persona i Client.

Pel que fa a les associacions, aquestes es poden implementar mitjançant la inclusió de camps a les classes d'entitat que siguin contenidors pertanyents al paquet estàndard de l'API *java.util.Collections*. Les associacions es creen i s'actualitzen mitjançant la inclusió i esborrament d'elements en aquests objectes amb els mètodes que exposen per a tals efectes. Per exemple, la relació 1..N existent entre Categoria i Producte s'ha implementat en el projecte mitjançant un objecte que compleix la interfície *Set* a Categoria, al qual podem afegir productes mitjançant el mètode *add()* i treure'n mitjançant el mètode *remove()*.

Tal com s'ha comentat, les operacions de creació, actualització, esborrament, etc. que es realitzen sobre les classes mapejades amb Hibernate s'han de realitzar dins de l'àmbit d'una sessió de treball per tal que els canvis en l'estat dels objectes es reflecteixin a la base de dades. Aquest concepte és equiparable a una seqüència transaccional d'una connexió amb base de dades, i de la mateixa manera pot finalitzar amb *commit* o *rollback*. En el projecte aquestes sessions es creen i s'utilitzen exclusivament dins de les classes que segueixen el patró de disseny DAO (*Data Access Object*) (J2EE) associades a cada entitat, de tal manera que resulta una estricta separació entre les entitats i l'accés a dades, fins i tot en els casos on s'involucren diverses entitats en una mateixa transacció. El canvi de Hibernate a qualsevol altre sistema a d'accés a base de dades, per exemple, mitjançant els objectes del paquet *java.sql*, només implicaria la reescriptura de les classes *Data Access Object*, sense que efectuar cap modificació a les classes d'entitat ni a les *Transfer Object*.

Capa de lògica de negoci

Tal com s'ha comentat en la descripció de l'arquitectura, les classes d'entitat encapsulen les regles de negoci, actuant com a *Wrappers* de l'estructura bàsica de les dades representada per la classe *Transfer Object* associada.

Les dades d'aquesta classe *Transfer Object* poden provenir de la base de dades o bé generar-se dins l'aplicació, com és el cas dels camps calculats o bé de les dades corresponents a elements auxiliars com mesos, anys, etc. Per tant, dividirem l'exposició de les principals classes que conformen la lògica de negoci o programari intermediari (*middleware*) segons presentin implementada la persistència o no. Finalment, farem un breu incís sobre les classes auxiliars.

Abans, però, observem el conjunt de classes d'entitat i de sessió de la de la capa de negoci juntament amb les seves relacions en el següent diagrama estàtic, sense considerar les classes *Data Access Object* ni les classes *Transfer Object* per obtenir un diagrama comprensible:

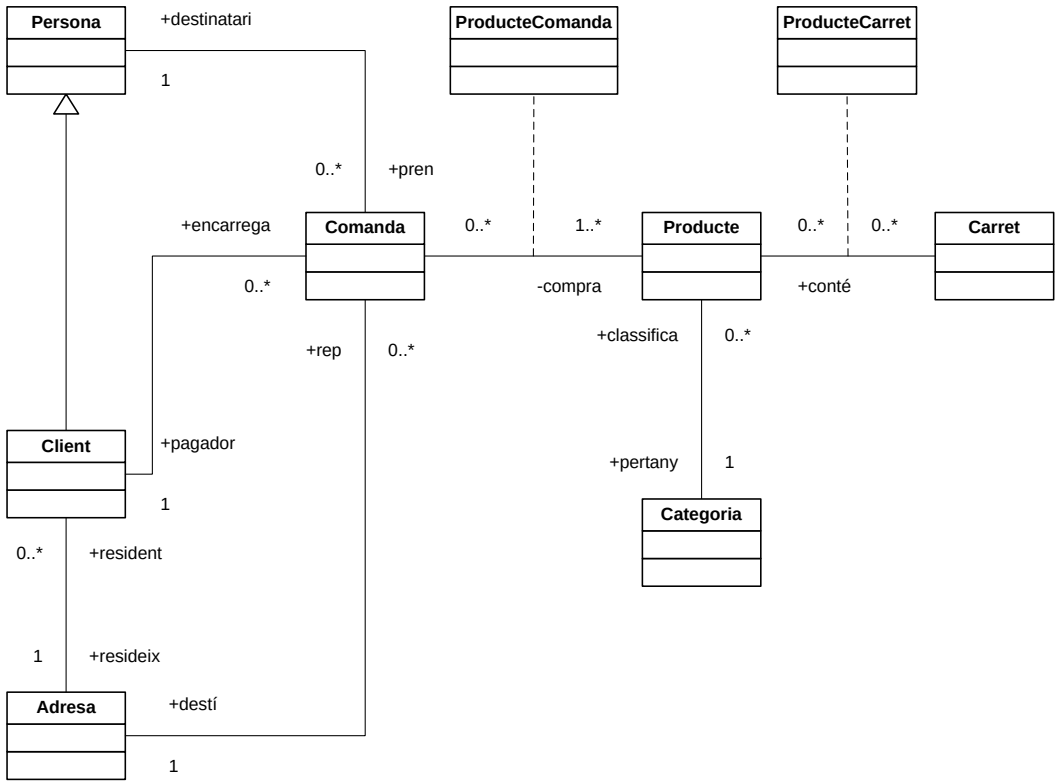


Diagrama estàtic de la capa de negoci

Classes d'entitat o persistents

Aquestes classes tenen com a missió principal estendre estendre la funcionalitat bàsica de les classes *Transfer Object* associades, que són l'objecte de mapeig amb Hibernate i nomes contenen les dades tal com s'emmagatzemen a la base de dades, juntament amb els seus accesors de lectura i escriptura. L'extensió de la funcionalitat inclou importants aspectes com la validació de les dades d'entrada, les relacions amb les classes *Data Access Object* associades per realitzar l'accés a les dades de manera transparent des de l'exterior de la classe, la implementació de les regles de negoci, etc.

El seu disseny presenta determinats elements comuns que cal considerar, els quals enumerem a continuació:

- s'instancien mitjançant un constructor que rep la classe *Transfer Object* associada com a paràmetre.
- poden actualitzar el seu estat mitjançant l'accesor d'escriptura (*setter*) de l'atribut corresponent, o bé rebent un *Transfer Object* que encapsuli les noves dades.
- disposen de mètodes per retornar els *Transfer Object* que encapsulen, o bé per generar-ne d'altres per als casos on es requereixi objectes compostos, els quals segueixen el patró de disseny *Composite Object*.
- disposen dels mètodes escaients per interactuar amb la base de dades: obtenció d'una instància a partir de l'identificador únic del sistema, creació d'una nova entitat mitjançant el *Transfer Object* associat, etc. En el cas de les altes es retorna una la nova instància creada, que ja disposarà de l'identificador únic assignat automàticament pel sistema.
- implementen la interfície *Comparable* per tal de generar llistats ordenats pels criteris corresponents.
- seguint les pràctiques recomanades en l'arquitectura J2EE, sobreescriuen els mètodes *equals()* i *toString()*

Detallem a continuació les classes d'entitat així com les seves responsabilitats i relacions. De la present enumeració, per motius d'espai, obviem les classes *Transfer Object* i *Data Access Object* associades:

Classe Persona

Responsabilitats: Encapsula les dades bàsiques d'una persona i realitza les operacions CRUD per a les dades d'aquesta entitat. S'utilitza per l'extensió cap a Client i per consignar els destinataris de les comandes, que poden ser la mateixa persona que el Client o bé una altra diferent.

Relacions: Client (especialització), Comanda (associació).

Notes: el sistema preveu la possibilitat d'actuar amb persones jurídiques, les quals no tindran cognoms.

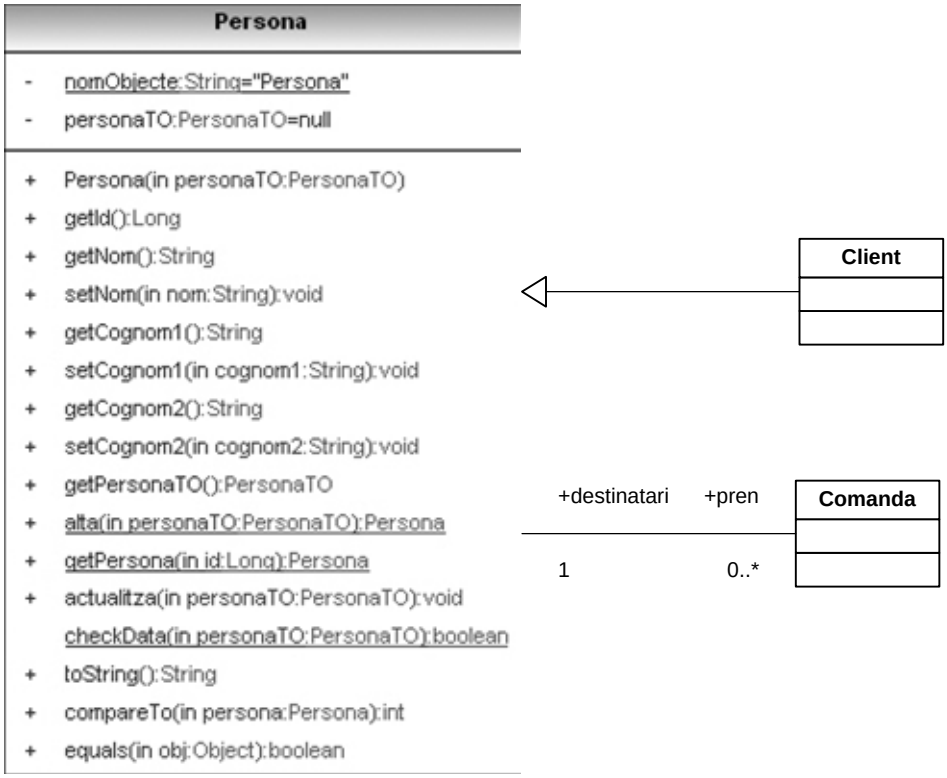


Diagrama estàtic de Persona

Classe Client (extensió de Persona)

Responsabilitats: Encapsula les dades d'un client i realitza les operacions CRUD per a les dades d'aquesta entitat. A més, retorna la instància d'Adresa associada i proporciona conjunts de instàncies amb tots el Clients de la base de dades.

Relacions: Persona (generalització), Adresa (associació), Comanda (associació).

Notes: Cada client s'identifica, a més del seu id assignat pel sistema, pel seu NIF, el qual haurà de ser únic. L'atribut telèfon indicarà el telèfon mòbil, a diferència del telèfon d'Adresa, el qual intenta representar el telèfon fix vinculat a una adreça. D'altra banda, no existeix un nom d'usuari per utilitzar conjuntament amb la contrasenya, donat que aquesta funcionalitat es pot complir amb el mateix NIF. S'observen dues sobrecàrregues al mètode d'alta, un que només rep com a paràmetre el Transfer Object de Client i un altre que a més rep el d'Adreça. D'aquesta manera es podran crear nous clients juntament amb les seves adreces associades de manera transaccional.

Diagrama estàtic de Client

Classe Adresa

Responsabilitats: Encapsula les dades d'una adreça i realitza les operacions CRUD per a les dades d'aquesta entitat. La classe s'utilitza per les associacions amb Client i Comanda, on indica l'adreça habitual d'un Client i el destí de la Comanda respectivament.

Relacions: Client (associació), Comanda (associació).

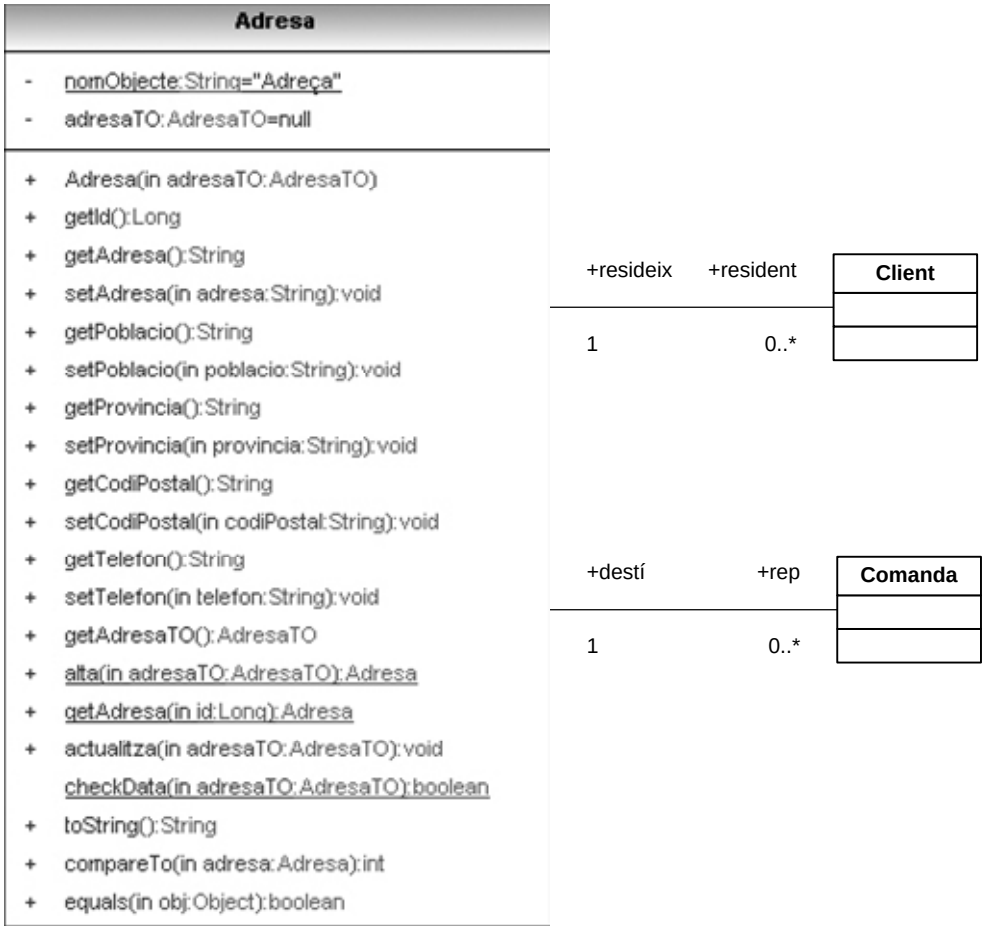


Diagrama estàtic d'Adresa

Classe Categoria

Responsabilitats: Encapsula les dades d'una categoria de productes i realitza les operacions CRUD per a les dades aquesta entitat. A més, retorna els productes associats a aquesta categoria i proporciona conjunts de instàncies amb totes les categories existents a base de dades .

Relacions: Producte (associació).

Notes: donat que l'aplicació es genèrica per tal d'emular el comerç electrònic en qualsevol sector comercial, les categories de productes són totalment abstractes i no requereixen més que un nom i una descripció com a atributs propis. No es contempla la possibilitat de crear subcategories ni qualsevol mena de relació entre aquestes.

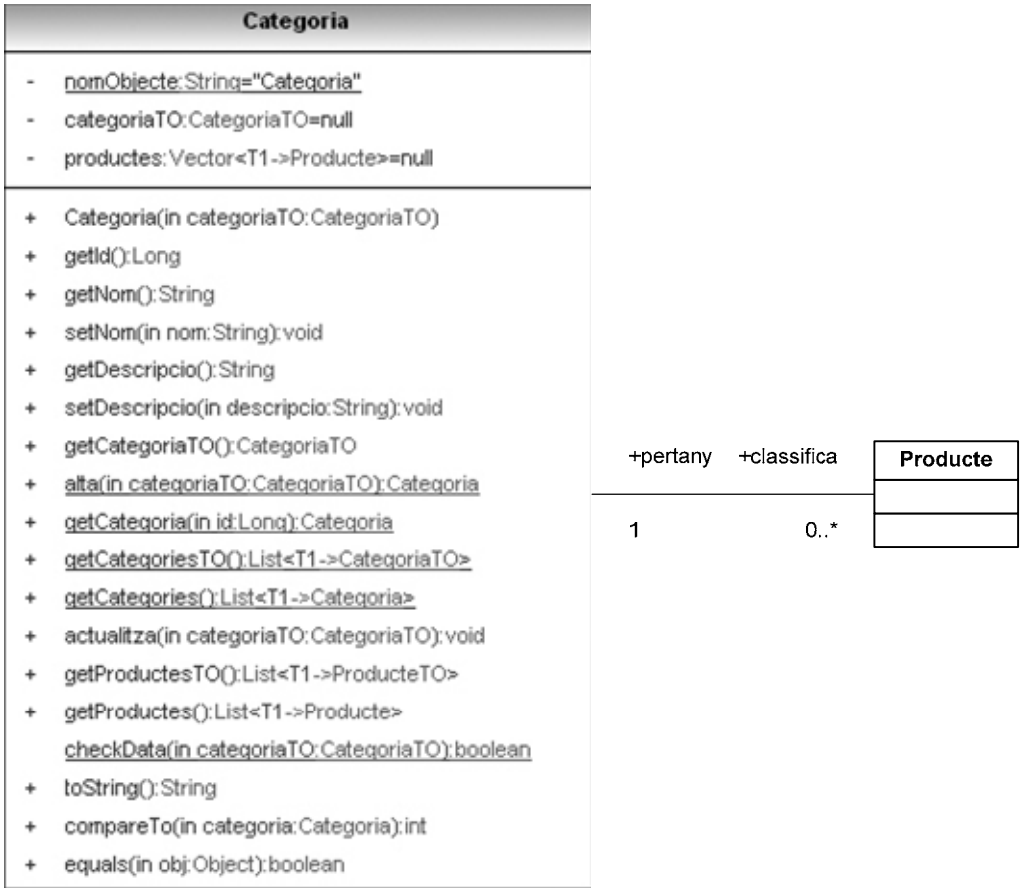


Diagrama estàtic de Categoria

Classe Producte

Responsabilitats: Encapsula les dades d'un producte i realitza les operacions CRUD per a les dades aquesta entitat. A més, retorna la Categoria associada i proporciona conjunts de instàncies amb tots els productes existents a base de dades .

Relacions: Categoria (associació), Comanda (associació parametritzada).

Notes: novament, la classe resulta prou genèrica per tal de representar productes de qualsevol tipus amb la finalitat de proporcionar una experiència de comerç electrònic desvinculada d'un sector comercial concret. Així, la classe es podria especialitzar per representar productes concrets. Per exemple, en el cas dels llibres s'inclourien camps per l'autor i l'ISBN, etc.

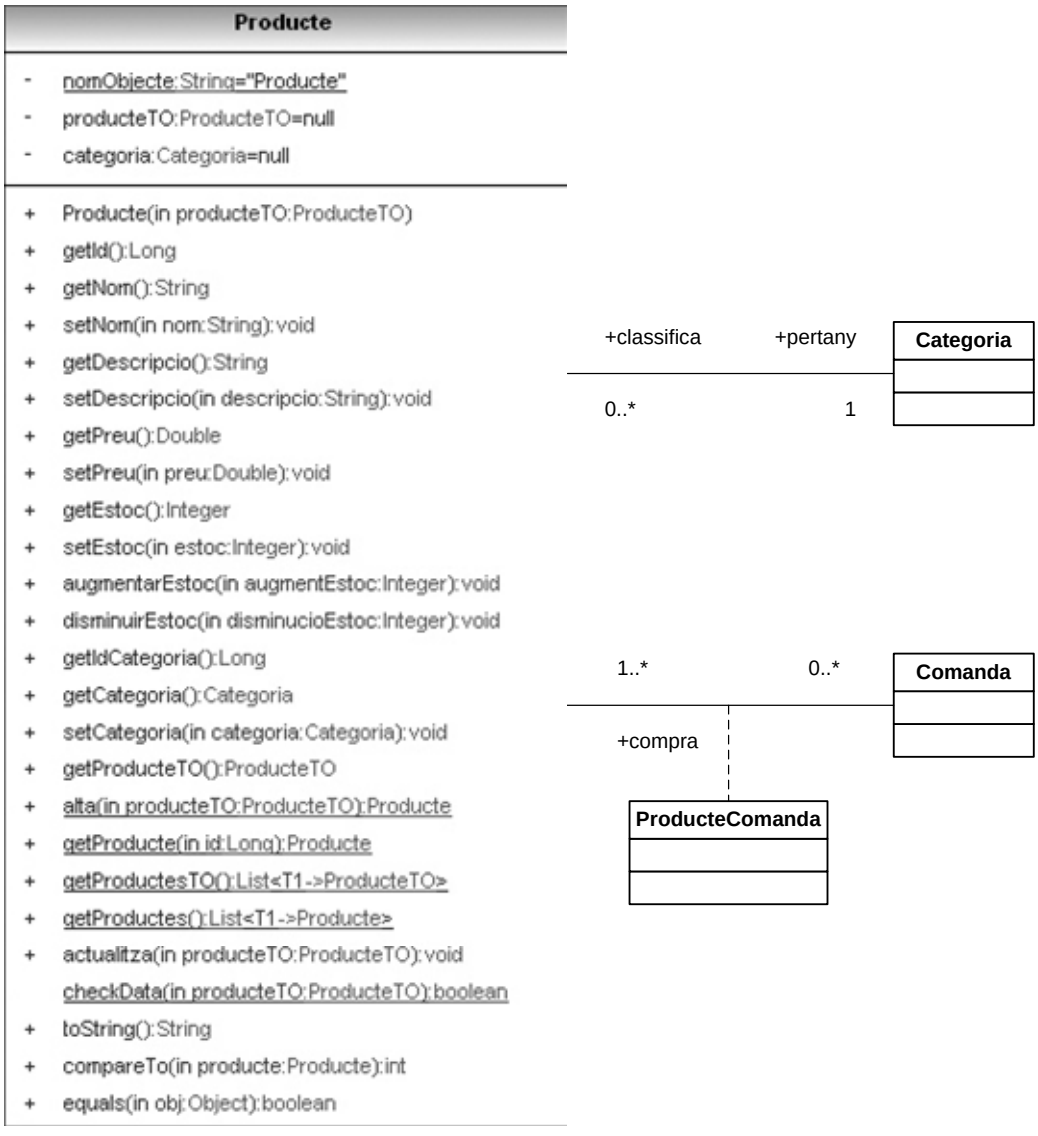


Diagrama estàtic de Producte

Classe Comanda

Responsabilitats: Encapsula les dades d'una comanda i realitza les operacions CRUD per a les dades aquesta entitat. A més, retorna els objectes associats Client, Persona i Adresa, a més de la relació de ProductesComanda que componen la Comanda i el import final d'aquesta.

Relacions: Client (associació), Persona (associació), Adresa (associació), Producte (associació parametritzada).

