PFC: Aplicació Gestió GCANEC

Estudiant: Jorge Casanovas Hernàndez

Enginyeria Informàtica

Consultor: Josep Maria Camps Riba

16/01/2012

2. DEDICATÒRIA I AGRAIMENTS

Per una banda vull agrair aquest treball al Departament d'Arquitectura de l'Institut Municipal d'Informàtica, gràcies al suport que m'han donat quant a documentació es refereix. Sense aquesta informació no m'haguès estat possible realitzar el Projecte.

Per altra banda m'agradaria agrair a la UOC haver-me acceptat la sol·licitud de "PFC Ad Hoc" per poder realitzar aquest Projecte, tot i que, el desenvolupament d'una aplicació no era l'objectiu de l'Àrea J2EE.

Per finalitzar, m'agradaríia fer una dedicació molt especial a la meva família, dóna i tres fills, per a totes les hores que no els hi he pogut dedicar durant la realització del PFC i durant tots els anys que he estat a la UOC.

3. RESUM

Per a la realització del PFC vaig fer una sol·licitud “Ad hoc”. Per les característiques del Projecte a realitzar, la UOC, em va donar el vist-i-plau i em va assignar a l’Àrea J2EE

El PFC consisteix en el desenvolupament d’una petita aplicació de gestió per poder mantenir les versions dels paquets que componen els productes distribuïts a tots els usuaris de l’Ajuntament de Barcelona.

L’aplicació s’encarregarà de mantenir el flux de canvis d’estat de les peticions des de que es creen fins que es tanquen, així com mantenir totes les versions de tots els productes de software distribuït.

Per al desenvolupament de l’aplicació s’utilitzaran els frameworks implementats a l’IMI per al desenvolupament d’aplicacions J2EE.

- Framework de desenvolupament J2EE: S’utilitzarà openFrame que té com a objectius principals: programació orientada a interfícies, configurar tots els serveis i elements de l’aplicació de forma declarativa sense afectar al codi, oferir components de desenvolupament, etc.

- Framework de presentació: L’arquitectura openFrame es basa en l’arquitectura MVC (Model-Vista-Controlador), així com una arquitectura en 3 capes (Capa Presentació, Capa de Lògica de Negoci, Capa d’Integració i Accés a Dades). OpenFrame utilitza com a framework de presentació, Struts, que proporciona un Controlador principal ja implementat, gestiona de forma automàtica els formularis, llibreria de tags, etc.

- Framework de persistència: S’utilitzarà Hibernate que proporciona les següents característiques: mapeig objecte-relacional, ús d’interfícies o classes concretes, llenguatge de queries, denominat HQL, independent de la base de dades, tota la configuració pot definir-se en fitxers de configuració XML, etc.

Algunes de les funcionalitats de l’aplicació seran:

- Creació i consulta de peticions.

- Consulta de productes amb els seus paquets corresponents.

- Creació de paquets nous o actualitzacions (noves versions).

4. ÍNDEX

5. PFC: APLICACIO GESTIÓ GCANEC

5.1. INTRODUCCIÓ

El PFC consisteix en el desenvolupament d'una petita aplicació de gestió per a l'Institut Municipal d'Informàtica. Per al desenvolpument d'aquesta aplicació s'utilitzarà l'entorn de desenvolupament de l'IMI i seguint les normatives corresponents quant a desenvolupament, creació scripts de base de dades, etc.

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

Els motius per els quals vaig fer la sol·licitud d'aquest PFC són els següents:

- Enfrentar-me a l'entorn de desenvolupament d'aplicacions, ja que el meu perfil és més Tècnic i, alhora, aprendre a treballar amb frameworks tant potents i utiltizats en l'actualitat com són OpenFrame, Struts, Spring, Hibernate, etc.

- La necessitat per a l'IMI, més concretament per al Departament de Gestió de Versions que és on desenvolupo les meves funcions, de tenir una aplicació d'aquestes característiques.

Per tant, les aportacions de la realització d'aquest PFC són, tant personals a nivell d'aprenentatge com professionals.

5.1.2. Objectius

Els objectius que pretenc assolir amb la realització d'aquest PFC, són els següents:

- Estudi de l'arquitectura d'un framework de desenvolupament J2EE: En aquest cas, l'aplicació es desenvoluparà utilitzant el framework openFrame. I per tant, hauré de conèixer les seves característiques, el serveis que ofereix, la seva arquitectura, etc.

- Estudi del funcionament d'un framework de persistència: En aquest cas, s'utilitzarà Hibernate, i per tant, com en el cas anterior, hauré de conèixer les seves característiques, tot el que pot oferir, etc. A més, també hauré d'aprendre HQL com a llenguatge de queries.

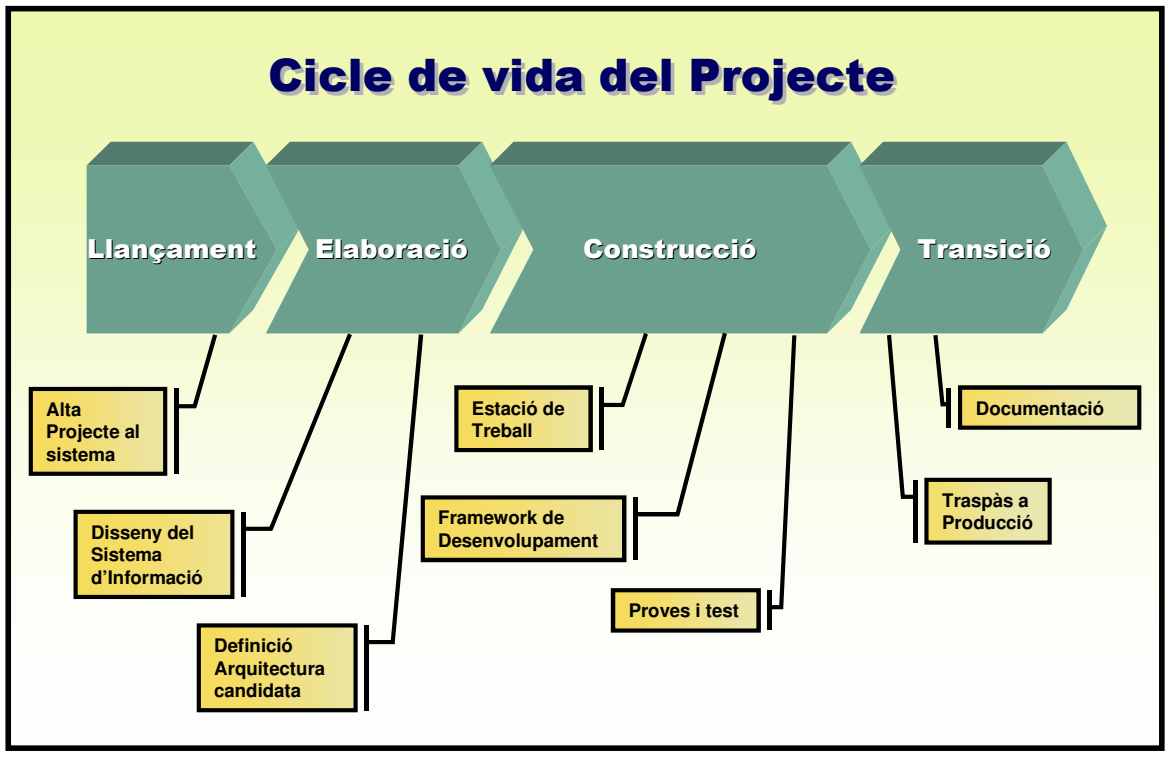
- Introducció en el desenvolupament d'aplicacions J2EE: Introduir-me al món del desenvolupament d'aplicacions, per tal d'adquirir experiència i afegir-la als meus coneixements més tècnics.

5.1.3. Enfocament i mètode seguit

Tal i com s'ha comentat anteriorment, el fet de realitzar una aplicació per a l'IMI (Institut Municipal d'Informàtica) m'ha obligat a seguir els mètodes i estàndards establerts per l'IMI.

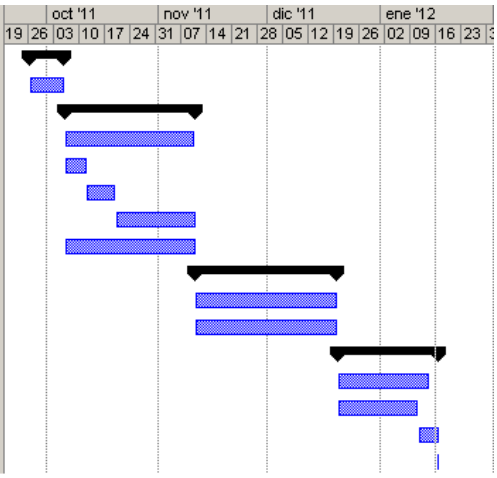
L'IMI té definida una metodologia de desenvolupament d'aplicacions que s'ha de complir. De la mateixa manera, l'IMI té definida una normativa i uns estàndards que han de complir totes les aplicacions que han de ser hostatjades a l'IMI.

La metodologia de treball, basada en RUP, s'anomena ADINET. Les fases es corresponen amb les del cicle de vida d'un projecte.



5.1.4. Planificació del Projecte

Nombre de tarea	Duración	Comienzo	Fin
PAC-1	**8,57 días**	**lun 26/09/11**	**mié 05/10/11**
Elaborar Pla Treball	8,57 días	lun 26/09/11	mié 05/10/11
PAC-2	**29,14 días**	**jue 06/10/11**	**jue 10/11/11**
Estudi Framework	29,14 días	jue 06/10/11	jue 10/11/11
Disseny Bases de Dades	4 días	jue 06/10/11	mar 11/10/11
Generació Scripts Bases de Dades	6,29 días	mar 11/10/11	mié 19/10/11
Implementació aplicació	17,71 días	jue 20/10/11	jue 10/11/11
Redacció memòria	29,14 días	jue 06/10/11	jue 10/11/11
PAC-3	**30,29 días**	**jue 10/11/11**	**lun 19/12/11**
Implementació aplicació	30,29 días	jue 10/11/11	lun 19/12/11
Redacció memòria	30,29 días	jue 10/11/11	lun 19/12/11
LLIURAMENT FINAL	**22,71 días?**	**mar 20/12/11**	**lun 16/01/12**
Implementació aplicació	21,14 días	mar 20/12/11	vie 13/01/12
Redacció memòria	17,71 días	mar 20/12/11	mar 10/01/12
Redacció presentació	4 días	mié 11/01/12	lun 16/01/12
Redacció manual usuari	1 día?	lun 16/01/12	lun 16/01/12



5.1.5. Productes Obtinguts

Amb la realització i finalització del PFC, obtenim:

- Una Base de dades amb tota la estructura i dades necessàries per al funcionament de l'aplicació. L'esquema s'ha generat en un Servidor Oracle 9.

- Una aplicació web de gestió per poder mantenir les versions dels paquets que componen els productes distribuïts a tots els usuaris de l'Ajuntament de Barcelona. L'aplicació s'ha generat mitjançant l'aplicació RAD (Rational Administration Developer) i sota un servidor WebSphere 6.0.

5.1.6. Descripció altres capítols

A continuació es detalla el contingut del següents apartats:

En l'apartat 5.2 es defineix l'esquema de Base de Dades necessari per al funcionament de l'aplicació. Es defineixen els scripts, seguint la metodologia de l'IMI, necessaris per a la creació de l'esquema, les taules que s'han creat amb un petita definició del que emmagatzemen i un diagrama de l'esquema relacional resultant.

En l'apartat 5.3 es defineixen els diferents elements del model dels casos d'ús, definint els actors que intervenen, els casos d'ús i el corresponent diagrama de casos d'ús.

En l'apartat 5.4 ja entrem en la definició, característiques i els 3 components base principals que formen el framework OpenFrame. Aquests components són Struts, Spring i Hibernate.

En l'apartat 5.5 es parla de l'arquitectura MVC (Model-Vista-Controlador) en que es basa el framework OpenFrame.

En l'apartat 5.6 entrem en detall amb l'arquitectura de 3 Capes (Capa Presentació, Capa Integració de Dades i Capa Lògica de Negoci). Per a cadascuna de les Capes es presenten els serveis que presten i s'explica amb més detall la part de la nostra aplicació per a cadascuna de les capes.

5.2. ESQUEMA BASE DE DADES

L'esquema de base de dades de l'aplicació s'ha creat seguint la normativa de creació d'scripts oracle al Institut Municipal d'Informàtica. Els scripts que s'han creat per generar l'esquema, són els següents:

- *GCANEC_U:* Creació de l'esquema. Es crea el propietari dels objectes, el seu espai físic i lògic.

- *GCANEC_T:* Creació estructura. Es creen les taules i estructures per l'emmagatzemament de dades.

- *GACANEC_C:* Creació objectes lògica. Creació d'objectes per la lògica de l'aplicació i rendiment.

- *GCANEC_I:* Inserció de dades a taules auxiliars

5.2.1. Taules Base de Dades

Es detallen totes les taules que s'han creat pel correcte funcionament de l'aplicació.

- *PETICIONS:* Taula on s'emmagatzemen totes peticions que es generen mitjançant l'aplicació.

  Una petició es genera per la instal·lació d'un nou producte o per una actualització d'aquest i a partir d'una RFC que crea el peticionari. A partir d'aquesta, Gestió del Canvi generarà la petició corresponent amb: les dades del producte, la seva versió, el nom del peticionari, amb la data de creació, etc.

- *PRODUCTES:* Taula on s'emmagatzema el catàleg de productes que s'instal·len.

  Previ a la generació d'una petició d'instal·lació, s'haurà de comprovar si el producte ja existeix i, si no es el cas, s'haurà de crear mitjançant l'aplicació amb les seves dades corresponents: nom del producte, versió, descripció, estat, etc.

- *PAQUETS:* Taula on s'emmagatzemen els paquets que formen un producte.

  Un cop generada la petició per la instal·lació o actualització d'un producte en concret, s'han de crear els paquets necessaris per al funcionament del producte.

Els paquets s'hauran de crear mitjançant l'aplicació amb les seves dades corresponents: versió del paquet, nom del paquet, descripció, estat, data alta, etc.

- *GRUPS:* Taula on s'emmagatzemen el nom dels grups assignats al producte en concret i, respectivament, als paquets associats.

  Qualsevol dels productes instal·lats ha de tenir un grup assignat. Aquest s'utilitzarà per assignar els usuaris als quals s'haurà d'instal·lar l'aplicació. Per tant, en el moment de la creació dels paquets, s'haurà de crear el grup en cas de que no existeixi.

- *ESTATS:* Taula on s'emmagatzemen els diferents estats en els quals pot estar una petició.

  Els diferents estats en els quals pot estar una petició són els següents:

  - Aturat: La petició s'atura per qualsevol motiu: disponibilitat, falta recursos, etc.

  - Paquetitzant: La petició està en procés, pendent de que el departament de paquetització generi els nous paquets.

  - Pendent maqueta: El departament de paquetització ha generat els nous paquets i passa la petició al departament de gestió de versions perquè aquests distribueixin l'aplicació en l'entorn de preproducció.

  - Llest maqueta: Un cop la petició s'ha distribuït a l'entorn de preproducció, està llesta perquè es facin les proves funcionals necessàries.

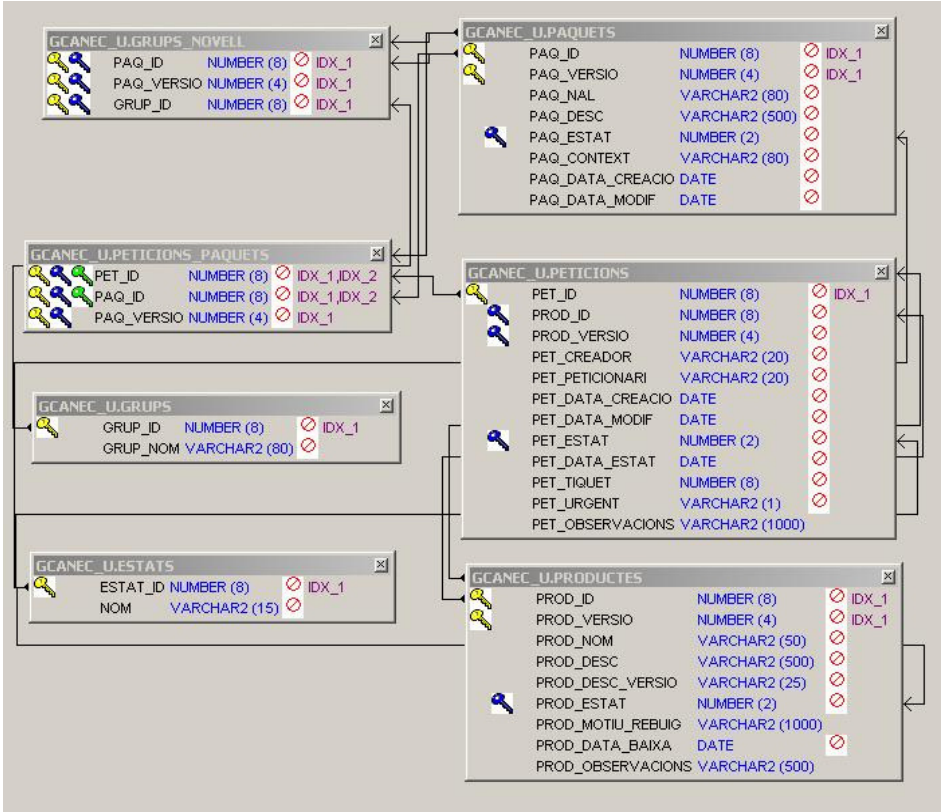
  - Distribució: Un cop s'han fet les proves i tot es correcte, s'autoritza el passi a l'entorn de producció.

  - Instal·lada: Un cop s'ha distribuït l'aplicació a l'entorn de producció.

  - Rebutjada: La petició es rebutja per qualsevol motiu.

5.2.2. Esquema Relacional

A continuació es mostra l'esquema relacional de la base de dades.



5.3. CASOS D'ÚS

A continuació es detallen el diferents elements del model dels casos d'ús.

5.3.1. Actors

Entenem com actor qualsevol cosa que es comunica o interacciona amb el sistema per realitzar diferents accions. Els actors no necessàriament coincideixen amb els usuaris i, aquests, poden interpretar diferents rols corresponents a diferents actors.

En el cas de la nostra aplicació, els actors representen papers (rols) que interpreten persones de diferents departaments (Gestió del Canvi, Gestió de Versions, Paquetitzadors). Es tracta d'actors primaris, ja que interaccionen amb el sistema per explotar la seva funcionalitat, treballant directament i freqüentment amb l'aplicació.

Actor	Gestió del Canvi
Descripció	Aquest actor el formen totes les persones que treballen en aquest departament i que utilitzaran l'aplicació.
Relacions	Aquest actor mantindrà relacions amb altres actors del sistema (Gestió de Versions, Paquetitzadors).
Referències	Casos d'ús en els que intervé: Gestió Peticions (Alta/Baixa/Modificació) Consulta Peticions, Gestió Productes (Alta/Baixa/Modificació), Consulta Productes, Consulta Paquets, Paquetitzant, Distribució, Aturat, Rebutjat.

Actor	Gestió de Versions
Descripció	Aquest actor el formen totes les persones que treballen en aquest departament i que utilitzaran l'aplicació.
Relacions	Aquest actor mantindrà relacions amb altres actors del sistema (Gestió del Canvi, Paquetitzadors).
Referències	Casos d'ús en els que intervé: Consulta Peticions, Consulta Productes, Consulta Paquets, Gestió Paquets (Alta/Baixa/Modificació), Paquetitzant, Alta Grup, Llest maqueta, Distribució, Instal·lada.

Actor	Paquetitzadors
Descripció	Aquest actor el formen totes les persones que treballen en aquest departament i que utilitzaran l'aplicació
Relacions	Aquest actor mantindrà relacions amb altres actors del sistema (Gestió de Versions, Gestió del Canvi)
Referències	Casos d'ús en els que intervé: Consulta Peticions, Consulta Productes, Consulta Paquets, Gestió Paquets (Alta/Baixa/Modificació), Alta Grup, Pendent Maqueta.

5.3.2. Casos d'ús del model

Els casos d'ús representen una unitat funcional coherent d'un sistema, subsistema o classe. L'especificació dels casos d'ús han de descriure la forma en que un actor interactua amb el sistema. Aquestes especificacions han de donar resposta a les següents preguntes:

- Quines són les principals funcions o tasques realitzades per l'actor?
- Quina informació del sistema adquireix, produeix o transforma l'actor?
- Ha d'informar l'actor al sistema dels canvis produïts a l'entorn?
- Quina informació del sistema desitja l'actor?
- Ha d'informar l'actor de qualcun canvi inesperat?

A continuació es detallen els diferents casos d'ús:

Cas d'ús	Gestió Peticions (Alta/Baixa/Modificació)
Actors	Participa l'actor Gestió del Canvi
Tipus	Essencial
Precondició	Donar d'alta el producte.
Postcondició	La petició queda registrada juntament amb el producte, donada de baixa o modificada segons correspongui.
Propòsit	
Registra l'alta, baixa o modificació d'una petició per la creació o modificació de software per distribuir als usuaris.	
Resum	
Arriba a Gestió del Canvi una RFC per la creació o modificació de paquets de software i la persona encarregada dona d'alta la petició corresponent. Un cop creada, la petició, també es podrà modificar o donar de baixa.	

