PROJECTE FINAL DE CARRERA

Desenvolupament d'una aplicació web d'habitatges

(Development of a housing listing web application)

Estudis: Enginyeria de Telecomunicació

Autor: Jordi Martí Carreras

Director/a: Jordi Forga Alberich

Any: 2016

Índex general

Col·laboracions

Departament d'Enginyeria Telemàtica. UPC

Agraïments

Gràcies als meus pares i a la meva germana, sense ells aquest projecte no hauria estat mai possible.

Resum del Projecte

L'aplicació LloguerJove.com (http://www.lloguerjove.com) és un portal web que ofereix la possibilitat a l'usuari de cercar i trobar diversos tipus d'habitatges (lloguer, habitacions en pisos compartits i lloguer per a temporades curtes) i proveeix el servei de publicació i gestió d'anuncis tan pels usuaris propietaris dels habitatges i agències, com pels usuaris que demanden habitatge, i que els permet també comunicar-se a través de missatges privats.

En aquest projecte es descriuen les diferents fases de desenvolupament de l'aplicació: l'anàlisi, el disseny i la posterior implementació del portal. Prèviament s'introdueixen de forma gradual i progressiva els conceptes fonamentals (HTTP, URL, plataforma Java, POO, *servlets*, *frameworks*), i principis sobre els quals es fonamenta l'estructura (arquitectura MVC) i el posterior disseny la solució (patrons de disseny: DAO, *Factory*, *Strategy*). També es presenta i s'avalua l'efectivitat del disseny guiat per el domini (DDD) en la programació orientada a objectes en el marc del desenvolupament d'una aplicació web.

Finalment es realitza una descripció de les tecnologies i frameworks que s'han utilitzat en el projecte (Struts, Hibernate, Solr, Apache shiro, til·les, etc..), i es descriuen les interfícies entre els seus components interns i sistemes externs i el sistema físic en el qual es despleguen les aplicacions.

Resumen del Proyecto

La aplicación LloguerJove.com (http://www.lloguerjove.com) es un portal web que ofrece la posibilidad al usuario de listar y encontrar varios tipos de viviendas (alquiler, habitaciones en pisos compartidos y alquiler para temporadas cortas) y provee el servicio de publicación y gestión de anuncios tanto para los usuarios propietarios de las viviendas y agencias como para los usuarios demandantes de vivienda, y que les permite también comunicarse a través de mensajes privados

En este proyecto se describen las diferentes fases de desarrollo de la aplicación: el análisis, el diseño y la posterior implementación del portal. Previamente se introducen de forma gradual y progresiva los conceptos fundamentales (HTTP, URL, plataforma Java, POO, servlets, frameworks), y principios sobre los que se fundamenta la estructura (arquitectura MVC) y el posterior diseño la solución (patrones de diseño: DAO, Factory, Strategy). También se presenta y evalúa la efectividad del diseño guiado por el dominio (DDD) en la programación orientada a objetos en el marco del desarrollo de una aplicación web.

Finalmente se realiza una descripción de las tecnologías y frameworks que se han utilizado en el proyecto (Struts, Hibernate, Solr, Apache shiro, Tiles, Freemarker, SLf4j, Log4j), se describen las interfaces entre sus componentes internos y sistemas externos y el sistema físico en dónde se despliega la aplicación.

Abstract

LloguerJove.com is a web application (http://www.lloguerjove.com) that provides people to list, find and rent for various types of housing (rent flats, rooms in shared flats and short term rentals) and provides listing publishing and management services for both owners and housing agencies and housing demanding users, and also allows them to communicate through private messages.

This project describes the different phases of the application development: analysis, design and subsequent implementation of the portal. Previously we introduce gradually and progressively fundamental concepts (HTTP, URL, Java platform, POO, servlets, frameworks) and principles which application structure (MVC), design and subsequent implementation (patterns design: DAO, Factory, Strategy) is based on. It also presents and evaluates the effectiveness of the domain driven design (DDD) in object-oriented programming in the context of a web application development.

We provide a description of the technologies and frameworks that we have been using in the project (Struts, Hibernate, Solr, Apache Shiro, tiles, etc…), a description of the interfaces between its internal components and external systems, and the physical system where the application is deployed.

1. Introducció

1.1 Context del projecte

El projecte s'emmarca dins de l'àmbit de la enginyeria de software i del desenvolupament d'aplicacions web. Els sistemes de software són inherentment complexos. Aquesta complexitat pot dominar-se, però no eliminar-se. L'estructura del sistema és clau en tot el cicle de vida d'un sistema de software. És important doncs, descriure l'estructuració del sistema de forma eficient en les etapes inicials del desenvolupament. Si no s'apliquen de forma efectiva tècniques de enginyeria de software , molts projectes produeixen software que és irrealitzable, lliurat fóra de termini tard o sobre pressupostat. D'altre banda, un sistema de software al llarg de la seva vida útil, pot necessitar millores, actualitzacions o integració de noves característiques. El cost d'aquestes correccions o actualitzacions pot ser pot ser molt elevat si no s'ha plantejat una bona arquitectura o disseny.

Sabem que no existeix un sistema *infal·lible* per abordar la enginyeria del software. La gran diversitat de sistemes i organitzacions implica que són necessaris diferents plantejaments a l'hora d' orientar el desenvolupament del software.

Aquest projecte aborda els principis que poden ajudar a la creació i estructuració d'un aplicació web complexa, de fàcil manteniment i de qualitat basant-se en la capacitat de la programació orientada a objectes, a través de l'abstracció, la descomposició en sistemes més petits i les relacions de jerarquia.

La majoria de tècniques i dissenys exposats en aquest projecte son vàlids per a desenvolupar software de dimensió industrial.

1.2 Objectius

L'objectiu del projecte es presentar els conceptes i principis sobre els quals es fonamenta el desenvolupament de l'aplicació Lloguerjove i que

serveixi com una guia de desenvolupament de software per tal d'ajudar a construir una aplicació robusta, de fàcil manteniment i de qualitat.

Tot i que en aquest projecte no empra una metodologia de desenvolupament concreta, també s'exposen totes aquestes tècniques amb l'objectiu d'accelerar el desenvolupament.

En aquest projecte vol mostrar també la efectivitat de la descomposició d'un sistema en nivells (*tiers*) i capes (*layers*) .

Es vol presentar el disseny guiat per el domini (DDD) i validar la seva efectivitat en el desenvolupament d'aplicacions web.

Finalment s'introdueixen un recopilatori de tecnologies i sistemes que conformen la aplicació LLOG i que li proveeixen una configuració estable, de fàcil manteniment, funcional i productiva d'acord amb els requeriments tan funcionals com no funcionals.