Notes: la classe disposa de mètodes per efectuar les operacions sobre l'objecte, com *cobrar*, *enviar* i *anular*, que modifiquen l'estat de la Comanda. Aquest estat es tracta separadament en la classe EstatComanda. Les dates de realització d'aquestes operacions s'introduiran automàticament, corresponent a les dates del sistema. El total és un atribut calculat amb el sumatori dels subtotals de cada ProducteComanda. Finalment, s'observa com la creació de la Comanda es realitza mitjançant una instància del Carret, un Client, una PersonaTO i una AdresaTO.

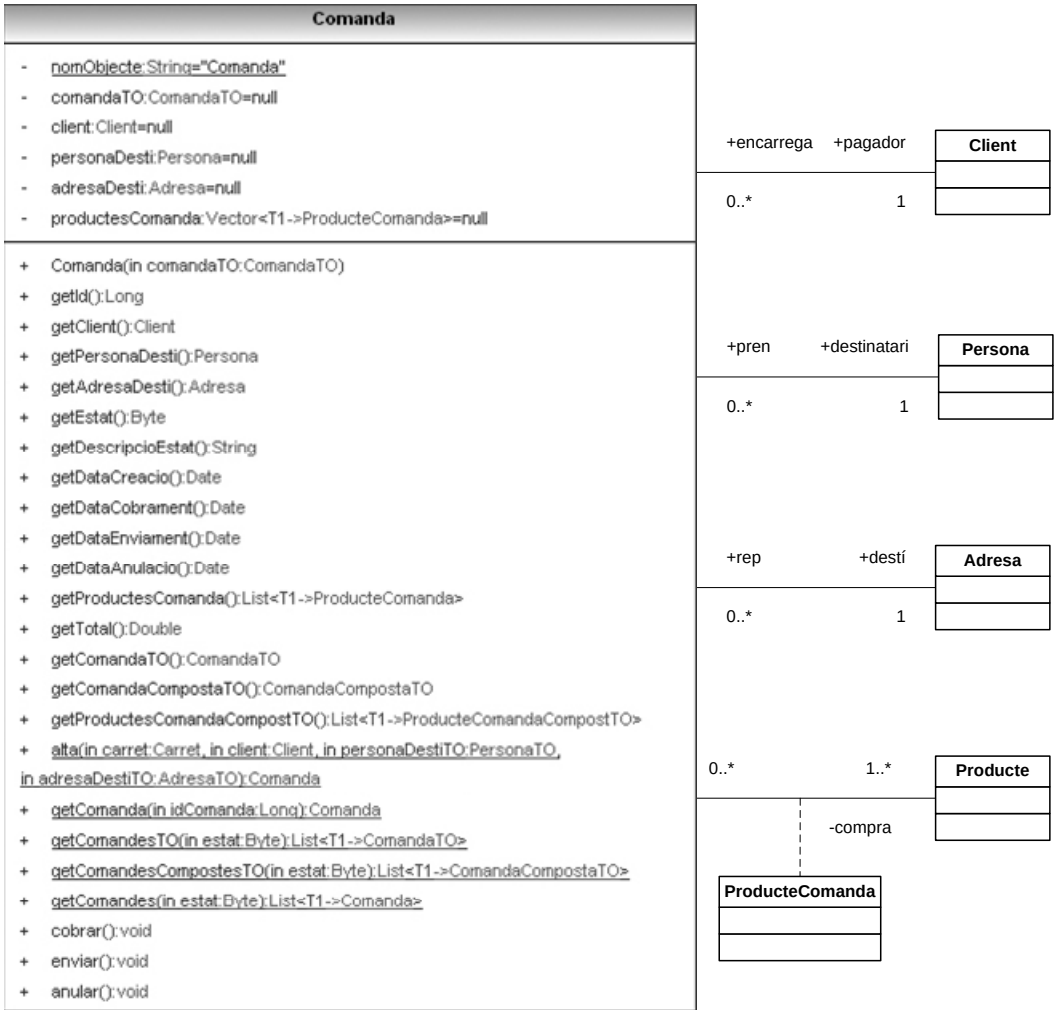


Diagrama estàtic de Comanda

Amb la finalitat de comprendre el cicle de vida d'aquest objecte, es mostra a continuació el seu diagrama d'estats:

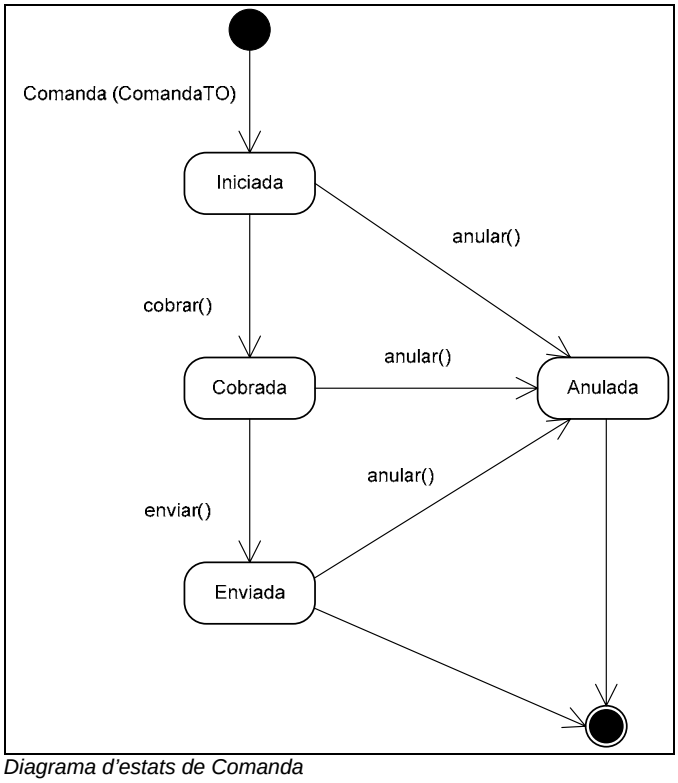


Diagrama d'estats de Comanda

Aquests estats s'emmagatzemen mitjançant constants numèriques en la classe EstatComanda, la qual s'utilitza també per proveir llistes desplegables, obtenir la descripció textual d'un estat, i altres funcions auxiliars. Com a exemple de classe auxiliar, mostrem el seu diagrama:

Diagrama estàtic d'EstatComanda

Classe ProducteComanda

Responsabilitats: Encapsula les dades de la relació entre un Producte i una Comanda. Retorna el Producte i la Comanda associats. Calcula el resultant de multiplicar el preu del producte per la quantitat per obtenir un subtotal del import final de la Comanda.

Relacions: Producte (associació), Comanda(associació).

Notes: la relació entre un producte i una comanda presenta atributs propis, que són la quantitat del producte que es demana, i el preu, que és el preu del producte en el moment d'efectuar la comanda. Aquest atribut cal emmagatzemar-lo en la relació per tal de mantenir la consistència de les dades en el cas de que el producte canviés de preu.

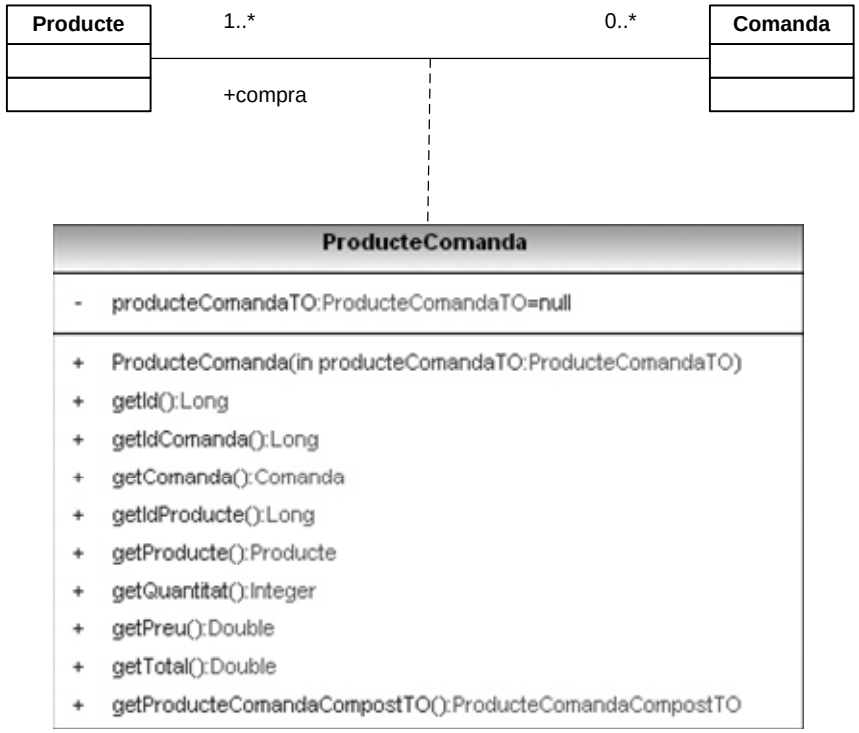


Diagrama estàtic de ProducteComanda

Classes de sessió o no persistents

Aquestes classes existeixen només dins l'objecte *Session* del contenidor J2EE, el qual abstrau el concepte de sessió de treball d'usuari, i per tant no emmagatzemen el seu estat a la base de dades.

Cal considerar, però, que dins del conjunt d'atributs emmagatzemats a *Session* poden existir classes persistents, com és el cas de Client en el cas de que aquest s'identifiqui correctament al sistema. Però donat que aquesta classe es caracteritza precisament per implementar persistència, no la consideraríem pertanyent a aquesta categoria.

Deixant de banda el cas de les classes auxiliars, com és el cas d'EstatComanda que hem vist prèviament, mostrem a continuació les classes de lògica de negoci no persistents, les quals estan totes directament relacionades amb el concepte de *carret de la compra virtual*.

Classe Carret

Responsabilitats: Representa el concepte de carret de la compra virtual. Disposa dels mètodes per afegir productes, treure'ls, i posar-se a zero.

Relacions: Producte (associació parametritzada).

Notes: tal com s'espera de les interrelacions amb un usuari habituat a utilitzar aquesta metàfora, la classe disposa dels mètodes escaients per afegir productes, treure'ls, i buidar-se. La responsabilitat de transformar-se en una Comanda es delega a la mateixa classe Comanda.

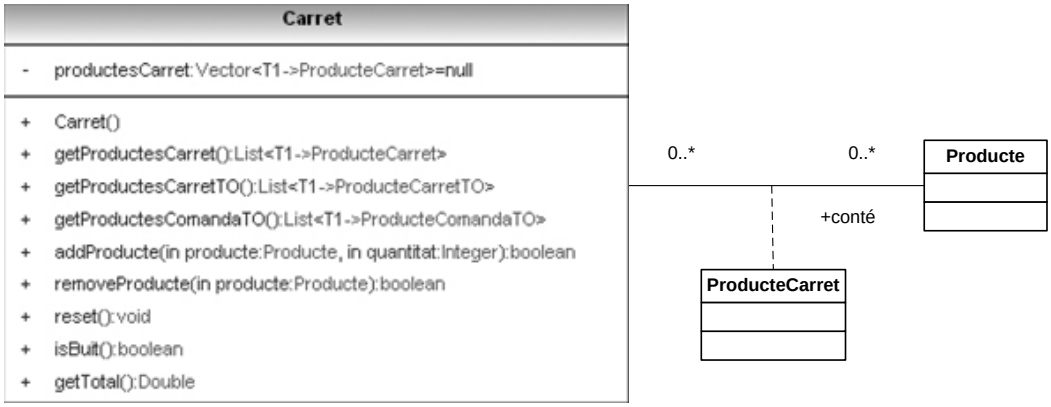


Diagrama estàtic de Carret

Classe ProducteCarret

Responsabilitats: Encapsula les dades de la relació entre un Producte i una Carret. Retorna el Producte associat. Calcula el resultant de multiplicar el preu del producte per la quantitat demanada per obtenir un subtotal del import final del Carret.

Relacions: Producte (associació), Carret(associació).

Notes: la relació entre un producte i una carret presenta com a atribut propis la quantitat del producte que es demana. La classe és molt similar a ProducteComanda, però donat que en aquest cas el preu no és un atribut propi de la relació i que la classe no requereix persistència, s'ha considerat adient la creació d'aquesta classe.

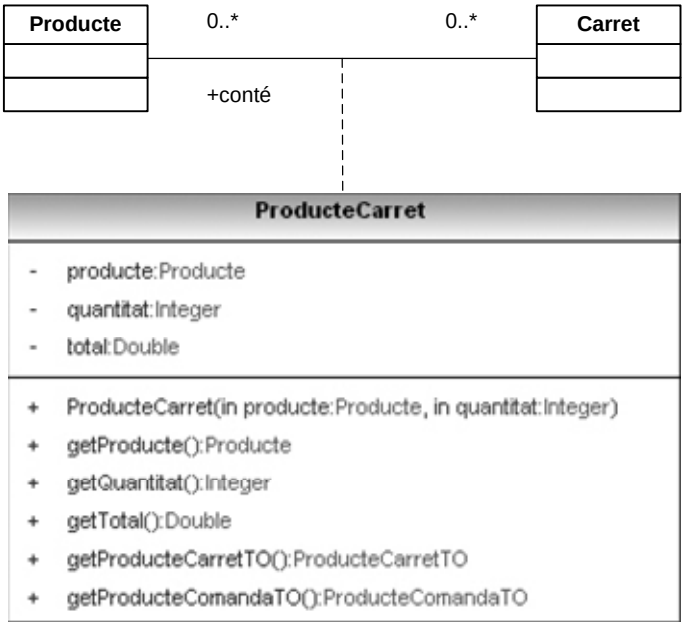


Diagrama estàtic de ProducteCarret

Classes auxiliars, d'utilitat i excepcions

Classes auxiliars

A part de les classes d'entitat i de sessió, s'ha comentat que existeixen altres classes per implementar el model de negoci, com EstatComanda, Mes, Any, etc. Aquestes classes són prou trivials, donat que la seva funció principal és la de proporcionar els elements que conformen les diferents llistes desplegables que es mostren a les pantalles de l'aplicació, però considerem adient comentar que segueixen el mateix esquema de classe de negoci que actua com a *Wrapper* d'una classe *Transfer Object*, com la resta de classes del model. D'aquesta manera el conjunt de classes de la capa de negoci ofereix un aspecte força coherent e intuïtiu, al seguir tots els elements els mateixos mecanismes i convencions.

Classes d'utilitat

La principal funció d'aquestes dades és la validació de dades, que protegeix l'estat dels objectes tant de les entrades incorrectes per part de l'usuari com dels errors de programació. Aquesta tasca es realitza en dues classes. La primera d'elles, ValidacioAlgorismes, conté els diferents algorismes de validació genèrics que s'utilitzen per a tota l'aplicació, de manera que en resulta un component molt apte per a la seva reutilització. La segona, ValidacioUtil, utilitza els algorismes codificats a la primera per tal de verificar les dades i llençar les excepcions pertinents si fos el cas.

Com a mostra dels algorismes implementats, mostrem el diagrama de ValidacioAlgorismes, que presenta uns mètodes prou indicadors de les comprovacions que pot realitzar:

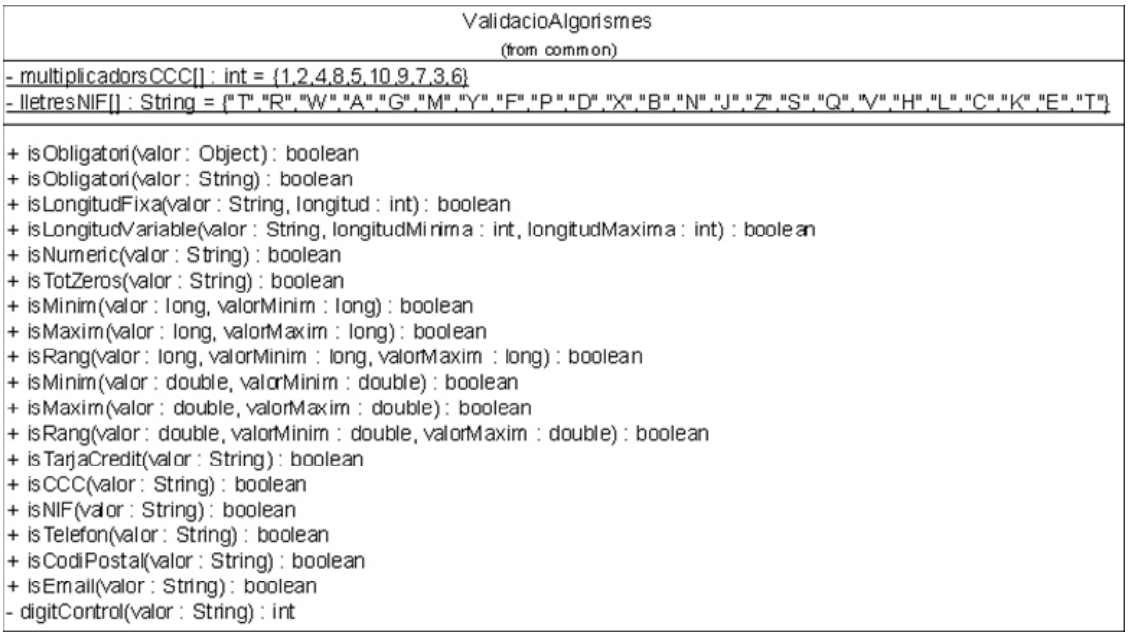


Diagrama estàtic de ValidacioAlgorismes

Les validacions implementades en aquesta classe i utilitzades a ValidacioUtil s'utilitzen tant a les classes de negoci com a les classes Action, resultant en alguns casos en comprovacions redundants al marge dels diversos mecanismes de validació que proporciona Struts. Aquest fet és degut a que les validacions realitzades a les classes més properes a la capa de presentació poden generar missatges més específics i més amigables cap a l'usuari, mentre que les de la capa de negoci necessàriament són més genèrics.

La doble comprovació, en tot cas, no afecta de manera important al rendiment i manté la independència del model de negoci respecte a les interactuacions amb la capa de presentació, a més d'obtenir una seguretat millorada.

Classes d'excepció

Considerant la possible reutilització del sistema de validacions, s'organitza una jerarquia d'excepcions pròpies de l'aplicació, que poden contenir una excepció pròpia del sistema o bé generar-se per una violació d'una regla de negoci. D'aquestes classes cal comentar que estan preparades per interactuar amb el sistema d'auditoria *log4j*, donat que emmagatzemen un missatge amigable per a l'usuari i un missatge de caire més tècnic per al log, a més del nivell de gravetat segons les constants de *log4j*. Al contenir la generació dels missatges en la mateixa excepció s'uniformitzen aquests per a totes les ocasions en que es produeixen, a més de permetre la seva internacionalització fàcilment.

Com en el cas dels algorismes de validació, aquests components estan pensats per la seva reutilització, obviant aquells propis del model de negoci de la present aplicació, que són les classes filles d'ExcepcioNegoci.

Exposem a continuació un diagrama estàtic molt parcial de les classes d'excepció, on s'observa la jerarquia d'aquestes:

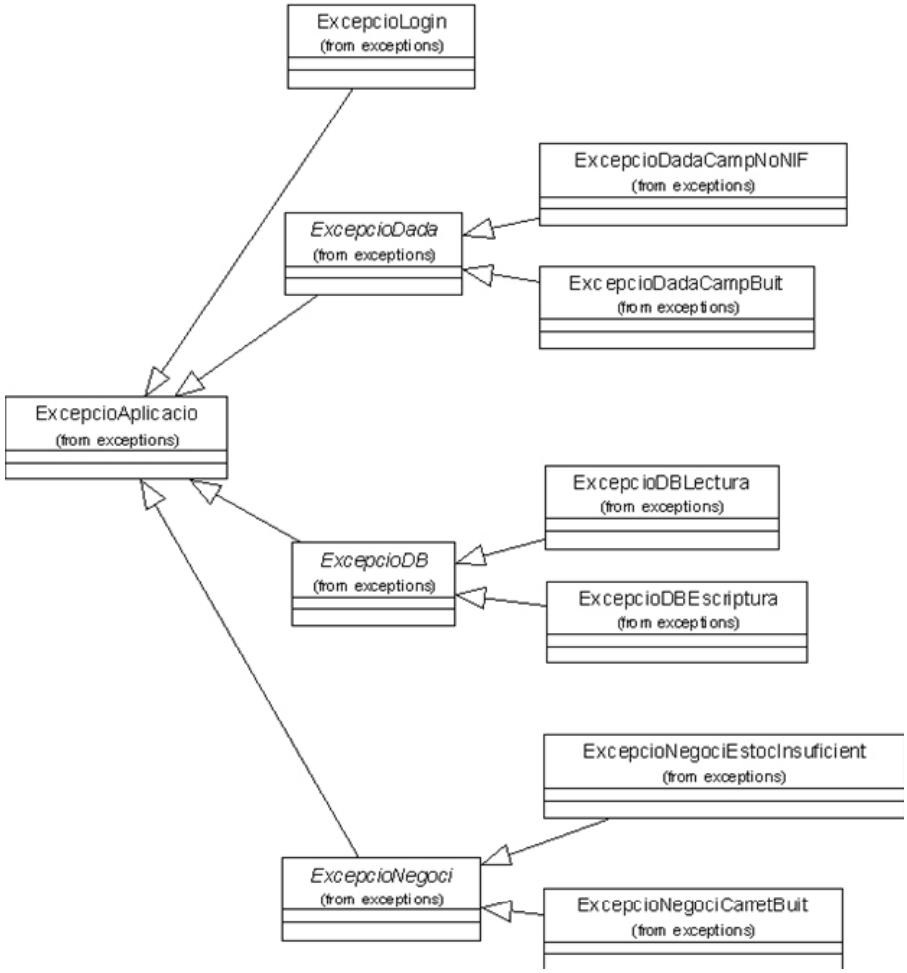


Diagrama estàtic de les classes d'excepció

Capa de presentació

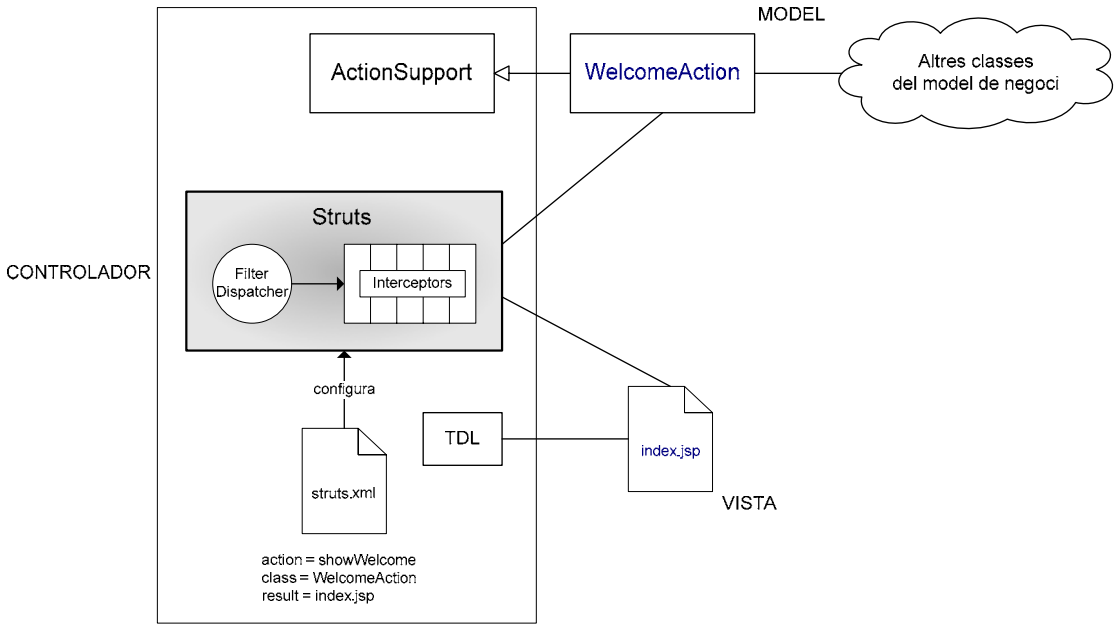
Implementació del patró *Model-View-Controler* amb Struts

La manera més habitual de implementar el patró *Model-View-Controler* en la plataforma J2EE consisteix en l'arquitectura anomenada *Servlet-centered design*, on el *Servlet* realitza les funcions de controlador, les planes JSP les de vista i altres classes Java planes formen el model. Usualment, només existeix un controlador principal per a tota la aplicació, el qual determina quines classes i mètodes del model s'executaran per atendre les peticions dels clients, i cap a quines planes JSP es redireccionarà la resposta.