Cas d'ús	Gestió Productes (Alta/Baixa/Modificació)
Actors	Participa l'actor Gestió del Canvi
Tipus	Essencial
Precondició	RFC per crear els paquets per al nou producte o per la seva modificació.
Postcondició	El producte queda registrat, donat de baixa o modificat segons correspongui.
Propòsit	
Registra l'alta, baixa o modificació d'un producte per poder donar d'alta la petició per la creació o modificació de software per distribuir als usuaris.	
Resum	
Arriba a Gestió del Canvi una RFC per la creació o modificació de paquets de software i la persona encarregada introdueix totes les dades per donar d'alta el producte corresponent	

Cas d'ús	Gestió Paquets (Alta/Baixa/Modificació)
Actors	Participen els actors Gestió de Versions i Paquetitzador
Tipus	Essencial
Precondició	Petició per crear els paquets per al nou producte o per la seva modificació.
Postcondició	El paquet o paquets queden registrats, donats de baixa o modificats segons correspongui.
Propòsit	
Registra l'alta, baixa o modificació d'un paquet per poder distribuir-los en els entorns de preproducció i producció.	
Resum	
Arriba als Paquetitzador una petició per la creació o modificació de paquets de software i la persona encarregada introdueix totes les dades per donar d'alta el paquet o paquets corresponents, en cas que no existeixin.	

Cas d'ús	Alta grup
Actors	Participen els actors Gestió de Versions o Paquetitzador
Tipus	Essencial
Precondició	Els paquets a distribuir són nous i s'ha de crear el seu grup corresponent.
Postcondició	El grup queda registrat juntament amb els seus paquets.
Propòsit	
Registra el grup per als nous paquets creats.	
Resum	
Arriba als Paquetitzador una petició per la creació de nous paquets i la persona encarregada introdueix totes les dades per donar d'alta el grup corresponent.	

Cas d'ús	Consulta Petició
Actors	Participen els actors Gestió del Canvi, Gestió de Versions o Paquetitzador
Tipus	Essencial
Precondició	
Postcondició	
Propòsit	
Consultar en qualsevol moment les peticions donades d'alta o, si escau, una en concret.	
Resum	
Qualsevol dels actors vol consultar l'estat d'una petició i la cerca per número de tiquet o per estat.	

Cas d'ús	Consulta Producte
Actors	Participen els actors Gestió del Canvi, Gestió de Versions o Paquetitzador
Tipus	Essencial
Precondició	
Postcondició	
Propòsit	
Consultar en qualsevol moment els productes donats d'alta o, si escau, un en concret.	
Resum	
Qualsevol dels actors vol consultar l'estat d'un producte i el cerca per nom o per estat.	

Cas d'ús	Consulta Paquet
Actors	Participen els actors Gestió del Canvi, Gestió de Versions o Paquetitzador
Tipus	Essencial
Precondició	
Postcondició	
Propòsit	
Consultar en qualsevol moment els paquets donats d'alta o, si escau, un en concret.	
Resum	
Qualsevol dels actors vol consultar l'estat d'un paquet i el cerca per nom o per estat o per producte.	

Cas d'ús	Estat Paquetitzant
Actors	Participa l'actor Gestió del Canvi
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba a l'actor Paquetitzador
Propòsit	
Passar la petició al Paquetitzador perquè generi i doni d'alta els nous paquets.	
Resum	
Gestió del Canvi dona d'alta la petició per la creació dels paquets per a un nou producte i la passa al departament corresponent prement el botó corresponent.	

Cas d'ús	Estat Pendent maqueta
Actors	Participa l'actor Paquetitzadors
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba a Gestió de Versions
Propòsit	
Passar la petició a Gestió de Versions perquè distribueixin els paquets creats a l'entorn de preproducció perquè es facin les proves necessàries.	
Resum	
Paquetitzacions ha creat els nous grups i passa la petició al departament corresponent prement el botó corresponent.	

Cas d'ús	Estat Llest Maqueta
Actors	Participa l'actor Gestió de Versions
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba al peticionari
Propòsit	
Passar la petició al peticionari perquè es facin les proves necessàries en l'entorn de preproducció.	
Resum	
Gestió de Versions, després d'haver rebut la petició, distribueix els nous paquets en un màquina de l'entorn de preproducció. Un cop distribuït es passa la petició al peticionari, prement el botó corresponent, perquè aquest faci les proves.	

Cas d'ús	Estat Distribució
Actors	Participa l'actor Gestió del Canvi o Gestió de Versions
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba a Gestió de Versions
Propòsit	
Passar la petició a Gestió de Versions perquè distribueixin els paquets creats a l'entorn de producció.	
Resum	
Gestió del canvi ha rebut l'ok per part del peticionari per poder distribuir l'aplicació a l'entorn de producció i, mitjançant el botó corresponent, passa la petició al departament corresponent perquè faci la distribució.	

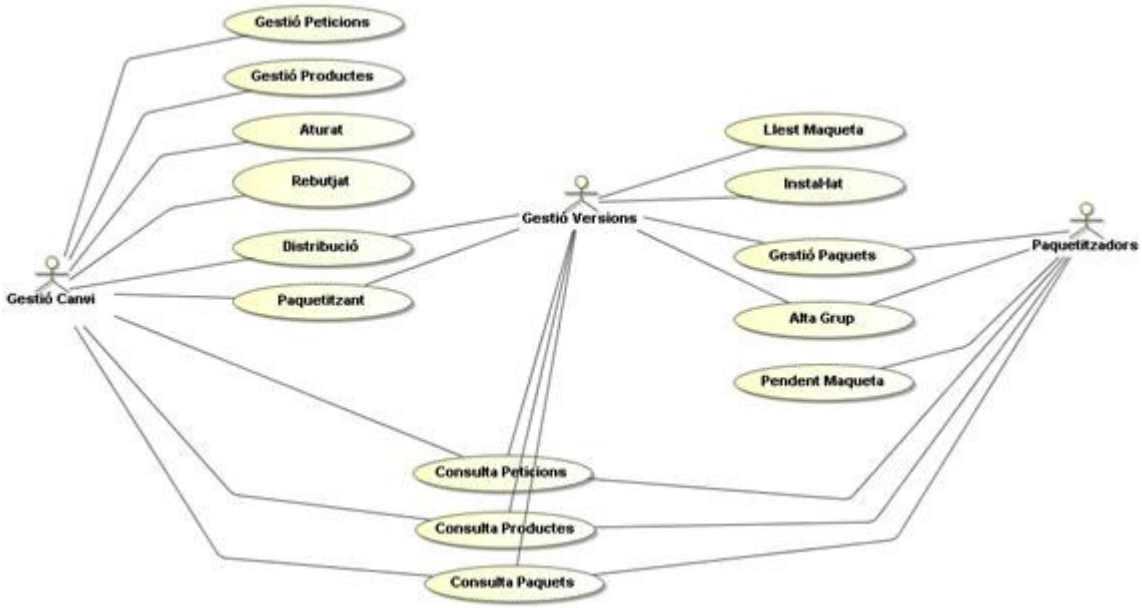
Cas d'ús	Estat Instal·lada
Actors	Participa l'actor Gestió de Versions
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba a Gestió del canvi
Propòsit	
Passar la petició a Gestió del Canvi per informar de que els nous paquets s'han distribuït en l'entorn de producció.	
Resum	
Gestió de Versions ha desplegat els nous paquets a producció i no ha donat cap error. Per tant, mitjançant el botó corresponent, passa la petició al departament corresponent	

Cas d'ús	Estat Aturat
Actors	Participa l'actor Gestió del Canvi
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba a Gestió del Canvi
Propòsit	
La petició s'atura per qualsevol motiu.	
Resum	
Gestió del canvi ha rebut la notificació per aturar la petició i, mitjançant el botó corresponent, canvia l'estat de la petició.	

Cas d'ús	Estat Rebutjat
Actors	Participa l'actor Gestió del Canvi
Tipus	Essencial
Precondició	Es vol canviar d'estat la petició
Postcondició	Un cop es canvia d'estat, la petició arriba a Gestió del Canvi
Propòsit	
La petició es rebutja per qualsevol motiu.	
Resum	
Gestió del canvi ha rebut la notificació per rebutjar la petició i, mitjançant el botó corresponent, canvia l'estat de la petició.	

5.3.3. Diagrama casos d'ús

Els diagrames de casos d'ús mostren les relacions entre els casos d'ús d'un sistema i els seus actors. Aquests només donen una visió general del model de casos d'ús. A continuació es mostra el diagrama del nostre model.



5.4. FRAMEWORK OPENFRAME

OpenFrame és un framework de desenvolupament J2EE que te com a missió integrar i estendre les diferents solucions en cada àmbit d'aplicació (traces, mailing, bases de dades, etc.).

OpenFrame proporciona una arquitectura:

- De cost reduït.
- Flexible i escalable.
- Oberta, basada en estàndards i no lligada a cap proveïdor.
- Fàcil d'evolucionar, ampliar i adaptar a les necessitats.
- Fiable, estable i provada.
- D'alt rendiment.

A més, permet, entre d'altres:

- Accelerar el desenvolupament d'aplicacions J2EE, simplificant el cicle de desenvolupament.
- Facilitar l'operació i gestió de les aplicacions generades.

- Proporcionar una arquitectura d'aplicacions consistent entre aplicacions, coherent per a tots els desenvolupaments.
- Minimitzar la corba d'aprenentatge dels desenvolupadors.

Els objectius principals d'aquest framework, són els següents:

- Programació Orientada a Interfícies: Oferir mitjançant interfícies l'accés a la implementació.
- Configuració declarativa: Configurar tots els serveis i elements de l'aplicació de forma declarativa sense afectar el codi.
- Solució Oberta: Poder afegir i intercanviar qualsevol peça amb un cost molt reduït.
- Simplificar la complexitat inherent a J2EE.
- Oferir components de desenvolupament.
- Proporcionar eines de suport que facilitin el cicle de desenvolupament.

5.4.1. Components base

Els tres mòduls principals d'openFrame són Struts, Spring i Hibernate.

5.4.1.1. Struts

Struts proporciona les següents facilitats:

- Un controlador principal ja implementat.
- Gestió automàtica dels formularis amb refresc entre pantalles i la seva possible validació.
- Gestió dels errors presentant-los a la vista.
- Internacionalització de l'aplicació (multiidioma).
- Llibreria de tags per a poder ser utilitzats a les vistes.
- Permet afegir filtres de procés (patró Decorating Filter).
- Un únic fitxer de configuració que lliga els controladors amb els formularis de l'aplicació i deixa clara la seva interacció i temps de vida.

5.4.1.2. Spring

Spring proporciona, entre d'altres, les següents facilitats:

- Un contenidor centralitzat d'objectes i serveis, totalment configurable amb fitxers XML.
- Mitjançant l'ús de la inversió de control, en particular la injecció de dependències permet la configuració d'objectes fora del codi de l'aplicació (el contenidor s'encarrega de la instanciació) i de manera no intrusiva (els objectes configurats no estan lligats a Spring, ni han de conèixer les seves classes).

- Redueix el codi d'aplicació dedicat a configurar i localitzar recursos (JNDI, JTA, etc.) en encarregar-se el framework. El codi de l'aplicació així es fa més llegible en tenir principalment lògica d'aplicació.
- Facilita best practices com ara programar contra interfícies en lloc de contra classes.
- Això promou el desacoblament de serveis (pensar en els objectes de l'aplicació com serveis, que expressen la seva funcionalitat com interfícies i abstreuen els seus detalls de configuració de la vista del programador, facilita el canvi d'una implementació concreta a una altra).
- Estructurar en serveis la lògica basant-se només en POJO's e interfícies facilita els tests unitaris (no és necessari el contenidor d'EJB per a les proves i els serveis són fàcils d'emular).
- Gestió de transaccions sense ús d'API's específiques mitjançant l'ús d'Aspect Oriented Programming (AOP).

5.4.1.3. Hibernate

Hibernate com a capa d'accés a dades proporciona les següents facilitats:

- Un mapeig objecte-relacional flexible (taula per classe, múltiples objectes per registre, múltiples taules per objecte, tot tipus de relacions 1-n, n-m, etc.).
- Persistència d'objectes de manera transparent (no intrusiu, sense imposar interfícies o classes estranyes, només arrays i collections estàndard java).
- Llenguatge de querys independent de la BD (HQL).
- Possibilitat d'accés natiu tradicional (T-SQL, PL-SQL, etc.) i crides a lògica a la BD (stored procedures, packages, etc.).
- Tota la configuració (mapejos, querys HQL, etc.) pot definir-se a fitxers de configuració XML, no hardcoded al codi java.
- Catxé (multi-layer, threadsafe, non-blocking, etc.).
- Altres optimitzacions (lazy initialitzation, etc.).
- Integració J2EE (EJB 3.0, JMX, JTA, etc.).
- Extensible (nous dialectes sql, generadors de claus propis, etc.).
- Suporta múltiples DB (Oracle, DB2, Sybase, etc.)

5.5. ARQUITECTURA GENERAL D'APLICACIONS OPENFRAME

L'arquitectura d'openFrame es basa en una arquitectura Model-Vista-Controlador (MVC), on existeix un procés d'abstracció que permet dividir una aplicació en components lògics que poden ser creats més fàcilment.

L'arquitectura MVC separa les dades d'una aplicació, la interfície d'usuari i la lògica de negoci en tres components ben diferenciats. Aquests components són els següents:

- *Model:* Representació específica de la informació amb la qual el sistema opera. El model es limita al relacionat amb la vista i el seu controlador facilita les presentacions complexes.

- *Vista:* Representa el model en un format adequat per interactuar, normalment la interfícies d'usuari.

- *Controlador:* Defineix el comportament de l'aplicació, interpreta els esdeveniments de l'usuari i els mapeja en accions a realitzar sobre el model. Depenent de l'acció, el controlador pot selecionar una nova vista a mostrar com resposta a la petició realitzada per l'usuari.

El flux d'aquesta arquitectura seria el següent:

1. L'usuari introdueix les dades i les accepta o prem un botó. D'aquesta manera, la Vista envia l'acció realitzada al Controlador.
2. El Controlador coordina i realitza els canvis en el Model com resposta a l'acció rebuda des de la Vista.
3. El Model canviat avisa a les Vistes associades dels seus canvis.
4. Les Vistes accedeixen al Model per obtenir les noves dades i s'actualitzen amb aquestes noves dades.
5. El Controlador selecciona i mostra la pantalla del resultat de la petició.

Així doncs, a més de l'estructura lògica dels components segons el patró MVC, openFrame defineix la seva ubicació segons una arquitectura en 3 capes.

5.6. ARQUITECTURA 3 CAPES.

Aquest tipus d'arquitectura té per objectiu principal la separació de la lògica de negoci de la lògica de disseny.

L'avantatge principal es que el desenvolupament es pot dur a terme en diferents nivells i, en cas d'haver canvis en qualcú, només s'ha d'actuar sobre el nivell requerit.

Quant a desenvolupament d'una aplicació, es refereix, el treball queda totalment distribuït en els tres nivells corresponents.

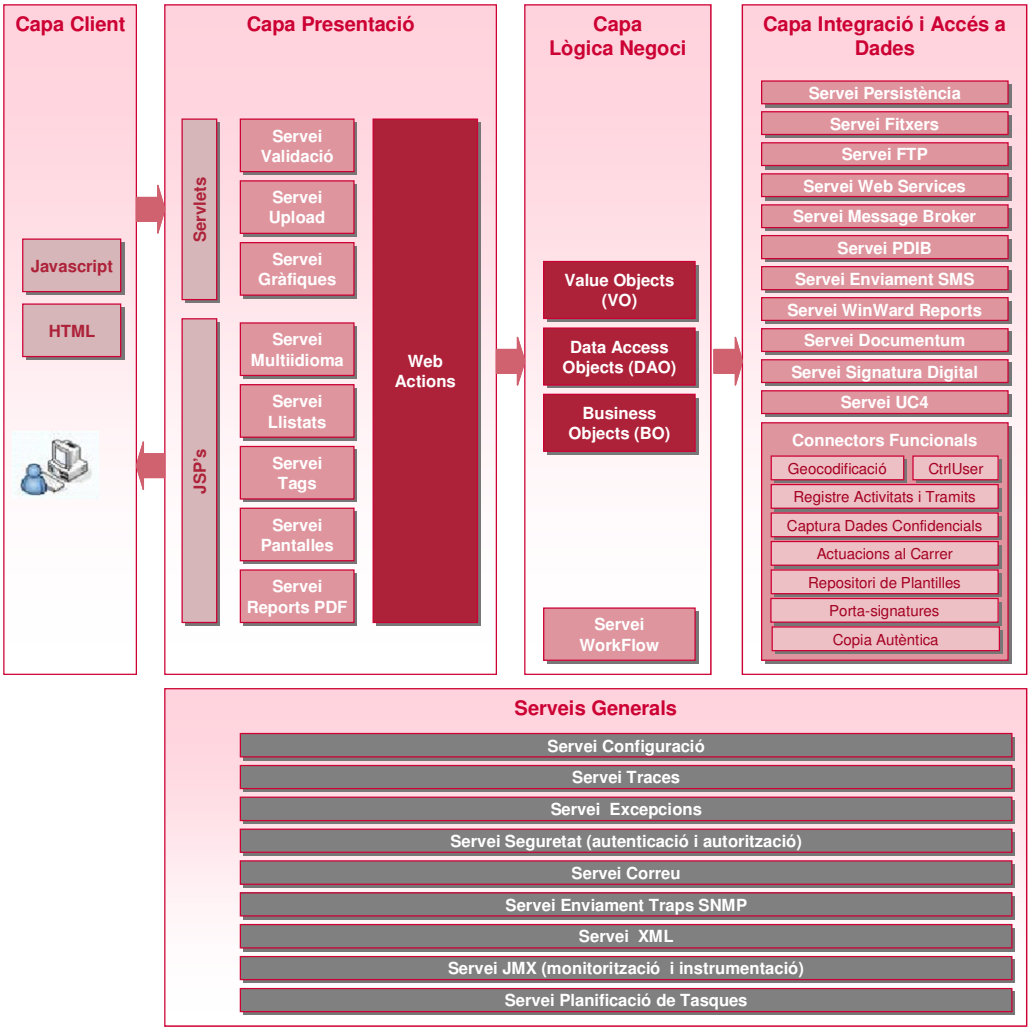
Les tres capes són les següents:

- *Capa de Presentació:* Aquesta és la capa que el sistema presenta a l'usuari per comunicar-li la informació i per capturar la informació per part de l'usuari en un mínim procés. Aquesta capa es comunica únicament amb la capa de negoci

- *Capa de Negoci:* Capa on es reben les peticions de l'usuari i s'envien les respostes un cop finalitzat el procés. S'estableixen totes les regles de negoci que

s'han d'acomplir. Aquesta capa es comunica amb la capa de presentació per rebre les sol·licituds i presentar les dades, i amb la capa de dades, per connectar amb el gestor de base de dades per emmagatzemar o recuperar dades.

- *Capa d'Integració i Accés a Dades*: Capa on resideixen les dades i, lògicament, l'encarregada d'accedir a les mateixes. Està formada per un o més gestors de base de dades els quals reben, des de la capa de negoci, les sol·licituds d'emmagatzemament o recuperació de la informació.



5.6.1. Capa Integració i Accés a Dades

Els serveis d'integració tenen com a objectiu principal l'accés a aspectes infraestructurals (tant sistemes interns com externs), en el qual es necessari definir el protocol d'integració. Dins els serveis d'integració, tenim:

- *Servei de Persistència:* Ús de bases de dades relacionals sense haver de realitzar tractaments específics de pas d'objectes de negoci a taules.

- *Servei FTP:* Permet l'enviament de fitxers mitjançant el protocol FTP.

- *Servei de Web Services:* Permet la integració amb sistemes externs mitjançant Web Services (protocol SOAP), tant consumir funcionalitat externa com exposar funcionalitat de la pròpia aplicació.

Pel desenvolupament de l'aplicació ens basarem, principalment, en el servei de persistència.