1.3 Estructura de la memòria

El projecte s'estructura en 3 parts grans seccions o capítols:

- Conceptes previs: en aquest capítol es presenten els conceptes i principis fonamentals que ajudaran a posar les bases en el posterior disseny. Aquest capítol presenta un enfoc de baix a dalt, és a dir, es comença per descriure els elements més bàsics i fonamentals de la web i de la programació orientada a objectes i progressivament es defineixen i construeixen les estructures sobre les quals es recolzarà l'aplicació web.
- Descripció del sistema: es descriu una especificació dels requisits tan funcionals com no funcionals d'acord amb els requeriments de l'aplicació per el cas particular de LLOG. En aquest capítol s'utilitza el llenguatge visual de UML per il·lustrar les idees i conceptes que es descriuen en el projecte.
- Disseny i implementació: es realitza una descripció de les tecnologies i *frameworks* utilitzats, es descriuen les interfícies entre els seus components interns i sistemes externs i la implementació de la solució concreta per l'aplicació.

1.4 Definicions, acrònims i abreviatures

LLOG - Acrònim de LloguerJove, el nom de l'aplicació.

RFC - Acrònim de *Request for Comments*, documents que descriuen diversos aspectes de funcionament d'internet i d'altres xarxes de computadores.

ASP – Acrònim de Application Service Provider.

CEN – Acrònim de Comité Européen de Normalisation.

JSP – Acrònim de Java Server Pages

JSTL – Acrònim de JavaServer Pages Standard Tag Library

BO – Acrònim de Business Object. Els BO abstrauen les entitats del domini que utilitza l'aplicació. Encapsulen les dades i el comportament associat a l'entitat que representa.

POJO – Acrònim de Plain Old Java Object. S'utilitza per anomenar classes que són ordinàries, és a dir que no implementa cap interfície ni estén cap classe d'un framework concret.

MTA - Mail Transport Agent.

SEO - Acrònim de Search Engine Optimitzation

WWW – Acrònim de World Wide Web

ASF - Acrònim de Apache Software Foundation

HTML - HyperText Markup Language

URL - Uniform Resource Locator

CGI - Common Gateway Interface

API - Application Programming Interface

EIS - Enterprise Information System

DDD - Acrònim de Domain Driven Design

JMS - Acrónim de Java Message System

DTO - Acrònim de Data transfer Object

ORM - Acrònim de Object-Relational mapping

GSP - Groovy Server Pages

AOP - Acrònim de Aspect-oriented programming (programació orientada a aspectes)

i18n - *numerònim* en angles "Internacionalització"

GoF - Acrònim de Gang of Four, el nom amb el qual es coneix als autors del llibre Design Patterns (Gamma,Erich; Helm , Richard;Johson Ralf;Vlissides,John, 1994).

UML - Acrònim de Llenguatge unificat de modelat

POO - Acrònim de Programació orientada a objectes

bean - Estàndard que defineix un tipus de classe Java.

lazy initialization - Patró de disseny que consisteix en demorar la creació d'un objecte.

2. Conceptes previs

2.1 Breu història de la WWW

Cap projecte o llibre sobre tecnologia web estaria complert sense una breu repassada de com la World Wide Web (WWW) s'ha convertit en quelcom tant popular com és avui en dia. La web ha recorregut un llarg trajecte des de que els primers documents d'hipertext s'enviaven per internet. En l'actualitat s'ha convertit en quelcom molt més complex i és la base dels sectors industrials i socials. El www és un sistema d'informació desenvolupat en el 1989 proposat pel enginyers **Tim Berners-Lee** i **Robert Cailliau** del laboratori CERN amb l'objectiu de compartir la informació d'una forma més efectiva. Timothy "Tim" John Berners-Lee va establir la primera comunicació entre un client i un servidor utilitzant el protocol HTTP. Ell i el seu grup van desenvolupar el que per les sigles en anglès s'anomena HTML (*HyperText Markup Language*) i el sistema de localització d' objectes a la web, URL (*Uniform Resource Locator*).

Tot i que va portar temps veure els beneficis d'utilitzar la web per aquells que estaven en el CERN, amb el temps el servei de www sobre internet, ha esdevingut un dels de major èxit.

Des del principi es va dissenyar la web per tractar amb documents estàtics, tot i que va ser una progressió natural poder comptar amb la possibilitat de generar dinàmicament el contingut del document.

La CGI (*Common Gateway Interface*) es va crear amb aquesta finalitat. CGI és un estàndard que permet als servidors Web interactuar amb aplicacions externes de manera que les pàgines de hipertext ja no tenen que ser estàtiques.

Un programa CGI pot recuperar resultats d'una base de dades i inserir aquests resultats com una taula en un document de hipertext. De la mateixa forma, les dades incorporades en una taula de hipertext es poden inserir en una base de dades. Aquesta tecnologia va obrir moltíssimes possibilitats i de fet, va iniciar la moda d'internet dels anys 90.

Tot i això, CGI presenta varis inconvenients, els llenguatges utilitzats per dissenyar aplicacions cal que es basin en procediments (porció de codi que realitza una tasca específica). En termes de rendiment, les aplicacions CGI requereixen de la creació d'una nova instancia de la aplicació per a

cada sol·licitud, crear un nou procés, amb el que pot afectar negativament el rendiment. Millores en CGI, com per exemple la extensió FastCGI) reduïren els problemes en rendiment que presentava inicialment.

CGI només descriu el contracte entre el servidor web i el programa. No proporciona serveis per ajudar a implementar sistemes centrats en els usuaris (manteniment de la identitat del client, formes de manteniment de la informació de l'usuari, restricció d'accés a la informació per usuaris autoritzats i l'emmagatzematge d'informació d'execució en l'aplicació).

Per tot això era necessari comptar amb una estructura per albergar les aplicacions creades per a la web.

El 1997, quan el llenguatge Java estava experimentant un enorme creixement, *Sun Microsystems* va crear la tecnologia *Java Servlet* per solucionar els problemes relacionats amb CGI anteriorment mencionats. Aquesta nova tecnologia obriria una nova via d'exploració per als desenvolupadors web.

2.2 El protocol HTTP

El protocol HyperText Transfer Protocol és l'estàndard de facto en la transferència de documents en el www tot i que ha estat dissenyat per ser extensible a un gran nombre de formats de documents. La versió 1.1 està documentada en el RCF 2616 (HTTP/1.1)

HTTP opera sobre connexions TCP, usualment escoltant al port 80, tot i també s'utilitzen altres ports. Després d'una connexió amb èxit, el client transmet un missatge de petició al servidor, el qual envia un missatge de resposta.