Existeixen diverses alternatives a l'ús d'aquest patró de disseny, com per exemple encapsular la lògica de negoci en el mateix *Servlet* o bé prescindir de les planes JSP i servir el codi HTML en el mateix *Servlet*. Òbviament, aquestes alternatives no són gens recomanables, motiu pel qual el patró *Model-View-Controler* apareix com la millor solució possible per enllaçar la capa de negoci amb la de presentació.

Un dels principals problemes que sorgeix amb la implementació del patró en les aplicacions orientades a Internet rau en el fet que les dades que indiquen les classes de negoci que s'han d'executar i les planes JSP on dirigir la resposta poden residir en una base de dades o bé estar codificades dins la mateixa aplicació, amb la qual cosa el seguiment de la navegació de l'usuari pot resultar difícil, a més de complicar el manteniment.

Struts és una implementació del patró *Model-View-Controler* en la que el Servlet controlador ja està proporcionat i disponible per la seva immediata utilització. El mapeig entre les classes de negoci i les planes JSP es realitza mitjançant l'arxiu de configuració en XML (struts.xml), tal com es mostra en el següent esquema d'exemple:



Esquema de mapeig classe-JSP amb Struts

En l'arquitectura resultant, el *Servlet* controlador resulta transparent per al desenvolupament i només cal considerar les classes del model, que anomenem classes *Action*, donat que estenen la classe ActionSupport proporcionada per Struts per facilitar el desenvolupament, i les planes JSP que constitueixen les vistes.

Classes Action

Les classes que anomenem *Action* són les gestores de la presentació. Constitueixen el model en el patró *Model-View-Controler* i són les que s'enllacen amb les planes JSP mitjançant el controlador que Struts proporciona.

En l'arquitectura del present projecte, aquestes classes s'encarreguen de cridar i utilitzar les classes de la capa de negoci descrites en l'apartat anterior segons les peticions dels clients, amb la finalitat de mantenir la separació entre la capa de negoci reutilitzable e independent de la presentació i les implementacions dependents de la navegació per l'aplicació. Constitueixen, per tant, el punt d'enllaç entre les capes de negoci i presentació, i es podrien considerar una implementació del patró de disseny *Business Delegate* (J2EE). Així, aquestes classes resulten molt senzilles i són fàcilment actualitzables, donat que tota la lògica de negoci es delega cap a les classes d'entitat i de sessió.

S'ha considerat convenient dividir aquestes classes entre les que gestionen el *front-end* i les que s'encarreguen de les operacions CRUD del *back-end*, malgrat que en tots els casos s'identifiquen pel sufix -Action, donat que s'empren tècniques diferents per la seva implementació.

Així, les classes del *front-end* responen directament a una petició de l'usuari i només executen un mètode per classe, per tal de disposar de diversos identificadors de mapeig que abstreguin la complexitat de la navegació per l'aplicació per part de l'usuari.

En canvi, cada classe del *back-end* s'encarrega del manteniment d'una entitat de negoci. Donat que les operacions CRUD resulten força repetitives (lectura, alta, baixa, modificació, etc.) aquesta tècnica permet centralitzar en una mateixa classe el manteniment complert d'una entitat de negoci, per aprofitar el fet de que la navegació per la zona de manteniment de l'aplicació no presenta cap complexitat.

Planes JSP

La interactuació amb el client s'obté mitjançant l'ús de documents HTML interpretables per un navegador estàndard, els quals, a més contenir hipertext, disposen de formularis amb els elements de control habituals en qualsevol interfície gràfica d'usuari, tal com s'ha observat en l'apartat corresponent del capítol d'Anàlisi. Aquests documents HTML s'obtindran com a resposta a les peticions dels clients que arribin mitjançant el protocol HTTP al contenidor web.

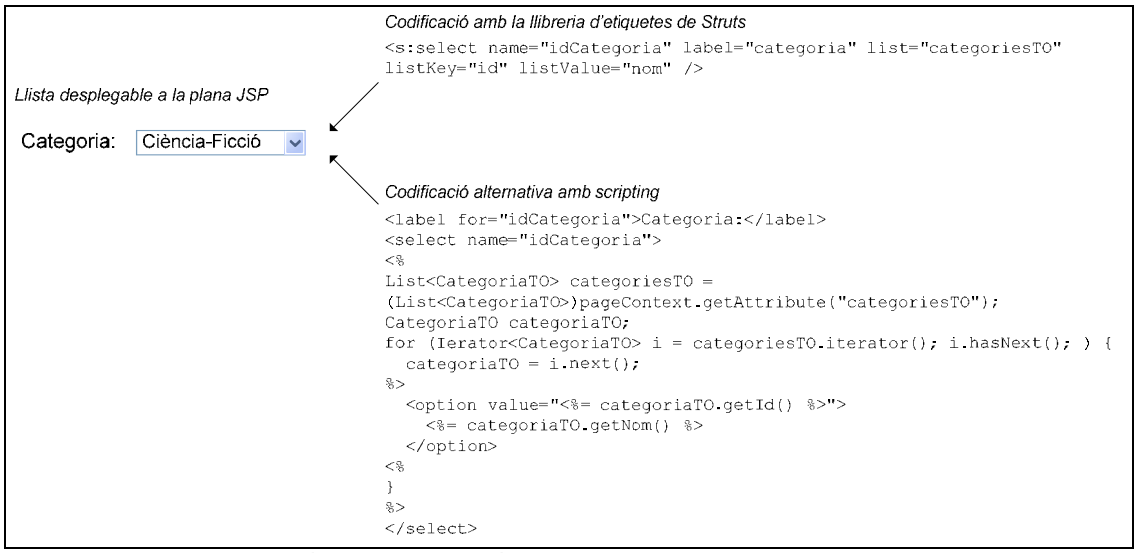
Els documents HTML seran el resultat del procés en la part servidora de planes JSP, les quals permeten la inserció de codi Java en una estructura de document. En rebre la primera petició, la plana JSP es compila i en resulta un *Servlet* tractable pel contenidor i amb les mateixes funcionalitats. D'aquesta manera s'evita l'escriptura de codi HTML dins les classes *Servlet* i s'aconsegueix la separació entre presentació i lògica de negoci. Ja hem vist, però, que en el model *Model-View-Controler* proporcionat per Struts, el *Servlet* controlador resulta completament transparent.

Així, les planes JSP de l'aplicació resulten en general prou senzilles, donat que en una arquitectura J2EE correctament implementada aquestes no han de contenir cap lògica de negoci sinó nomes la codificació mínima per la presentació. Malgrat aquesta simplicitat, existeixen determinats elements dignes de consideració que esmentem tot seguit:

TDLs

Els formularis i els seus elements no s'obtenen directament amb els elements d'HTML corresponent, sinó mitjançant les llibreries d'etiquetes (TDL, *Tag Descriptor Library*) que proporciona Struts, amb la finalitat d'evitar determinades codificacions i alleugerir la plana fins on sigui possible. Per exemple, s'utilitzen etiquetes per mostrar directament una llista desplegable a partir d'un objecte *List*, i per tant estalviem la codificació del bucle i la sortida per pantalla manuals. Es disposa també de mecanismes per avaluar les dades d'entrada del client en el seu mateix navegador mitjançant *JavaScript*. La codificació d'aquestes validacions es genera automàticament mitjançant l'ús de les etiquetes corresponents.

Es mostra a continuació un exemple de les diferències entre l'ús d'etiquetes i la codificació tradicional mitjançant *scripting* per obtenir el mateix resultat:



Esquema d'exemple de les diferències de codificació entre TDLs i scripting

També s'utilitzen les etiquetes pròpies de JSP, que no en formen part actualment de l'especificació, especialment per als elements de controls com comprovacions i presa de decisions, donat que permeten una utilització més fàcil que les de Struts en aquest aspecte. L'ús de totes dues llibreries en la mateixa plana no comporta cap conflicte d'incompatibilitat.

Expression Language

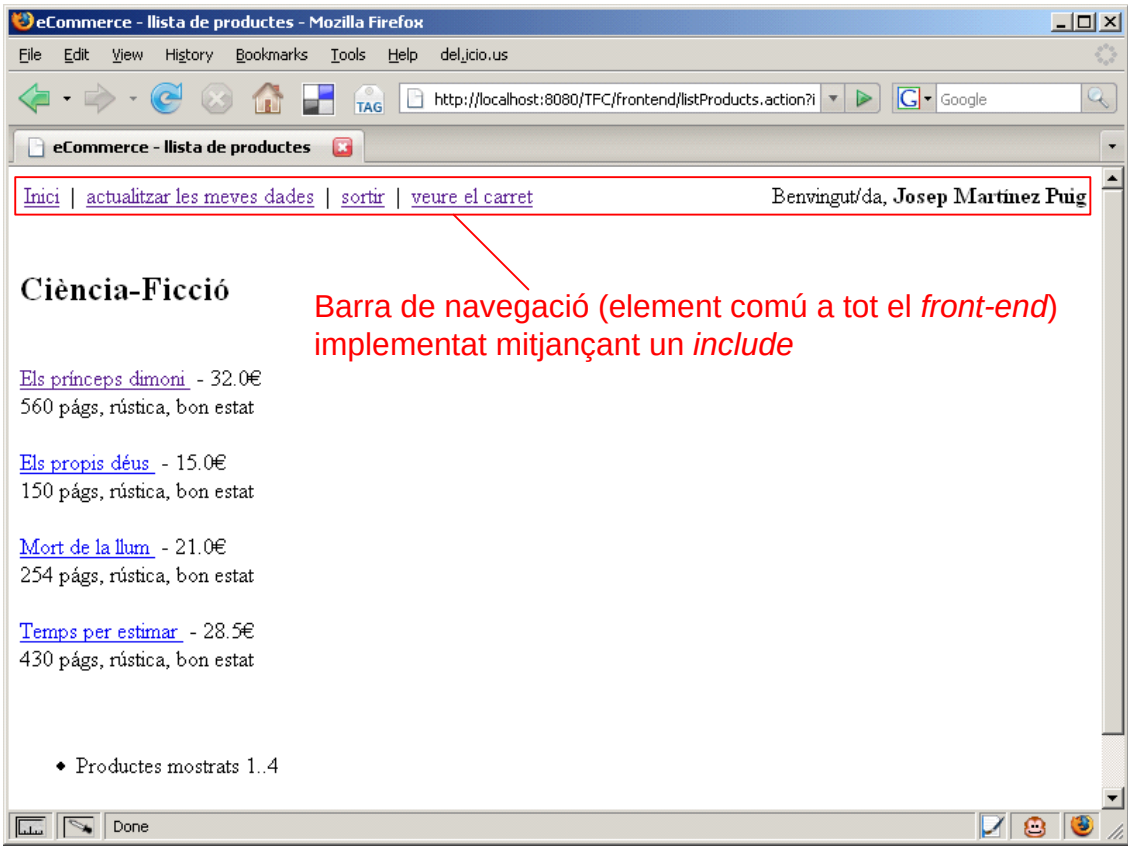
Seguint les pràctiques recomanades, l'escadussera codificació requerida en les planes JSP s'implementa mitjançant EL (*Expression Language)*. La bibliografia afirma que EL resulta de més fàcil utilització per al personal encarregat de la presentació que no ha de conèixer Java necessàriament. Sense contradir aquestes afirmacions, considerem que l'*scripting* ben planificat resulta més flexible i potent, tot i que cal disposar dels coneixements adients.

includes

Per tal de centralitzar les parts comunes de les planes, s'utilitzen fragments de JSP que s'insereixen a la plana principal. Aquests fragments es coneixen com *includes* i els hem caracteritzat amb l'extensió .inc en el sistema d'arxius.

Existeixen diverses tècniques per a la seva implementació. En el present projecte s'ha optat per utilitzar la que uneix el fragment amb la plana en temps de compilació, amb la finalitat d'evitar el redireccionament de la *Request* per tal de no afectar el rendiment. L'inconvenient resideix en que aquesta tècnica obliga a recompilar la plana principal en cas de que es modifiqui el *include* per tal de notificar els canvis. És suficient amb una simple modificació del contingut del JSP per tal que el contenidor recompili la plana el proper cop que la serveixi, per tant els inconvenients derivats són mínims.

Els arxius fragmentaris s'utilitzen per implementar la barra de navegació del *front-end* i per controlar que l'administrador estigui correctament identificat en el sistema abans de mostrar qualsevol plana del *back-end*. En aquest darrer cas el fragment inclòs és invisible.



Plana del font-end que mostra la utilització de includes per implementar elements comuns

Conclusions

Un finalitzada la fase de implementació del present treball i efectuades les proves corresponents per tal de garantir el funcionament esperat, podem considerar que el disseny s'ha vist reflectit en el codi de l'aplicació resultant, sense que en cap cast les circumstàncies concretes de la fase de implementació hagin obligat a obviar cap element del disseny ni a la violació de les regles de l'arquitectura establerta.

Per tant, podem afirmar que la utilització dels bastiments Struts e Hibernate és perfectament compatible amb una arquitectura J2EE formalment correcta que segueixi les pràctiques més recomanades segons l'estat de l'art actual de l'enginyeria del programari.

Tots dos bastiments ofereixen unes funcionalitats madures i suficientment provades, com es demostra pel volum de la comunitat d'usuaris que posseeixen i per la bibliografia existent. Les possibilitats de mapejar mitjançant arxius XML fàcilment actualitzables les planes JSP amb les classes que els tractaran en el cas de Struts, o bé les classes contenidores de dades amb les taules d'una base de dades relacional en el cas de Hibernate, sens dubte alliberen el desenvolupament de tasques monòtones i propenses a errors de codificació. En el cas de Struts, addicionalment, cal considerar la potència dels seus elements auxiliars, com les llibreries d'etiquetes.

Per tant, podem afirmar que la inversió inicial en temps que comporta l'aprenentatge de tots dos bastiments supera amb escreix la rendibilitat que se'n treu.

Evidentment, la utilització d'eines, bastiments, IDEs i altres elements que facilitin un desenvolupament ràpid i que protegeixin dels errors habituals comporta determinats riscos per la implementació d'una arquitectura formalment correcta, especialment en els casos de cicle de vida del programari amb prototipatge. Analitzarem a continuació els riscos que considerem més greus i de més fàcil aparició que se'n deriven de la utilització de Struts e Hibernate.

En el cas de Struts, el risc més evident consisteix en implementar tota la lògica de negoci i l'accés a dades en el mètode d'execució de la classe directament mapejada. Aquesta pràctica incorrecta seria l'equivalent, en l'entorn dels IDEs gràfics, a codificar tots els elements en els mètodes escoltadors dels botons dels formularis. El resultat seria un fort acoblament entre la capa de negoci i la de presentació que dificultaria qualsevol actualització. Per evitar-ho, hem de disposar d'un disseny correcte a la capa de negoci, pensat per la seva reutilització e independència respecte a la capa de presentació, i considerar les classes directament mapejades amb Struts com el punt d'entrada al sistema, seguint el patró de disseny *Business Delegate* (J2EE) o *Facade*(GoF).

De manera similar, en el cas de Hibernate, el risc consisteix en implementar la lògica de negoci en les classes que es mapegen directament amb les taules de la base de dades. La conseqüència seria novament un fort acoblament, aquest cas entre la capa de lògica de negoci i la d'accés a dades. L'escenari es presta a la implementació del patró de disseny *Transfer Object* (J2EE) per les classes directament relacionades amb les taules, amb els beneficis addicionals que se'n deriven per la claredat del codi i la facilitat de transmissió per la xarxa en el cas d'aplicacions distribuïdes.

Malgrat que aquests riscos poden tenir greus conseqüències a llarg termini en la fase de manteniment d'un projecte d'aplicació empresarial, considerem que són fàcilment evitables mitjançant un disseny eficient i el coneixement de les pràctiques recomanades per l'arquitectura J2EE, i per tant no haurien de suposar cap inconvenient per a la utilització dels bastiments.

Glossari

Administrador: en el context de l'aplicació, és el propietari del programari o bé una persona delegada per aquest, amb els privilegis escaients per operar amb el *back-end*.

Back-end: zona privada del programari, només accessible per a l'Administrador, on es realitzen les tasques de manteniment.

Business Delegate: patró de disseny del catàleg J2EE, consistent a introduir classes intermediàries entre els clients i la lògica de negoci com a punts d'entrada a un sistema complex. Similar al patró *Facade* (GoF) però orientat per aplicacions J2EE.

Carret: metàfora usual en el comerç electrònic, consistent en un objecte actiu durant una sessió de treball d'un Usuari en el front-end que actua com a contenidor dels Productes que es compraran i a partir dels quals es generarà una Comanda.

Categoria: en el context de l'aplicació, representa un conjunt de Productes agrupats pels criteris que consideri l'Administrador.

Client: en el context de l'aplicació, representa un usuari enregistrat, del qual es disposa de les seves dades personals a la base de dades

Comanda: compra en ferm d'un conjunt de Productes per part d'un Client en el *front-end* de l'aplicació, que es transmetrà de forma física a la persona i adreces consignades.

Data Access Object (DAO): patró de disseny del catàleg J2EE, consistent en la utilització de classes encarregades de l'accés a base de dades separades de les classes de negoci, per tal de reduir l'acoblament entre els orígens de dades i la lògica de l'aplicació.

Decorator: patró de disseny del catàleg GoF, pel qual una classe actua com embolcall d'una altra per estendre'n la seva funcionalitat, tot delegant les funcions originals a una instància de l'objecte embolcallat.

Facade: patró de disseny del catàleg GoF, pel qual una classe d'utilització amigable actua com a punt d'entrada a un sistema complex, amagant a l'exterior els detalls d'aquesta complexitat.

Front-end: zona pública del programari, accessible per qualsevol usuari de Internet, on es duu a terme l'acció comercial.

Hibernate: Bastiment que, mitjançant arxius XML, mapeja classes Java amb taules d'una base de dades, establint la correspondència entre els atributs de totes dues i proporcionant sistemes per automatitzar les tasques habituals de lectura, actualització, inserció i esborrament.

J2EE (*Java 2 Enterprise Edition*): Conjunt d'especificacions de la plataforma Java per aplicacions empresarials, especialment les orientades a Internet, juntament amb les seves APIs, programari, patrons de disseny, etc.

JSP (*Java Server Pages*): en el context de l'arquitectura J2EE, és un arxiu amb l'estructura d'un document HTML que inclou codi Java, i que és transformat en un *Servlet* automàticament pel contenidor.

Model-View-Controler (MVC): patró de disseny que desacobla la lògica de l'aplicació (el model) de la seva presentació (la vista) introduint un tercer element, el controlador.

Producte: en el context de l'aplicació, són els objectes genèrics comercialitzats mitjançant l'aplicació.

Servlet: en el context de l'arquitectura J2EE, és una classe que s'executa dins un contenidor web i que compleix un contracte que la facilita per atendre peticions HTTP.

Struts: Bastiment que segueix el patró de disseny *Model-View-Controler* per aplicacions J2EE, utilitzat per mapejar mitjançant arxius XML classes Java amb (usualment) planes JSP. S'acompanya de nombrosos elements auxiliars, com TDLs, etc.

TDL (*Tag Descriptor Library*): conjunt d'etiquetes que estenen la funcionalitat de les etiquetes HTML per la programació que contenen, per a ser utilitzades en documents JSP.

Transfer Object (TO): patró de disseny del catàleg J2EE, consistent en la utilització de classes d'estructura molt simple destinades exclusivament a emmagatzemar les dades d'una classe de negoci, per tal de facilitar la seva transmissió per la xarxa entre altres motivacions.

Usuari: en el context de l'aplicació, representa un usuari de Internet anònim que interactua amb el programari, i que pot esdevenir Client en cas de que s'enregistri al sistema.

Wrapper: denominació informal equivalent al patró *Decorator*(GoF), especialment en els casos en que l'objecte embolcallat no és un objecte complex,

Bibliografia

ALUR, Deepak; CRUPI, John; MALKS, Dan; (2003) *Core J2EE™ Patterns: Best Practices and Design Strategies*. Santa Clara (California): Prentice Hall / Sun Microsystems Press.

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert (2004) *Head First Servlets & JSP. Passing the Sun Certified Web Component Developer exam*. Sebastopol (California): O'Reilly Media.

BAUER, Christian; KING, Gavin (2007) *Java persistence with Hibernate*. Cherokee Station (New York): Manning Publications.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar (1999) *El Lenguaje Unificado de Modelado. Manual de referencia*. Madrid: Addison-Wesley.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar (2000) *El proceso unificado de desarrollo de software*. Madrid: Addison-Wesley.