5.6.1.1. Servei de Persistència

Permet persistir i recuperar dades entre l'aplicació i el motor de base de dades. La base tecnològica està basada en Spring i Hibernate, i entre d'altres característiques ofereix:

1. Suport per a DAO (Data Acces Object) orientat principalment a facilitar la feina amb les tecnologies d'accés a dades.

2. Traducció d'excepcions específiques de la tecnologia utilitzada a una jerarquia d'excepcions genèrica.

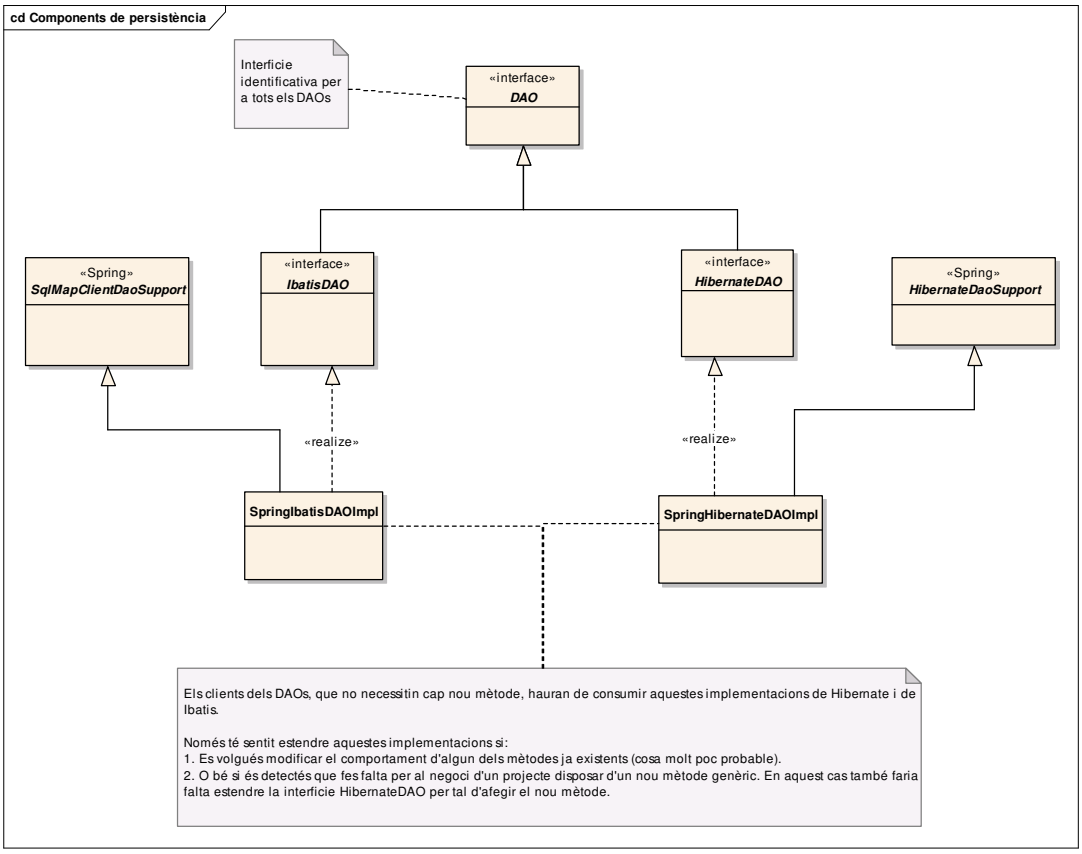
3. Transaccionalitat declarativa.

El servei ofereix dos possibles implementacions per manegar la persistència de les aplicacions:

- iBATIS
- Hibernate

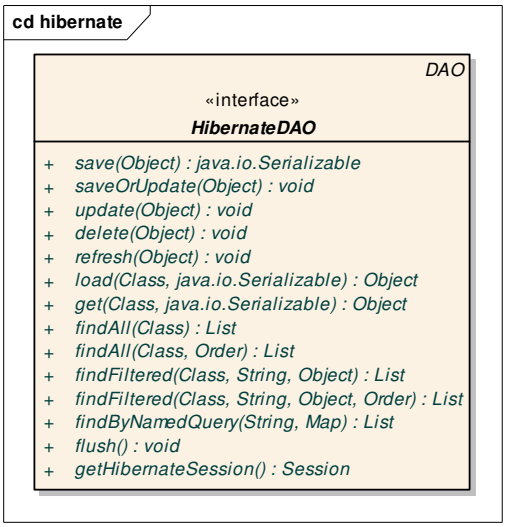
Interfícies i components genèrics orientats a Hibernate

El servei de persistència suporta treballar amb iBATIS i Hibernate, i defineix el següent conjunt d'interfícies i components:



Del conjunt de components que es mostren al diagrama, els que s'apliquen per Hibernate, són els següents:

- *HibernateDAO:* Interfície de negoci a consumir per les aplicacions que vulguin treballar amb Hibernate. Els serveis que ofereix es mostren a la següent imatge, i corresponen a la major part de mètodes típics per a interactuar amb Hibernate:



- *SpringHibernateDAOImpl:* És la implementació per Spring de la interfície HibernateDAO que ofereix el framework. Aquesta classe estén de la classe de suport que ofereix Spring per treballar amb Hibernate:

  org.springframework.orm.hibernate3.support.HibernateDaoSupport

- *LocalSessionFactoryBean:* Aquesta classe estén la classe de Spring "org.springframework.orm.hibernate3.LocalSessionFactoryBean" per evitar possibles excepcions al carregar les propietats d'Hibernate, ja que el comportament per defecte és que si una propietat no es pot carregar provoca una excepció.

Transaccionalitat declarativa

És aconsellable gestionar la transaccionalitat de manera declarativa ja que permet mantenir els serveis de negoci lliures de codi repetitiu de tractament de les transaccions en cada mètode. Encara més, hi ha semàntica com el comportament de propagació i el nivell d'aïllament poden canviar-se en un fitxer de configuració i no afectar a la implementació dels serveis de negoci.

- Definició d'un proxy base: Cal definir la configuració per defecte que tindran els nostres objectes de negoci. Cal editar el fitxer openFrame-services-persistence.xml, definim un TX Manager de tipus JTA i l'utilitzem per crear un TransactionProxyFactoryBean abstracte. Aquesta serà la configuració que heretaran tots els nostres objectes de negoci transaccionals.

  El baseProxy és una plantilla que utilitzarem per crear instàncies dels nostres objectes de negoci transaccionals i, per tant, caldrà redefinir l'atribut target indicant el bean de negoci i possiblement també redefinir els transactionAttributes.

```
<!--  Base Proxy -->
<bean id="baseProxy"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean" abstract="true">
    <property name="transactionManager">
        <ref local="transactionManager" />
    </property>
    <property name="target">
        <bean class="java.lang.Object" />
    </property>
    <property name="transactionAttributes">
        <props>
            <prop key="get*">PROPAGATION_SUPPORTS,readOnly</prop>
            <prop key="find*">PROPAGATION_SUPPORTS,readOnly</prop>
            <prop key="load*">PROPAGATION_SUPPORTS,readOnly</prop>
            <prop key="store*">PROPAGATION_REQUIRED</prop>
            <prop key="save*">PROPAGATION_REQUIRED</prop>
            <prop key="delete*">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>
```

5.6.1.2. Implementació per la nostra aplicació

Per a la nostra aplicació haurem de fer el següent:

- Creació dels artefactes d'Hibernate (hibernate.cfg.xml, fitxers de mappings *.hbm.xml i VOs): En el nostre cas haurem de crear els següents fitxers Productes.hbm.xml, Peticions.hbm.xml, Paquets.hbm.xml, Grups.hbm.xml, Peticions_paquets.hbm.xml i Grups_novell.hbm.xml. A continuació es mostra el contingut del fitxer hibernate.cfg.xml:

```
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property name="hibernate.connection.url">jdbc:oracle:thin:@172.16.17.49:1541:CORDE1</property>
        <property name="hibernate.connection.username">gcanec_u</property>
        <property name="hibernate.connection.password">gcanec_p</property>
        <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>

        <!-- Application Mapping Files -->

        <mapping resource="hibernate/mappings/Peticions.hbm.xml" />
        <mapping resource="hibernate/mappings/Productes.hbm.xml" />
        <mapping resource="hibernate/mappings/Paquets.hbm.xml" />
        <mapping resource="hibernate/mappings/Peticions_Paquets.hbm.xml" />
        <mapping resource="hibernate/mappings/Grups.hbm.xml" />
        <mapping resource="hibernate/mappings/Grups_Novell.hbm.xml" />

        <!-- END OF Application Mapping Files -->
    </session-factory>
</hibernate-configuration>
```

5.6.2. Capa Lògica Negoci

Un cop acabada la capa de persistència és el moment de dedicar-nos a la capa de negoci de la nostra aplicació. Degut a que la nostra aplicació té diferents casos d'ús per a la gestió de l'aplicació, es necessari crear un objecte de negoci per cadascun dels casos d'ús. Per poder desacoblar la capa de negoci de la seva implementació concreta que normalment pot estar lligada a la capa de persistència, en aquest cas Hibernate, farem servir una interfície com objecte de negoci.

Utilitzant aquest procediment aconseguim que la nostra aplicació sigui més flexible als canvis, ja que si es modifica la capa de persistència no s'ha de canviar la capa de presentació ni la capa de persistència directament, però si indirectament.

5.6.2.1. Value Objects

Objectes de transferència que s'utilitzen per a la comunicació entre les 3 capes, i representen les dades de les **Entitats de Negoci** que, en general, es corresponen amb entitats del model de dades:

- Cada entitat es designa amb un nom Entitat que intervé en la nomenclatura i facilita la identificació dels fitxers a partir del nom: el VO d'una entitat es crea com un POJO anomenat EntitatVO en un fitxer EntitatVO.java

Els Vos han de complir les següents característiques:

- Ser serialitzables.
- Tenir les seves propietats protegides, oferint mètodes setPropietat i getPropietat (isPropietat en cas de booleans) per accedir a les mateixes.
- No tenir dependències de cap capa.

5.6.2.2. Implementació per la nostra aplicació

Per a la nostra aplicació, s'han de crear els VOs necessaris per a representar cadascuna de les entitats del model de dades. S'han de crear al package ***es.bcn.gca.nec.common.model*** al projecte gcanecCommon:

5.6.2.3. Tipus Objectes

Es poden construir utilitzant dos tipus d'objectes que representen el comportament de les Entitats de Negoci:

- BOs (Business Objects): Són POJOS en els que la lògica de negoci es programa en els seus mètodes:
    - El noms dels fitxers són: EntitatBO.java (la interfície) i EntitatBOImpl.java (la implementació).
    - Al fitxer applicationContext.xml es defineix cada BO com un bean d'Spring. Al bean d'un BO se li injecten els DAO's (genèrics o específics) i els serveis amb els quals hagi d'interaccionar en les seves operacions amb la Capa d'Integració i Accés a Dades.
    - El nom d'aquest bean ha de ser entitatBO si no té necesstitat de transaccionalitat i entitatBOTarget si requereix transaccionalitat. En el segon cas, es defineix a més un altre bean entitatBOTransaccional, que

hereta d'un bean transaccional del servei de persistència (baseProxy) i que té al entitatBOTarget com target al que aplicar transaccionalitat.

- Stateless Session EJBs, que hauran de ser locals si no s'espera accés remot RMI/IIOP als mateixos des d'altres aplicacions, o remots si s'espera aquest accés. Com el cas local està totalment desaconsellat ja que es recomana fer ús de BOs, la normativa es refereix al cas remot:

    - Es recomana que la lògica s'implementi igualment en un BO (interfície EntitatBO amb implementació EntitatBOImpl), i que el EJB només faci de façana del mateix.
    - Cal definir també una interfície EntitatBORemote com la EntitatBO, però amb els mètodes llençant també una RemoteException.
    - Els noms dels fitxers EJB han de ser: EntitatEJBHome.java (interfície EJBHome), EntitatEJB.java (interfície EJBObject que a més estén EntitatBORemote) i EntitatEJBBean.java (classe AbstractStatelessSessionBean que a més implementa EntitatBORemote).
    - Al fitxer applicationContext.xml es defineix cada EJB com un bean d'Spring. Al bean d'un EJB se li injecta la interfície del BO del que és façana.
    - El nom del bean corresponent a un EJB ha de ser entitatBOImplEjb.

5.6.2.4. Implementació per la nostra aplicació

Tal i com s'ha comentat anteriorment es recomana, sempre que no sigui accés remot, BOs. Per tant, per a la nostra aplicació fem el següent:

El primer que hem de fer es crear una interfície per cada objecte de negoci de la nostra aplicació (xxxBO.java). Es creen al package ***es.bcn.gca.nec.common.model.bo*** al projecte gcanecCommon. Cal que els noms siguin identificatius dels objectes de la capa de persistència que representen.

Aquestes interfícies han de contenir tots els mètodes abstractes que necessitem per la gestió dels nostres objectes de negoci. Alguns dels mètodes que contenen aquestes interfícies són per poder consultar, modificar o esborrar elements de les taules, corresponents, de la base de dades.

Les classes que s'han de crear estaran vinculades amb la capa de persistència utilitzada, en aquest cas Hibernate, fent servir els objectes Value Objects (VO).

Aquestes classes (xxxBOImp.java) s'han de crear al projecte **gcanecBusiness** al package ***es.bcn.gca.nec.business.model.bo.impl.*** Implementaran les pròpies interfícies que s'han creat anteriorment.

Amb tot això s'han creat els objectes necessaris per a la nostra capa de negoci. El següent pas és preparar la configuració d'aquesta capa per poder integrar-la amb els serveis d'openframe.

Per aconseguir això hem d'afegir els beans de negoci al fitxer ***applicationContext.xml*** del projecte **gcanecBusiness** al directori /**resources/spring**.

Hem de definir els beans de negoci i és aquí, on s'ha d'afegir, si volem que la comunicació amb la base de dades sigui transaccional.

```
<!-- Beans de negoci de la teva aplicacio. Pots moure'les a fitxers externs i importar-los -->
<!-- Bean de negoci amb un DAO univaersal per a Hibernate per a accedir a dades -->
<bean id="peticionsBO"
    class="es.bcn.gca.nec.business.model.bo.impl.PeticionsBOImpl"
    singleton="true" lazy-init="false">
    <property name="logService" ref="loggingService" />
    <property name="hibernateDao" ref="hibernateGenericDao"/>
</bean>
<bean id="productesBO"
    class="es.bcn.gca.nec.business.model.bo.impl.ProductesBOImpl"
    singleton="true" lazy-init="false">
    <property name="logService" ref="loggingService" />
    <property name="hibernateDao" ref="hibernateGenericDao"/>
</bean>
<bean id="paquetsBO"
    class="es.bcn.gca.nec.business.model.bo.impl.PaquetsBOImpl"
    singleton="true" lazy-init="false">
    <property name="logService" ref="loggingService" />
    <property name="hibernateDao" ref="hibernateGenericDao"/>
</bean>
<bean id="grupsBO"
    class="es.bcn.gca.nec.business.model.bo.impl.GrupsBOImpl"
    singleton="true" lazy-init="false">
    <property name="logService" ref="loggingService" />
    <property name="hibernateDao" ref="hibernateGenericDao"/>
</bean>
<bean id="target_peticionsBO" parent="baseProxy" lazy-init="true">
    <property name="target">
        <ref bean="peticionsBO" />
    </property>
</bean>
<bean id="target_productesBO" parent="baseProxy" lazy-init="true">
    <property name="target">
        <ref bean="productesBO" />
    </property>
</bean>
<bean id="target_paquetsBO" parent="baseProxy" lazy-init="true">
    <property name="target">
        <ref bean="paquetsBO" />
    </property>
</bean>
<bean id="target_grupsBO" parent="baseProxy" lazy-init="true">
    <property name="target">
        <ref bean="grupsBO" />
    </property>
</bean>
```

Les primeres entrades defineixen el nostres beans de negoci. Les segones hereten del bean transaccional baseProxy aplicant (property target) transaccionalitat als nostres beans de negoci. En el fitxer openFrame-services.persistence.xml dins el directori /resources/spring del projecte gcanecBusiness, es defineixen els nivells de transaccionalitat que s'aplicaran als diferents mètodes dels nostres bean.

```
<beans>
    <!-- Hibernate SessionFactory -->
    <bean id="sessionFactory"
        class="net.opentrends.openframe.services.persistence.hibernate.LocalSessionFactoryBean" lazy-init="true">
        <property name="configLocations">
            <list>
                <value>classpath:hibernate/config/hibernate.cfg.xml</value>
            </list>
        </property>
        <property name="dataSource"><ref local="dataSource" /></property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">${hibernate.dialect}</prop>
                <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
                <prop key="hibernate.jdbc.use_scrollable_resultset">${hibernate.jdbc.use_scrollable_resultset}</prop>
                <prop key="hibernate.transaction.factory_class">${hibernate.transaction.factory_class}</prop>
                <prop key="hibernate.transaction.manager_lookup_class">${hibernate.transaction.manager_lookup_class}</prop>
            </props>
        </property>
    </bean>

    <!-- Transaction manager for a single Hibernate SessionFactory -->
    <!-- Generic TX for any App Server
    <bean id="transactionManager"
        class="org.springframework.transaction.jta.JtaTransactionManager" lazy-init="true">
        <property name="autodetectTransactionManager" value="false" />
    </bean>
    -->
    <!-- Specific TX for IBM WAS App Server -->
    <bean id="transactionManager"
          class="org.springframework.transaction.jta.WebSphereUowTransactionManager" lazy-init="true">
    </bean>

    <!--  Base Proxy -->
    <bean id="baseProxy"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean" abstract="true">
        <property name="transactionManager">
            <ref local="transactionManager" />
        </property>
        <property name="target">
            <bean class="java.lang.Object" />
        </property>
        <property name="transactionAttributes">
            <props>
                <prop key="get*">PROPAGATION_SUPPORTS,readOnly</prop>
                <prop key="find*">PROPAGATION_SUPPORTS,readOnly</prop>
```

El segon bean, el transaccional, serà el que utilitzarem des de la resta de codi.

5.6.2.5. Data Acces Objects (DAO)

Objectes per al accés a dades:

- El servei de persistència basat, en aquest cas, en Hibernate de l'arquitectura, proporciona DAOs genèrics que resolen les operacions habituals d'accés a dades (creació d'entitats –INSERT-, consultes -SELECTS-, modificació –UPDATE-, etc.).
- En el cas poc habitual de requerir operacions més complexes, l'aplicació pot definir-se els seus propis DAOs estenent dels anteriors. La notació seria EntitatDAO (fitxer EntitatDAO.java).

5.6.3. Capa Presentació

Un cop acabades les capes de persistència i negoci, és el moment de desenvolupar la capa de presentació. Ja que aquest procés no és senzill, es decideix dividir-lo en subapartats per fer-lo més entenedor i fàcil de seguir.

5.6.3.1. Serveis de Presentació

El Serveis de presentació (figura pag. 14) tenen com objectius proporcionar la infraestructura necessària per a desenvolupar aplicacions amb Interfície Gràfica d'Usuari (GUI) de tipus Web:

- Al Navegador Web es mostra la part client de la nostra aplicació com HTML.
- S'utilitza Javascript per a proporcionar un GUI mes àgil a l'usuari.
- S'habiliten mecanismes Ajax per a carregar informació sense recarregar la pagina.

En el punt central de l'arquitectura de la capa de presentació es troben els *Actions*, gestors de les peticions rebudes.

En aquesta capa es troben els serveis següents:

- Servei de Pantalles. Permet definir pantalles consistentment, amb un Look & Feel comú, afavorint la reutilització d'elements gràfics i de maquetació, i aïllant aquestes dels dades i continguts presentats.
- Servei de Presentació amb Tags. Utilitzats en les JSPs per a generar pàgines HTML amb la presentació final.
- Servei de Llistats. Per a mostrar llistats a l'usuari amb capacitat d'ordenar per columna, paginar els resultats i navegar per a les pagines, i generar formats més convenients per al seu tractament (PDF, Excel).
- Servei Multiidioma. Permet internacionalitzar el GUI d'aplicacions dirigides a usuaris amb diferents idiomes.
- Servei de Validació. Per a validar les entrades de l'usuari (dates, rangs, etc.) amb possibilitat de realitzar-la tant a la part client com a la part servidora.
- Servei de Reports PDF. Per a generar documents PDF amb format incorporant dades rebudes des de la capa de negoci.
- Servei d'Upload de Fitxers. Per a que l'usuari pugi fitxers i per a tractar-los quan arriben al servidor.
- Servei de Generació de Gràfiques (charts). Per a generar diversos tipus de gràfiques (àrees, línies, pastissos...) partint de series de dades.

Alguns d'aquests serveis comparteixen el mateix *jar* (*openFrameIMI-services-web*):

- Servei de Pantalles.
- Servei de Presentació amb Tags.
- Servei de Llistats.

A continuació s'ofereix un resum dels objectius i característiques de cada servei.

- **Servei de Pantalles**

Tota aplicació Web requereix d'un **Look & Feel (L&F)** comuna. Si el L&F es defineix en cada pàgina de forma manual, un canvi de L&F en la aplicació provocaria la necessitat de modificar una a una cadascuna de les pàgines de la aplicació. En un bon disseny és important que siguem capaços de separar el L&F del contingut de les pàgines.

El Servei de Pantalles permet definir l'estil comú especificant, per exemple, que usarem una capçalera, un cos, un menú a l'esquerra i un peu. Les pàgines de la aplicació no haurien de definir aquestes parts, només haurien de preocupar-se de definir el seu contingut. La pàgina final es construirà de forma dinàmica usant les parts comunes i les parts específiques de la pàgina.

El Servei de Pantalles permet:

- La creació de pantalles mitjançant l'acoblament de vàries parts.
- Definir herència entre pantalles, pel que podem definir el cas general i les seves particularitats.
- Definir diferents tipus de pantalles: vertical, portal, horitzontal...

Implementació basada en Spring, Struts i Tiles

La solució adoptada pel framework es basa en la utilització de Spring amb Struts i Tiles.