HTTP és un protocol de comunicacions *sense estat,* és a dir, que tracta cada petició com una transacció independent que no té relació amb qualsevol sol·licitud anterior,

Els missatges HTTP són intel·ligibles per l'home de manera que es pot operar manualment un servidor HTTP amb comandes com :

```
$ telnet server 80
```

```
>GET /es/Intro.do
```

HTTP està basat en un model petició/ resposta, per al que existeixen dos tipus de missatges HTTP: la petició i la resposta. En navegador obre una connexió a un servidor i realitza una petició. El servidor processa la petició del client i retorna la resposta.

Ambdós tipus de missatges consten d'una línia de partida, zero o més camps d'encapçalament i una línia buida que indica el final dels encapçalaments del missatge. Ambdós tipus de missatges també poden contenir un cos de missatge opcional. El format i disseny dels missatges de petició i resposta son molt similars, però existeixen algunes diferències.

La Petició HTTP

La línia inicial d'una petició HTTP es coneix com la línia de petició. Sempre és la primera línia d 'un missatge de petició i conté tres camps:

- Un mètode HTTP: Tot i que existeixen diversos mètodes HTTP per a recuperar dades de un servidor, els dos que s'utilitzen més sovint son GET i POST. El mètode GET sol·licita un recurs del servidor, indicat per la petició URI. Si el URI apunta a un recurs de generació de dades com un servlet, les dades es retornaran dins del missatge resposta. Tot i que el missatge GET pot passar informació en la cadena de consulta, el mètode POST s'utilitza per passar explícitament dades al servidor que es poden utilitzar per al processament per part del URI de petició.

- Un Identificador Universal de Recursos: El URI identifica el recurs que hauria de processar la petició. Una petició amb una URI invàlida retornarà un codi d'error (normalment 404).

- Una versió del protocol HTTP : La versió del protocol de petició HTTP identifica davant del servidor a quina versió de l'especificació HTTP s'ajusta la petició. El següent exemple il·lustra la línia de petició per a una petició GET:

```
GET /index.html HTTP/1.0
```

Com s'ha mencionat anteriorment, la petició HTTP pot contenir zero o més camps d'encapçalament. Els camps d'encapçalament permeten que el client passi al servidor informació addicional sobre la petició i el propi client.

El format del camp encapçalament, tant per a la petició com per la resposta, és el nom del camp d'encapçalament, seguit de dos punts (:) i el valor. Si s'especifiquen múltiples valors per a un sol camp d'encapçalament, tenen que estar encapçalats per comes.

A més de les peticions GET, els clients poden enviar també peticions del tipus HEAD i POST, de les quals POST en són els més importants. Les peticions POST s'utilitzen per als formularis de HTML i altres operacions que requereixen a client transmetre un bloc de dades al servidor.

La petició HEAD és idèntica al GET excepte en que el servidor no ha de retornar un missatge en la resposta. La meta informació continguda en les capçaleres HTTP en la resposta a una petició HEAD hauria de ser idèntica a la informació enviada en resposta a una petició GET. Aquest mètode es pot utilitzar per obtenir meta informació sobre les entitats implicades en la petició sense haver-les de transferir. Aquest mètode també es pot utilitzar per fer un test sobre la validesa, accessibilitat i modificació recent dels enllaços. Després d'enviar la petició HEADER i la línia en blanc, el client transmet les dades. El HEADER ha d'haver inclòs una capçalera *Content-Length*: camp, que permet al servidor determinar quan s'han rebut tots les dades

2.3 URL

Uniform Resource Locators (URLs) són seqüències que especifiquen com accedir als recursos de la xarxa , tals com documents HTML. Són part de la classe més general dels Universal Resource Identifiers (URIs). L'ús més important de URLs està en els documents de l'HTML per a identificar els destins dels enllaços (hyperlinks). Quan s'utilitza un navegador, normalment els enllaços estan destacats i cada un té una URL associat a ella, que és accessible quan l'enllaç és activat amb un clic del ratolí.

URLs relatives especifiquen només una part de la URL complerta, la informació que s'omet es pot deduir a partir del context del document font.

La URLs es van documentar inicialment en RFC 1738 (http://www.rfc-editor.org/info/rfc1738). URLs relatives es documenten en RFC 1808 [3]. URIs es documenten en RFC 1630 [4]

.

La següent URL descriu la pagina d'inici de Lloguer Jove:

http://www.lloguerjove.com/ca/Intro.do

El significat dels camps que la componen és el següent:

```
http
```

Indica que s'utilitza HTTP per tal d 'accedir al document. Altres possibles valors en aquest camp són https (secure HTTP), ftp (File Transfer Protocol), entre molts d'altres.

```
www.lloguerjove.com
```

Indica el nom de host que s'ha de resoldre usant el DNS (Domain Name Service)

```
/ca/Intro.do
```

Indica el directori i el nom del fitxer que han de ser enviats a través de la petició http per tal de identificar el document d'entre qualsevol d'altres en el servidor.

2.4 MIME (Multipurpose Mail Extensions)

El tipus MIME d'un document es refereix a l'estàndard oficial d 'internet que defineix el format amb el qual s'han d'ajustar els missatges de manera que es puguin intercanviar entre diferents sistemes d' email.

Els tipus de continguts MIME també tenen molta importància fora del context dels correu electrònic. Per exemple, en el protocol HTTP, en una petició de tipus POST, s'indica el tipus MIME de la petició a través de la capçalera Content-type, que pot prendre els valors: multipart/form-data (transmissió de arxius binaris) o bé application/x-www-form-urlencoded.

El tipus MIME és un format molt flexible, que permet incorporar virtualment qualsevol tipus d'arxiu en un missatge d'email.

En essència, el camí a seguir a l'hora de transmetre un fitxer és el mateix: codificar el fitxer no ASCII a US-ASCII (fent-lo al mateix temps compatible amb l'estàndard SMTP "Simple Mail Protocol") transmetre en aquest format i reconvertir-lo al destí en el format original (descodificar-lo).

Sense entrar en una descripció detallada, podem destacar les següents funcions del sistema MIME:

- Classificar els continguts a transmetre segons diversos tipus.
- Establir quina acció s' ha de prendre per a cada tipus de fitxer que es transmet, és a dir, quin tipus de codificació s'ha d'utilitzar per a cada tipus de fitxer.

Per aconseguir-ho, d'entre la gran varietat de tipus de formats (còdecs) de fitxers utilitzats, es va acordar establir unes taules el més complertes possibles en les que s'indiqui quina codificació ha d'utilitzar-se per a cada tipus i subtipus.