CASTELLS, Manuel (1997) *La era de la información. Economía, sociedad y cultura*. (3 vols.). Madrid: Alianza.

CAVANESS, Chuck (2004) *Programming Jakarta Struts*. Sebastopol (California): O'Reilly Media.

CRAWFORD, William; KAPLAN, Jonathan (2003) *J2EE Design Patterns. Patterns in the real world*. Sebastopol (California): O'Reilly Media.

DUDNEY, Bill; ASBURY, Stephen; KROZAK, Joseph; WITTKOPF, Kevin (2003) *J2EE AntiPatterns*. Indianapolis: Wiley Publishing.

ELLIOTT, James (2004) *Hibernate: A developer's notebook*. Sebastopol (California): O'Reilly Media.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John (1995) *Desing patterns. Elements of reusable Object-Oriented Software*. Indianapolis: Addison-Wesley.

HAY, Kam; ROTH, Mark (2003) *Code Conventions for the Java Server Pages Technology Version 1.x Language.* [article en línia]. Sun. <http://java.sun.com/developer/technicalArticles/javaserverpages/code_convention/index.html> [Data de consulta: 13/11/2007]

HOLMES, James (2007) *Struts: the Complete Reference, Second Edition*. Two Penn Plaza (New York): McGraw-Hill.

JENDROCK, Eric; BALL, Jennifer; CARSON, Debbie; EVANS, Ian; FORDIN, Scott; HAASE, Kim (2006) *The Java EE 5 Tutorial*. Santa Clara (California): Sun Microsystems Press.

JOHNSON, Robert (2005, gener) "J2EE development frameworks". Computer (vol.38, nº1, pàg. 107-110).

KING, Peter; NAUGHTON, Patrick; DEMONEY, Mike; KANERVA, Jonni; WALRATH, Kathy; HOMMEL, Scott *et al*. (1999) *Code conventions for the Java™ Programming Language*. [article en línia]. Sun. <http://java.sun.com/docs/codeconv/index.html> [Data de consulta: 01/10/2007]

KUMAR, Vinod (2007) *Struts 2 Tutorial*. [tutorial en línia]. Rose India. <http://www.roseindia.net/struts/struts2/index.shtml> [Data de consulta: 10/11/2007]

LARMAN, Craig (1997) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Upper Saddle River (New Jersey): Prentice Hall.

NAFTALIN, Maurice; WADLER, Philip (2006) *Java generics and Collections*. Sebastopol (California): O'Reilly Media.

NAUGHTON, Patrick; SCHILDT, Herbert (1997) *Java. Manual de referencia*. Madrid: McGraw-Hill.

SUHA, Deepak (2007) *Complete Hibernate 3 Tutorial*. [tutorial en línia]. Rose India. <http://www.roseindia.net/hibernate/index.shtml> [Data de consulta: 10/11/2007]

WANG, Jia-Shun, CHEN, Jia (2006) "The research and application on Business Tier Based on J2EE" A: *International Conference on Computational Inteligence for Modelling Control and Automation*. Dalian: Dalian University.

WHEELER, David (2003) *Secure programmer: validating input. Best practices for accepting user data*. [article en línia]. IBM.<http://www.ibm.com/developerworks/linux/library/l-sp2.html> [Data de consulta: 23/11/2007]

WONG, Wang-chan; WYADAT, Mohammad; NEG, Shane (2006) "Degree of Freedom - Experience of Applying Software Framework" A: *Third International Conference on Information Technology* (pàgs. 143-148). San Francisco: California State University.

Annexos

Annex I - notes de instal·lació i programari utilitzat

Seguint les especificacions de l'arquitectura J2EE, l'aplicació es transporta mitjançant un arxiu .WAR que compren tant el codi font com el codi objecte, les llibreries i tots els arxius de configuració necessaris pel seu correcte funcionament.
Per la instal·lació, nomes cal deixar aquest arxiu dins el directori /webapps de Apache Tomcat i procedir a l'arrencada d'aquest.

En un servidor Apache Tomcat configurat per defecte, les URLs per accedir a l'aplicació són:
- *front-end*: http://localhost:8080/TFC/showWelcome.action
- *back-end*: http://localhost:8080/TFC/backend/showFormLoginAdmin.action

Les planes d'entrada ja porten anotats els noms d'usuari i contrasenya vàlids per accedir a totes les funcionalitats. En tot cas, d'usuari i contrasenya de l'administrador es pot modificar a web.xml, i els dels clients mitjançant els formularis de manteniment dels seus comptes.

La configuració de la base de dades en MySQL és la següent:
- URL: jdbc:mysql://localhost
- user: jordi
- password: 1234
- schema: TFC

Aquesta configuració es pot modificar, si cal, a l'arxiu hibernate.cfg.xml

Les classes de l'aplicació s'estructuren en els següents paquets:
- edu.uoc.eCommerce.actions : classes Action genèriques.
- edu.uoc.eCommerce.actions.front : classes Action del front-end.
- edu.uoc.eCommerce.actions.common : classes de utilitat per log i validació.
- edu.uoc.eCommerce.actions.entitites : classes que encapsulen la lògica de negoci, normalment cada entitat disposa de la seva classe d'entitat, una classe TO (Transfer Object) que és encapsulada per aquesta, i una classe DAO (Data Access Object) per l'accés a base de dades mitjançant Hibernate.
- edu.uoc.eCommerce.actions.exceptions : classes d'excepció, que representen els diversos errors que es poden produir durant el funcionament de l'aplicació.
- edu.uoc.eCommerce.actions.entitites : conté un experiment amb *listeners*, que treu els paràmetres de la *Request* al log.
- edu.uoc.eCommerce.actions.test : proves molt parcials e incomplertes d'accés a dades mitjançant les classes TO e Hibernate.

Les planes JSP del front-end s'ubiquen immediatament sota el directori /pages. Les del back-end es localitzen a /pages/back, que al seu cop conté directoris amb les planes per al tractament de cada entitat de negoci.

Per tal de garantir la correcta utilització en un entorn en el que les especificacions canvien ràpidament, fem constar les versions dels diversos programaris utilitzats en el desenvolupament::
- JDK 6.0.3
- Apache Tomcat 6.0.14
- MySQL 5.0.45 amb drivers JDBC 5.1.15
- Struts 2.0.11
- Hibernate 3.2.5
- Eclipse Lomboz 3.3

Annex II - Patrons de disseny en l'aplicació

Detallem a continuació els patrons de disseny del catàleg J2EE que considerem més importants per comprendre l'arquitectura de l'aplicació i per mostrar com aquesta segueix les pràctiques recomanades per al desenvolupament d'aplicacions empresarials. Aquests patrons són el DAO (*Data Access Object*) i el *Transfer Object*. La seva importància en el disseny de l'aplicació segons els objectius del projecte es deu al fet que són els més directament relacionats amb la utilització dels bastiments Struts e Hibernate.

Obviem de la present relació altres patrons del mateix catàleg també utilitzats, com *Composite Entity* o *Business Delegate*, donat que la seva participació és molt menor en l'esquema general de l'arquitectura.

Data Acces Object (DAO)

Context

L'accés a les dades varia en funció de les fonts d'aquestes. L'accés a l'emmagatzematge persistent, com és una base de dades, canvia molt depenent del tipus d'emmagatzematge (bases de dades relacionals, bases de dades orientades a objecte, arxius de text, etc.) i la implementació del fabricant.

Problema

Moltes aplicacions J2EE al món real necessiten utilitzar dades persistents en algun moment. Per moltes aplicacions, l'emmagatzematge persistent s'implementa amb diferents mecanismes, i hi ha marcades diferències a les APIs utilitzades per accedir aquests diferents sistemes d'emmagatzematge. Altres aplicacions poden necessitar dades que resideixen a sistemes separats. Per exemple, les dades poden residir en sistemes *mainframe*, repositoris LDAP, etc. Altres exemples el constitueixen quan les dades són proveïdes per sistemes externs com sistemes d'integració B2B (*Business-to-Business*), serveis de targes de crèdit, etc.

Típicament, les aplicacions utilitzen components distribuïts com *Entity Beans* per representar dades persistents. Es considera que una aplicació empra persistència gestionada pel bean (BMP: *Bean Managed Persistence*) per als seus *Entity Beans* quan aquests *Entity Beans* accedeixen explícitament a les dades persistents, així, l'*Entity Bean* inclou codi per accedir directament a les dades emmagatzemades. Una aplicació amb requeriments més simples pot obviar l'ús d'*Entity Beans* i en el seu lloc utilitzar *Session Beans* o *Servlets* per accedir a la capa de dades i directament efectuar lectures i modificacions. O bé l'aplicació pot emprar *Entity Beans* amb persistència gestionada pel contenidor (CMP: *Container Managed Persistence*) i per tant deixar al que el contenidor gestioni els detalls de la persistència.

Les aplicacions poden utilitzar l'API JDBC per accedir a les dades residents en un sistema de gestió de bases de dades relacional (SGBDR). L'API JDBC permet un accés i manipulació estandarditzat de les dades de la capa de persistència, com és el cas d'una base de dades relacional. L'API JDBC permet a les aplicacions l'ús de sentències SQL, que constitueixen el sistema estandarditzat d'accedir a les taules d'un SGBDR. Però, fins i tot a l'entorn d' un entorn SGBDR, la sintaxi i format de les sentències SQL poden variar en funció de cada fabricant.

Hi ha encara una major varietat en els diferents tipus d'emmagatzematge de dades. Mecanismes d'accés, APIs, i funcionalitats poden variar entre els diferents sistemes d'emmagatzematge com SGBDRs, sistemes de gestió de bases de dades orientades a objecte,

arxius de text, etc. Les aplicacions que necessiten accedir dades de sistemes dispars es poden veure obligades a emprar APIs que poden ser propietàries. Aquests sistemes de dades dispars poden potencialment crear una dependència directa entre el codi de l'aplicació i el codi d'accés a les dades. Quan els components de negoci com *Entity Beans* o *Session Beans*, o fins i tot els components de presentació com els Servlets o els JSP necessiten accedir a les dades poden utilitzar l'API requerida per aconseguir la connectivitat i la manipulació de les dades. Però incloure el codi per la connectivitat i l'accés a les dades dins aquests components introdueix un fort acoblament entre els components i la implementació de la capa de dades. Aquestes dependències de codi en els components fa que sigui difícil migrar l'aplicació d'un tipus d'emmagatzematge de dades a un altre. Quan l'accés a dades canvia, els components han de ser modificats per poder tractar el nou tipus d'accés a dades.

Forces

- Components com *Entity Beans* amb persistència gestionada pel *Bean*, *Session Beans*, *Servlets* i altres objectes necessiten recuperar i emmagatzemar informació de la capa de persistència, que pot estar composada de diverses fonts de dades com SGBDRs, sistemes propietaris B2B, LDAP, etc.
- Les APIs d'emmagatzematge persistent canvien en funció del fabricant. Altres fonts de dades poden tenir APIS que no siguin estàndard o bé que siguin propietàries. Aquestes APIs i les seves capacitats també varien en funció del tipus d'emmagatzematge: SGBDRs, SGBDOOs, documents XML, arxius de text, etc. Hi ha una carència de APIs uniformitzades per gestionar els requeriments d'accés a sistemes tan dispars.
- Els components típicament utilitzen APIs propietàries per accedir a sistemes externs per recuperar i emmagatzemar dades.
- La portabilitat dels components es veu directament afectada quan sistemes específics d'accés a les dades s'inclouen directament en aquests.
- Els components necessiten ser transparents respecte a la implementació d'accés a les dades per tal d'aconseguir una fàcil migració entre diferents fabricants, diferents sistemes d'emmagatzematge i diferents orígens de dades.

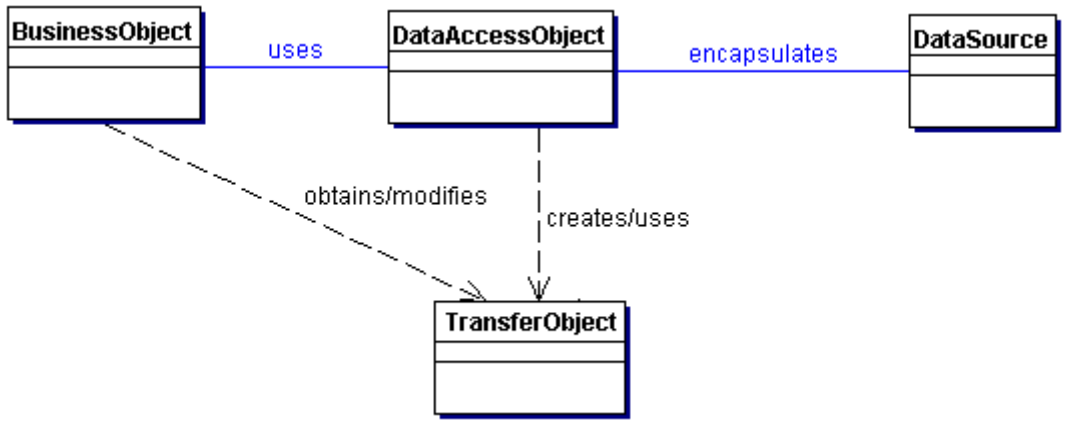
Solució

Utilitzar un *Data Access Object* (DAO) per abstraure i encapsular tots els accessos als orígens de dades. El DAO gestiona la connexió amb l'origen de dades per obtenir i emmagatzemar dades.

El DAO implementa el mecanisme d'accés requerit per treballar amb els orígens de dades. Els orígens de dades poden ser bases de dades relacionals, serveis externs com un intercanviador B2B, un repositori com una base de dades LDAP, o un servei de negoci al qual s'accedeix via CORBA o *sockets* de baix nivell. El component de negoci que es recolza en el DAO utilitza la interfície exposada al DAO pels seus clients. El DAO amaga completament la implementació dels orígens de dades als seus clients. Donat que la interfície exposada pel DAO als seus clients no canvia quan la implementació dels orígens de dades ho fa, aquest patró permet que el DAO s'adapti a diferents sistemes d'emmagatzematge sense afectar als seus clients o als components de negoci. Essencialment, el DAO és un *Adapter* entre el component i l'origen de dades.

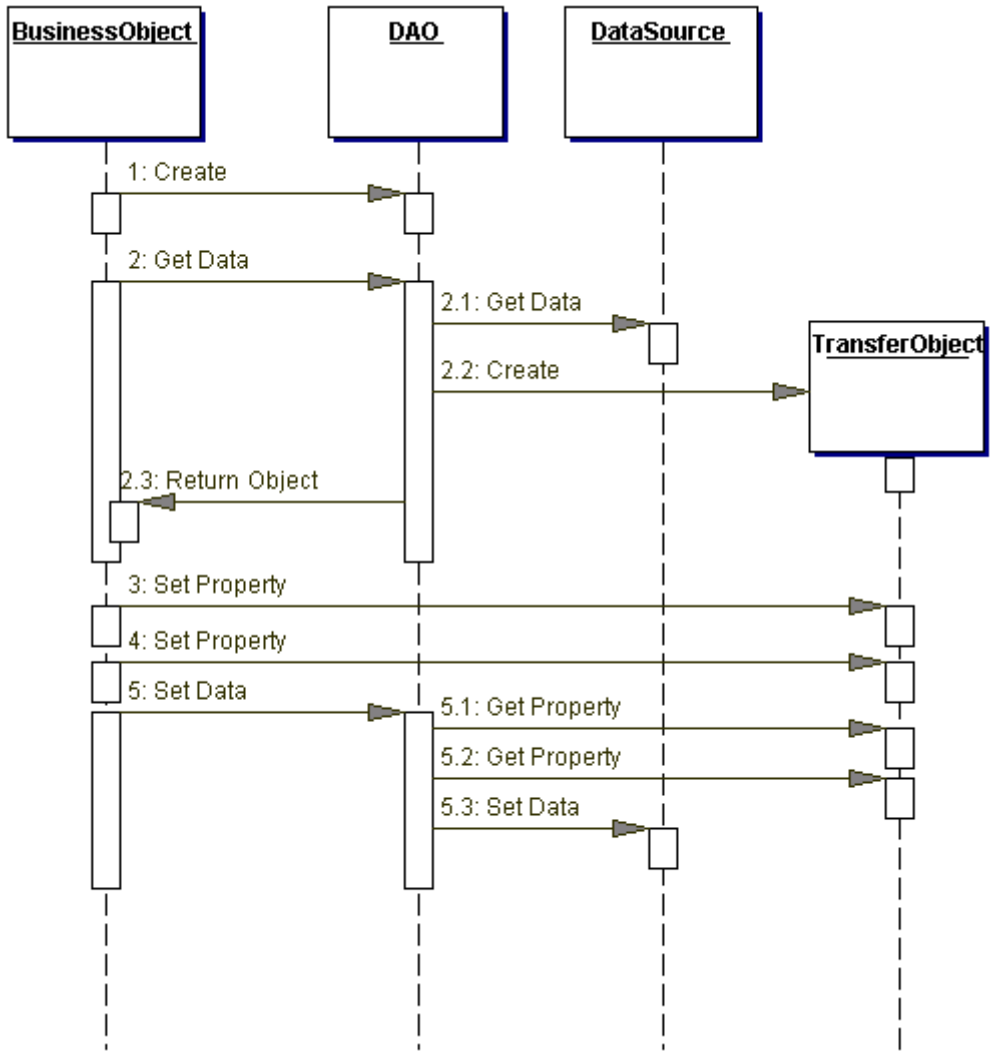
Estructura

Es mostra el diagrama de classes que representa les relacions d'un DAO:



Participants i responsabilitats

Es mostra el diagrama de seqüència que mostra les interaccions entre els diferents participants d'aquest patró



BusinessObject
El *BusinessObject* representa les dades del client. És l'objecte que requereix l'accés a l'origen de dades per obtenir i emmagatzemar dades. Un BusinessObject pot ser implementat per un Session Bean, Entity Bean, o qualsevol altre objecte Java, a més de un Servlet o Bean auxiliar que requereixi accés a les dades.

DataAccessObject
El *DataAccessObject* és l'objecte principal d'aquest patró. El *DataAccessObject* abstrau la implementació d'accés a les dades pel *BusinessObject* per permetre l'accés transparent als orígens de dades. El *BusinessObject* delega la càrrega i emmagatzematge de dades al *DataAccessObject*.

DataSource
Representa la implementació d'un origen de dades. Un origen de dades pot ser una base de dades d'un SGBDR, SGBDOO, un repositori XML, un arxiu de text, etc. Un origen de dades pot ser també un altre sistema, un servei extern, o algun tipus de repositori com LDAP.

TransferObject
Representa el *TransferObject* utilitzat pel transport de dades. El *DataAccessObject* pot utilitzar un *TransferObject* per retornar les dades al client. El *DataAccessObject* por, a més, rebre les dades del client en un *TransferObject* per actualitzar les dades a l'origen de dades.

Conseqüències

- Permet transparència: Els objectes de negoci poden emprar els orígens de dades sense conèixer els de detalls específics de la implementació d'aquests. L'accés és transparent donat que els detalls de la implementació s'amaguen dins el DAO.

- Permet migracions fàcils: Una capa DAO facilita que una aplicació pugui migrar a una diferent implementació de la base de dades. Els objectes de negoci no tenen coneixement de la implementació de l'accés a dades. Per tant, la migració implica canvis només a la capa DAO.
- Redueix la complexitat del codi als objectes de negoci: Donat que el DAO tracta totes les complexitats de l'accés a dades, simplifica el codi als objectes de negoci i altres clients de dades que utilitzen el DAO. Tot el codi relacionat amb la implementació, com les sentències SQL, es contenen en el DAO i no en els objectes de negoci. Això facilita la legibilitat del codi i la productivitat del desenvolupament.
- Centralitza tots els accessos a dades en una capa separada: Donat que totes les operacions d'accés a dades es deleguen als DAOs, la capa d'accés a dades es pot contemplar com la capa que aïlla la resta de l'aplicació de la implementació de l'accés a dades. Aquesta centralització fa que l'aplicació sigui més fàcil de mantenir i tractar.
- Afegeix una capa extra: Els DAOs creen una capa addicional d'objectes entre les dades dels clients i els orígens de dades que ha de ser dissenyada e implementada per tal d'obtenir els beneficis d'aquest patró, però es considera que el benefici obtingut per escollir aquesta aproximació compensa aquest esforç addicional.

Ús del patró en l'aplicació

En el present cas trobem que l'origen de dades és únic i està completament predeterminat pels requeriments, i a més no és probable que sorgeixi la necessitat d'efectuar alguna migració en aquest aspecte. A més, l'accés a dades s'efectua mitjançant Hibernate, que en una visió

general de l'arquitectura actuaria com una capa DAO la qual resultaria completament transparent a l'aplicació respecte a l'origen de dades, sempre i quan aquests origen sigui una de les bases de dades que pugui tractar Hibernate.

Sense obviar aquests elements, considerem que una arquitectura correcta ha de preveure els escenaris descrits en aquest patró, com poden ser la necessitat de migrar els orígens de dades o bé l'existència d'un origen diferent de les dades emprades a l'aplicació. D'altra banda, cal no oblidar els aspectes que poden ser determinats i objectius en tots els escenaris possibles, com són la claredat del codi i les interrelacions amb els altres patrons emprats, com és el cas de *TransferObject*.

Per tant, hem optat per la implementació del patró DAO per encapsular els accessos a dades. Així, a cada classe d'entitat que requereixi accés a la persistència li correspon una classe DAO que gestiona les transaccions amb Hibernate. El fet de seguir aquesta nomenclatura facilita la localització del codi, a més de efectivament separar la lògica de negoci de l'accés a dades, amb el resultat d'obtenir unes classes de mida tractable i correctament estructurades, amb mètodes de poca extensió que es concentren en la lògica del negoci o en la delegació de l'accés a les dades.

Totes les classes DAO hereten d'una classe abstracta, *AbstractDAO*, que proporciona l'obtenció de sessions de treball amb Hibernate, per tal de reutilitzar codi i centralitzar aspectes com l'auditoria o la gestió d'errors. Dins d'una classe DAO resideixen tots els mètodes necessaris per crear, llegir, actualitzar i esborrar les dades corresponents als objectes de negoci. La comunicació amb les classes de negoci s'efectua mitjançant l'ús del patró TransferObject per tal d'alleugerir el pes de la comunicació i simplificar els accessos a dades, és a dir, que per exemple, podem prescindir de mètodes d'actualització per tots i cadascun dels diversos camps d'una classe d'entitat.