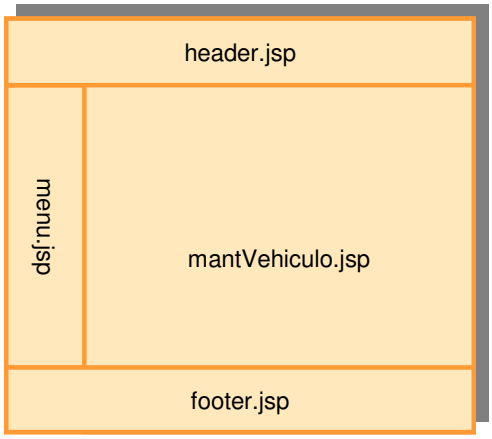
La peça que encaixa tot aquest mecanisme és com el contingut de les seccions 'header', 'menu', 'body' i 'footer' s'insereixen conjuntament per a formar la pàgina final. Per a això, existeix el fitxer de definicions de pàgines **tiles-definitions.xml**.

```
<definition name="pages.peticions" extends="site.mainLayout">
      <put name="quickSearch" value="/WEB-INF/jsp/peticions/peticioListQuickSearchPg.jsp" />
   <put name="body" value="/WEB-INF/jsp/peticions/peticioListPg.jsp" />
   <put name="useCaseNameKey" value="jsp.pet.title" type="string" />
</definition>
```

Codi 1 Exemple de Definició d'una Pantalla

Definició d'una pantalla basada en una plantilla

Amb la definició dalt mostrada, estem creant un nova pàgina definida pel layout de nom '**site.mainLayout.jsp**', on el contingut de cadascuna de les parts del layout està definida amb cadascun dels tags put. D'aquesta forma, es bolca el contingut de cadascuna de les pàgines definides en cadascun dels paràmetres en el layout.



Configuració

Per a configurar l'ús de plantilles és necessari incorporar en el fitxer descriptor de Struts '**struts-config.xml**', el següent contingut:

```
<controller>
   <set-property property="processorClass"
            value="net.opentrends.openframe.services.web.
             struts.ExtendedDelegatingTilesRequestProcessor"/>
</controller>

<plug-in className="org.apache.struts.tiles.TilesPlugin">
   <set-property property="definitions-config"
         value="/WEB-INF/classes/tiles/tiles-definitions.xml"/>
    <set-property property="moduleAware" value="true" />
</plug-in>
```

on per un costat se li indica a Struts la classe que realitzarà el processat de cadascuna de les peticions, i per un altre se li indica el fitxer de configuració Tiles.

Ús

La utilització del servei de pantalles basat en Tiles implica 2 passos diferenciats:

- Definició del template: En Tiles l'element bàsic és el template. El template no és més que una pàgina JSP que usa una llibreria de tags JSP per a definir el layout d'una pàgina. El template serveix com definició de com es situaran les pàgines sense especificar el contingut. Aquest s'insereix en temps d'execució en la pàgina template. Totes les pàgines de l'aplicació poden compartir un mateix template o fer ús de templates diferents.

o Definició de les pantalles basades en un o més templates: Un cop definit el template en el qual basarem les nostres pantalles, podem definir com serà el contingut de cada pantalla.

Per a definir una pantalla utilitzarem el següent format:

```
<tiles-definitions>

<definition name="nombre" path="path">
   <put name="nombreParam"  value="valor" />
   ...
   <putList name="nombreList">
            <add value="valor1"/>
            <add value="valor2"/>
   </putList>
</definition>
```

Podem definir els següents atributs per a la definició:

- name: Nom de la pantalla. Aquest nom ha de ser únic.
- path: Ruta a la pàgina JSP que construeix la pantalla (el template definit).

Per a especificar quins són els valors específics a substituir en el template podem usar:

- put: Amb cada paràmetre a passar a la definició definirem una entrada put.
- putList: Serveis per a poder passar un paràmetre que estigui compost per una llista de valors.

Suposem per exemple la següent definició:

```
<definition name="site.mainLayout"
            path="/WEB-INF/jsp/layouts/mainLayoutPg.jsp">
            <put name="header" value="/WEB-INF/jsp/includes/headerPg.jsp" />
            <put name="menu" value="/WEB-INF/jsp/includes/menuPg.jsp" />
            <put name="useCaseTitle"
                     value="/WEB-INF/jsp/includes/useCaseTitlePg.jsp" />
            <put name="errors" value="/WEB-INF/jsp/includes/errorsPg.jsp" />
            <put name="quickSearch" value="/WEB-INF/jsp/includes/blankPg.jsp" />
            <put name="body" value="/WEB-INF/jsp/includes/blankPg.jsp" />
            <put name="footer" value="/WEB-INF/jsp/includes/footerPg.jsp" />
            <put name="useCaseNameKey" value="inherit.me" type="string" />
   </definition>
```

Mitjançant ‘put name=’header’ value=’ /WEB-INF/jsp/includes/headerPg.jsp’ ’ especifiquem que l’espai de la plantilla amb el codi ‘tiles:insert attribute=”header” ’ serà substituït per la pàgina ‘headerPg.jsp’.

Per una altra banda, és molt comú que determinades parts de les nostres pantalles siguin comunes a moltes pantalles (capçalera, peu...) Per a cobrir aquests casos, podem especificar que una definició de pantalla estén un altra definició de pantalla.

Per a estendre des d'una pantalla a un altra pantalla s'utilitzarà l'atribut extends, el valor del qual, serà el nom de definició pare.

```
<definition name="pages.peticions" extends="site.mainLayout">
        <put name="quickSearch" value="/WEB-INF/jsp/peticions/peticioListQuickSearchPg.jsp" />
       <put name="body" value="/WEB-INF/jsp/peticions/peticioListPg.jsp" />
       <put name="useCaseNameKey" value="jsp.pet.title" type="string" />
</definition>
```

En l'exemple es mostra com s'especifica una definició 'site.mainLayout' que utilitza el layout definit per a la pàgina 'mainLayout.jsp'. Aquesta definició ha estat estesa per 'pages.peticions' especificant únicament com serà el títol i el cos del layout, mentre que manté invariable la capçalera, menú i peu del pare.

- **Servei de Presentació amb Tags**

Proporciona *Tags* per a utilitzar en les fulles JSP. Entre d'altres es proporcionen les següents possibilitats:

- Crear una pàgina única per a presentar formularis per a diferents propòsits: creació, edició o consulta.
- Marcar camps requerits automàtica en la pàgina (si s'ha configurat que el camp és requerit en el servei de validació).
- Campos de tipus data amb assistent calendari.
- Conversions automàtiques de dades (eliminació de blancs, pas a majúscules).
- *Tooltips* d'ajuda associada.
- Campos d'edició rica HTML (amb negreta, justificats…).
- Accés directe als components per teclat utilitzant combinacions de tecles.
- Tabulació automàtica al superar el màxim de caràcters d'un component.
- Obtenció dels valors d'un component segons el valor modificat en un altre component sense recarregar la pàgina, per exemple, refresc dels valors d'un *combo* al modificar un altre.

Arquitectura i components

L'arquitectura del Servei de Tags es basa en l'ús de llibreries de tags JSP.

A manera de resum, l'arquitectura que hi ha al darrere de les llibreries de tags és la mostrada a continuació:

1) El contenidor del Servidor d'Aplicacions al generar el contingut de sortida a partir de la pàgina JSP troba la referència '%taglib' on s'indica la localització de la llibreria de tags que resol tot tag de la pàgina que contingui el prefix 'fwk'.
2) Quan troba el tag fwk:text, per exemple, accedeix al descriptor tld i troba que és la classe que generarà el contingut a substituir en la localització del tag
3) El tag realitza la sortida a partir dels atributs passats en la pàgina JSP. Aquests continguts s'incorporen a la pàgina.

Seguint la filosofia d'independència màxima de la implementació, openFrameIMI defineix interfícies per a tots els tags. Actualment s'ofereixen implementacions basades en JSP i Struts.

A l'optar per aquest tipus de configuració basada en interfícies, la migració a una altra tecnologia es podrà realitzar de forma senzilla si és necessari.

Arquitectura de Procediment d'Inicialització Interna

La configuració dels tags es realitza mitjançant el Servei de Configuració com si de classes es tractés. El model inherent a JSP, pel qual és el contenidor qui instància els tags no permet fer ús directe del patró 'Dependency Injection'. En qualsevol cas, s'utilitzen els mecanismes definits per al Servei de Configuració per a inicialitzar els tags.

El procediment per a la inicialització d'un tag és el següent:

(1) Es rep la petició de l'usuari (per exemple un submit) en el qual arriba un paràmetre 'reqCode' que indica el mètode de l'acció Struts a executar. Una vegada finalitzada l'acció, aquesta retorna la pàgina JSP que mostrarà el contingut de la resposta.

(2) En la generació del contingut de sortida el contenidor del servidor crida a la inicialització de tots els tags inclosos en la pàgina JSP. Imaginem que en la pàgina JSP tenim un component definit de la següent manera:

```
<fwk:text styleId="petId" property="petId"/>
```

És obligatori que definim aquestes 2 propietats en les nostres pàgines. L'atribut 'property' és equivalent al que s'utilitza en Struts, que permet associar un atribut de l'objecte a representar al formulari. L'atribut 'styleId' servirà per a obtenir la configuració del tag del fitxer de configuració.

El propi tag, en el seu mètode d'inicialització, crida a la classe 'TagUtil' indicant-li com és la seva 'styleId' per a obtenir les seves propietats.

(3) La classe TagUtil consulta el fitxer de configuració '**action-servlet.xml**' i obté el bloc definit en l'entrada corresponent al 'reqCode' que rep com paràmetre i les propietats definides pel 'styleId' del tag.

```
<property name=" tagsConfiguration" >
<map>
   <entry key=" *"  (1)>
   <list>
   ...
   <bean id=" ..."  class=" ..." >
   <property name=" styleId"  value=" nombreEstiloId" />
   </bean>
   </list>
   </entry>
   <entry key=" reqCode" >
   <list>
   <bean ...>
   <property name=" styleId"  value=" nombreEstiloId"  (2)/>
   </bean>
   </list>
   </entry>
</map>
```

(1) Segons el reqCode que arribi com paràmetre a la url '...?reqCode=...' se seleccionarà una entry o una altra per a configurar els tags de la pàgina.

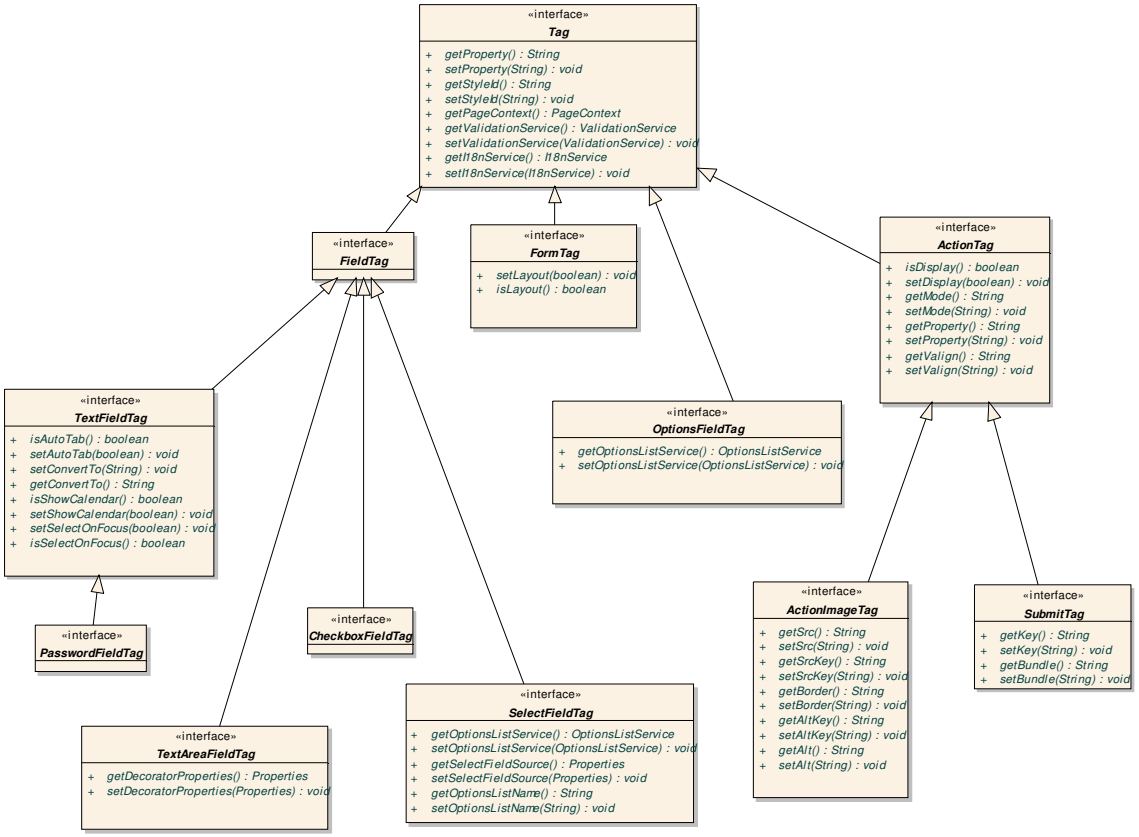
(2) L'estil indicat enllaça amb l'atribut 'styleId' que s'ha definit en la pàgina JSP

(4) El tag, mitjançant commons beanutils, copia les propietats definides sobre les seves propietats. A continuació realitza la generació de la sortida segons les propietats definides.

Interfícies i Components Genèrics

Els tags del servei es troben definits mitjançant interfícies que defineixen els mètodes que s'espera de les implementacions.

En el següent gràfic es mostra la jerarquia d'interfícies definida per openFrameIMI.



Component	Package	Descripció
Tag	net.opentrends.openframe.services.web.taglib	Interfície base de tots els tags. Defineix principalment la necessitat que totes les implementacions hagin definit mètodes d'accés als atributs: • 'styleId'. Identificador del tag en la pantalla • 'pageContext'. Obtenció del context de la pàgina • validationService. Referència al servei de validació • i18nService. Referència al servei d'internacionalització
FieldTag	net.opentrends.openframe.services.web.taglib	Interfície de definició de tots els tags que permeten l'entrada de dades en un formulari
FormTag	net.opentrends.openframe.services.web.taglib	Interfície de definició d'un tag que representa un formulari
OptionsFieldTag	net.opentrends.openframe.services.web.taglib	Interfície de definició d'un tag que representa els valors d'una llista d'opcions

ActionTag	net.opentrends.openframe.services.web.taglib	Interfície tags d'acció d'un formulari
TextFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag entrada d'una dada d'una única línia en un formulari
TextAreaFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag entrada d'una dada de diverses línies en un formulari
CheckboxFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag de marcar un valor en un formulari
FileFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag per a enviar un fitxer en un formulari
SelectFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag de selecció d'una llista de valors
ActionImageTag	net.opentrends.openframe.services.web.taglib	Interfície tag per a usar una imatge de botó per a acceptar un formulari i enviar-lo
SubmitTag	net.opentrends.openframe.services.web.taglib	Interfície tag per a usar una botó per a acceptar un formulari i enviar-ho
PasswordFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag entrada de dades que recullen contrasenyes de l'usuari
RadioFieldTag	net.opentrends.openframe.services.web.taglib	Interfície tag per a selecció d'un ràdio button

Implementació basada en Struts

La implementació basada en Struts de les interfícies mostrades es troben en el package '**net.opentrends.openframe.services.web.struts.taglib.forms.fields**'. Els seus noms coincideixen amb els de les interfícies.

Els tags actuals d'openFrameIMI es basen en extensions dels tags de Struts-Layout.

- **Servei de Llistats**

El servei de llistats permet la presentació de llistats parcials HTML en els quals es permet a l'usuari:

- Ordenar per columna de manera ascendent o descendent.
- Paginació dels resultats i navegació per pàgines mitjançant "avanç" i "reculada" (primera, última, següent, anterior).
- Presentar un nombre determinat de resultats per pàgina.
- Exportació automàtica del llistat a PDF o Excel.

A més, proporciona ajuda a l'usuari, presentant:

- *Rollover* de la fila al passar el ratolí per damunt.
- Desactivació dels botons de navegació depenent de la pàgina en la qual s'estigui situat.
- Indicació de la columna per la qual s'està ordenant, mitjançant icones, mostrant si és ascendent o descendent.
- L&F configurable.

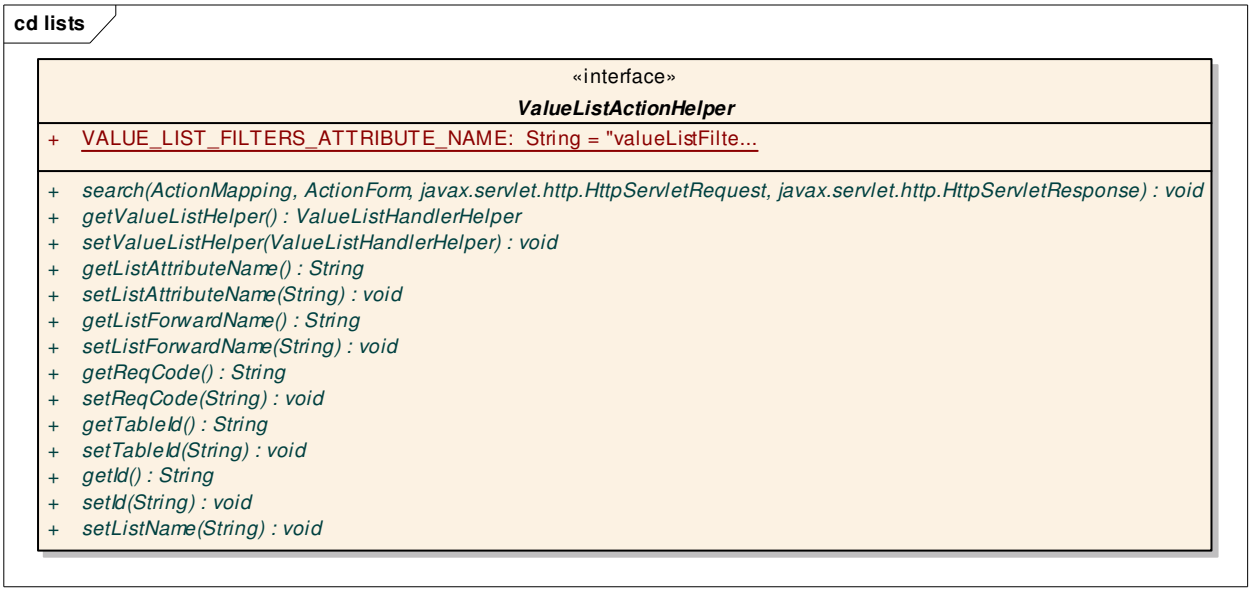
Arquitectura i Components

Existeixen tres tipus de components. Podem classificar-los en:

- Interfícies i components genèrics.
- Implementació dels interfícies.
- Llibreria de tags per a la presentació de resultats.

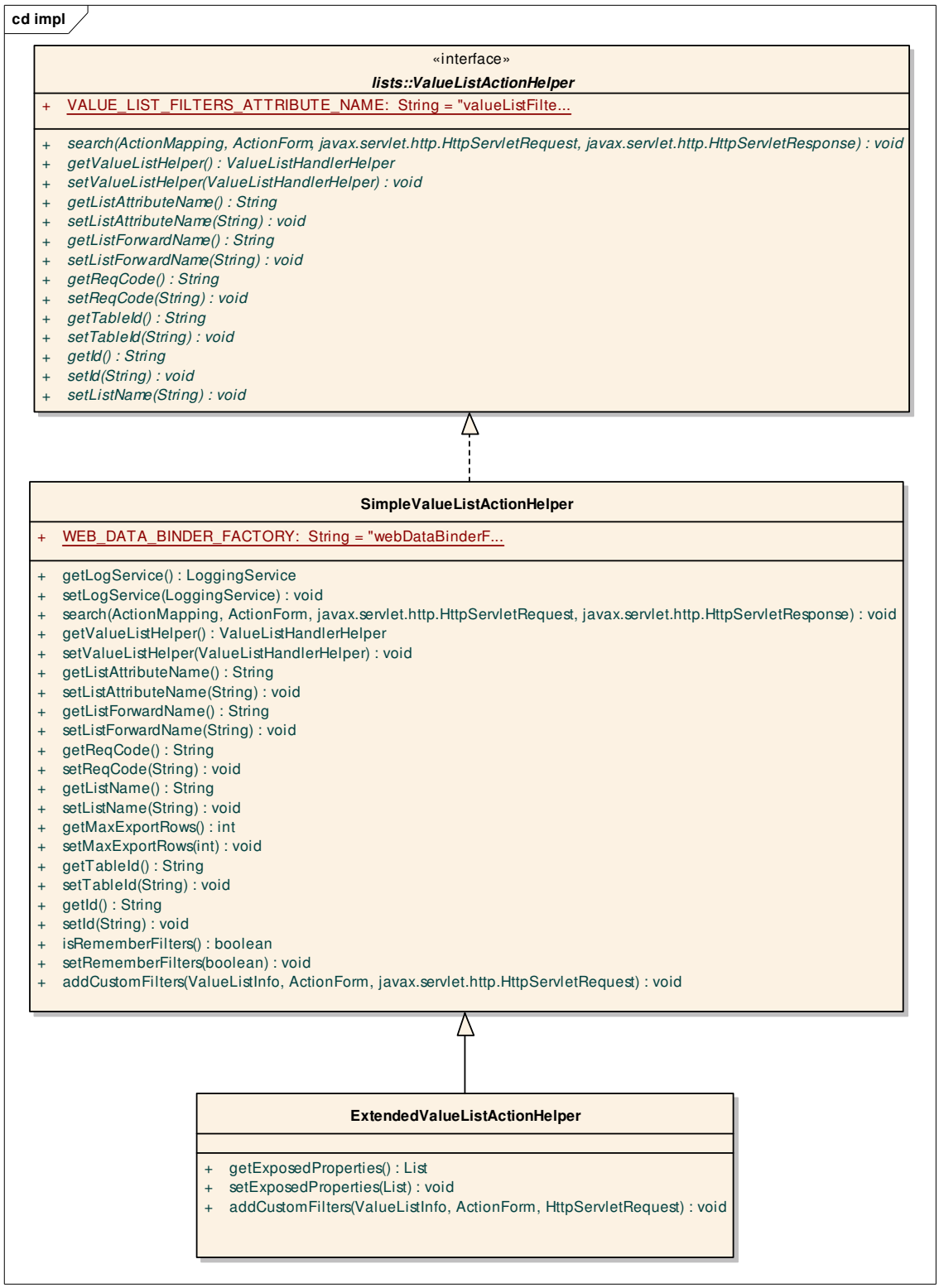
Interfícies i components genèrics

El servei de llistats defineix les següents interfícies:



Implementació de les interfícies

El servei de llistats defineix els següents implementacions:



Configuració

La configuració del servei implica:

- Definir el servei i injectar-li les seves dependències

La definició del servei requereix manipular el fitxer ***openFrame-services-web-lists.xml***, i configurar:

- Un bean ”pare” amb identificador *valueListActionHelper*, que identifica la implementació concreta del servei a utilitzar.