El sistema té l'avantatge de que quan apareix un nou tipus de format només cal actualitzar en els Navegadors i Agents de Correu els parells de valors tipus de fitxer/tipus de còdec.

Per a evitar una excessiva proliferació de codificadors diferents, la IANA (*Internet Assignet Numbers Authority*) s'encarrega d'establir les taules.

Veure Annex: Exemples de tipus MIME

2.5 CGI

El CGI és el primer mecanisme que permet als servidors generar contingut dinàmicament. CGI defineix una interfície que permet als clients accedir a les aplicacions de la mateixa forma estàndard a través de HTTP.

Una URL de programa CGI té el següent aspecte:

```
http://www.myserver.com/cgi-bin/MyExecutable?name1=value1&name2=value2
```

La part de la URL /cgi-bin/ informa al servidor que cal executar el programa CGI especificat en la següent part de la URL.

Veure Annex: Exemples de tipus MIME

2.6 La Plataforma Java

En l'actualitat Java és un dels llenguatges de programació més elaborats i més utilitzats per a la creació de software d'empresa. L' evolució de Java que ha passat de ser un medi de desenvolupament d'*applets* per a ser executats en navegadors a ser un model de programació capaç de manejar les aplicacions d'una empresa d'avui en dia ha estat extraordinari.

Amb el transcurs dels anys, Java ha desenvolupat tres edicions de plataformes diferents, cada una d'elles destinades a cobrir un conjunt diferent de necessitats de programació.

- La plataforma Java , Micro Editon (Java ME).
- La plataforma Java Standard Edition (Java SE).
- La plataforma Java Enterprise Edition (Java EE).

Java SE es una col·lecció de APIs del llenguatge de programació Java útils per a molts dels programes de la Plataforma Java. Java SE és una plataforma àmpliament utilitzada per al desenvolupament i desplegament d'aplicacions portàtils per a entorns d'escriptori i servidors.

La plataforma Java EE és una extensió de la plataforma Java SE que proveeix una API per a desenvolupar aplicacions d'empresa.

La plataforma que s'utilitza en aquest projecte és la Java SE 7, degut a que la complexitat de la especificació JavaEE i el cost d'implementació no justifica la utilització ni la sobrecàrrega d'aquesta plataforma. Per exemple, per modelar la presidència es fa ús de la API de **Hibernate**, i per la seguretat s'utilitza la API de **Apache Shiro**.

En sentit estricte, Java SE és una especificació de la plataforma. Defineix una àmplia gamma de propòsits generals APIs, com les API de Java per a la biblioteca de classes de Java, i també inclou l'especificació del llenguatge Java i la Java Virtual Machine Specification. [15] Una de les implementacions més conegudes de Java SE Development Kit és Java d'Oracle Corporation (JDK), que és la que està pensada per crear aplicacions de servidor.

El següent diagrama conceptual que il·lustra tots els components tecnològics de la plataforma Java SE i com encaixen entre si.

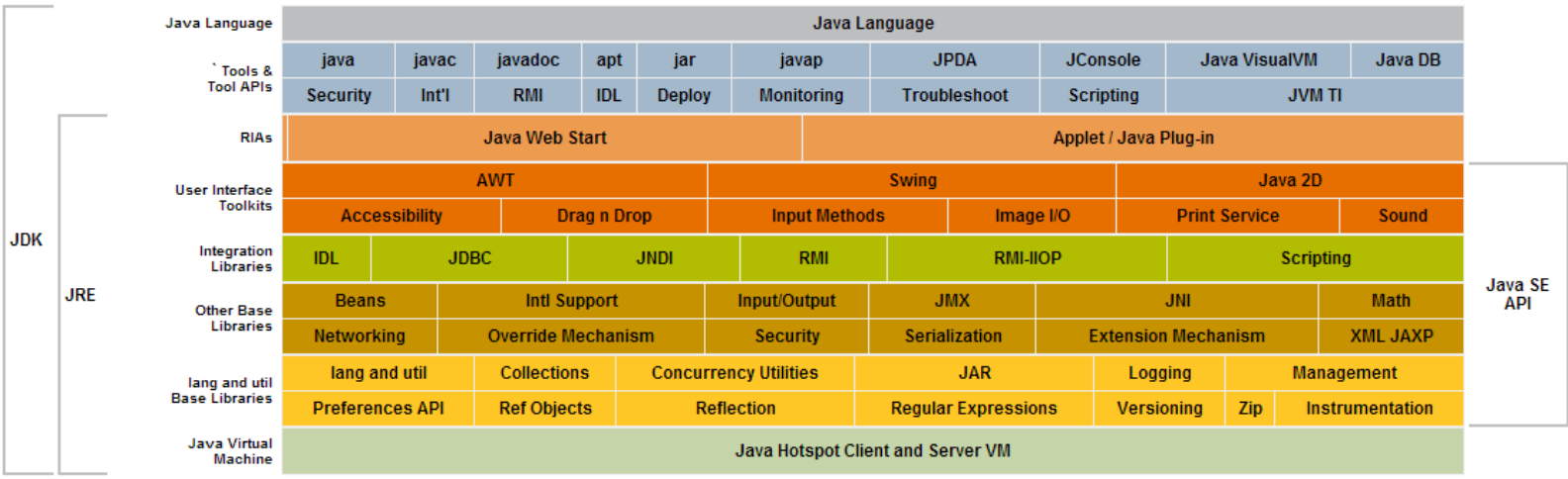


Figura 2-1 Components tecnològics Java SE

- https://docs.oracle.com/javase/7/docs/api/

Característiques principals

- **Orientat a objectes.**
- Amb Garbage Collector.
- Segur.
- Portable.
- Multithread
- Tipat estàticament
- Interpretat.

Les característiques mes destacades dels llenguatges orientats a objectes són les següents:

- Abstracció: És el procés en el qual separem les propietats més importants d'un objecte, on es capturen els seus comportaments. Cada objecte en el sistema serveix com a model d'un "agent" abstracte que pot realitzar un treball, informar i canviar el seu estat, i "comunicar-se" amb altres objectes del sistema sense revelar com s'implementen aquestes característiques.
- Encapsulament: També referit com ocultació de la informació, l'encapsulament és la propietat de l'orientació a objectes que ens permet assegurar que la informació d'un objecte li és desconeguda als altres objectes en l'aplicació.
- Jerarquia: la tecnologia orientada a objectes ens permet definir jerarquies entre classes i jerarquies entre objectes. Les dues jerarquies més importants que existeixen són la jerarquia "és un" que necessita la generalització i especificació entre classes (herència en java) i la jerarquia "és part de" en la qual es delimita l'agregació d'objectes (agregació).
- Polimorfisme: El polimorfisme és la propietat per la qual una entitat pot prendre diferents formes. Generalment aquesta entitat és una classe, i la manera per la qual s'aconsegueix que prengui diferents formes és a través de nomenar els mètodes d'aquesta classe amb un mateix nom però amb implementacions diferents.