Cal considerar que en l'aplicació existeixen dades persistents que no resideixen a la base de dades, com poden ser el nom d'usuari i la contrasenya d'administrador, que s'emmagatzemen com paràmetres de l'aplicació a l'arxiu de configuració de l'aplicació web web.xml. En aquest cas, donat que el seu accés és molt puntual i la forma d'accedir està descrita a l'especificació J2EE, a més de no vincular-se directament amb cap classe de negoci, no hem considerat adient desenvolupar un DAO específic per aquestes dades.

Finalment, cal assenyalar que existeixen transaccions que impliquen diversos objectes de negoci, com per exemple la conversió del contingut d'un Carret de la compra en una Comanda, que no només implica la creació de la Comanda sinó també la disminució automàtica dels estocs disponibles dels productes comprats. Per aquests casos hem considerat la possibilitat d'utilitzar mètodes a les classes de negoci als quals se'ls hi passaria com a paràmetre la sessió de Hibernate per tal que actuessin dins una transacció general més àmplia i externa a l'objecte. Per exemple, el mètode que crea la Comanda al DAO de Comanda seria l'encarregat 'obtenir la sessió de Hibernate i aniria cridant a un mètode de disminuir els estocs existent a cada producte del Carret el qual rebria com a paràmetre la sessió de Hibernate, i d'aquesta manera es durien a terme totes les actualitzacions de manera transaccional.

Finalment, hem considerat que per tal d'acomplir els objectius de total aïllament de l'accés a dades que proposa el patró DAO no poden existir mètodes als objectes de negoci que rebin com a paràmetre una sessió de Hibernate. Cal considerar, per exemple, que en cas de canviar l'accés a les dades aquest paràmetre podria ser una instància de *java.sql.Connection*, un *java.io.File*, o qualsevol altre objecte diferent, i per tant els canvis a l'accés a les dades afectarien a les classes de negoci obligant a una nova implementació. Per tant, totes les classes DAO efectuen les transaccions de manera atòmica segons les dades que reben per paràmetre en els seus mètodes, i les sessions de treball de Hibernate s'obren i es tanquen

exclusivament dins aquests mètodes sense que l'objecte Session que representa la sessió de Hibernate es passi per referència fora del context d'una classe DAO.

Transfer Object (TO)

Context

Els clients de l'aplicació necessiten intercanviar dades amb els components de negoci.

Problema

Les aplicacions J2EE implementen els components de negoci de la banda del servidor com *Session Beans*, *Entity Beans* o bé objectes Java simples (POJO, *Plain Old Java Object*) . Alguns mètodes exposats pels components de negoci retornen dades als clients. Sovint, el client invoca els mètodes accesors d'un objecte de negoci diversos cops fins que obté tots els valors dels atributs.

Els *Session Beans* representen els serveis de negoci i no es comparteixen entre usuaris. Un *Session Bean* proveeix mètodes de servei detallats quan s'implementa amb el patró *Session Facade*.

Per altra part, els *Entity Beans* són objectes multiusuari i transaccionals que representen dades persistents. Un *Entity Bean* exposa els valors dels atributs proveint un mètode accesor (és a dir, els getters) per cada atribut que desitja exposar.

Cada crida a un mètode de l'objecte de negoci, ja sigui un *Entity Bean* o un *Session Bean*, és potencialment remot. Per tant, en una aplicació EJB aquestes invocacions remotes utilitzen la capa de xarxa sense considerar la proximitat del client al *Bean*, creant una sobrecàrrega a la xarxa. Les crides a mètodes de components de negoci poden traspassar les capes de xarxa del sistema fins i tot si el client i el contenidor EJB que emmagatzema l'*Entity Bean* estan tots dos funcionant sota la mateixa JVM, sistema operatiu i màquina física. Alguns fabricants poden implementar mecanismes per reduir aquesta sobrecàrrega utilitzant un accés més directe i sobrepassant la xarxa.

A mesura que s'incrementa la utilització d'aquests mètodes remots, el rendiment de l'aplicació es va degradant significativament. Per tant, utilitzar múltiples crides als mètodes getter per obtenir simples valors d'atributs és ineficient per obtenir els valors de les dades d'un *Enterprise Bean*.

Forces

Tots els accesors a un *Enterprise Bean* es realitzen per les interfícies remotes del *Bean*. Cada crida a un *Enterprise Bean* és potencialment una crida remota a un mètode amb sobrecàrrega de xarxa.

Típicament, les aplicacions tenen una freqüència més elevada de transaccions de lectura que de transaccions d'actualització. El client requereix les dades de la capa de negoci per presentació, mostrar detalls i altres tipus de processos de només lectura. El client actualitza les dades de la capa de negoci amb molta menys freqüència amb la que llegeix les dades.

El client normalment requereix valors per més d'un atribut o objecte depenent d'un *Enterprise Bean*. Per tant, el client por invocar múltiples crides a mètodes per obtenir les dades requerides.

El nombre de crides que fa el client al component de negoci repercuteix en el rendiment de xarxa. Les aplicacions que tenen un elevat tràfic de dades entre les capes client i de negoci sovint degraden el rendiment de la xarxa.

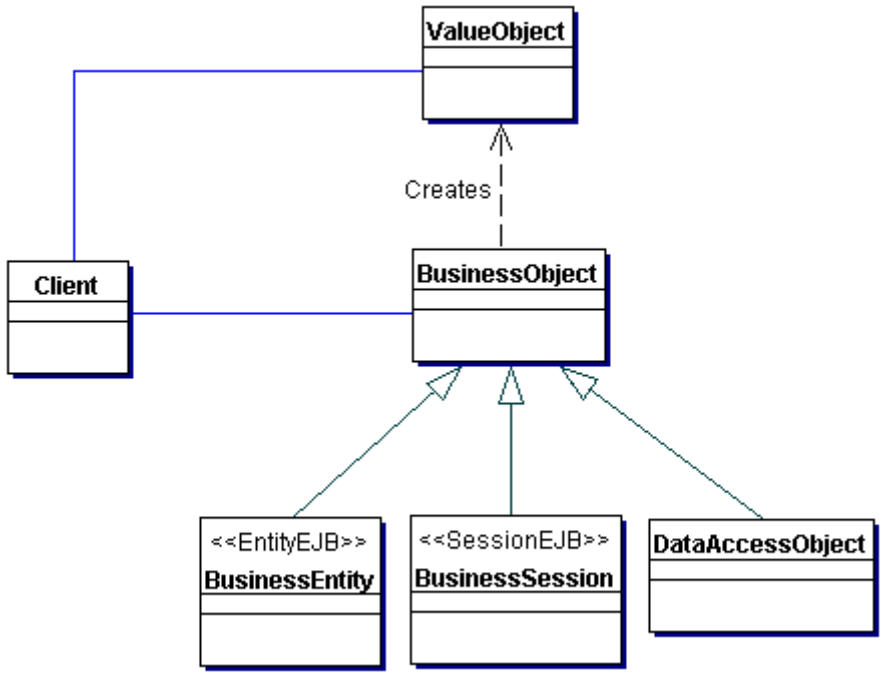
Solució

Utilitzar un *Transfer Object* per encapsular les dades de negoci. S'utilitza una única crida a mètode per enviar i rebre el *Transfer Object*. Quan el client requereix el component de negoci per les dades de negoci, aquest pot construir el *Transfer Object*, omplir-lo amb els valors dels seus atributs, i passar-lo per valor al client.

Quan un component de negoci utilitza un *Transfer Object*, el client fa una sola crida remota de invocació al mètode d'aquest en comptes de nombroses crides remotes per obtenir valors individuals d'atributs. El component de negoci construeix una nova instància del *Transfer Object*, copia els valors dins l'objecte i el retorna al client. El client rep el *Transfer Object* i llavors pot utilitzar els mètodes accesors d'aquest per obtenir els valors individuals dels atributs. O bé el *Transfer Object* pot estar implementat de tal manera que faci els atributs públics. Donat que el *Transfer Object* es passa per valor al client, totes les crides a la instància del *Transfer Object* són locals en comptes d'invocacions remotes a mètode.

Estructura

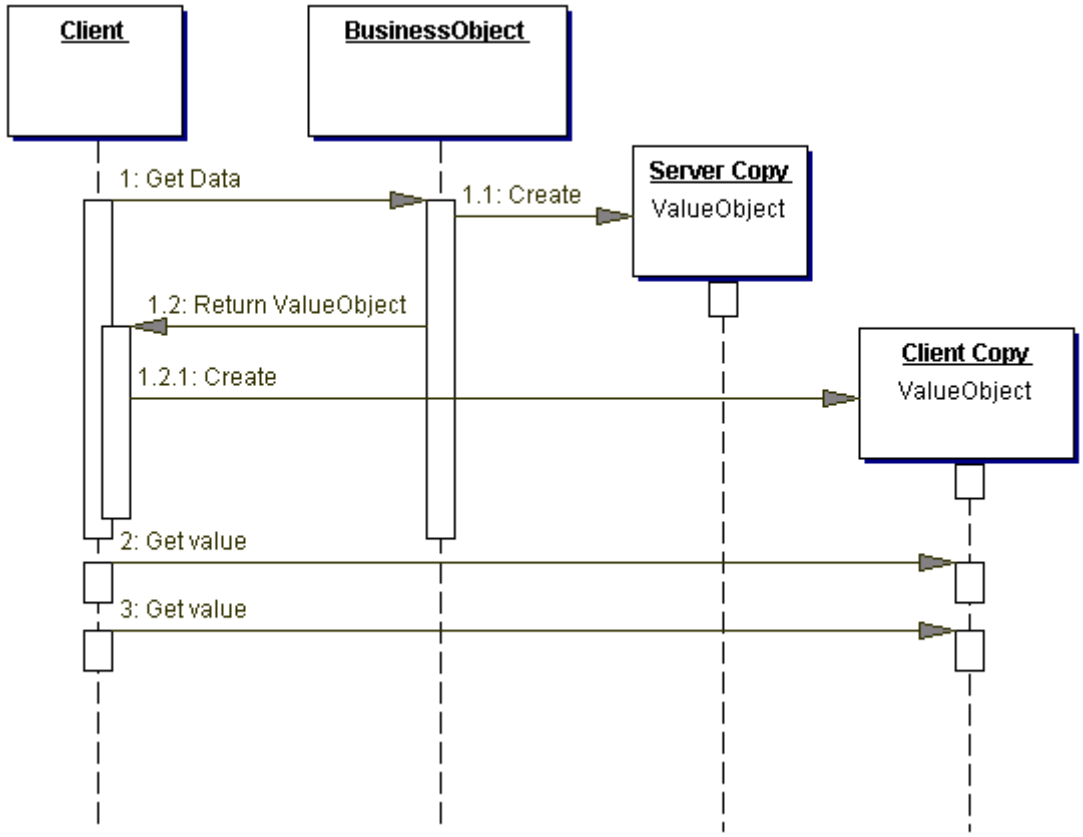
Es mostra a continuació el diagrama de classes que representa el *Transfer Object* en la seva forma més simple:



Com es mostra en el diagrama de classes, el Transfer Object es construeix per petició pel component de negoci i es retorna al client remot. No obstant això, existeixen diferents estratègies per implementar aquest patró.

Participants i responsabilitats

Es mostra el diagrama de seqüència amb les interaccions del patró *Transfer Object*:



Client
Representa el client del component de negoci. El client pot ser una aplicació d'usuari final, com és el cas d'una aplicació client dissenyada per accedir directament als components de negoci. El client pot ser també un Business Delegate o un altre objecte de negoci.

Business Object
El *Business Object* representa un rol en aquest patró que pot ser complert per un *Session Bean*, un *Entity Bean*, o un *Data Access Object* (DAO). El *Business Object* és responsable de crear el *Transfer Object* i retornar-lo al client segons la seva petició. El *Business Object* a més pot rebre dades del client en la forma d'un *Transfer Object* i utilitzar aquestes dades per efectuar una actualització.

Transfer Object
El *Transfer Object* és un objecte Java arbitrari serialitzable referit com a *Transfer Object*. Una classe *Transfer Object* pot proveir un constructor que accepti tots els atributs requerits per crear el *Transfer Object*. El constructor pot acceptar tots els valors dels atributs d'un *Entity Bean* que els *Transfer Object* ha d'emmagatzemar segons el seu disseny. Normalment, els membres d'un *Transfer Object* es defineixen com públics, eliminant així la necessitat de mètodes accesors. Si cal algun grau de protecció, llavors els membres poden ser definits com privats o protegits, i es

proveeix a l'objecte de mètodes accesors. Si no s'ofereixen mètodes accesors d'escriptura per fixar els valors, un *Transfer Object* està protegit davant possibles modificacions després de la seva creació. Si només es permet la modificació d'uns pocs membres per tal de facilitar les actualitzacions, llavors es poden proveir d'accesors d'escriptura per aquests membres. Per tant, la creació d'un *Transfer Object* varia en funció dels requeriments de l'aplicació. És una elecció de disseny decidir si els atributs d'un *Transfer Object* són privats i accessibles mitjançant mètodes accesors, o bé tots els atributs es fan públics.

Ús del patró en l'aplicació

En el present cas, donat que els components de negoci no són *Entity Beans* ni *Session Beans*, semblaria que l'ús d'aquest patró aportaria una càrrega de complexitat innecessària en el disseny. A més, cal considerar que tota la part servidora de l'aplicació resideix en la mateixa màquina, funcionant sota la mateixa JVM i el mateix SO, i no es preveu la utilització de cap sistema d'accés remot a dades (RMI, CORBA, etc.). Per tant, no existeix més tràfic de xarxa que el que es produeix entre el servidor i el client per HTTP. Així, la principal motivació que justifica l'ús d'aquest patró està, en aquest cas, absent.

Malgrat això, hem considerat dos factors que fan molt recomanable el seu ús. En primer lloc, les classes amb les que interactua Hibernate estan obligades a definir uns atributs que es corresponguin amb els camps de la taula de la base de dades que representen, juntament amb els seus mètodes accesors de lectura i escriptura. Aquesta unitat atòmica representa un registre de la base de dades i es pot conservar com a referència directa d'aquesta representació de les dades si no afegim camps calculats ni regles de negoci. Així, en la seva forma més simple que permet una interactuació directa amb base de dades, aquestes classes compleixen la definició de *Transfer Object* en la modalitat definida amb accesors de lectura i escriptura.

Les classes de negoci, que complirien el rol dels *Entity Beans* en la definició de Transfer Object segons el catàleg J2EE, actuen com un *Wrapper* de les classes amb les que interactua Hibernate, afegint regles de negoci, comprovacions i validacions, ordenacions de llistes, camps calculats, mètodes de creació, actualització, esborrat, etc. és a dir, esdevenen un Decorator (GoF) dels *Transfer Object*. D'aquesta manera aconseguim una separació entre les dades primàries que provenen de la base de dades i les regles de negoci, fet que sens dubte facilita el manteniment i la legibilitat global de l'aplicació.

En segon lloc, malgrat que, com hem assenyalat anteriorment, no es preveu cap tràfic de xarxa entre les classes de negoci i els seus clients, considerem que una aplicació correctament dissenyada ha de preveure aquesta possibilitat. Per exemple, és força habitual en les aplicacions distribuïdes dividir la part servidora entre els components de negoci i els *Servlets* i JSP, residint totes dues parts en màquines diferents connectades per xarxa. També es podria seguir l'estratègia de dividir la part de negoci entre les classes de negoci pròpiament dites i els DAO, en un escenari hipotètic en el que aquestes darreres classes operin des de una màquina propera a una altra en la que residiria el SGBDR i els components de negoci estarien més a prop del client. En tots dos casos, que no han de ser considerats com mútuament excloents, la utilització del patró *Transfer Object* facilitaria les possibles adaptacions.

Per tant, hem optat per la implementació del patró *Transfer Object* en l'aplicació. Les classes T*ransfer Object* són les que s'intercanvien entre els components de negoci i els DAO per una banda, i entre els mateixos components de negoci i els JSP per una altra. En aquest darrer cas aconseguim simplificar el nombre de propietats que ha de tenir presents un JSP per les lectures i actualitzacions, obtenint el benefici addicional de simplificar el codi i fer-los més tractables. Determinats components de negoci disposen de més d'una classe de *Transfer Object*, a més de la que representa el seu registre en base de dades, donat que en determinats casos la capa

client ha d'accedir a camps calculats que no apareixen en el *Transfer Object* original. En comptes d'afegir majors complicacions, aquest esquema de funcionament allibera als JSPs de la necessitat d'obtenir valors per camps calculats i beneficia l'objectiu d'aconseguir la màxima simplificació possible de la capa client.

Pel que fa a la implementació, seguint les recomanacions d'ús del patró hem fet que totes les classes *Transfer Object* implementin la interfície *Serializable*, per tal de preveure la seva serialització i deserialització en el context d'una aplicació distribuïda. A més, també implementen la interfície *Cloenable*, per tal de poder utilitzar clons fàcilment dels objectes que es recuperen de la base de dades mitjançant els mètodes de Hibernate. Amb aquesta tècnica, aquests clons són utilitzables tant per lectura com per escriptura fora del context d'una sessió de treball de Hibernate, de tal manera que les operacions d'accés a la base de dades queden centralitzades exclusivament dins les classes DAO.

Anàlisi, disseny e implementació d'una aplicació de comerç electrònic en la plataforma J2EE amb Struts e Hibernate

Jordi Raya Lapuente
ETIG

Albert Grau Perisé

14/01/2008

- El comerç electrònic
  - L'economia global comporta l'èxit del comerç electrònic
  - El comerç electrònic beneficia a les dues parts implicades
  - Les organitzacions demanen aplicacions de comerç electrònic segures i eficients
  - Les aplicacions de comerç electrònic depenen de la tecnologia existent

- El context tecnològic
  - Irrupció de Internet
  - Aparició de les aplicacions distribuïdes
  - Èxit de la programació orientada a l'objecte
  - Acceptació universal de UML per l'enginyeria del programari
  - Ús dels patrons de disseny de programari

- La plataforma J2EE
  - Està especialment preparada per treballar amb Internet
  - Està dissenyada per construir aplicacions empresarials fiables, escalables i eficients
  - Disposa d'àmplia documentació sobre les pràctiques que han demostrat la seva eficiència
  - Disposa de bastiments (*framewoks*) per facilitar el desenvolupament ràpid

- Els bastiments de codi obert
  - Faciliten el desenvolupament ràpid
  - Alliberen el desenvolupament de tasques repetitives i propenses a generar errors
  - Ofereixen funcionalitats ja provades en molts altres projectes
  - Requereixen una inversió inicial en temps per tal de comprendre el seu funcionament
  - La seva utilització pot comportar pràctiques poc recomanables

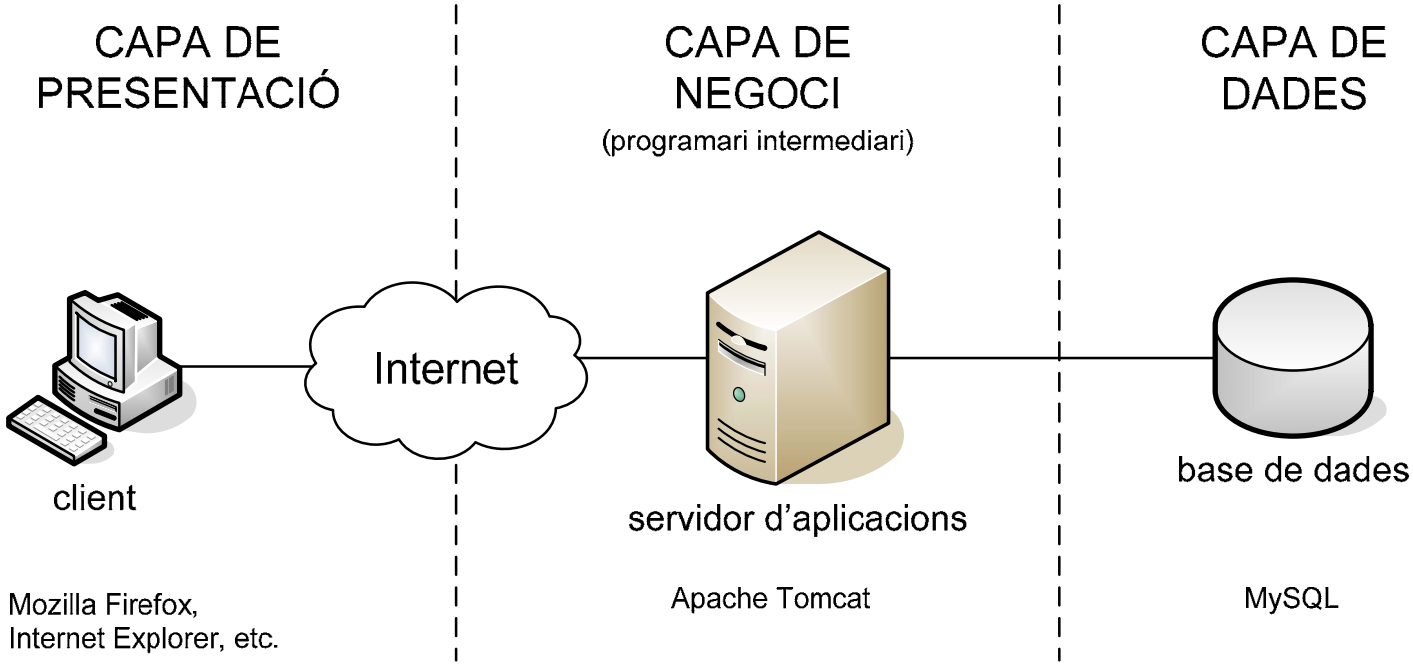
- Objectius del present treball
  - Desenvolupar una aplicació de comerç electrònic en la plataforma J2EE
  - Utilitzar una arquitectura el més correcta possible segons l'estat de l'art
  - Utilitzar els bastiments Struts e Hibernate
  - Estudiar, avaluar i resoldre els possibles conflictes entre el disseny de l'arquitectura i la implementació amb els bastiments

- Enfocament i metodologia
  - Arquitectura a tres capes
  - Ús de tècniques estandarditzades: diagrames UML, diagrames E-R, etc.
  - Ús de patrons de disseny de programari

- Planificació
  - Es seguiex el cicle de vida del programari amb prototipatge
  - Fases:
    - presa de requeriments
    - anàlisi
    - disseny
    - implementació del prototipus
    - implementació final

- Requeriments funcionals
  - L'aplicació consistirà en una botiga virtual que segueixi les convencions d'aquestes aplicacions
  - S'utilizaran les metàfores habituals: *carret de la compra*, *compte de client*, etc.
  - Disposarà d'una part pública o *front-end* accesible als usuaris de Internet
  - Disposarà d'una part privada o *back-end* per les tasques d'administració i manteniment