  Per exemple:

```
...
   <bean name="valueListActionHelper" class="net.opentrends.openframe.services.web.
            lists.impl.SimpleValueListActionHelper" lazy-init="true">
            <property name="valueListHelper">
                     <ref bean="valueListHelper" />
            </property>
            <property name="maxExportRows" value="0"></property>
            <property name="logService" ref="loggingService"/>
   </bean>

...
```

- Configurar un bean “simplificador” , amb identificador *valueListHelper*, per a executar les cerques,

  Exemple:

```
...
   <bean id="valueListHelper"
            class="net.mlw.vlh.web.mvc.ValueListHandlerHelper">
            <property name="valueListHandler">
                     <ref bean="valueListHandler" />
            </property>
   </bean>
...
```

- Podem veure que en aquest bean “simplificador” és fa referència al bean amb identificador *valueListHandler*. Aquest bean correspon a un bean “delegat” per a localitzar els cerques.

  Exemple:

```
<bean name="valueListHandler"
   class="net.mlw.vlh.DefaultValueListHandlerImpl" lazy-init="true">
   <property name="config.adapters">
            <map>
            <!-- Template Definition for each navigable list in application -->
               <entry key="peticioList">
                     <bean parent="baseHibernateAdapter">
                       <property name="hql">
                              <value>
                                  FROM
                                  es.bcn.gca.necn.common.model.PeticionsVo AS vo
                                  WHERE 1=1
                                  /~petId: AND lower(vo.petId) LIKE lower('[petId]%') ~/
                              </value>
                        </property>
                     </bean>
                </entry>
                          ...
            </map>
      </property>
   </bean>
```

- A més a més, cal configurar els beans “adaptadors”, que són els encarregats d’executar les cerques. En l’exemple anterior, podem veure que l’entrada *accountList* fa referència a un bean adaptador amb identificador *baseHibernateAdapter* (aquest és l’adaptador a fer servir quan les cerques són contra base de dades, encara que podem tenir els nostres propis adaptadors). Els atributs d’aquest bean *baseHibernateAdapter* són:

  Exemple:

```
...
   <bean name="baseHibernateAdapter"
            class="net.opentrends.openframe.services.web.vlh.Hibernate30Adapter"
               abstract="true">
            <property name="sessionFactory" ref="sessionFactory"/>
            <property name="defaultNumberPerPage" value="5"/>
            <property name="defaultSortColumn" value="id"/>
            <property name="defaultSortDirection" value="asc"/>
            <property name="removeEmptyStrings" value="true"/>
          <property name="statementBuilder" ref="statementBuilder" />
   </bean>
...
```

- Veiem que aquest bean fa referència a un bean amb identificador *statementBuilder*. Aquest bean permet controlar el procés de generació de la sentència HQL i com són formatejats els atributs dintre de la consulta.

  Exemple:

```
<bean id="statementBuilder" class="net.opentrends.openframe.services.web.vlh.util.StatementBuilder"
lazy-init="true">
   <property name="setters">
     <map>
      <entry key="lower18">
            <bean class="net.opentrends.openframe.services.web.vlh.setter.BooleanSetter">
              <property name="logService" ref="loggingService"/>
              <property name="booleanEditor" ref="booleanEditor"/>
            </bean>
      </entry>
      </map>
    </property>
   </bean>
```

- Per últim, la definició del servei requereix configurar un bean "configurador" per a la presentació de resultats.

  Exemple:

```
...
   <bean name="vlConfig" class="net.mlw.vlh.web.ValueListConfigBean"
           singleton="false">
      <property name="messageSource" ref="messageSource"/>
      <property name="displayProviders">
         <map>
            <entry key="ExportPDF">
               <bean class="net.opentrends.openframe.services.web.vlh.tag.
               support.PdfDisplayProvider" singleton="false">
                     <property name="logService" ref="loggingService"/>
                     <property name="i18nService" ref="i18nService"/>
                     <property name="skipColumns"> ❶
                       <list>
                               <value> </value>
                       </list>
                     </property>
               </bean>
             </entry>
            </map>
       </property>
   </bean>
...
```

- Enllaç d'una acció amb el seu "Value List"

Aquesta relació és defineix al fitxer action-servlet-xxx.xml de configuració del bean de l'acció. Cada acció que implementi una cerca necessita la definició d'un bean que hereti del bean "pare" del servei.

Exemple:

```
...
<property name="valueListActionHelper">
   <bean parent="valueListActionHelper">
            <property name="listName">
                     <value>peticionsList</value>
            </property>
            <property name="tableId" value="PETICIONS"/>
   </bean>
</property>
...
```

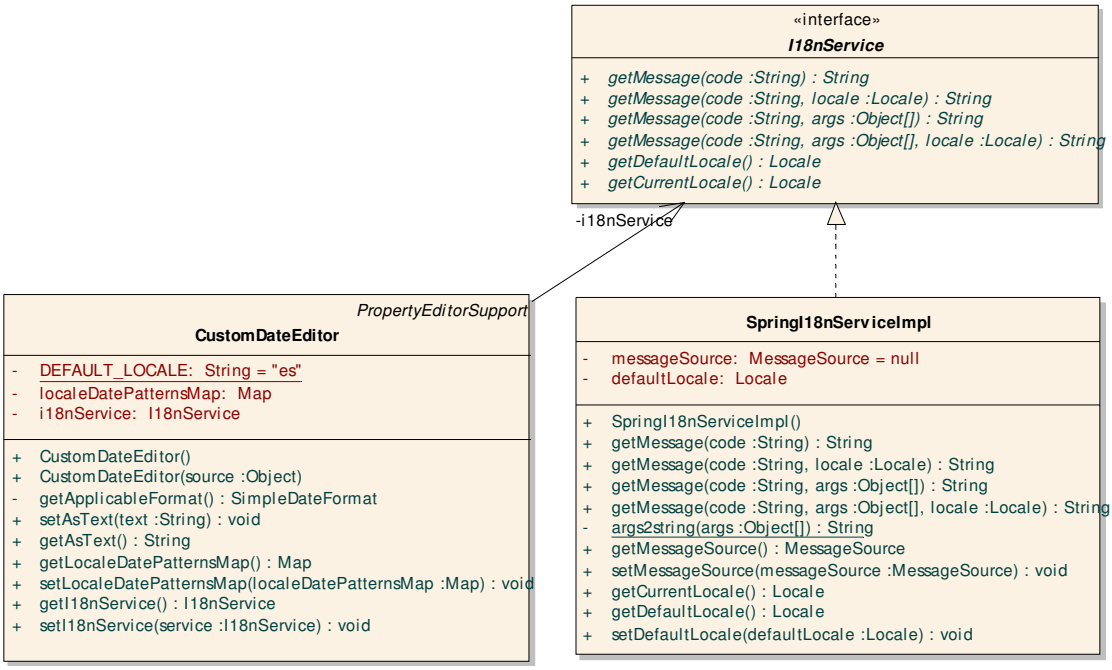
➢ **Servei Multiidioma (o de Internacionalització)**

El Servei de Multiidioma té com objectiu el desenvolupament d'aplicacions que presentin els seus texts i recursos en diferents llenguatges i sense que sigui necessària cap modificació del codi cada vegada que s'incorpori un nou llenguatge a la aplicació.

Els texts es emmagatzemen en fitxers internacionalitzats.

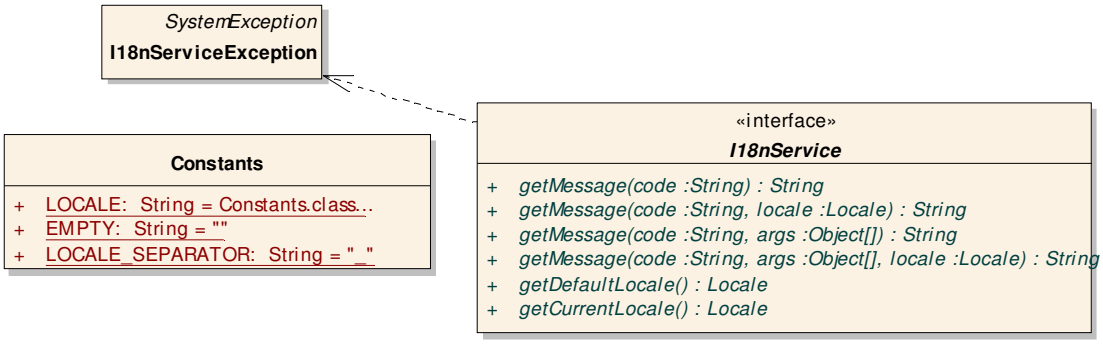
Interfícies i Components

Interfície genèrica i Components Implementació Spring



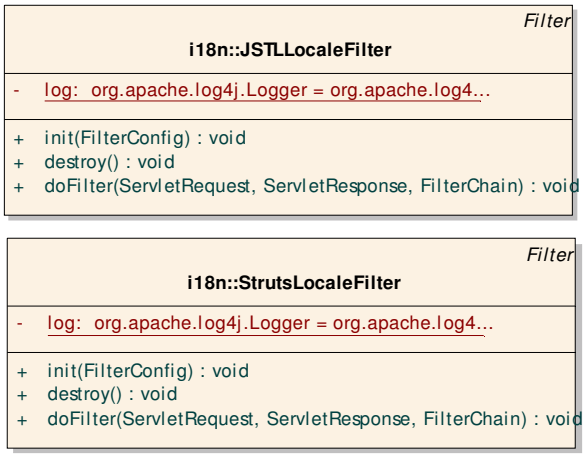
- **I18nService:** Interfície del servei.
- **SpringI18nServiceImpl:** Implementació del servei basada en Spring.
- **CustomDataEditor:** Editor específic de dades.

Components per la gestió d'excepcions i elements genèrics



- **I18nServiceException:** Excepció llençada pel servei.
- **Constants:** Classe de Constants del servei.

Components d'integració amb la capa de presentació.



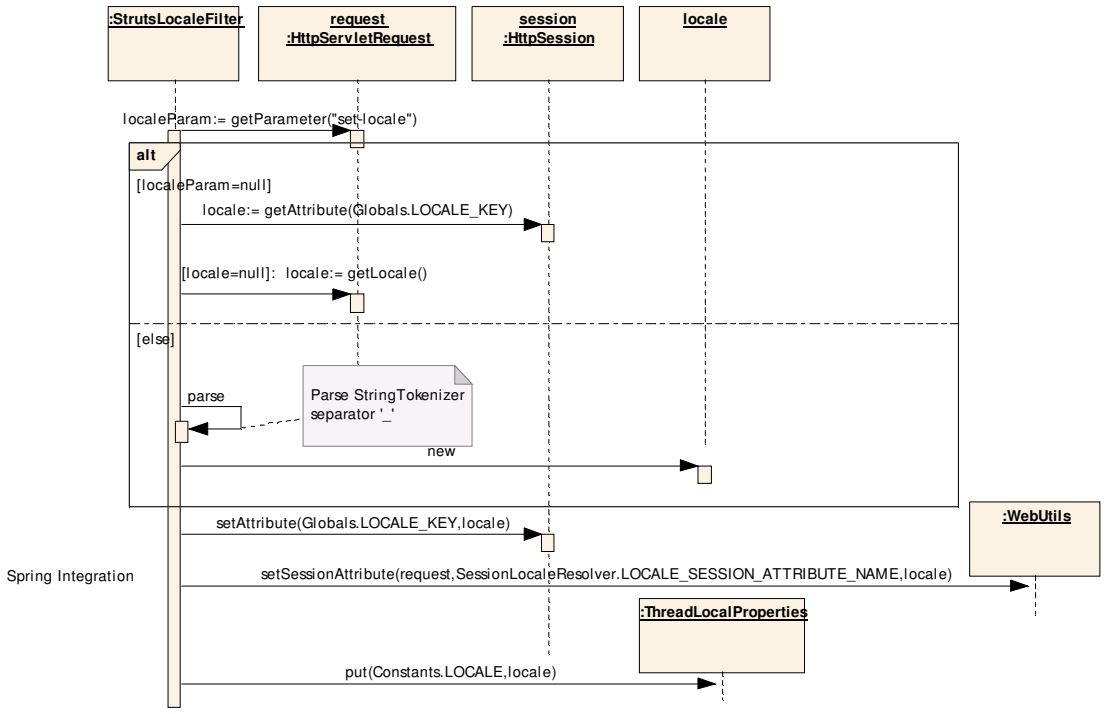
- **JSTLocaleFilter:** Filtre WEB d'integració amb JSTL. Principalment s'encarrega de fer disponible als tags de JSTL el Locale de l'usuari, mitjançant un atribut de la sessió.
- **StrutsLocaleFilter:** Filtre WEB d'integració amb Struts. Principalment s'encarrega de fer disponible als components de Struts el Locale de l'usuari, mitjançant un atribut de la sessió.

Arquitectura dels filtres

La internacionalització té en consideració la configuració del idioma de l'usuari al seu navegador. D'aquesta forma, els literals mostrats s'obtindran del fitxer de traduccions associat a aquest idioma.

En qualsevol cas, és possible especificar un paràmetre 'set-locale' amb el llenguatge i el país com a valor separats per '_'. En el següent diagrama podem veure quin és el procediment de tractament de tota petició:

1. Si no es rep el paràmetre 'set-locale', es considera si existeix la variable de Struts que especifica el idioma de la sessió de l'usuari . Si aquest no està especificat, el Locale s'obté de la petició HTTP. Aquest idioma serà el que és troba configurat en primer lloc a la llista d'idiomes del navegador (dins 'Preferències').
2. Si es rep el paràmetre 'set-locale', el seu valor és parsejat en 2 parts i es crea l'objecte 'Locale' corresponent.
3. Finalment, amb el locale obtingut es realitzen 3 passos:
    1) Introducció del locale escollit a la sessió de Struts
    2) Introducció del locale a la sessió de Spring
    3) Introducció del locale a l'objecte 'ThreadLocalProperties' d'openFrameIMI. Aquest objecte permet passar informació entre els objectes dins el fil d'execució actual, i serà usat pel mètode 'getCurrentLocale' de la implementació del servei d'internacionalització.



- **Servei de Validació**

En la gestió de les peticions és necessària, pel general, una validació prèvia de les dades entrades per l'usuari. Aquesta validació inclou, entre d'altres:

- Comprovar que s'han introduït dades en un camp definit com obligatori.
- Comprovar que la dada introduïda es pot transformar a un tipus definit (sencer, flotant, data…).
- Comprovar que la dada compleix amb una expressió regular (correu electrònic...) o amb una formula (nombre de NIF...).

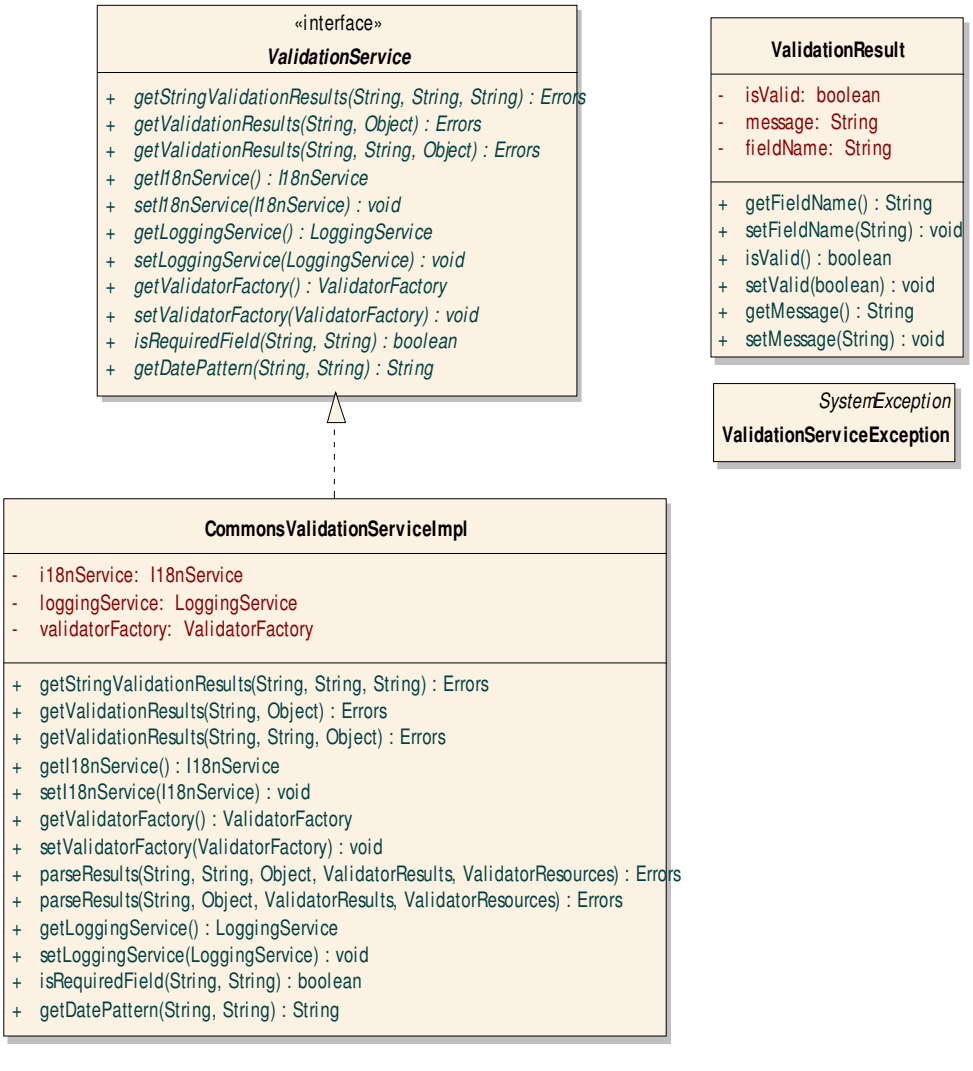
La validació realitzada en la presentació té com objectiu filtrar les dades entrades abans que arribin a la lògica de negoci.

El Servei de Validació permet definir quines validacions cal realitzar en cada camp mitjançant l'ús de fitxers externs XML que el Servei de Presentació amb Tags utilitza per a generar el codi Javascript necessari per a validar les dades en el navegador de l'usuari, si es decideix que es vol fer la validació al client en comptes de fer-la al servidor.

El Servei de Validació internament inclou una implementació que pot ser invocada des de capa de negoci amb possibilitat de validar qualsevol tipus. A més, inclou la extensió especifica utilitzada a la capa de presentació, on els paràmetres, independentment de que representin, son de tipus text.

Independència del Sistema de Validació Utilitzat

OpenFrameIMI ofereix la possibilitat d'utilitzar diferents implementacions de validació mitjançant la definició d'interfícies. En l'actualitat s'ofereix una implementació basada en l'ús de Commons Validator, que és un estàndard de facto com a API de validació.


«interface»
ValidationService
+ getStringValidationResults(String, String, String) : Errors
+ getValidationResults(String, Object) : Errors
+ getValidationResults(String, String, Object) : Errors
+ getI18nService() : I18nService
+ setI18nService(I18nService) : void
+ getLoggingService() : LoggingService
+ setLoggingService(LoggingService) : void
+ getValidatorFactory() : ValidatorFactory
+ setValidatorFactory(ValidatorFactory) : void
+ isRequiredField(String, String) : boolean
+ getDatePattern(String, String) : String
ValidationResult
- isValid: boolean
- message: String
- fieldName: String
+ getFieldName() : String
+ setFieldName(String) : void
+ isValid() : boolean
+ setValid(boolean) : void
+ getMessage() : String
+ setMessage(String) : void
SystemException
ValidationServiceException
CommonsValidationServiceImpl
- i18nService: I18nService
- loggingService: LoggingService
- validatorFactory: ValidatorFactory
+ getStringValidationResults(String, String, String) : Errors
+ getValidationResults(String, Object) : Errors
+ getValidationResults(String, String, Object) : Errors
+ getI18nService() : I18nService
+ setI18nService(I18nService) : void
+ getValidatorFactory() : ValidatorFactory
+ setValidatorFactory(ValidatorFactory) : void
+ parseResults(String, String, Object, ValidatorResults, ValidatorResources) : Errors
+ parseResults(String, Object, ValidatorResults, ValidatorResources) : Errors
+ getLoggingService() : LoggingService
+ setLoggingService(LoggingService) : void
+ isRequiredField(String, String) : boolean
+ getDatePattern(String, String) : String

Configuració

Per a configurar el Servei de Validació, l'únic que s'ha de definir dintre de les aplicacions és un bean amb identificador ***validatorFactory*** i de tipus ***org.springmodules.validation.commons.DefaultValidatorFactory*** en el qual es defineixi la llista de fitxers de configuració a fer servir.