- Modularitat: permet repartir l'aplicació en mòduls bàsics amb funcionalitats o mètodes relacionats. Per exemple, els Servlets de Java o les pàgines de JavaServer proporcionen un esquema per modularitzar les aplicacions, que ens guia a separar les aplicacions en diversos nivells i tasques individuals.
- Re usabilitat: fer servir l'orientació a objecte per a encapsular la funcionalitat compartida és una forma de reutilització.

Plataforma Java Standard Edition

Com s'ha comentat anteriorment, *Sun Microsystems* va posar en el punt de mira els CGI alhora de dissenyar els Servlets. A continuació es fa una breu descripció d'aquesta tecnologia.

Servlets java

Són programes que permeten incorporar la lògica d'aplicació al procés de sol·licitud-resposta HTTP. Amplien la funcionalitat del servidor web per admetre contingut dinàmic. A diferència de CGI, que s'inicia un procés per a cada sol·licitud, els Servlets s'executen en un sol procés utilitzant *threads* més lleugers per a cada petició. Els Servlets representen una arquitectura més eficient.

Pàgines JavaServer

Les JavaServer Pages (JSP) proporcionen un mètode d'incorporar components a una pàgina i fer que realitzin la seva funció per generar la pàgina que serà enviada al client. Les pàgines JSP són de fet una extensió del model de programació dels Servlets. Quan un usuari sol·licita una pàgina JSP, el contenidor web compila la pàgina JSP en un Servlet. El contenidor web invoca al Servlet i retorna el contingut resultant al navegador web. Les pàgines JSP són únicament documents basats en text fins que el contenidor web els compila en els corresponents Servlets. D'aquesta manera es proporciona una presentació precisa entre lògica d'empresa i lògica de presentació.

Llibreries d'etiquetes (Tags) JSP

Les etiquetes o Tags JSP són elements del llenguatge JSP definits per l'estàndard o personalitzats per l'usuari que encapsulen tasques repetitives o funcionalitats comunes per la majoria d'aplicacions web. Les etiquetes personalitzades es distribueixen en una biblioteca d'etiquetes, que defineix un conjunt d'etiquetes personalitzades relacionats i conté els objectes que implementen les etiquetes.

Models d'Implementació Java EE

Tal i com es descriu en (Mark Cade, Humphrey Sheil, 2010) cal avaluar primerament l'escenari com a criteri per decidir quin enfoc és més adient a l'hora d'escollir una arquitectura JEE.

En (Mark Cade, Humphrey Sheil, 2010) es proposen 2 models :

- EJB Centric (JavaEE) : Es fa ús d'un contenidor d'aplicacions, EJB de sessió, cues JMS
- WebCentric (Java SE+ Servlets+Frameworks)

En el següent apartat es descriu l'arquitectura del model *Web-centric*, que és el model que s'ha utilitzat en la aplicació LLOG. La descripció més detalla del model EJB cèntric queda fora de l'abast del projecte. Més informació sobre l'enfoc EJB-Centric es pot trobar en (Deepak, et al., 2003).

Web-Centric Application Structure

Una *Web-centric application* (aplicació centrada en la web) és una de les possibles configuracions que organitza components d'un *middleware*. Aquest tipus d'aplicació està molt estès, ja que resulta adequada quan l'ús de solucions més complexes representen una sobrecàrrega innecessària en temps d'execució i el desenvolupament. Normalment, aquestes aplicacions es divideixen en tres nivells (*tiers*) que agrupen la presentació, la lògica i la persistència de les dades. Aquestes es construeixen al voltant d'un servidor web que controla la interacció de l'usuari i realitza la lògica transaccional:

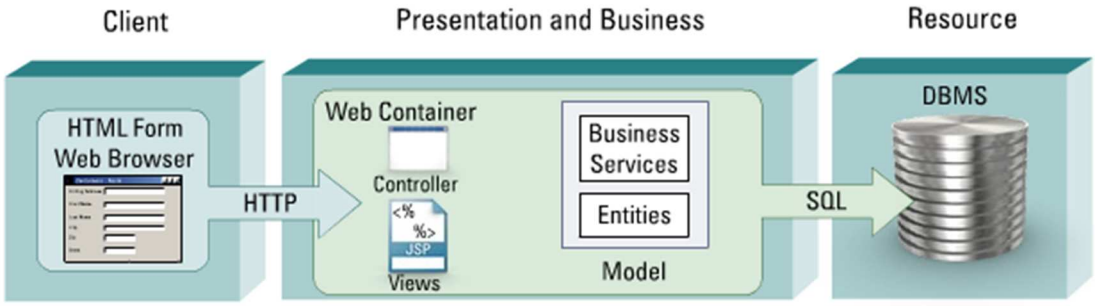


Figura 2-2 Webcentric application tiers

- Nivell Client
- Nivell mig o *Web tier*: s'ajunta el nivell de presentació i de lògica de negoci.
- Nivell de recursos o EIS (Enterprise Information System): base de dades o qualsevol altre repositori.

En el nivell mig o *web-tier* s'executaran els processos de negoci. Tot aquest conjunt es troba manejat dins el mateix contenidor web (Tomcat, etc...).

Arquitectura d'aplicacions Java EE

De la mateixa forma que la plataforma Java SE/EE conté una sèrie de capes (*layers*), les aplicacions en Java SE/EE es poden beneficiar de l'arquitectura en capes .

La majoria d'aplicacions web comparteixen un conjunt de requeriments comuns que la plataforma JEE no proveeix per ella mateixa. Una capa de software anomenada *application framework* pot complir aquests requeriments i pot ser compartida entre aplicacions (Deepak, et al., 2003).

Una *Web-tier application framework* s'estructura a la capa superior de la plataforma, i proporciona funcionalitats comunes com servir peticions, invocar mètodes del model, i seleccionar i assemblar les vistes. Les classes de *framework* i les interfícies són estructurals, són com els elements d'una construcció, sobre les quals es suporta l'aplicació.