- Requeriments tecnològics
  - Es seguirà l'arquitectura a tres capes



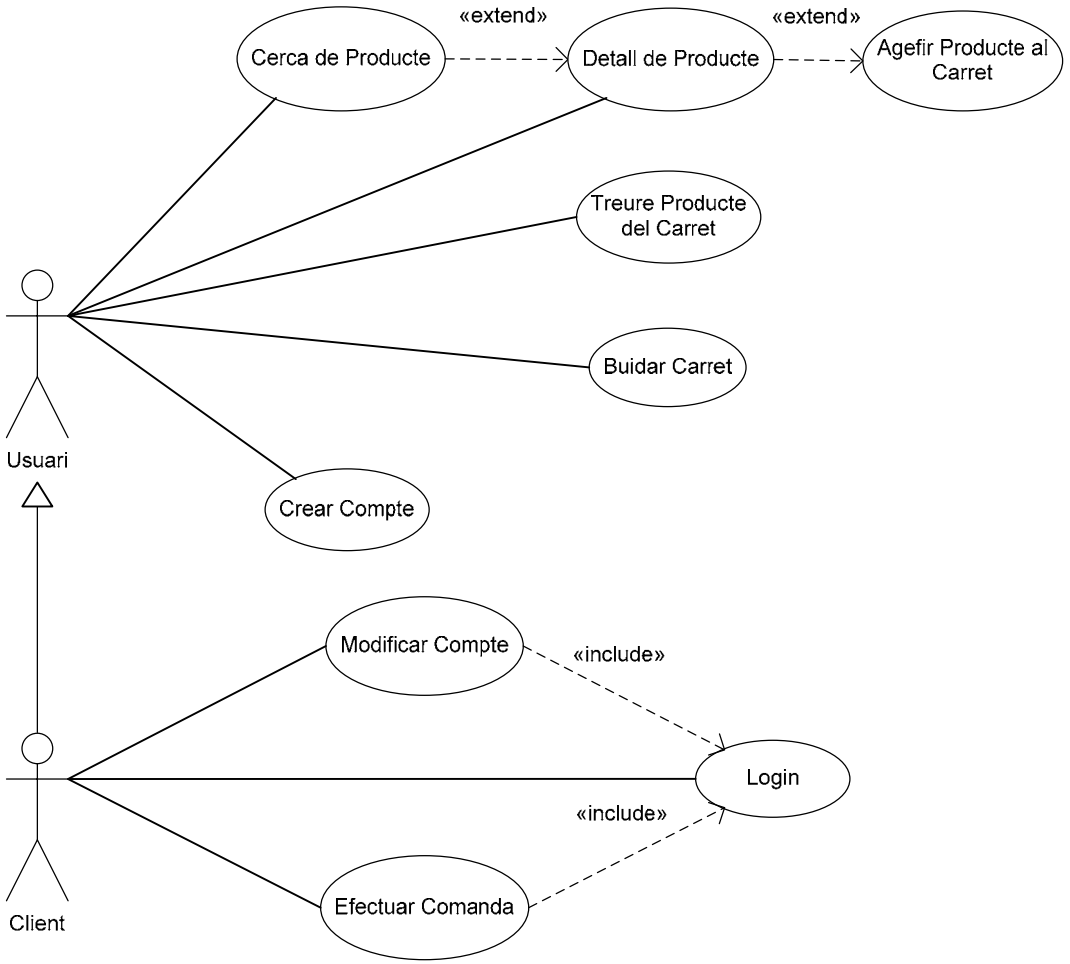
- Capa client
  - Presenta les dades en documents HTML
  - S'utilitza un navegador web estàndard
- Capa de negoci
  - Conté el programari de l'aplicació
  - Utilitza els bastiments per comunicar-se amb les altres capes
  - S'executa en un contenidor web, Apache Tomcat
- Capa de dades
  - Conté les dades de l'aplicació
  - S'utilitza el SGBD MySQL, accesible per JDBC

- Els bastiments en la lògica de negoci
  - Struts comunica la capa de negoci amb la capa de presentació
  - Hibernate comunica la capa de negoci amb la capa de dades

- Rols en l'aplicació
  - Administrador
    - propietari de la botiga o personal delegat facultat per realitzar les tasques de manteniment
  - Usuari
    - usuari anònim de Internet que accedeix a l'aplicació
  - Client
    - usuari registrat en l'aplicació facultat per exercir l'acció comercial

- # Casos d'ús per usuari i client

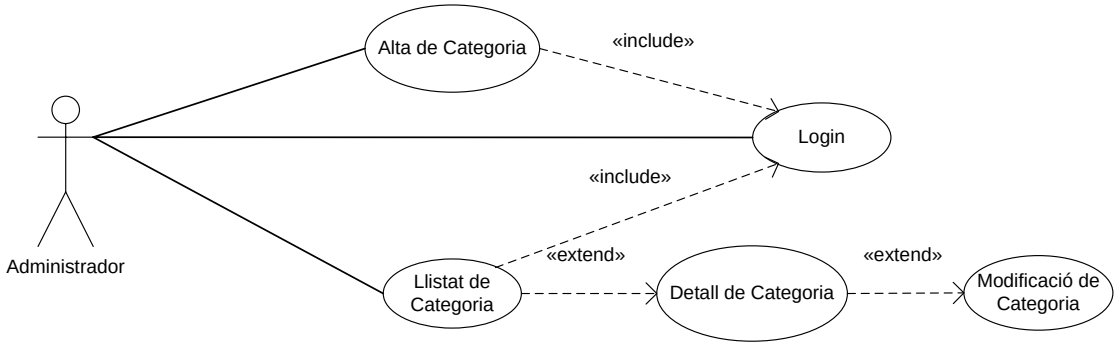
  Corresponen a les accions habituals dels usuaris d'una botiga virtual



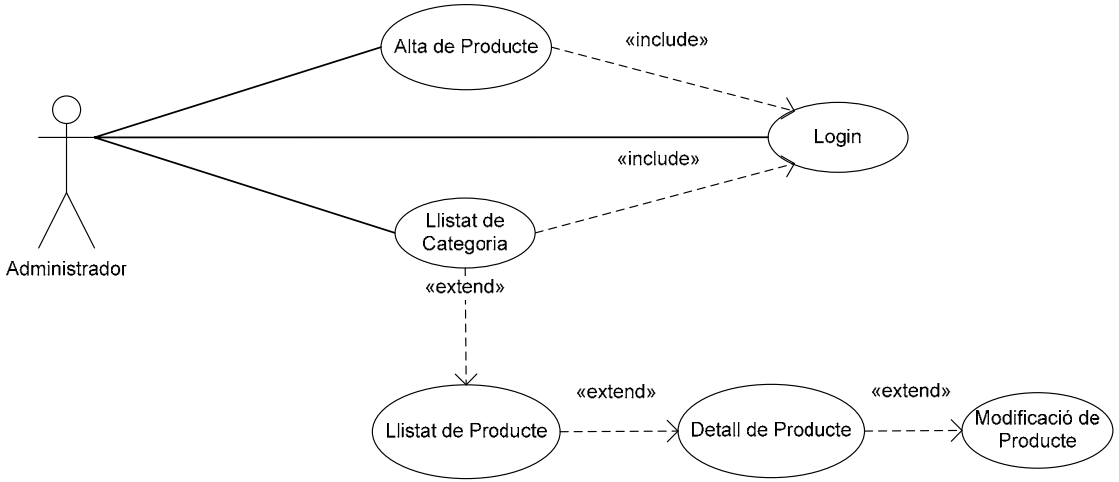
Casos d'ús per administrador

Corresponen a les accions CRUD (Create-Read-Update-Delete) de les entitats del model de negoci

Per categories



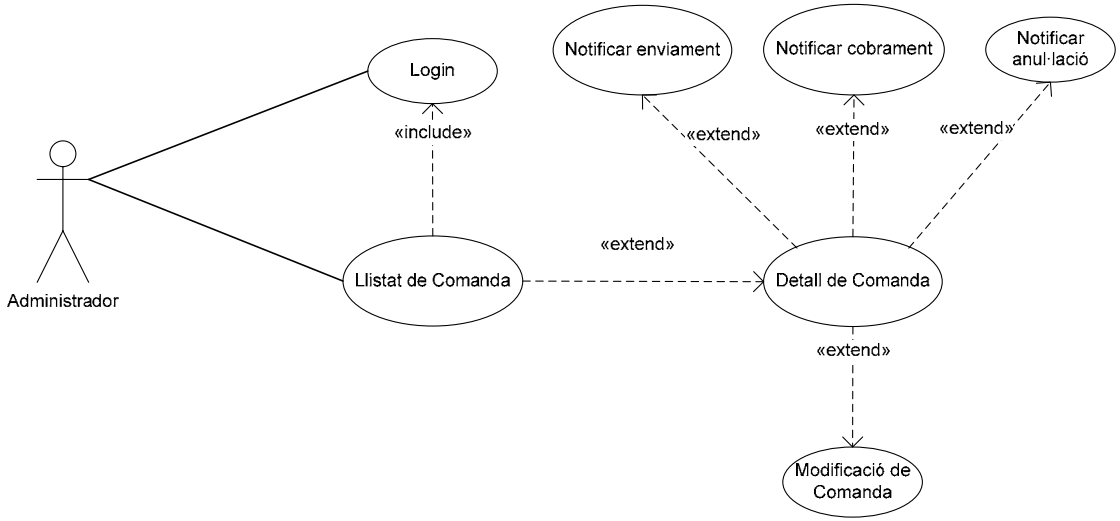
Per productes



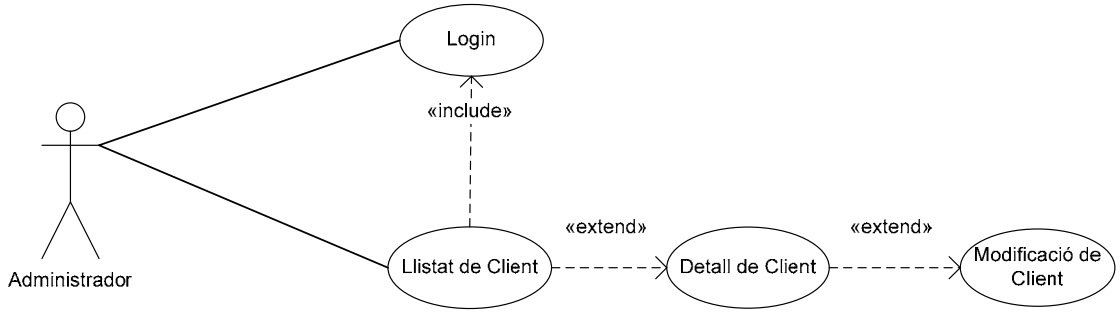
Casos d'ús per administrador II

Corresponen a les accions CRUD (Create-Read-Update-Delete) de les entitats del model de negoci

Per comandes



Per clients



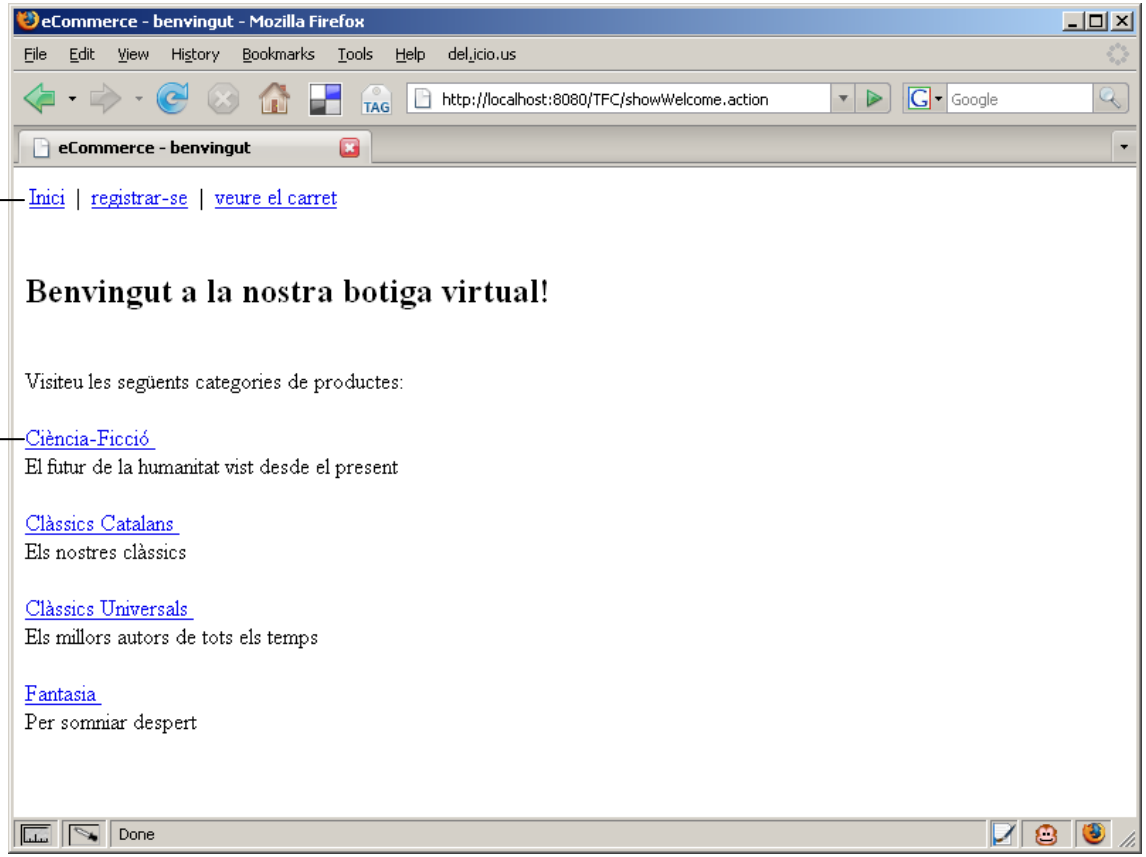
- Interfície Gràfica d'Usuari
  - Es generarà amb planes JSP (*Java Server Pages*) que es transformaran en documents HTML comprensibles pels clients
  - S'aprofitaran els recursos del bastiment Struts
  - Serà bàsica sense considerar els elements estètics ni d'usabilitat
  - Tampoc es consireren les qüestions de web semàntica ni els estàndards W3C