També han d'incloure el import del fitxer de configuració ***openFrame-services-validation-default.xml***

```
<import resource="classpath:/spring/openFrame-services-validation-default.xml" />

<bean name="validatorFactory" class="org.springmodules.commons.validator.DefaultValidatorFactory">
            <property name="validationConfigLocations">
              <list>
                    <value>/WEB-INF/classes/validation/validator-rules.xml</value> ❶
                    <value>/WEB-INF/classes/validation/validation.xml</value> ❷
              </list>
            </property>
   </bean>
```

❶ Fitxer de definició de les regles de validació disponibles.

❷ Fitxer de definició de les validacions a aplicar als components dels formularis de l'aplicació

Fitxer de Regles de Validació

En el fitxer **'validation-rules.xml**' es defineixen els tipus de validacions que podem realitzar des d'openFrameIMI. Aquestes regles de validació es basen en Spring i tenen la següent estructura:

```
<validator name="required"
         classname="org.springmodules.commons.validator.FieldChecks"
         method="validateRequired"
         methodParams="java.lang.Object,
                       org.apache.commons.validator.ValidatorAction,
                       org.apache.commons.validator.Field,
                       org.springframework.validation.Errors"
                 msg="errors.required">

         <javascript><![CDATA[
            function validateRequired(form) {
                var isValid = true;
                var focusField = null;
```

Cada validador defineix quin és la classe que realitza la validació (en el cas mostrat 'FieldChecks') .

Els validadors actualment disponibles són:

Validador	Descripció	Disponibilitat
required	Validació de camps requerits	CLIENT, SERVER
requiredif validwhen	Validació de camp requerit si es dóna una altra condició	SERVER
minlength	Validació de longitud mínima de caràcters	CLIENT, SERVER
maxlength	Validació de longitud màxima de caràcters	CLIENT, SERVER
mask	Validació de format segons expressió regular	CLIENT, SERVER
byte	Validació que el camp pot ser convertit a Byte (entre -2^7 i 2^7-1)	CLIENT, SERVER
short	Validació que el camp pot ser convertit a Short (entre -2^{15} i $2^{15}-1$)	CLIENT, SERVER
integer	Validació que el camp pot ser convertit a Integer (entre -2^{31} i $2^{31}-1$)	CLIENT, SERVER
long	Validació que el camp pot ser convertit a Long (entre -2^{63} i $2^{63}-1$)	SERVER
float	Validació que el camp pot ser convertit a Float (entre 2^{-149} i $(2-2^{-23})\cdot 2^{127}$)	CLIENT, SERVER
double	Validació que el camp pot ser convertit a Double (entre 2^{-1074} i $(2-2^{-52})\cdot 2^{1023}$)	SERVER
date	Validació que el camp és una data correcta	CLIENT, SERVER
intRange	Validació que un camp Integer es troba dintre un rang de valors	CLIENT, SERVER
floatRange	Validació que un camp Float es troba dintre un rang de valors	CLIENT, SERVER
longRange	Validació que un camp Long es troba dintre un rang de valors	SERVER
email	Validació que un camp té format de correu electrónic	CLIENT, SERVER
nif	Valida que un camp tingui format de NIF o CIF correcte	CLIENT, SERVER
ccc	Valida que un camp corresponent a una Compte Corrent, tingui el Dígit Control correcte.	CLIENT, SERVER

Nota: Si es fa ús de validació de formularis al costat client, cal tenir en compte que només actuaran aquelles validacions del llistat anterior amb Disponibilitat CLIENT.

Fitxer de Validacions

En aquest fitxer és en el qual definirem quines validacions volem realitzar per als nostres objectes o formularis. Definirem 2 seccions diferenciades:

- **Constants:** Secció en la qual es defineixen expressions regulars comunes que seran utilitzades des de diverses definicions de validació.

```
<global>
   <constant>
            <constant-name>any</constant-name>
           <constant-value>^([0-9][0-9][0-9][0-9])$</constant-value>
            </constant>
</global>
```

- **Definició dels formularis a validar:** Tot i que hem explicat que la validació serà a nivell d'objectes i/o formularis, Commons Validator utilitza el concepte *form* per a referir-se a ells. Per a cada camp especificarem quines validacions realitzar.

```
<formset>
       <form name="pet">
            <field property="petId" depends="required">
                     <arg0 key="forms.peticioForm.field.petId"/>
            </field>
            <field property="productes.prodId" depends="required">
                     <arg0 key="forms.peticioForm.field.productes"/>
            </field>
            <field property="petCreador" depends="required">
                     <arg0 key="forms.peticioForm.field.petCreador"/>
            </field>
       </form>
</formset>
```

L'esquema d'aquest fitxer és el mostrat dalt. Cada objecte o formulari a validar serà definit en una secció 'form' i cada camp del formulari en una subsecció 'field'

En l'atribut 'name' de la secció 'form' definim el nom del validador. Aquest nom pot ser qualsevol literal, i és el que es farà servir a posteriori per a enllaçar un objecte o un formulari amb les validacions que haurà de verificar.

Per a cada camp especificarem:

- property. Indicar com valor el nom de l'atribut de la classe que vulguem validar.

- depends. Llista de validacions a realitzar separades per comes. El nom de les validacions correspon a noms de validacions definides en el fitxer de regles de validació (***validation-rules.xml***).

Integració amb altres Serveis

El servei de validació és un servei transversal del framework, i, com a tal, es pot fer servir des de qualsevol component de l'aplicació. Simplement s'ha d'afegir la dependència del component del servei a la classe des de la qual volem invocar a la validació de dades, i gestionar la resposta en el cas d'errors en la validació.

Ara bé, openFrameIMI integra el servei de validació dintre dels components de la capa de presentació de manera no intrusiva, com podrem veure a continuació.

Integració amb el Servei d'Internacionalització

En els fitxers de configuració es defineixen claus que permeten especificar quins missatges retornar en cas de validació errònia. Per a poder traduir aquestes claus és necessari especificar al Servei de Validació que usarà el Servei d'Internacionalització.

En determinats casos podem requerir de diferents tipus de validació segons el idioma seleccionat (dates, formats de moneda, etc.). En aquest cas, crearem diversos formsets.

Integració amb el Servei de Presentació

En el cas de voler validar les dades introduïdes per l'usuari per pantalla, el framework ofereix tres mecanismes de validació:

- Validació al propi client amb javascript (CLIENT)
- Validació al servidor amb AJAX (SERVER)
- Validació al servidor.

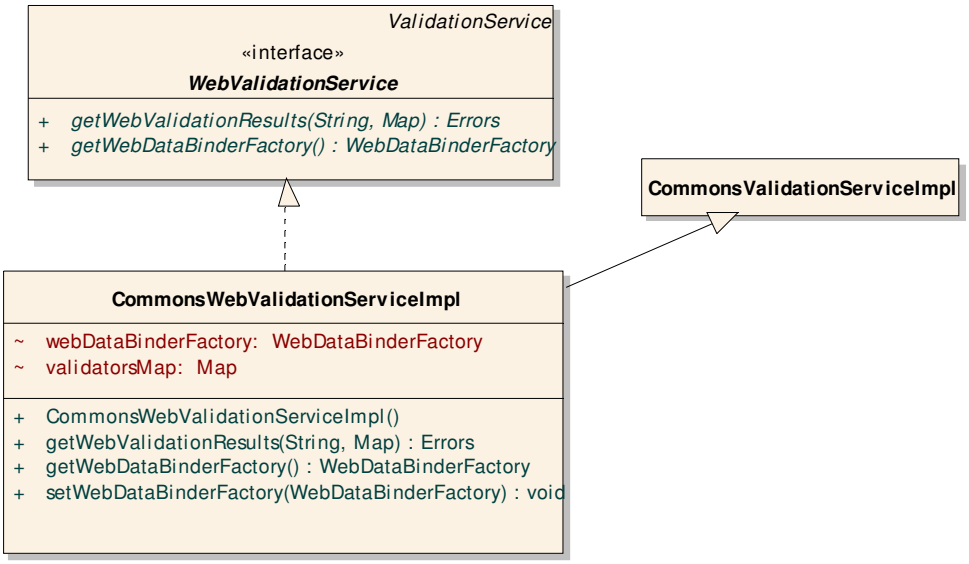
La diferència entre els 2 primers mecanismes és que en el cas de Client la validació es fa des del navegador del Client (mitjançant javascript) i en el cas de Servidor s'utilitza Ajax per a comunicar des del navegador amb el servidor i presentar els missatges sense haver de recarregar de nou la pàgina.

En cas que la validació es faci per mitjà de Javascript incrustat a la pàgina, no cal realitzar cap procés addicional. En el moment de renderitzar la pàgina JSP, **openFrameIMI** s'encarrega d'afegir el codi de validació Javascript basat en el validador configurat.

Si utilitzem la validació des del Servidor amb AJAX, **openFrameIMI** s'encarrega d'afegir a la pàgina JSP el codi necessari per a fer la invocació via AJAX al component Web del servidor encarregat de realitzar la validació, així com la posterior gestió dels possibles errors.

Per últim, en el cas de voler implementar la validació en servidor (sense crida AJAX), l'aplicació tampoc haurà d'afegir cap codi addicional a part del corresponent a la pròpia configuració.

En el cas de validació en servidor (amb o sense AJAX), el component que s'encarrega de realitzar la validació de las dades que venen de la pantalla és el que es mostra al següent diagrama:



A continuació es detalla la configuració necessària en cadascuna de les possibilitats. Com podrem veure, aquesta configuració es molt semblant i sempre s'especifica sobre el bean associat al formulari de la pantalla:

- ***Validació en client:***

```
<bean parent="formTag">
   <property name="styleId" value="actionForm"/>
   <property name="layout" value="true"/>
   <property name="validationProperties">
     <props>
       <prop key="validationType">CLIENT</prop>  ❶
       <prop key="validatorName">pet</prop>  ❷
     </props>
   </property>       /bean>
```

On

❶Identifica que es vol fer la validació de les dades d'aquest formulari en client.

❷Nom del validador que es vol aplicar a les dades del formulari. Aquesta propietat és opcional, i en cas de no informar-se, es pren com a nom del validador el nom de la propietat *pojoClass* del bean Action.

- ***Validació en servidor, amb crida AJAX:***

```
<bean parent="formTag">
   <property name="styleId" value=" actionForm"/>
   <property name="layout" value="true"/>
   <property name="validationProperties">
     <props>
       <prop key="validationType">SERVER</prop> ❶
       <prop key="validatorName">pet</prop> ❷
     </props>
   </property>
</bean>
```

On el significat de les propietats és el mateix excepte que en aquest cas el valor de la propietat *validationType* ha de ser SERVER.

- ***Validació en servidor:***

```
<bean parent="formTag">
   <property name="styleId" value=" actionForm "/>
   <property name="layout" value="true"/>
   <property name="validationProperties">
     <props>
           <prop key="validatorName">pet</prop> ❶
           <prop key="reqCodesToValidate">save*:altaPeticions,update</prop> ❷
     </props>
   </property>       </bean>
```

El significat dels paràmetres en aquest cas és el següent:

❶ Nom del validador a fer servir per a validar les dades.

❷ Llistat que permet especificar per a cadascuna de les accions del formulari (*reqCode*), si aquesta s'ha de validar i quina validació es vol aplicar.

El format d'aquest llistat és una seqüència d'elements del tipus

reqCodePattern:validador

separats pel caràcter ','. En la definició de cada element solament és obligatori indicar el patró

del reqCode, ja que sinó s'especifica el validador es pren el corresponent a la propietat *validatorName*.

Per exemple, en aquest exemple s'està indicant que només per les accions amb reqCode save* o update s'han de validar les dades que s'envien al servidor**.** A més a més, en el cas de les accions de tipus *save**, s'han d'aplicar les regles de validació definides al validador amb nom *altaPeticions*; i en el cas de l'acció *update*, s'han d'aplicar les regles definides al validador *pet*.

En el cas de la validació al servidor, a part de la configuració vista anteriorment sobre el bean formTag, existeix l'opció de definir la configuració de la validació a nivell de classe Action:

```
<bean name="/accounts" parent="accountBaseDefinition" lazy-init="true">
<property name="pojoClass"
            value="es.bcn.gca.necn.web.pojo.PeticionsTo" />
   <property name="validationProperties">
     <props>

            <prop key="validatorName">pet</prop>
            <prop key="reqCodesToValidate">save*:altaPeticions,update</prop>
     </props>
   </property>
  ….
</bean>
```

Com podem veure, la configuració és exactament igual que en el cas anterior.

5.6.3.2. Implementació per la nostra aplicació

Un cop s'ha definit com estarà formada la nostra pantalla final, s'especifica amb més detall la composició de pantalles (Tiles) i la creació de vistes (JSPs) per a la nostra aplicació.

Tal i com es van definir anteriorment els casos d'ús de la nostra aplicació, aquesta s'utilitzarà principalment, per a gestionar Peticions, Productes, Paquets i Grups. Per a cadascun dels casos d'ús es crearan dues pantalles diferents, una per mostrar un formulari de cerca i un llistat i l'altre per poder crear, consultar o modificar. A continuació es detallarà com s'han creat aquestes pantalles amb les seves corresponents vistes.

- **Composició de pantalles amb Tiles**

Per a definir la composició de pantalles es fa servir Tiles, que permet generar pantalles definint distintes porcions, cadascuna generada per un JSPs, fins a obtenir una pantalla completa.

Tiles es configura al fitxer "**tiles-definitions.xml**", ja existent al directori /**resources/tiles** del projecte ***gcanecnWeb*** amb lo següent:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
       "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>

   <definition name="pages.logoff" path="/WEB-INF/jsp/logoffPg.jsp">
   </definition>

   <definition name="pages.error" path="/WEB-INF/jsp/error403Pg.jsp">
            <put name="header" value="/WEB-INF/jsp/includes/headerPg.jsp" />
            <put name="footer" value="/WEB-INF/jsp/includes/footerPg.jsp" />
   </definition>

   <definition name="site.mainLayout" path="/WEB-INF/jsp/layouts/mainLayoutPg.jsp">
            <put name="header" value="/WEB-INF/jsp/includes/headerPg.jsp" />
            <put name="menu" value="/WEB-INF/jsp/includes/menuPg.jsp" />
            <put name="useCaseTitle" value="/WEB-INF/jsp/includes/useCaseTitlePg.jsp" />
            <put name="errors" value="/WEB-INF/jsp/includes/errorsPg.jsp" />
            <put name="quickSearch" value="/WEB-INF/jsp/includes/blankPg.jsp" />
            <put name="body" value="/WEB-INF/jsp/includes/blankPg.jsp" />
            <put name="footer" value="/WEB-INF/jsp/includes/footerPg.jsp" />
            <put name="useCaseNameKey" value="inherit.me" type="string" />
   </definition>

   <definition name="pages.home" extends="site.mainLayout">
            <put name="useCaseNameKey" value="page.home.inici" type="string" />
   </definition>

   <definition name="pages.i18n" extends="site.mainLayout">
            <put name="body" value="/WEB-INF/jsp/i18n/i18nPg.jsp" />
            <put name="useCaseNameKey" value="jsp.i18n.title" type="string" />
   </definition>

   <!-- Afegeix les teves pantalles a continuacio -->
   <!-- cas ús cerca peticions -->
   <definition name="pages.peticions" extends="site.mainLayout">
   <put name="quickSearch" value="/WEB-jsp/peticions/peticioListQuickSearchPg.jsp"
   <put name="body" value="/WEB-INF/jsp/peticions/peticioListPg.jsp" />
   <put name="useCaseNameKey" value="jsp.pet.title" type="string" />
   </definition>

   <!-- cas ús edita peticions -->
   <definition name="pages.peticioEdit" extends="site.mainLayout">
   <put name="body" value="/WEB-INF/jsp/peticions/peticioPg.jsp" />
   <put name="useCaseNameKey" value="jsp.pet.title" type="string" />
   </definition>
```

```
</tiles-definitions>
```

Aquesta configuració ja defineix algunes pantalles com la home (pages.home) i la de canvi d'idioma (pages.i18n), que tenen l'aspecte següent:

Tots dos hereten del template "site.mainLayout", que defineix una pantalla amb capçalera, peu, menú a la esquerra, etc. Tal i com es pot veure al fitxer de configuració i a les imatges, tant pages.home com pages.i18n estenen de manera diferent el títol i la part central de la pantalla.

Tal i com s'ha comentat anteriorment, el manteniment de les entitats PETICIONS, PRODUCTES, PAQUETS i GRUPS requereixen dues pantalles, la primera amb un formulari de cerca i un llistat, i la segona amb un formulari per a creació/modificació/consulta d'una entitat.

La configuració de Tiles per a la primera pantalla, la del formulari de cerca amb llistat de PETICIONS, seria:

```
    ...
    </definition>

    <!-- Afegeix les teves pantalles a continuacio -->

    <definition name="pages.productes" extends="site.mainLayout">
        <put name="quickSearch" value="/WEB-INF/jsp/productes/producteListQuickSearchPg.jsp" />
        <put name="body" value="/WEB-INF/jsp/productes/producteListPg.jsp" />
        <put name="useCaseNameKey" value="jsp.pro.title" type="string" />
    </definition>
</tiles-definitions>
```

Estenem la pàgina pare per a poder reaprofitar gran part dels JSPs necessaris per a composar les nostres pàgines. A continuació, el que es fa es reescriure aquelles parts de la pàgina que son especifiques, en concret la "quickSearch", el "body" i "useCaseNameKey", ja que ens interessa afegir un formulari de cerca ràpida, un llistat a la

part central i canviar el títol de la pàgina que veurà l'usuari. Els dos primers elements reescrits apunten a dos JSPs que descriurem a continuació; l'altre és un literal que haurem d'afegir als nostres fitxers d'internacionalització.

La configuració de Tiles per a la segona pantalla, per a PETICIONS, seria:

```
    ...
    </definition>

    <!-- Afegeix les teves pantalles a continuacio -->

    <definition name="pages.producteEdit" extends="site.mainLayout">
            <put name="body" value="/WEB-INF/jsp/productes/productePg.jsp" />
            <put name="useCaseNameKey" value="jsp.pro.title" type="string" />
    </definition></tiles-definitions>
```

Un cop configurat Tiles, s'han de crear els JSP corresponents per a completar les vistes.

- **Creació de vistes (JSP)**

JSP de cerca de PETICIONS

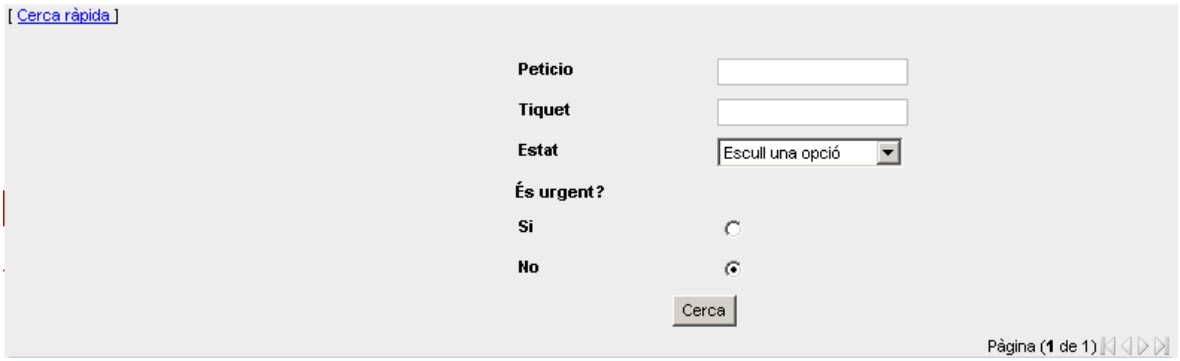
El JSP definit com a quickSearch, "**peticioListQuickSearchPg.jsp**", que s'ha de crear al lloc on s'indica al fitxer de configuració de Tiles al projecte ***gcanecnWeb*** al directori /**WebContent/WEB-INF/jsp/ peticions** amb el següent codi:

```
...