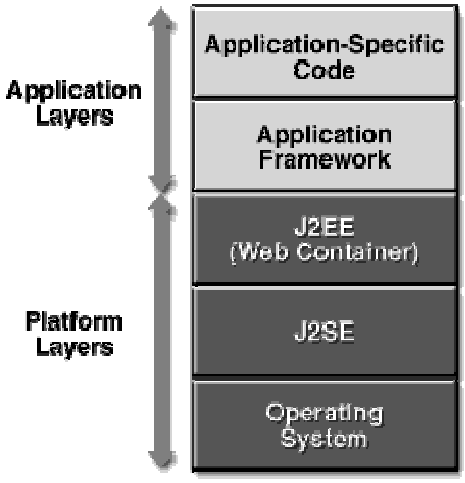


Figura 2-3 Capes d'aplicació

Els desenvolupadors d'aplicacions amplien, utilitzen o implementen les classes del marc de treball per realitzar funcions específiques de l'aplicació.

Per exemple, un marc de treball o *framework*, pot oferir una classe abstracta que un desenvolupador pot estendre per executar lògica de negoci en resposta a successos que es produeixen a l'aplicació. Una *Web-tier application framework* fa que les tecnologies d'una *Web-tier* siguin més senzilles d'usar, ajudant als desenvolupadors d'aplicacions a concentrar-se a la lògica de negoci.

2.7 Paradigma MVC: Model View Controller

Descompondre és una tècnica per dominar la complexitat coneguda des de temps remots : *dividi et impera* (divideix i venceràs).

Trygve Reenskaug fou el primer en descriure el paradigma MVC o Model View Controller al 1979 en un paper anomenat "applications programming in SmallTalk : How to User Model-View-Controller" (kayal, 2008).

El paradigma MVC divideix les aplicacions en tres capes (*layers*), model, vista i controlador, i desacobla les seves respectives responsabilitats. Cada capa maneja responsabilitats específiques i té responsabilitats específiques cap a les altres capes.

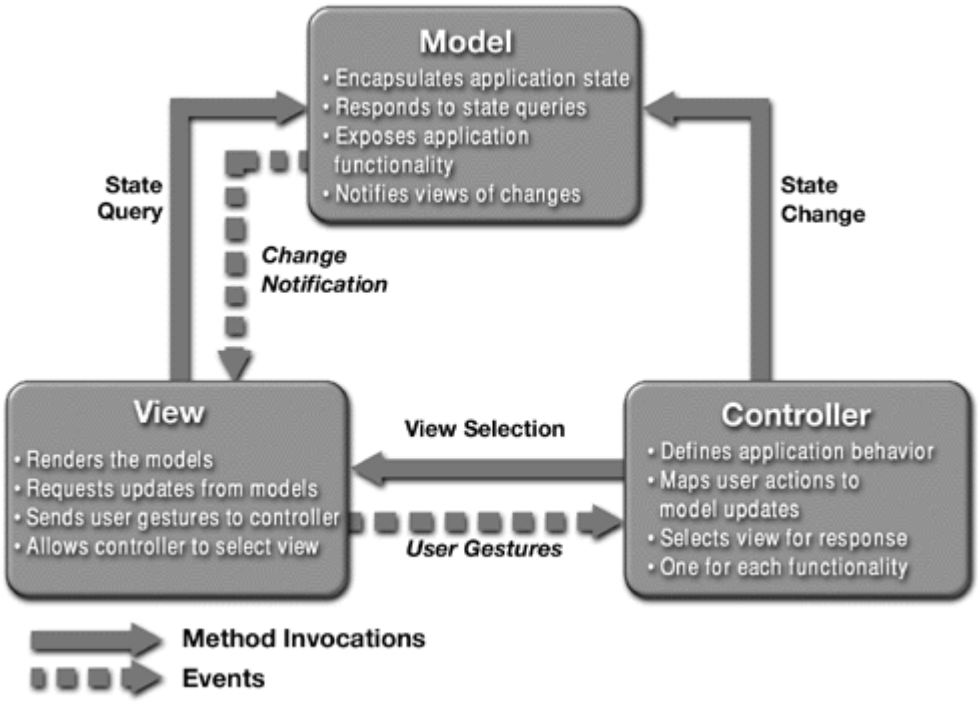


Figura 2-4 Diagrama conceptual arquitectura MVC

- Model: és l 'objecte que representa les dades del programa. Maneja les dades i controla totes les seves modificacions. El model serveix sovint com una aproximació software a la funcionalitat real de la aplicació. El model notifica a la Vista quan canvia i proporciona la capacitat per a que la Vista pregunti al Model sobre el seu estat. Aquesta capa és el cor del *software* de negoci (Evans, 2004).

- La Vista: serveix els continguts d'un model. Té accés a les dades del model i especifica com han de ser presentades totes aquestes dades. Actualitza la presentació de les dades quan canvia el model. La vista també dirigeix la informació d'entrada d'usuari al controlador

- El Controlador: defineix el comportament de l'aplicació. Envia les peticions de l'usuari i escull la Vista per a la presentació. Interpreta la

informació d'entrada de l'usuari i la *mapeja* en les accions que es realitzaran per al model (Figura 2-5 Controlador). En un client GUI *stand-alone*, l'entrada de l'usuari inclou clics de botons i seleccions de menús. En una aplicació Web, són les peticions GET i POST. Un controlador escull la vista per mostrar basant-se en les interaccions de l'usuari i en els resultats de les operacions del model. Una aplicació sol tenir un controlador per a cada grup de funcionalitats relacionades. Algunes aplicacions utilitzen un controlador diferent per a cada tipus de client, perquè la interacció i selecció de la vista pot variar entre diferents tipus de clients.

El controlador en una aplicació web realitza les següents funcions:

- Intercepta peticions HTTP des de un client
- Tradueix cada petició en una operació específica de negoci a portar a terme.
- Invoca l'operació de negoci i la delega a un gestor.
- Ajuda a seleccionar la següent vista a mostrar al client.
- Retorna la vista al client.