Interfície Gràfica d'Usuari II

Front-end: pantalla d'inici

Barra de navegació per a un usuari no registrat

Llistat de categories de productes



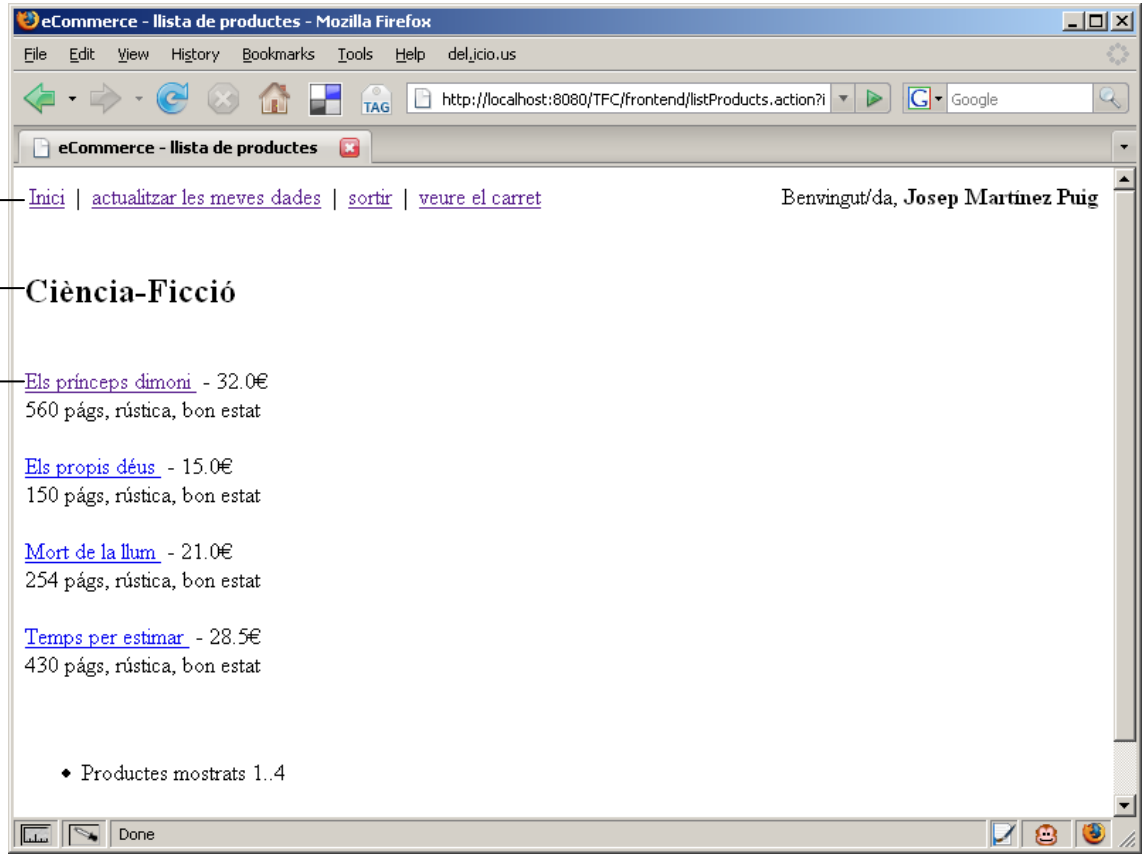
Interfície Gràfica d'Usuari III

Front-end: continguts de categoria

Barra de navegació per a un usuari registrat

Títol de la categoria

Contingut de la categoria



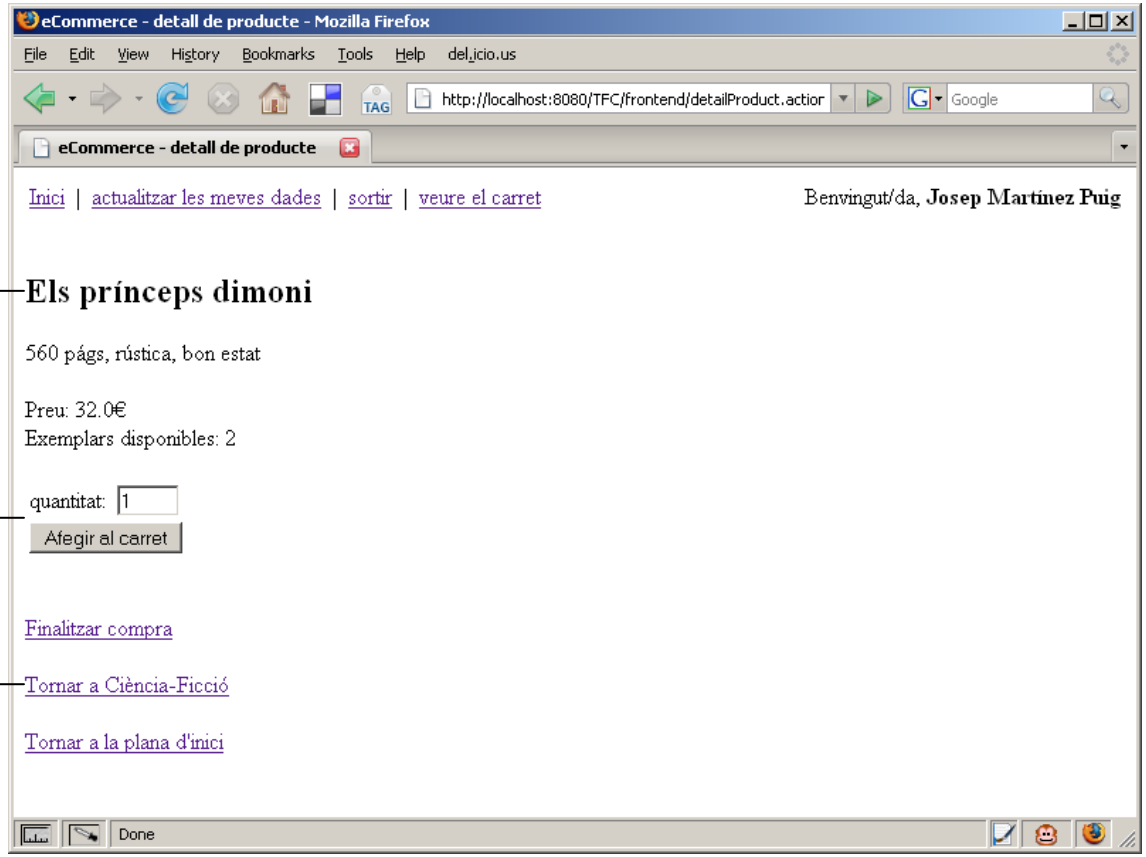
Interfície Gràfica d'Usuari IV

Front-end: detall de producte

Dades públiques del producte escollit

Elements de control de formulari

Enllaços de navegació



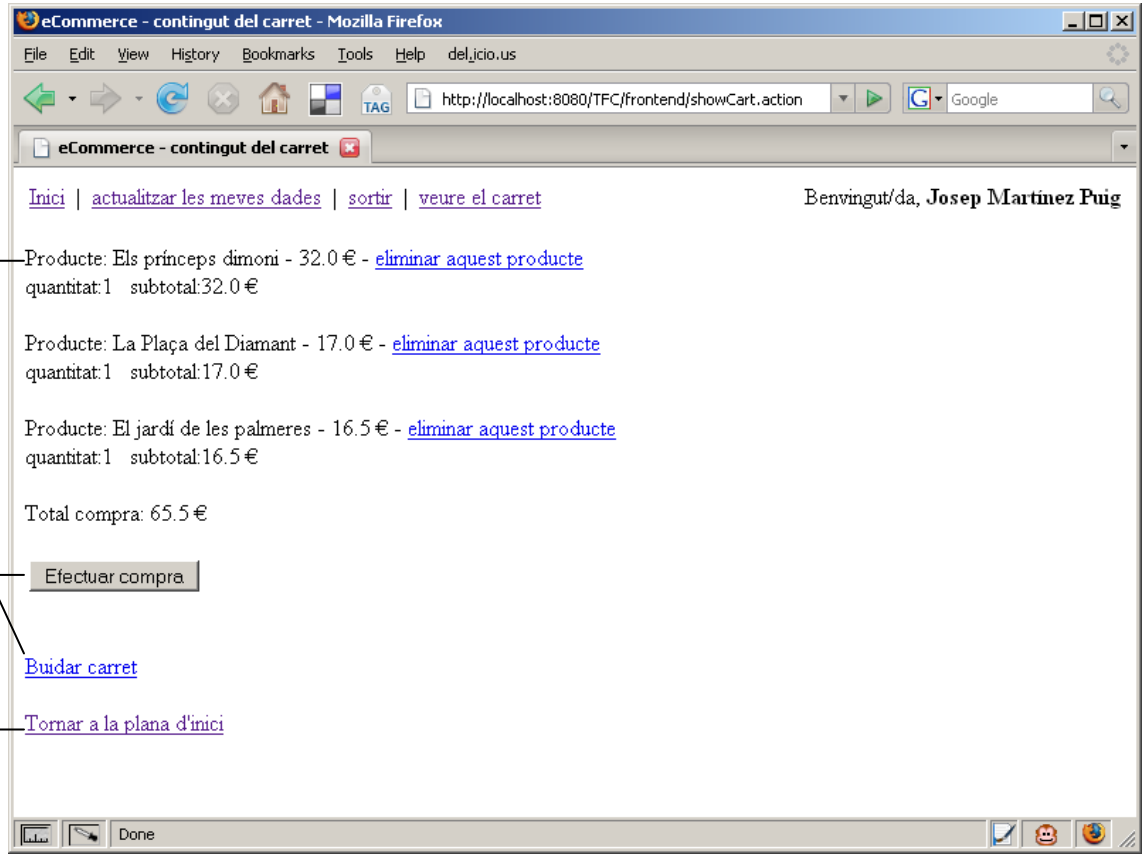
Interfície Gràfica d'Usuari V

Front-end: continguts del carret

Accions sobre el carret

Botó per transformar el carret en una comanda

Enllaços de navegació

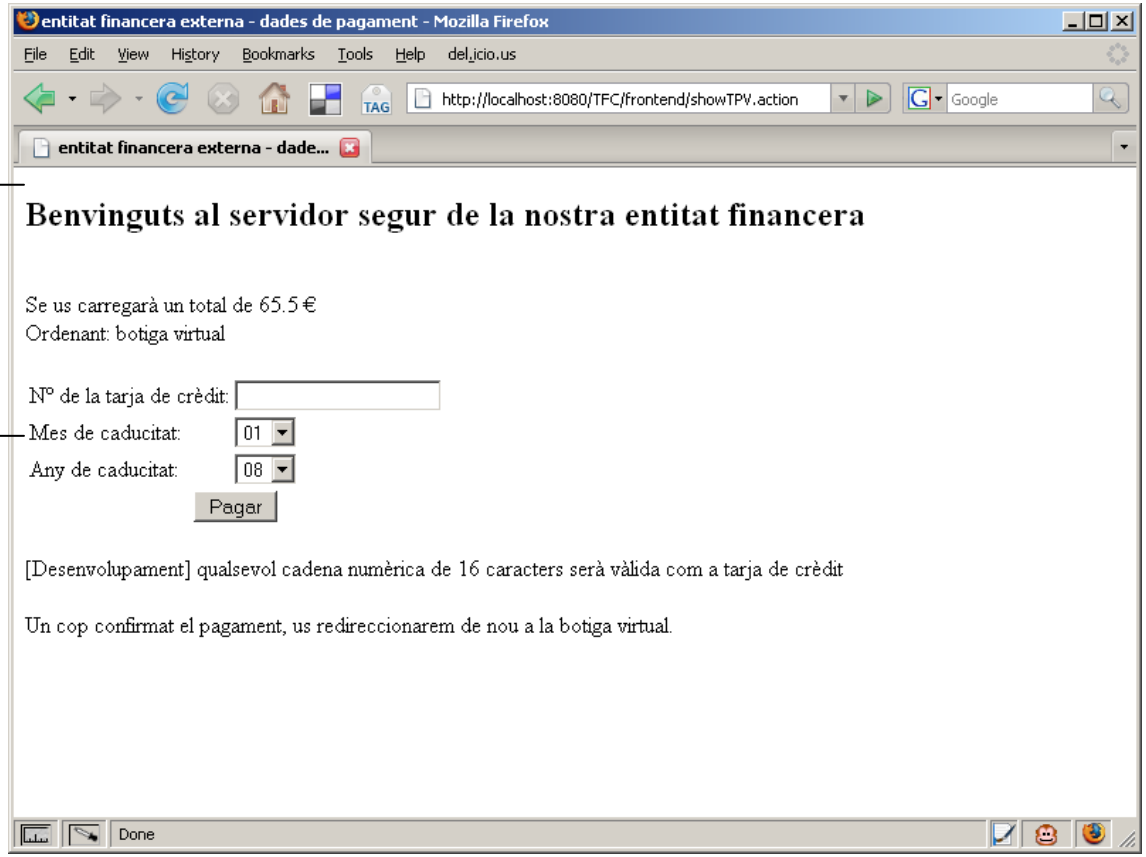


Interfície Gràfica d'Usuari VI

Front-end: simulació de pagament en línia

Absència de barra de navegació per indicar que l'acció es realitza externament a la botiga

Dades usuals requerides per les entitats financeres



Interfície Gràfica d'Usuari VII

Front-end: confirmació de comanda

Aparició de la barra de navegació per indicar que s'ha tornat a la botiga

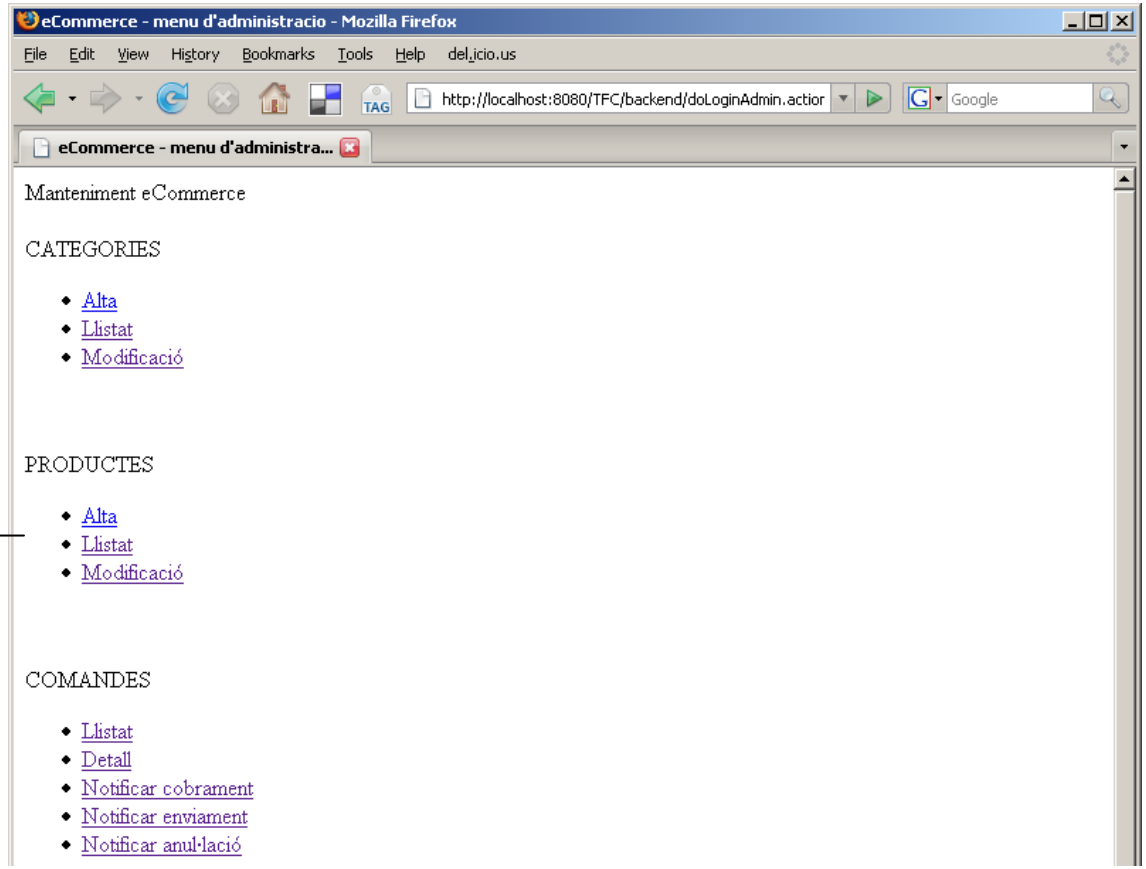
Dades de la comanda

Enllaços de navegació

Interfície Gràfica d'Usuari VIII

Back-end: menu principal

Accions de manteniment sobre les entitats de negoci



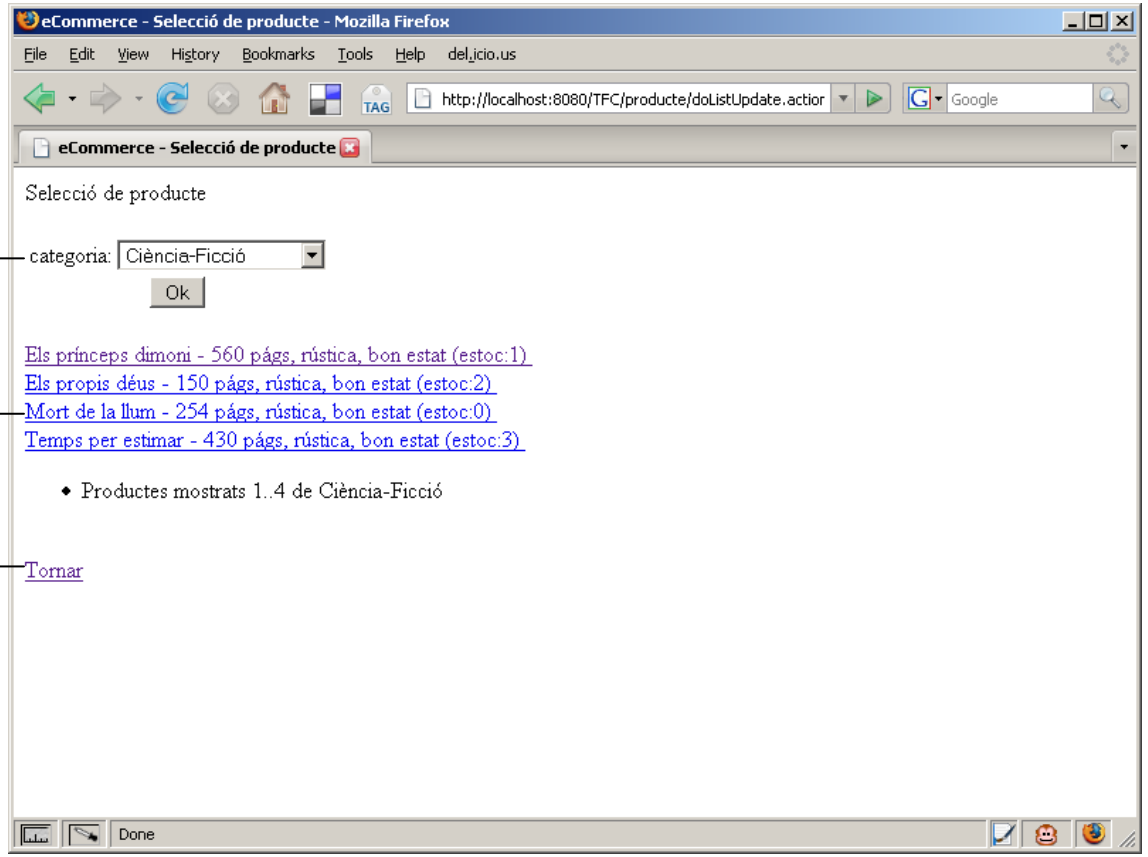
Interfície Gràfica d'Usuari IX

Back-end: llistat de productes

Selecció de categoria mitjançant una llista desplegable

Llistat de productes amb enllaços per efectuar altres accions sobre ells

Enllaços de navegació

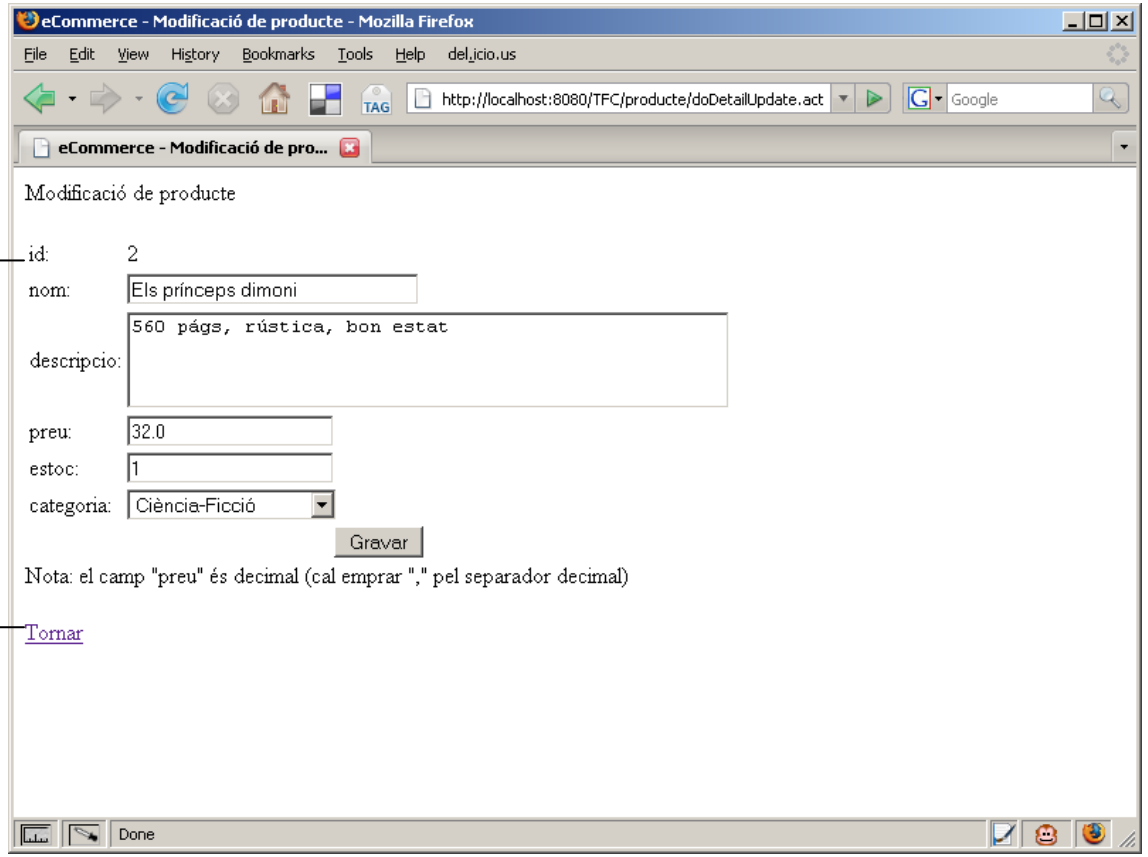


Interfície Gràfica d'Usuari X

Back-end: formulari de productes

Elements de control del formulari, que serien text simple per a les consultes

Enllaços de navegació



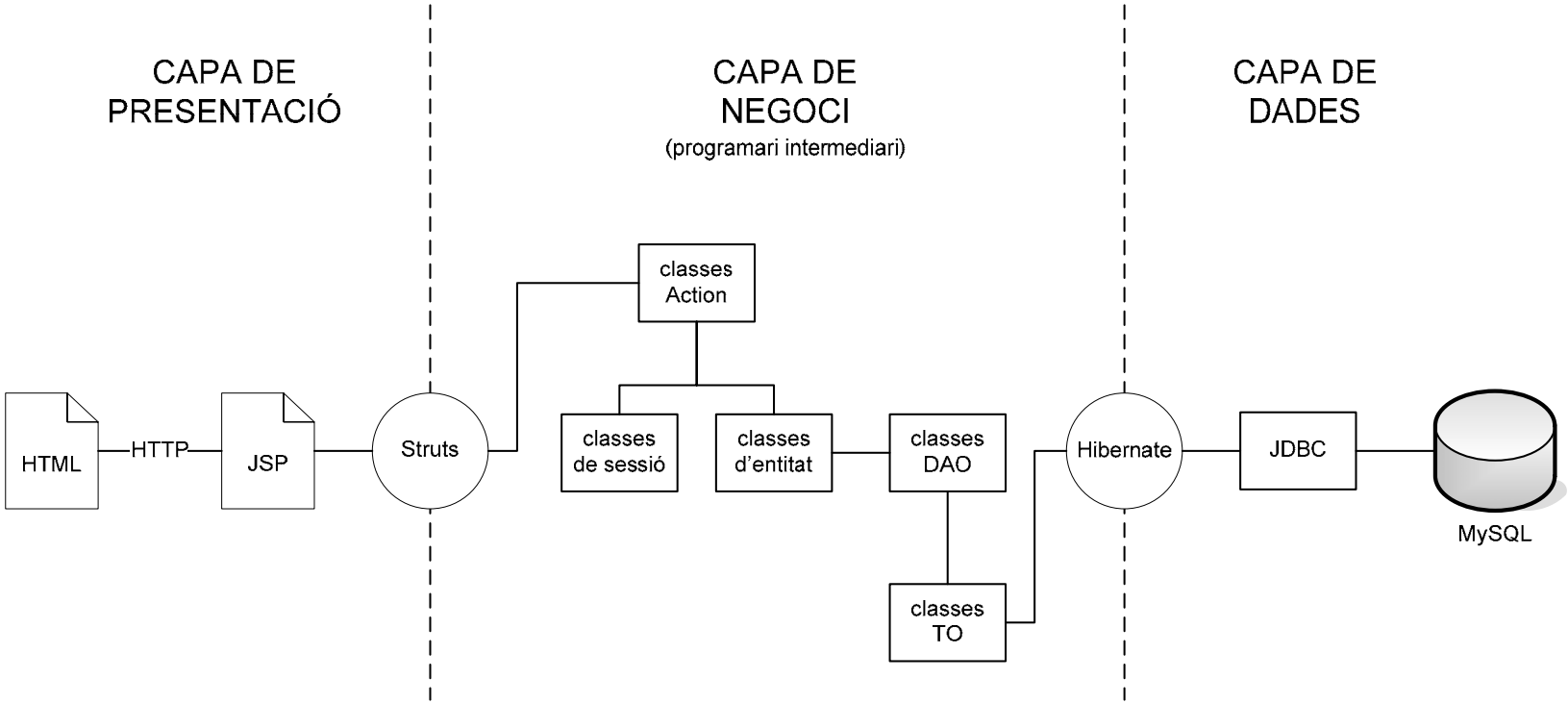
- Visió general de l'arquitectura
  - Tipus de classes del model de negoci:
    - D'*entitat* o persistents: requereixen accedir a la base de dades per emmagatzemar el seu estat
    - De *sessió* o no persistents
    - Auxiliars i d'utilitat
  - Patrons de disseny més importants utilitzats
    - *Data Acces Object* (DAO)
    - *Transfer Object* (TO)

- Visió general de l'arquitectura II
  - El patró DAO (J2EE)
    - Consisteix en delegar l'accés a les dades en una altra classe diferenciada de la classe d'entitat
    - Manté la independència de les classes de negoci de la capa de dades
  - El patró TO (J2EE)
    - Consisteix en implementar l'estructura bàsica de les dades en una altra classe diferenciada de la classe d'entitat o de sessió
    - Redueix la càrrega de transmissió per la xarxa i ajuda a la claretat del codi

- Visió general de l'arquitectura III
  - Classes d'entitat
    - Disposaran d'una classe DAO per accedir a les dades
    - Disposaran d'una classe TO per comunicar-se amb la classe DAO i per transmetre les seves dades a la capa de presentació
  - Classes de sessió
    - Disposaran d'una classe TO per transmetre les seves dades a la capa de presentació

Visió general de l'arquitectura IV

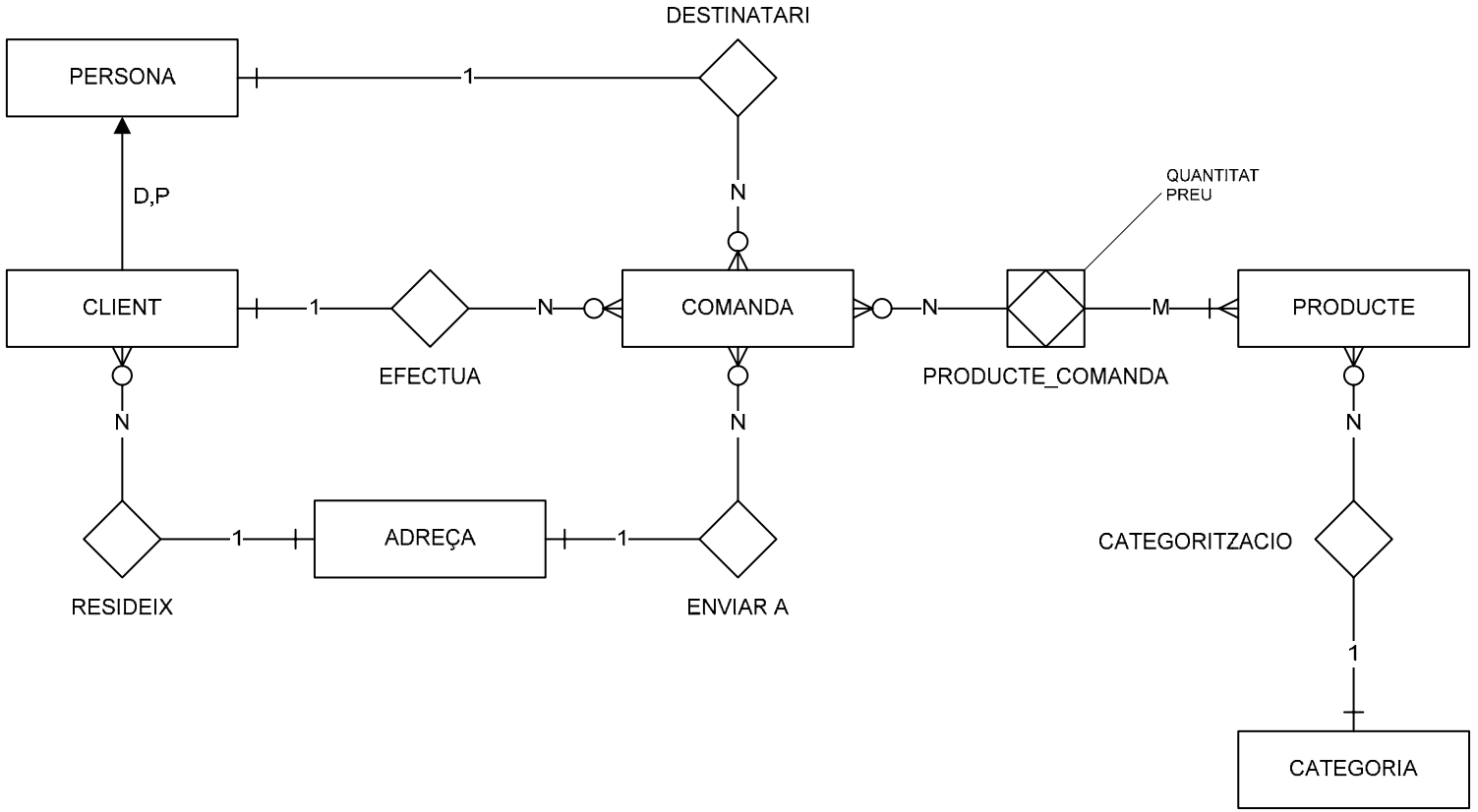
- Ubicació dels bastiments en l'arquitectura de tres capes