<table >
   <tr>
    <td >
      <fwk:form action="peticions.do" styleId="actionForm" reqCode="search" width="100%"
            method="post" styleClass="edit">
       <fwk:text styleId="petId" property="petId" key="forms.peticioForm.field.petId"
            styleClass="selectOnFocus campofwk" />
       <fwk:text styleId="petTiquet" property="petTiquet" key="forms.peticioForm.field.petTiquet"
            styleClass="selectOnFocus campofwk" />
       <fwk:select otherKey="blank" styleId="petEstat" property="petEstat"
            key="forms.peticioForm.field.petEstat" styleClass="selectOnFocus campofwk" />
```

En aquest JSP es poden veure tot un conjunt de literals que seran informats en temps d'execució gràcies al servei multiidioma del framework si introduïm les propietats corresponents (per exemple, forms.peticioForm.field.petId) als fitxers de multiidioma.

També es pot veure que per a definir el formulari s'utilitzen tags d'openFrame (fwk:form, fwk:text, fwk:submit...).

El resultat d'aquest JSP és el següent:



JSP de llistat de PETICIONS

El JSP encarregat de generar el llistat de la part central de la pantalla és un JSP anomenat "**peticioListPg.jsp**" que hem de crear al directori apuntat en la definició de Tiles **/WebContent/WEB-INF/jsp/peticions** del projecte ***gcanecnWeb*** amb el codi següent:

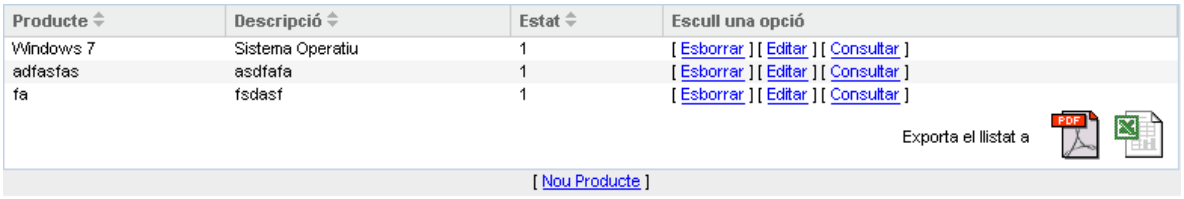
```
...
<fwk:vlhrow bean="llista" display="<%=displayProvider%>">
                                                        <fwk:vlhattribute
name="onmouseover">this.className='selected';</fwk:vlhattribute>
                                                        <fwk:vlhattribute
name="onmouseout">this.className='zebra0';</fwk:vlhattribute>

                                                        <fwk:vlhcolumn
titleKey="forms.peticioForm.field.productes" property="productes.prodId" sortable="asc">


                                                        </fwk:vlhcolumn>
                                                        <fwk:vlhcolumn
titleKey="forms.peticioForm.field.petDataCreacio" property="petDataCreacio" sortable="asc">


                                                        </fwk:vlhcolumn>
                                                        <fwk:vlhcolumn
titleKey="forms.peticioForm.field.petTiquet" property="petTiquet" sortable="asc">


                                                        </fwk:vlhcolumn>
                                                        <fwk:vlhcolumn
titleKey="forms.peticioForm.field.petEstat" property="petEstat" sortable="asc">


                                                        </fwk:vlhcolumn>
                                                        <fwk:vlhcontrols
titleKey="blank">
                                                                        [
                                                                <fwk:vlhaction
url="editPeticio.do?">
```

```
   <fwk:vlhaddParam name="reqCode" value="delete" temp="false" /> <bean:message
key="jsp.peticions.peticioList.esborrar"/>
                                                                  <fwk:vlhaddParam name="petId"
property="petId" temp="false" />
                                                                          </fwk:vlhaction>
                                                                                  ]
                                                                                  [
                                                                          <fwk:vlhaction
url="editPeticio.do?">

   <fwk:vlhaddParam name="reqCode" value="edit" temp="false" /> <bean:message
key="jsp.peticions.peticioList.editar"/>
                                                                  <fwk:vlhaddParam name="petId"
property="petId" temp="false" />
                                                                          </fwk:vlhaction>
                                                                                  ]
                                                                                  [
                                                                          <fwk:vlhaction
url="editPeticio.do?">

   <fwk:vlhaddParam name="reqCode" value="inspect" temp="false" /> <bean:message
key="jsp.peticions.peticioList.consultar"/>
                                                                  <fwk:vlhaddParam name="petId"
property="petId" temp="false" />
                                                                          </fwk:vlhaction>
                                                                  ]

                                                                  </fwk:vlhcontrols>
                                                          </fwk:vlhrow>

   ....
```

Es pot comprovar que per a definir el llistat s'utilitzen tags d'openFrame (fwk:vlhroot, fwk:vlhpaging, fwk:vlhrow...).

També es pot ressenyar que, jugant amb la propietat display de fwk:vlhroot, el mateix llistat serveix per a mostrar el HTML que veurà l'usuari com per a exportar les dades a PDF o MS Excel.

El resultat és el següent llistat:

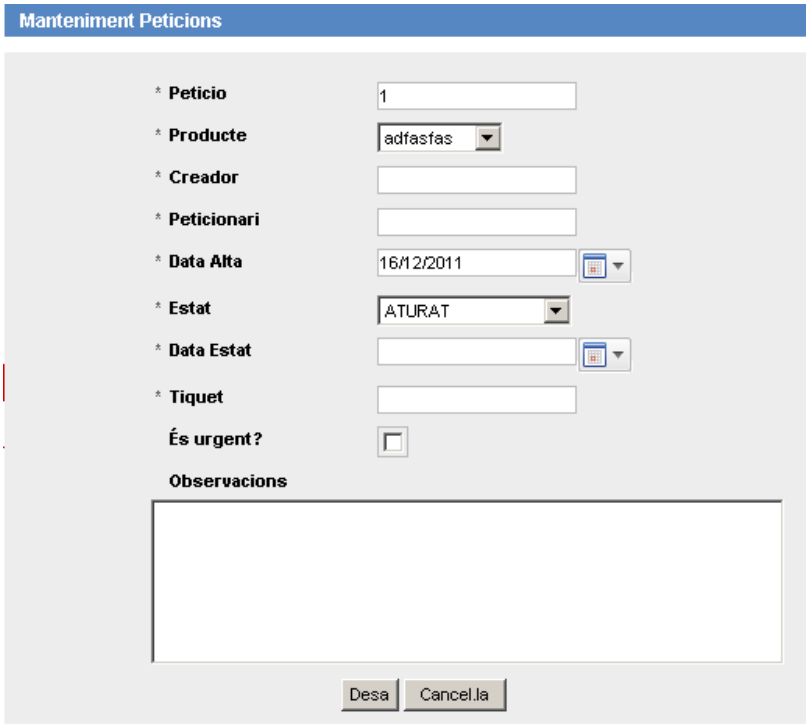


JSP de creació/edició/consulta d'una PETICIÓ

És el JSP per a crear, editar i consultar una petició. És un JSP anomenat "peticioPg.jsp" que s'ha de crear al directori apuntat en la definició de Tiles **/WebContent/WEB-INF/jsp/peticions** del projecte ***gcanecnWeb*** amb el codi següent:

```
...

<fwk:form action="editPeticio.do" styleId="actionForm" reqCode="edit" width="100%" method="post"
styleClass="edit">
                                <table>
                                        <fwk:text styleId="petId" property="petId"
styleClass="selectOnFocus campofwk"/>
                                        <fwk:select styleId="productes.prodId"
property="productes.prodId" styleClass="selectOnFocus campofwk"/>
                                        <fwk:text styleId="petCreador" property="petCreador"
styleClass="selectOnFocus campofwk"/>
                                        <fwk:text styleId="petPeticionari"
property="petPeticionari" styleClass="selectOnFocus campofwk"/>
                                        <fwk:text styleId="petDataCreacio"
property="petDataCreacio" styleClass="selectOnFocus campofwk"/>
                                        <fwk:select styleId="petEstat" property="petEstat"
styleClass="selectOnFocus campofwk"/>
                                        <fwk:text styleId="petDataEstat" property="petDataEstat"
styleClass="selectOnFocus campofwk"/>
                                        <fwk:text styleId="petTiquet" property="petTiquet"
styleClass="selectOnFocus campofwk"/>
                                        <fwk:checkbox styleId="lower18" property="lower18"
styleClass="selectOnFocus campofwk"/>
                                        <fwk:textarea styleId="petObservacions"
property="petObservacions" maxlength="256" styleClass="selectOnFocus campofwk"/>
                                        </table>
                                        <table>
                                    <fwk:row>
...
```

El resultat és el següent formulari:



- **Definició i configuració dels objectes DAO i Actions**

Objectes de transferència de dades cap a les vistes

Per a poder enviar i rebre informació a les vistes (JSPs) s'utilitzen POJOs serialitzables. En el cas que estem veient, manteniment de la entitat PETICIONS, podríem fer servir directament els VOs de la capa de persistència, però es bona pràctica utilitzar POJOs específics, que anomenarem DTOs (Data Transfer Object).

En el nostre cas, caldrà crear un DTO, per al cas de les peticions, anomenat PeticionsTO, al projecte ***gcanecnWeb*** al package **es.bcn.gca.necn.web.pojo**.

Creació de classes Action

Amb openFrame, a la capa de presentació es treballa amb el patró MVC (Model-Vista-Controlador). Fins al moment ja tenim definit el model representat per els BOs i les vistes. Ens falta definir el controlador, per al flux de navegació, que ho farem més tard, però abans haurem de crear les classes, que hereten de **DispatchActionSuport**, on es crearan els mètodes corresponents a les accions que el usuari podrà realitzar a les pantalles.

Els Action que crearem, en aquest cas per al cas d'ús de les Peticions, s'anomerà PeticionsAC.java i el crearem al projecte ***gcanecnWeb*** al package **es.bcn.gca.necn.web.struts.action.**

```
...

public ActionForward searchIni(ActionMapping mapping, ActionForm form,
            javax.servlet.http.HttpServletRequest request,
            javax.servlet.http.HttpServletResponse response)
            throws Exception {
             logService.getLog(this.getClass()).debug("searchIni");
        request.getSession().setAttribute(valueListActionHelper.getTableId(), null);

        return this.search(mapping,form,request,response);
    }

    /**
     * Búsqueda, realiza una consulta
     * @param mapping
     * @param form
     * @param request
     * @param response
     * @return
     * @throws Exception
     * @see @see
es.bcn.fwk.prototipus.web.struts.action.actionaccount2#search(org.apache.struts.action.ActionMappin
g, org.apache.struts.action.ActionForm, javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
     */
    public ActionForward search(ActionMapping mapping, ActionForm form,
            javax.servlet.http.HttpServletRequest request,
            javax.servlet.http.HttpServletResponse response)
            throws Exception {
             logService.getLog(this.getClass()).debug("search");

             SpringBindingActionForm actionForm = (SpringBindingActionForm) form;
             PeticionsTO to = (PeticionsTO) actionForm.getTarget();

             BeanUtils.copyProperties(form, to);
             valueListActionHelper.search(mapping,form,request,response);

             return mapping.findForward("success");
    }
...
```

Atès això podem determinar el següent:

- No s'accedeix directament a les dades des dels mètodes, sinó que fem ús dels BOs de la capa de negoci definits anteriorment.
- Al invocar mètodes dels BOs, utilitzem Vos de la capa de persistència.
- Els objectes que ens arriben o que enviem a les vistes són DTOs.
- A l'entrar a la vista de creació/edició/consulta de qualsevol de les entitats, en aquest cas, de les Peticions, informem el mode (CREATE_MODE, EDIT_MODE, INSPECT_MODEL), el que farà que alguns dels camps estiguin bloquejats.
- Com cal fer per a preparar la vista per a la exportació del llistat a PDF o a MS Excel.

- **Configuració del controlador i vistes**

Configuració del controlador

La navegabilitat de la nostra aplicació es troba al fitxer '**struts-config.xml**' que es troba dins el projecte ***gcanecnWeb*** al directori /**resources**/**struts.** És en aquest fitxer on hem d'incloure les dues vistes definides anteriorment, per a cadascun dels casos d'ús (Peticions, Productes, Paquets i Grups), dins els action-mappings. El codi serà el següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>
   <data-sources />
   <form-beans>
            <form-bean name="actionForm"
                     type="net.opentrends.openframe.services.web.struts.SpringBindingActionForm"/>
   </form-beans>
   <global-exceptions />

   <action-mappings>
            <!-- Generic Forward Action  -->
            <action path="/page*"
                     type="org.apache.struts.actions.ForwardAction"
                     parameter="pages.{1}"/>

            <action path="/home"
                     type="org.apache.struts.actions.ForwardAction"
                     parameter="pages.home"/>

            <action path="/i18n"
                     type="org.apache.struts.actions.ForwardAction"
                     parameter="pages.i18n"/>


   </action-mappings>

   <controller>
            <set-property property="processorClass"

   value="net.opentrends.openframe.services.web.struts.ExtendedDelegatingTilesRequestProcessor"/>
   </controller>
   <message-resources parameter="i18n.applicationResources" null="false" />


   <plug-in className="org.apache.struts.tiles.TilesPlugin">
       <set-property property="definitions-config" value="/WEB-INF/classes/tiles/tiles-
definitions.xml"/>
       <set-property property="moduleAware" value="true" />
   </plug-in>

</struts-config>
```

Pel nostre cas d'ús inclourem les dues vistes definides anteriorment introduint al *action-mappings* el següent codi:

```
<action path="/peticions" name="actionForm" scope="request"
         parameter="reqCode">
         <forward name="error" path="pages.peticions" />
         <forward name="atras" path="pages.home" />
         <forward name="siguiente" path="pages.siguiente" />
         <forward name="success" path="pages.peticions" redirect="true" />
</action>

<action path="/editPeticio" name="actionForm" scope="request"
         parameter="reqCode">
         <forward name="error" path="pages.peticioEdit" />
         <forward name="success" path="pages.peticioEdit" />
         <forward name="list" path="/peticions.do?reqCode=search" redirect="true" />
         <forward name="deleteSuccess" path="/peticions.do?reqCode=search"
                redirect="true" />
</action>
```

Mitjançant aquest codi estem definint el funcionament de la nostra aplicació, la navegació que es donarà en funció del resultat de cada pas per un Action. Per exemple, si ens fixem en el Action /peticioAccount podem comprovar que si l'Action PeticioAc retorna un success mostrarà al usuari la pàgina definida a Tiles amb nom "***pages.peticioEdit***". I si el resultat del Action es list, llavors el que fa es una nova requests amb el parametre search contra la URL /peticions.do?reqCode=search. Aquesta nova requests acabarà aquí i serà de nou l'Action qui processarà la petició.

Configuració de les vistes

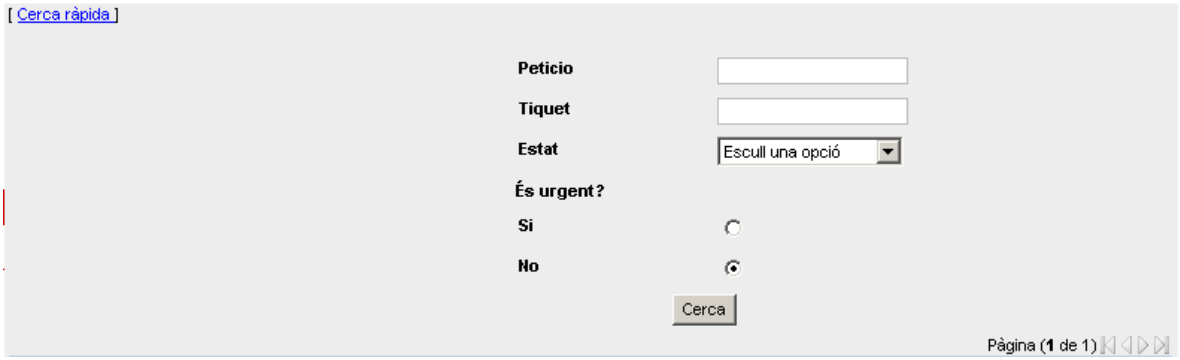
OpenFrameIMI defineix declarativament molts dels comportaments i funcionalitats de les vistes. En el nostre cas, definirem un fitxer de configuració que anomenarem **action-servlet-peticions.xml** i que posarem al projecte ***gcanecnWeb*** al directori **/resources/spring**.

La primera cosa a introduir en aquest fitxer la Action que havien creat, injectant-li els serveis i objectes de negoci que utilitza:

Configuració de desplegables

A continuació explicarem com es configuren els desplegables dels nostres formularis i com s'integren a la nostra pantalla.

Si ens fixem per exemple a la pantalla de cerca d'una nova petició podem veure una cosa semblant a aquesta:



Podem observar un desplegable que s'informa en temps d'execució directament des de la base de dades. En concret és un desplegable d'estats. Per poder aconseguir això és necessari seguir els següents passos.

Tornem a obrir el fitxer de configuració de les vistes (fitxer "**action-servlet-peticions.xml**" del directori /**resources**/**spring** del projecte ***gcanecnWeb***), que és on s'ha de configurar quins camps son desplegables i quins valors mostren. Per això, tenim:

```
<bean parent="selectFieldTag">
     <property name="key" value="forms.peticioForm.field.petEstat" />
     <property name="styleId" value="petEstat" />
     <property name="optionsListName" value="estatsList" />
     <property name="layout" value="true" />
     <property name="mode" value="E,E,E" />
</bean>
```

El camp és un desplegable per tenir com parent a selectFieldTag. Els valors a mostrar son referenciats per la property optionsListName, a la que donem valor "estatsList" i que configurarem al fitxer "openFrame**-services-web-options-lists.xml**" ja existent al directori /**resources**/**spring** del projecte ***gcanecnWeb***:

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

<beans>
......
<entry key="estatsList">
   <bean parent="defaultOptionBaseHibernateAdapter">

            <property name="hql">
              <value>
                select distinct new es.bcn.gca.necn.common.model.EstatsVO(vo.estatId,vo.nom)
                FROM es.bcn.gca.necn.common.model.EstatsVO
                AS vo WHERE 1=1
                /~nom: AND lower(vo.nom) LIKE lower ('[nom]%')~/
               ORDER BY vo.nom
               </value>
            </property>
   </bean>
</entry>

..

   <bean id="estatsOptionListSource" parent="defaultOptionListSource" lazy-init="true">
                    <property name="optionListName" value="estatsList"/>
                    <property name="optionLabelName" value="nom"/>
                    <property name="optionLabelProperty" value="estatId"/>

   </bean>

....

<bean id="optionsListService"
class="net.opentrends.openframe.services.web.taglib.util.options.OptionsListServiceBase" lazy-
init="true">
            <property name="optionsListSources">
                    <map>
                             <!-- Afegeix els teus combos a continuacio -->

                             <entry key="estatsList"><ref bean="estatsOptionListSource"/></entry>
                                      <entry key="productesList"><ref
                                        bean="productesOptionListSource"/></entry>
                    </map>
            </property>
   </bean>
</beans>
```

En aquest fitxer trobem primer tres beans que serveixen com base per a alimentar desplegables amb valors que es recuperan de base de dades amb Hibernate, que és el nostre cas. En principi no cal modificar aquest beans.

Per a aconseguir el nostre desplegable, cal introduïr configuració en altres dos beans:

- defaultOptionValueListHandler, on afegirem una nova entry amb el “estatsList” com key (es correspon amb el valor utilitzat al configurar el camp a la vista), amb la

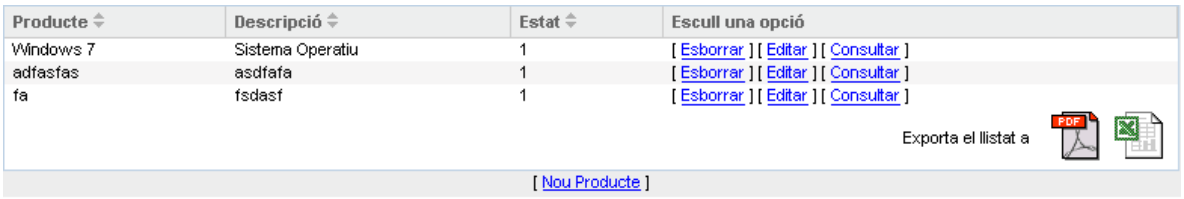
query a realitzar en base de dades (aqui s'ha utilitzat hql, però es pot fer ùs de SQL)

- optionsListService, on afegirem una nova entry amb el "estatsList" com key (es correspon amb el valor utilitzat al configurar el camp a la vista) i referenciant a un bean de nom "**estatsOptionListSource**".

A més, cal definir un nou bean, el anomenat "**estatsOptionListSource**", indicant quin camp és el que conté el valor a manegar internament pel camp i quins valors mostrar al desplegable.

Configuració de llistats

Ara es passarà a explicar com generar llistats com el que es mostra a la taula que ja em vist a pantalles anteriors.



Per generar un llistat com aquest hem de fer dos passos, el primer ja l'hem fet quan hem definit els parametres que s'han d'informar al Action PeticionsAc per presentar aquesta pantalla, al fitxer "**action-servlet-peticions.xml**".

Si ens fixem a aquella definició trobarem aquest extracte:

```
....
<property name="valueListActionHelper">
   <bean parent="valueListActionHelper">
            <property name="listName"><value>peticioList</value></property>
            <property name="tableId" value="PETICIONS"/>
   </bean>
</property>
...
```

Si ens fixem en al propietat "**valueListActionHelper**" aquest acaba cridant a un bean que a la seva vegada té associat una propietat de nom "listName" i valor "**peticioList**", a més d'indicar la taula d'on s'han d'extreure les dades.

Configurarem el llistat al fitxer "**openFrame-services-web-lists.xml**" ja existent al directori /**resources/spring** del projecte ***gcanecnWeb***:

```
....
   <entry key="producteList">
     <bean parent="baseHibernateAdapter">
       <property name="hql">
            <value>
              FROM
              es.bcn.gca.necn.common.model.ProductesVO AS vo
              WHERE 1=1
              /~prodId: AND lower(vo.prodId) LIKE lower('%[prodId]%')~/
              /~prodDesc: AND lower(vo.prodDesc) LIKE lower('%[prodDesc]%')~/
              /~prodEstat: AND lower(vo.prodEstat) LIKE lower('%[prodEstat]%')~/
            </value>
         </property>
     </bean>
   </entry>

   ....
```

Al bean valueListHandler hem afegit una nova entry utilitzant com key el nom "**peticioList**" utilitzat com valor per a la propietat "listName" al definir abans la vista. A aquesta entry definim la query a realitzar en base de dades (aqui s'ha utilitzat hql, però es pot fer ùs de SQL)

Configuració de validacions

Un cop s'ha implementat tota la lògica de la nostra aplicació passarem a la configuració de la validació de les dades dels formularis. Aquesta configuració es divideix en dues parts:

- La definició on executar la validació, que s'indica a la configuració de les vistes, que es on es configuren els formularis.
- La definició de les validacions a fer a cada camp.

Si recordem, la configuració de les vistes que es realitzaven al fitxer "**action-servlet-accounts.xml**" al directori /**resources/spring** del projecte ***gcanecnWeb*** trobàvem parts del codi com les següents:

```
<bean parent="formTag">
   <property name="styleId" value="actionForm"/>
   <property name="layout" value="true"/>
   <property name="validationProperties">
            <props>
                     <prop key="validationType">SERVER</prop>
                     <prop key="validatorName">pet</prop>
            </props>
   </property>
</bean>
```

o també:

```
<bean parent="formTag">
   <property name="styleId" value="actionForm"/>
   <property name="layout" value="true"/>
   <property name="validationProperties">
            <props>
                    <prop key="validationType">CLIENT</prop>
                    <prop key="validatorName">pet</prop>
   </props>
   </property>
</bean>
```

Aquí podem veure dos possibles opcions de configuració de les validacions. D'aquest configuració és important fixar-se en la propietat "**validationType**" i "**validatorName**" de les dues definicions. Veiem que una de les dues definicions té de "**validationType**" SERVER i l'altre com a CLIENT. Així, en el primer cas, la validació del formulari es farà al costat del servidor fent una crida AJAX, i en el segon cas, es farà al navegador mitjançant javascript.

Les validacions específiques a realitzar en cada camp de cada formulari es defineixen al fitxer "**validation.xml**" del directori /**resources/validation**/ del projecte ***gcanecnWeb***. Per als formularis de les nostres pantalles, el codi és el següent:

```
...
<form-validation>

    <global>
        <constant>
            <constant-name>any</constant-name>
            <constant-value>^([0-9][0-9][0-9][0-9])$</constant-value>
        </constant>
        <constant>
            <constant-name>defaultDatePattern</constant-name>
            <constant-value>dd/MM/yyyy</constant-value>
        </constant>
    </global>
    <formset>
            <form name="pro">
                    <field property="prodId" depends="required">
                            <arg0 key="forms.producteForm.field.prodId"/>
                    </field>
                    <field property="prodDesc" depends="required">
                            <arg0 key="forms.producteForm.field.prodDesc"/>
                    </field>
                    <field property="prodEstat" depends="required">
                            <arg0 key="forms.producteForm.field.prodEstat"/>
                    </field>
```

```
                    <field property="prodDataCreacio" depends="required,date">
                            <arg0 key="forms.producteForm.field.prodDataCreacio"/>
                                    <var>
                                            <var-name>datePattern</var-name>
                                            <var-value>${defaultDatePattern}</var-value>
                                    </var>
                    </field>
            </form>

    </formset>
...
```

D'aquest fitxer és important fixar-se que el nom dels forms que han de coincidir amb els noms que hem donat durant la configuració de les nostres vistes. En aquest cas, vam definir "**pro**" com a "**validatorName**", que coincideixen amb el name dels dos forms aquí definits.

D'aquestes definicions de validació podem veure quin tipus de validació s'aplicaran. La més extessa és la "required".

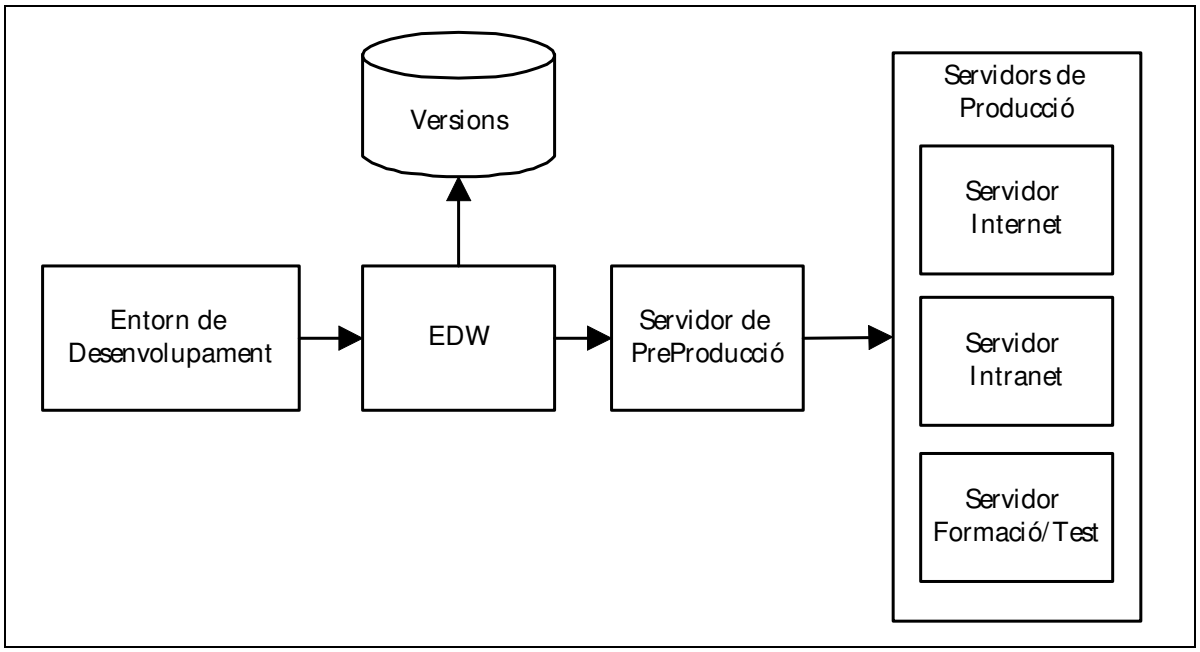
Al fitxer "**validation-rules.xml**" que es troba al directori /**resources**/**validation** del projecte ***gcanecnWeb*** es defineixen els javascript per al navegador i les classes Java per al servidor que es faran servir per els diferents tipus de validació.

5.7. Entorns desenvolupament IMI

L'IMI té una estructura formada per 4 entorns: local, integració, pre-producció, producció Internet i producció Intranet).

- Local: S'hi creen, desenvolupan i proven les aplicacions.
- Desenvolupament: Es fan les primeres proves d'integració.
- PreProducció: Té la mateixa configuració que Producció. Serveix per acceptar les aplicacions abans d'implantar-les.
- Producció: Té un entorn per Internet i un altre per Intranet. És l'únic entorn que té comunicació externa. A Desenvolupament i PreProducció, les aplicacions que necessitin comunicar amb l'exterior ho han de simular.

SIA és una eina feta a l'IMI per guardar les versions de les aplicacions i regular el seu pas als entorns productius. Del desplegament de les aplicacions en els servidors de PreProducció, Formació/Test, Producció-Internet i Producció-Intranet se n'encarrega el departament de Producció.



Entorn local

El desenvolupament de les aplicacions es realitza sempre en local.

L'entorn d'IDE de J2EE està format per l'eina de desenvolupament Rational Application Developer (RAD) que conté un servidor Websphere Application Server (WAS) v6.0 integrat.

Dins el workspace del RAD es disposa d'un plug-in amb un conjunt de tasques, anomenades tasques IMI, que faciliten algunes accions a realitzar: desplegar aplicacions en el servidor local, desplegar aplicacions en el servidor de desenvolupament, configurar l'aplicació dins del servidor local, etc... També hi ha integrats altres plug-ins, per facilitar altres tasques: un visualitzador de logs i un plug-in de validació de codi.

Abans de traspassar aquest desenvolupament local a qualsevol de la resta d'entorns de l'IMI, es desa en una unitat de xarxa. Aquesta unitat realitza les funcions de repositori i també forma part de l'entorn de treball. El repositori és Z:\desenv i conté una carpeta per cada aplicació existent. Existeix una tasca IMI, dins el RAD que executa una exportació del codi sobre la carpeta de Z:\desenv corresponent a l'aplicació. Igualment, n'existeix una altra que realitza el procés contrari. És important fer aquest pas de l'aplicació a Z:\desenv, ja que és necessari a l'hora de fer els traspassos a pre-producció i producció.

Entorn de Desenvolupament

L'entorn de Desenvolupament consta d'aquests elements:

- IBM Rational Application Developer 6.0 (RAD), que inclou el WebSphere Application Server 6.0 Express (WAS)

- Dos workspaces personals **d:\desenv\WS6\evolucio\workspace** i **d:\desenv\WS6\correccio\workspace**, sota els quals es fa el desenvolupament de les aplicacions des del RAD. Permeten que convisquin versions evolutives i correctives d'una mateixa aplicació.

- El directori de xarxa **z:\desenv**, entrada al producte SIA, on s'han de copiar les versions de les aplicacions per poder ser transferides als entorns productius,.

Per seguretat, també s'ha d'utilitzar aquest directori per desar diàriament les aplicacions modificades en els directoris personals

- Un conjunt d'extensions fetes al RAD que el doten de funcionalitats addicionals.

El RAD (i productes compatibles (Eclipse 3.0), com ara WebTool d'Eclipse) veu les aplicacions com un conjunt de projectes que pengen directament del workspace. Els projectes estan agrupats per descriptors situats en el projecte ear, el primer d'ells.

Aquesta seria, per exemple, la forma com el RAD tractaria l'aplicació anomenada **IdjExemple**:

d:\desenv\
WS6\
evolucio\
workspace\
Idjexemple
IdjexempleCommon
IdjexempleBusiness
IdjexempleEJB
IdjexempleWeb

Estructura desenvolupament RAD

Abans de promoure la versió d'una aplicació cap a altres entorns, els seus projectes RAD s'han d'exportar a la carpeta **src** de l'estructura de l'aplicació a **z:\desenv**. Aquesta estructura la crea automàticament SIA quan l'aplicació s'hi dóna d'alta. Els seus nivells superiors són:

z:\desenv\idjexemple.ws6\
dist
document
src\
Idjexemple
IdjexempleCommon
IdjexempleBusiness
IdjexempleEJB
IdjexempleWeb
sql

Estructura estàndard

El nom del directori arrel de l'estructura estàndard té sempre el format sss*aplicació***.ws6**, on:

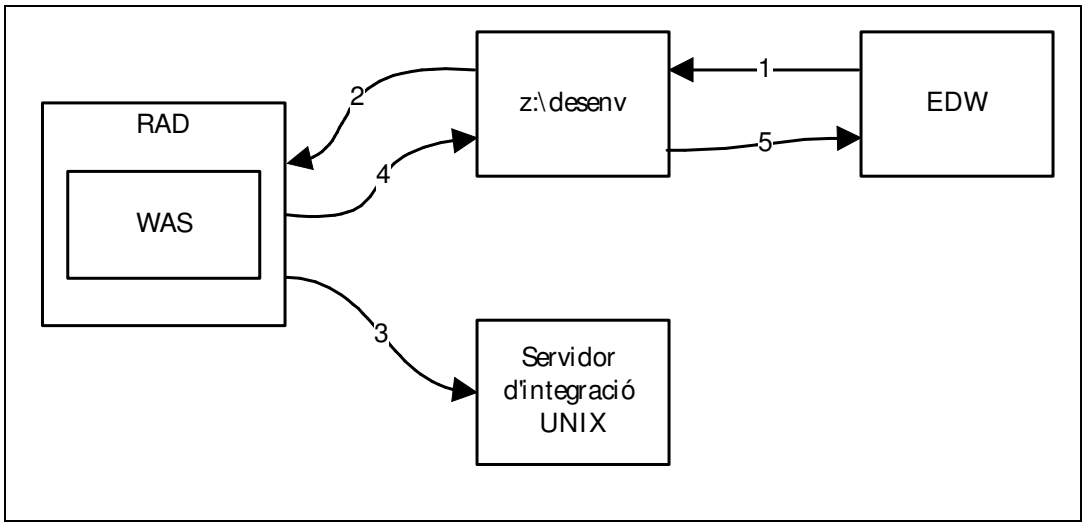
- *sss* són tres lletres que identifiquen un dels sistemes d'informació definits a l'IMI
- *aplicació* és un nom (senzill o NomCompost) que identifica l'aplicació dins el sistema d'informació
- **.ws6** és l'extensió que indica a SIA que l'aplicació correspon a la tecnologia WebSphere, versió 6.0
- **.w61** és l'extensió que indica a SIA que l'aplicació correspon a la tecnologia WebSphere, versió 6.1

El nom del directori arrel s'escriu en minúscules.

El contingut de la carpeta **src** es:

- Un conjunt de projectes RAD, *sssaplicació*{**Common**,**Business**,**EJB**,**Web**}, cada un dels quals té el seu propòsit donat per l'arquitectura. Cada projecte té la seva carpeta dins de **src**.
- Una carpeta **sql** per contenir els scripts de creació del model de dades de l'aplicació.

Esquemàticament, l'entorn de Desenvolupament es pot representar així:



Les fletxes indiquen:

1) Quan es defineix una aplicació a SIA, se li genera a **z:\desenv** una estructura estàndard d'acord amb les característiques declarades.

2) Còpia d'una versió de **z:\desenv** en el RAD, on es desenvolupa i prova amb el servidor WAS local

3) Desplegament i proves d'una versió en el servidor WAS d'integració

4) Còpia a **z:\desenv** d'una versió en format instal.lable

5) Pas d'una versió a Producció.

Configuracions IMI

1. Directoris shared:

   - C:\IBM\Rational\SDP\6.0\imi-shared\ext: per ubicar els jars d'utilitat no desenvolupats a l'IMI però que formen part dels nostres estàndards.

   - C:\IBM\Rational\SDP\6.0\imi-shared\int:
     - Per ubicar els jars d'utilitat o de client d'Ejb desenvolupats a l'IMI.
     - Per ubicar els jars del framework de desenvolupament openFrameIMI, i els jars de les seves dependències no inclosos a C:\IBM\Rational\SDP\6.0\imi-shared\ext.

- C:\IBM\Rational\SDP\6.0\imi-shared\conf: per ubicar fitxers de configuració com els log4j, o mq.

2. S'han definit aquestes llibreries com *shared libraries* i s'identifiquen amb un **nom:**

   - Tots els .jar són a les *shared libraries* excepte ojdbc14.jar i log4j.jar, providerutil.jar i fscontext.jar que són en el *classpath* del servidor.

   - Les *shared libraries* d'openFrameIMI (Fwk*) es defineixen mitjançant subcarpetes de C:\IBM\Rational\SDP\6.0\imi-shared\int; tot jar inclòs en alguna subcarpeta d'una *shared library* forma part del classpath de l'aplicació a la qual s'assigna aquesta *shared library*.

Shared Libraries:

Shared Library	Fitxer / Carpeta	Versió
IdjRutinesUg	idjrutinesugCommon.jar	Última
IdjDadesAplicUg	idjdadesaplicugCommon.jar	Última
IdjEdicioUg	idjediciougCommon.jar	Última
BigFacelessPDF	bfopdf.jar	1.2.5
DocFramework	docproxy_fwkCommon.jar docproxy_fwkEJBClient.jar	Última
GbgGeocodificacioUg	gbggeocodugEjbClient.jar	Última
RatTramitsUg	rattramitsugEjbClient.jar	Última
CdcCapturadadUg	cdccapturadadugEjbClient.jar	Última
CtuCtrlUserUg	ctuctrluserugEjbClient.jar	Última
IdjLoginUg	idjLoginugEjbClient.jar	Última
AceAcer	aceacerEjbClient.jar	Última
IdjCripto	idjcriptoCommon.jar	Última
BigFacelessReport	Bforeport.jar	1.1.5
Xerces	xercesImpl	2.6.2
Xml-Apis	xml-apis.jar	2.0.1
Xstream	xstream.jar	1.1
Commons-Net	commons-net	1.2.1
Jakarta-Oro	jakarta-oro.jar	2.0.8
Shareds del Portal de Tràmits		
Shareds dels mòduls comuns d'administració electrònica (eRegistre)		
FwkCore	\fwk\aop \fwk\core \fwk\core\common \fwk\core\configuracio	

Shared Library	Fitxer / Carpeta	Versió
	\fwk\core\excepcions \fwk\core\i18n \fwk\core\persistencia \fwk\core\traces \fwk\core\validacions \fwk\core\web	
FwkCaptcha	\fwk\captcha	
FwkCharts	\fwk\charts	
FwkFiles	\fwk\file	
FwkFilesUpload	\fwk\fileupload	
FwkFTP	\fwk\ftp	
FwkMonitor	\fwk\jmx	
FwkMail	\fwk\mail	
FwkPDIB	\fwk\pdib	
FwkPortasig	\fwk\portasignatures	
FwkReports	\fwk\reports	
FwkScheduler	\fwk\scheduler	
FwkSecurity	\fwk\seguretat	
FwkSignatura	\fwk\signatura	
FwkSNMP	\fwk\snmp	
FwkUc4	\fwk\uc4	
FwkWebServices	\fwk\ws	
FwkWorkflow	\fwk\workflow	
FwkXML	\fwk\xml	

La primera vegada que una aplicació es desplegui en un servidor se li hauran d'associar les shareds Libraries que utilitzi.

3. Providers d'Oracle.

   Primer es dóna valor a la variable *ORACLE_JDBC_DRIVE_PATH*: *C:\IBM\Rational\SDP\6.0\imi-shared\ext\ojdb14.jar* i després es defineixen els proveïdors, un *Oracle JDBC Driver* i un altre *Oracle JDBC XA Driver*.

4. Log4j:

   El jar de log4j s'ubica a *c:/IBM/Rational/SDP/6.0/runtimes/base_v6/lib/ext*.

   Configuració per donar suport al log de les aplicacions:

      El fitxer de configuració per a les aplicacions s'ubica a *c:\IBM\Rational\SDP\6.0\imi-shared\conf*:

      Per tal que les aplicacions facin servir aquest fitxer, cal configurar-lo des de la consola del WAS: *Application servers ->server1->ProcessDefinigion> Java Virtual Machine→Generic JVM arguments*:

         -Dlog4j.configuration=file:///c:/IBM/Rational/SDP/6.0/imi-shared/conf/log4j-config.xml

El fitxer de logs applications-log4j.log es grava a *C:\IBM\Rational\SDP\6.0\runtimes\base_v6\profiles\default\logs\server1*

5.8. Conclusions

Les meves conclusions a la finalització del PFC són molt bones i es corresponen amb la obtenció dels objectius marcats per a la seva realització i de les perspectives marcades per mí mateix. És a dir, he pogut estudiar i treballar amb frameworks molt potents i actualment molt utilitzats i, sobretot, aprendre moltíssim sobre el seu funcionament, la seva configuración i el desenvolupament d'aplicacions sobre aquests frameworks.

Sincerament, la realització d'aquest PFC ha sigut un repte molt important per a mí, ja que els entorns de desenvolupament així com el desenvolupament d'aplicacions no han estat mai un punt fort per a mí, degut al meu perfil més tècnic.

Tot i que he patit en algunes parts del desenvolupament, he gaudit moltíssim realitzant-lo i, principalment, sobrepossant-me a les adversitats i obtenint els resultats esperats.

6. Bibliografia

Tota la documentació que s'ha utitlizat per a realizar el PFC es documentació interna del Departament d'Arquitectura de l'Institut Municipal d'Informàtica.

- CatalegServeisFrameworkJ2EE.doc
- FW-001-Introduccio-openFrame-1.0.6.CAT.doc
- FW-110-Best-practices.doc
- GuiaPracticaRAD.doc
- NormativaAplicacionsWebSphere.doc
- ArquitecturaAplicacionsWebSphere.doc
- DescripcioEntornDsvWebSphere.doc
- FW-007-Servei-persistència-1.0.6.CAT.doc
- FW-007-Servei-pantalles.doc
- FW-007-Servei-llistats-1.0.2.CAT.doc
- FW-007-Servei-multiidioma.doc
- FW-007-Servei_presentacio_tags-1.0.9.CAT.doc
- FW-007-Servei-validacio.doc
- FW-007-Servei-persistència-1.0.6.CAT.doc
- FW-007-Servei-correu-electronic-1.0.4.CAT.doc
- FW-007-Servei-configuracio-1.0.6.CAT.doc
- FW-007-Servei-excepcions.doc
- FW-007-Servei-persistència-1.0.6.CAT.doc
- FW-007-Servei-persistència-1.0.6.CAT.doc
- FW-007-Servei-persistència-1.0.6.CAT.doc