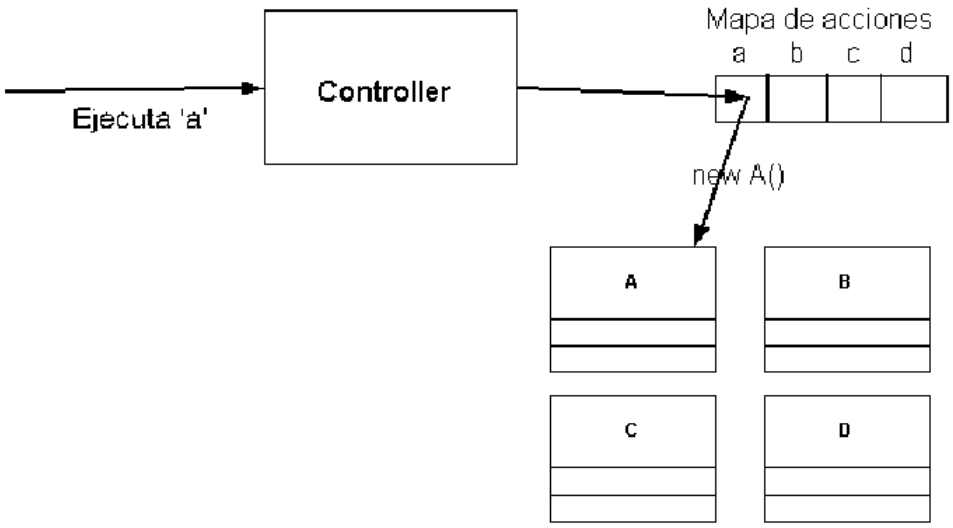


Figura 2-5 Controlador

Avantatges

Les avantatges que obtenim d'aquest model són òbvies, una separació total entre lògica de negoci i presentació. Separant les responsabilitats entre els objectes del model, vista i controlador es redueix la duplicació del codi, i facilita que les aplicacions siguin fàcils de mantenir. També ajuda a manejar les dades, tant si s'afegeixen noves fonts com si es produeixen canvis en la presentació, ja que la lògica del negoci es manté separada de les dades. És més senzill suportar nous tipus de clients, perquè no és necessari canviar la lògica amb l'addició de cada nou client. A això se li poden aplicar opcions com el multi llenguatge, diferents dissenys de presentació, etc... sense alterar la lògica de negoci.

- La separació de capes com la presentació, lògica de negoci, accés a dades és fonamental per al desenvolupament d'arquitectures consistents, reutilitzables i de més fàcil manteniment, fet que al final resulta en un estalvi de temps en desenvolupament en posteriors projectes.

2.8 Que és un marc de treball (framework)?

El terme marc de treball cal que sigui definit amb cura, comprendre el seu significat i entendre perquè afegeix valor en un desenvolupament software.

En la seva acceptació més senzilla, un marc de treball és un conjunt de classes e interfícies que cooperen pera solucionar un tipus específic de problema de software i té les següents característiques. (Cavaness, 2004)

Consta de múltiples classes i components, cada un dels quals pot proporcionar una abstracció d'algun concepte determinat.

- Defineix com aquestes abstraccions treballen conjuntament per a solucionar un problema.
- Els seus components són reutilitzables.
- Un marc de treball organitza patrons en un nivell superior.

Un bon marc hauria de proporcionar un comportament genèric que poguessin utilitzar molts tipus d'aplicacions diferents.

Existeixen diverses interpretacions del que constitueix un *framework*; algunes podrien considerar les classes e interfícies que proporciona el llenguatge Java com un entorn de treball, però aquestes són en realitat una llibreria. Existeix una diferència molt important entre una llibreria de software i un marc de treball. Una llibreria de software consta de funcions i rutines que la aplicació pot invocar. Un marc de treball, per el contrari, proporciona components genèrics i cooperatius que la seva aplicació amplia per proporcionar un conjunt determinat de funcions. Les parts per on el marc es pot ampliar s'anomenen punt d'extensió.

La figura següent mostra les diferències existents entre marc i llibreria software.

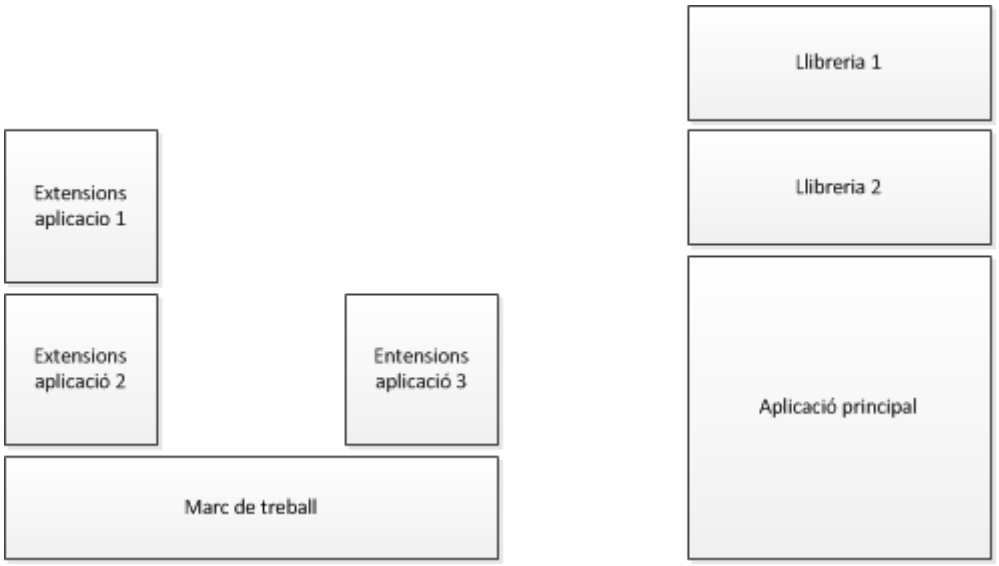


Figura 2-6 Marc i llibreria

Totes aquests avantatges tenen en un cost, per descomptat. Sempre existeix un menor control d'un disseny que s'ha adquirit vers un que hagi estat creat per un mateix. Alguns marcs han de ser comprats, encara que aquests en general esta lligats amb un conjunt d'eines de desenvolupament. Altre codi de la gent en el seu ús vol dir altres errades de la gent en el seu ús. De totes maneres la major part de projectes de desenvolupament troben que un *web-tier framework* milloren el disseny i la qualitat de la implementació.

2.9 Disseny guiat pel domini (Domain Driven Design)

La programació orientada a objectes és una eina molt potent degut a que es basa en un paradigma de modelatge i proveeix una implementació de les construccions del model. Java i altres eines permeten crear objectes i relacions directament anàlogues al model conceptual d'objectes.

El disseny guiat pel domini és un enfoc per al desenvolupament de software amb necessitats complexes mitjançant la connexió de la implementació amb un model en evolució dels conceptes clau del negoci. (Domain Language, Inc. and contributors, 2007).

Aquest terme va ser encunyat per Eric Evans, en el llibre "Domain Driven Design" (Evans, 2004). Aquesta forma de pensar complementa molt bé l'arquitectura MVC , especialment en la capa del model.