- Visió general de l'arquitectura V
  - Funcions dels bastiments
    - **Struts** relaciona la presentació amb la lògica de negoci segons el patró *Model-View-Controler*, establint relacions entre les vistes (planes JSP) i les classes del model, que són les gestores de presentació (classes *Action*) mitjançant un *Servlet* controlador transparent
    - **Hibernate** relaciona la capa de dades amb la lògica de negoci establint correspondències entre taules de la base de dades i les classes, *i.e.*, vinculant camps amb atributs

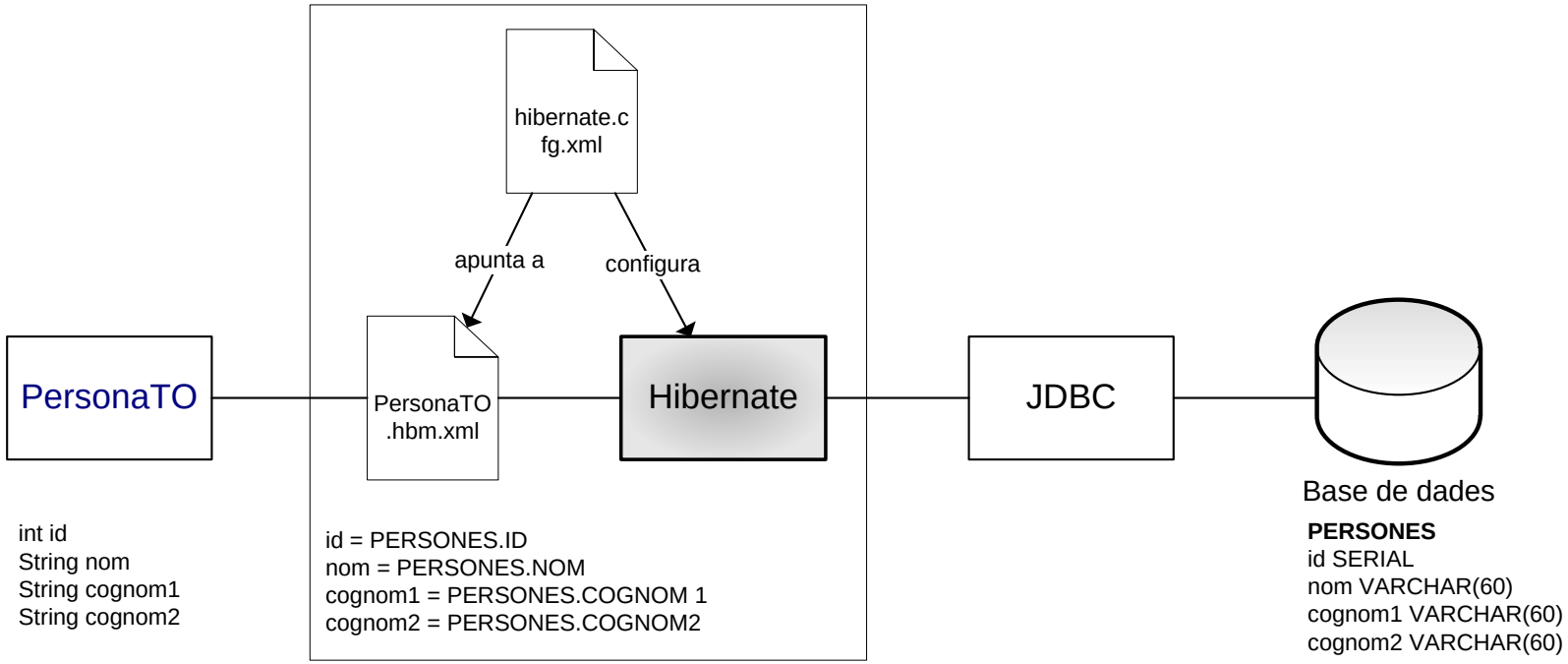
Capa de dades

- Model E-R de la persistència



Capa de dades II

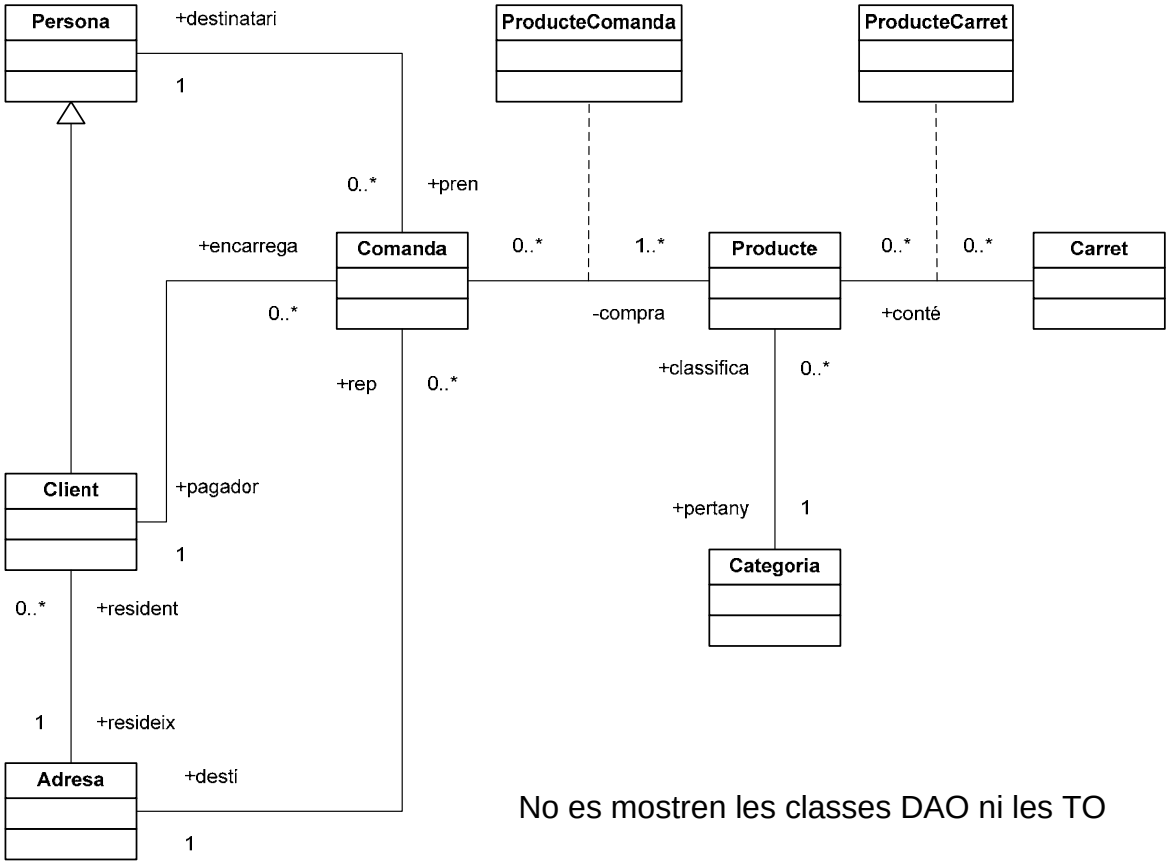
- Accés a dades amb Hibernate



- Capa de dades III
  - Funcionament de Hibernate
    - El mapeig entre les classes i les taules de la base de dades es realitza mitjançant arxius XML fàcilment mantenibles
    - Les classes que es mapegen seran les classes TO
    - D'aquesta manera, les classes d'entitat i de sessió esdevenen *Decorators*(GoF) de les classes TO, en les que deleguen l'accés a les seves dades bàsiques
    - Només s'utilitza Hibernate dins les classes DAO per mantenir la independència de la capa de negoci respecte a l'origen i forma d'accès de les dades

Capa de negoci

Diagrama estàtic simplificat



No es mostren les classes DAO ni les TO

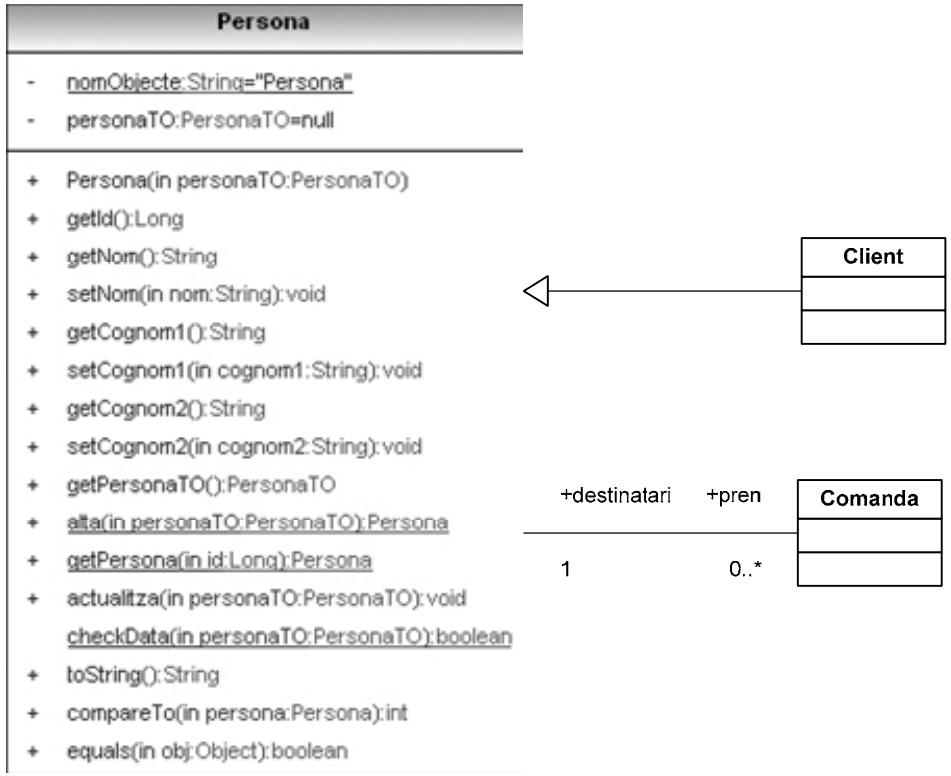
- Capa de negoci II
  - Classes d'entitat
    - S'instancien mitjançant un constructor que rep la classe TO associada com a paràmetre.
    - Poden actualitzar el seu estat mitjançant l'accesor d'escriptura (*setter*) de l'atribut corresponent, o bé rebent un *Transfer Object* que encapsuli les noves dades.
    - Disposen de mètodes per retornar els *Transfer Object* que encapsulen, o bé per generar-ne d'altres per als casos on es requereixi objectes compostos, els quals segueixen el patró de disseny *Composite Object*.
    - Disposen dels mètodes escaients per interactuar amb la base de dades, delegats en la classe DAO
    - Implementen la interfície *Comparable* per tal de generar llistats ordenat

- Capa de negoci III
  - Classes de sessió
    - Resideixen dins del contenidor web, apuntats per l'objecte *Session* que abstrau la sessió d'usuari.
    - Dins aquest contenidor poden existir objectes d'entitat, com el Client un cop identificat pel sistema
    - Disposen de mètodes per retornar els *Transfer Object* que encapsulen, o bé per generar-ne d'altres per als casos on es requereixi objectes compostos, els quals segueixen el patró de disseny *Composite Object*.
    - En general, estan directament relacionades amb les abstraccions del comerç electrònic per a l'usuari: *carret de la compra*, *compte de client*, etc.

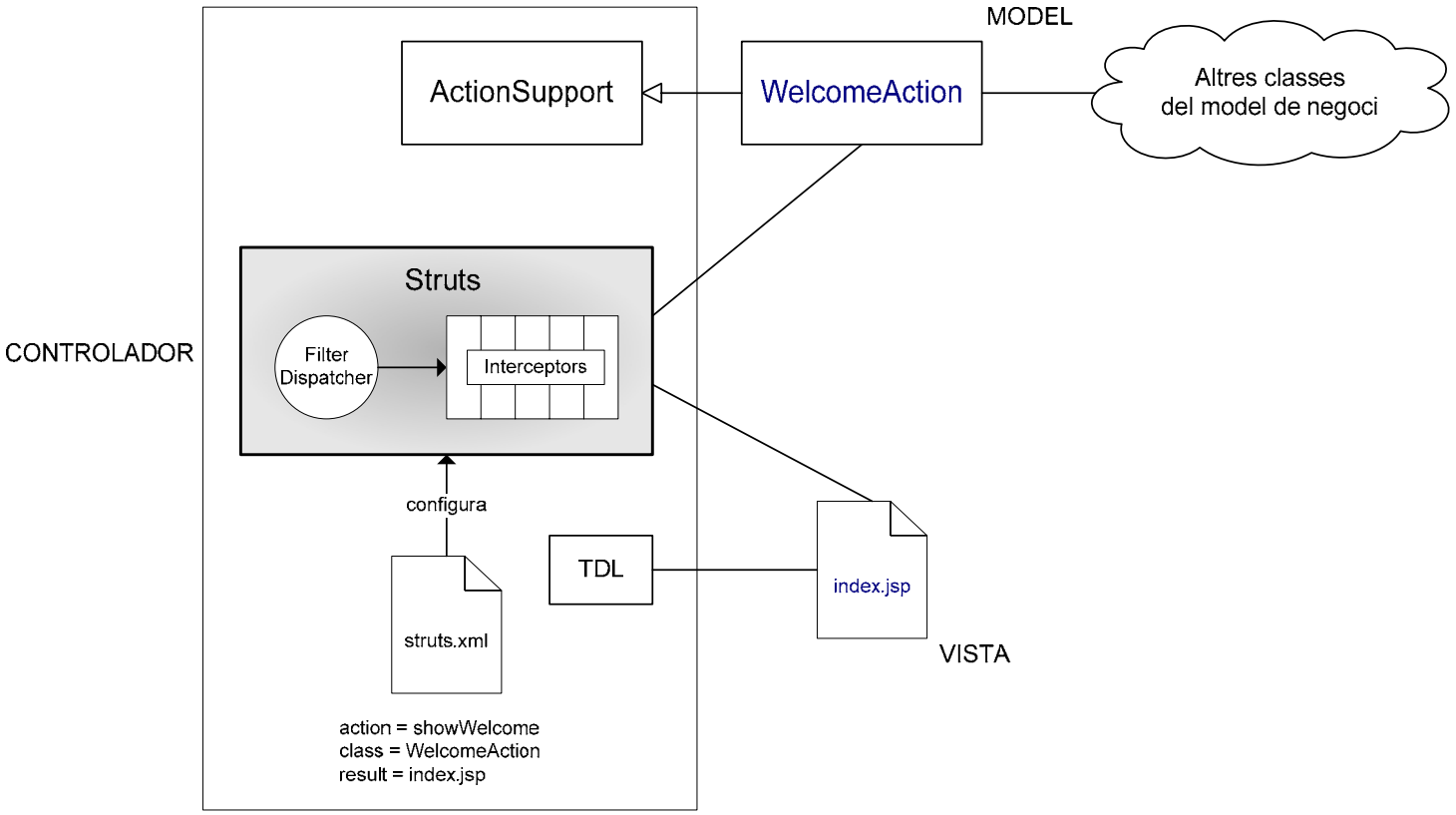
- Capa de negoci IV
  - Altres classes
    - Les classes de *utilitat* centralitzen els algorismes de validació de les de dades d'entrada
    - Les classes d'*excepció* encapsulen les excepcions que es poden produir durant el funcionament de l'aplicació i faciliten les tasques d'auditoria mitjançant el sistema *log4j*
    - Tots dos tipus de classes han estat dissenyades pensant en la seva reutilització per altres aplicacions

Capa de negoci V

- Exemple de diagrama estàtic detallat



- Capa de presentació
  - El patró *Model-View-Controler* amb Struts



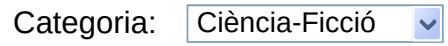
- Capa de presentació II
  - Funcionament de Struts
    - Les classes del model (classes *Action*) es mapegen amb les vistes (planes JSP) mitjançant arxius XML fàcilment mantenibles
    - El controlador és completament transparent
    - Les classes *Action* utilitzen les classes d'entitat i de sessió i transmeten les classes TO a les planes JSP
    - Les classes *Action* són el punt d'entrada a la lògica de negoci i es pot considerar que segueixen els patrons *Facade* (Gof) o *Business Delegate* (J2EE)

- Capa de presentació III
  - Les planes JSP
    - No contenen cap lògica de negoci
    - Utilizen les TDLs (*Tag Descriptor Library*) de Struts i de JSP per simplificar la presentació dels controls i llistats, les quals actuen com les etiquetes de HTML
    - Utilizen EL (*Expression Language*) per simplificar la programació en lloc de *scripting* Java
    - Utilizen fragments inclosos per les parts comunes, com la barra de navegació, per simplificar el manteniment

Capa de presentació IV

- Exemple d'utilització de les TDL comparat amb el tradicional *scripting* en Java

Llista desplegable a la plana JSP



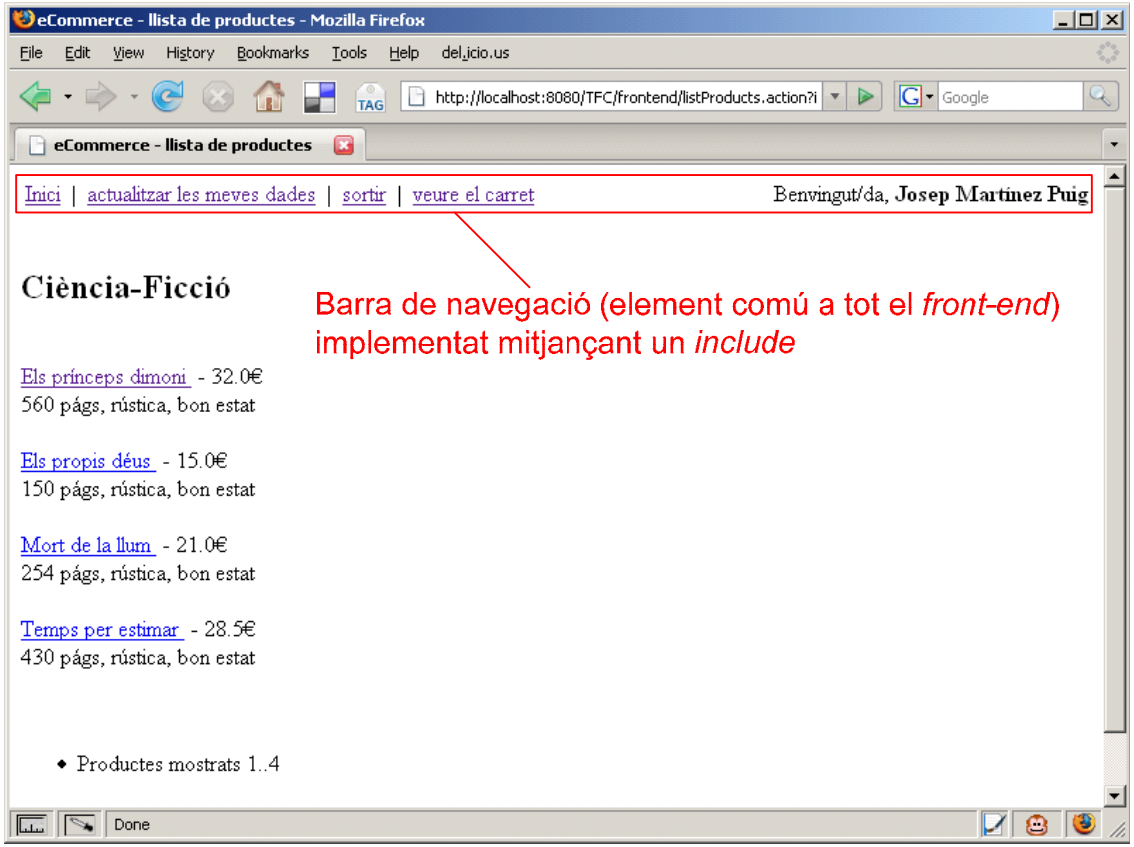
Codificació amb la llibreria d'etiquetes de Struts

```
<s:select name="idCategoria" label="categoria" list="categoriesTO"
listKey="id" listValue="nom" />
```

Codificació alternativa amb scripting

```
<label for="idCategoria">Categoria:</label>
<select name="idCategoria">
<%
List<CategoriaTO> categoriesTO =
(List<CategoriaTO>)pageContext.getAttribute("categoriesTO");
CategoriaTO categoriaTO;
for (Ierator<CategoriaTO> i = categoriesTO.iterator(); i.hasNext(); ) {
  categoriaTO = i.next();
%>
  <option value="<%= categoriaTO.getId() %>">
    <%= categoriaTO.getNom() %>
  </option>
<%
}
%>
</select>
```

- Capa de presentació V
  - Exemple d'utilització de frament inclosos (includes) a les planes JSP per als elements comuns



- Conclusions
  - L'aplicació resultant és una mostra d'arquitectura J2EE segons l'estat de l'art actual
  - La utilització de bastiments en cap moment ha entrat en conflicte amb l'arquitectura
  - Els bastiments han facilitat un desenvolupament ràpid i lliure d'errors, per tant la inversió en temps que ha requerit el seu aprenentatge es compensa amb els beneficis
  - En general, la seva utilització és molt recomanable per les aplicacions empresarials orientades a Internet

- Conclusions II
  - L'ús de bastiments pot comportar determinats riscos que cal tenir presents
    - Per Struts, el risc resideix en implementar la lògica de negoci en les classes Action, ignorant els patrons *Facade*(GoF) i *Business Delegate*(J2EE)
    - Per Hibernate, el risc resideix en implementar la lògica de l'aplicació en les classes mapejades amb la base de dades, ignorant el patró *Transfer Object*(J2EE)
  - Ignorar aquests patrons implicaria un fort acoblament entre les diferents capes de l'arquitectura
  - Aquests riscos són fàcilment evitables amb un bon disseny