El disseny guiat o orientat pel domini (DDD) no és exactament una tecnologia, sinó una guia de pràctiques i terminologies per d'organitzar les aplicacions i estructurar el codi. DDD es basa abstraure i "naturalitzar" la forma en què s'interactua amb el sistema.

En el projecte de LLOG utilitzem alguns dels conceptes descrits en DDD per organitzar l'aplicació i estructurar el codi.

Terminologia utilitzada en el disseny guiat pel domini:

- Entitat: Un objecte definit fonamentalment per un fil de continuïtat i identitat (Evans, 2004). El concepte d'identitat adoptat per DDD va més enllà de les operacions dels llenguatges orientats a objectes que proveeixen una "identitat" a cada objecte. (per exemple l'operador == en java). El concepte d'identitat, des del punt de vista del DDD, és un atribut més metafísic, que emergeix del model i pren significat a través del domini. És aquest concepte el que guia el disseny de l'operació d'identitat entre objectes del domini. Per exemple, en l'aplicació LLOG, l'atribut "email" s'utilitza com a identificador de dues instàncies d'objecte de tipus Usuari, ja que dos usuaris no poden tenir el mateix email.

- Value Object: un objecte que no té una identitat. Normalment aquest tipus d'objectes són immutables.

- Repositori: Objectes utilitzats per emmagatzemar, recuperar, i eliminar objectes del domini de diverses implementacions d'emmagatzematge.

- Servei: Tipus d'objecte que realitza operacions que no pertanyen conceptualment a un objecte concret.

- *Aggregate*: clúster d'objectes del domini que pot ser tractat com a una única unitat.

- Factoria (Factory): Tipus d'objecte utilitzat per crear *Aggregates* complexes. No confondre amb el patró Factoria de GoF descrit en el següent capítol, el qual defineix un patró a un nivell més tècnic.

En l'aplicació de LLOG s'estructura el codi en objectes de tipus Entitats, Value Object, Repositoris i Serveis. En l' apartat 2.10 es descriu a nivell més tècnic els patrons utilitzats per implementar aquest tipus d'objectes.

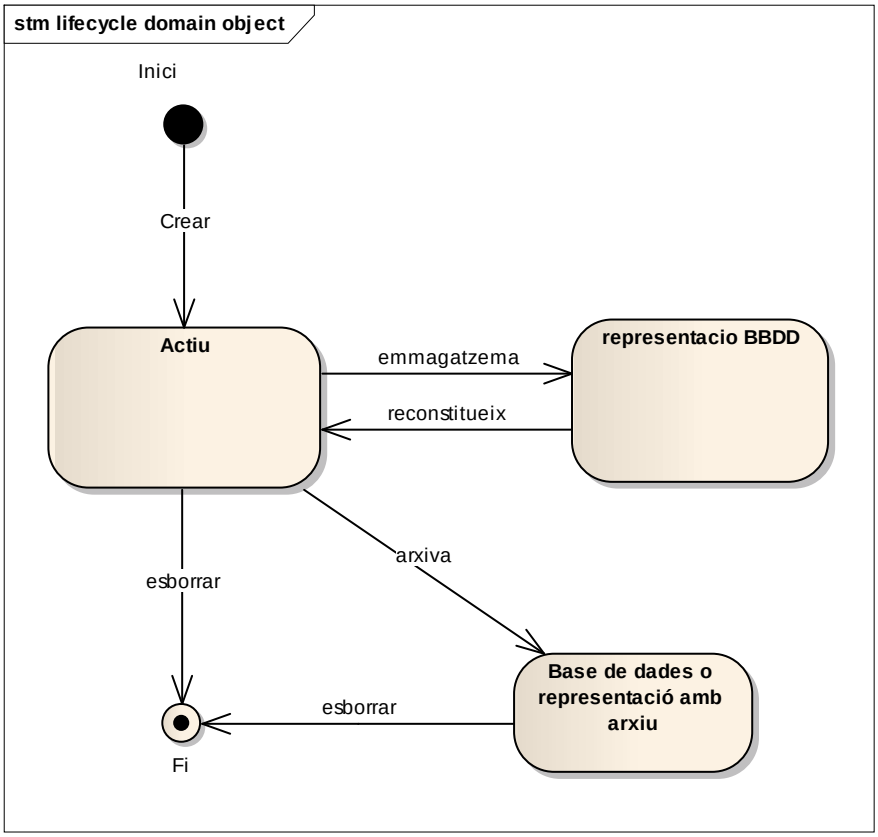


Figura 2-7 Cicle de vida d'un objecte del domini

2.10 Patrons de disseny en Arquitectura MVC + DDD

El disseny d'aplicacions es pot simplificar de manera considerable aplicant patrons de disseny Java EE. Els patrons de disseny Java van ser documentats en els Sun's Java Blueprints i posteriorment en el llibre Core J2EE Patterns (Deepak, et al., 2003).

Els patrons de disseny JEE consisteixen en solucions recurrents i documentades de problemes comuns de disseny d'aplicacions JavaEE. Moltes d'aquestes solucions es basen en patrons de disseny fonamentals orientats a objectes descrits en Design Patterns : Elements of Reusable Objected-Oriented (Gamma, Erich et al, 1995), així que es poden trobar solucions JavaEE que són aplicades al disseny d'aplicacions de software en general ((Deepak, et al., 2003), (kayal, 2008)).

Un patró JavaEE consisteix en un document basat en una plantilla que defineix les següents seccions:

- Context: Espai d'entorns sota els quals el patró existeix.
- Problema: Descriu el problema de disseny al que s'enfronta el desenvolupador.
- Causes: Llista les raons i motivacions que afecten al problema i a la solució. Aquesta llista subratlla la raó per la qual s'ha d'escollir utilitzar el patró en discussió en un problema de disseny i de la justificació de la seva utilització.

Controlador

Front Controller

El patró Front Controller, que es part dels patrons de disseny de Java Enterprise Edition (J2EE) descriu com s'hauria d'implementar un controlador de nivell web. Com que totes les peticions i respostes del client passen per el controlador, existeix un punt centralitzat de control per a la aplicació web. Aquest fet és una ajuda quan s'afegeixen noves funcionalitats. El codi que normalment es necessitaria situar en cada

pàgina JSP es pot situar en el servlet de controlador, que processa totes les peticions. El controlador també ajuda a desacoblar els components de presentació (vistes) de les operacions de negoci, afavorint al desenvolupament.

El patró de disseny Front Controller J2EE utilitza un sol controlador per canalitzar totes les peticions dels clients a través d'un punt central. Una de les moltes avantatges que aporta aquest patró a la funcionalitat de la aplicació, és que els serveis relacionats amb la seguretat, internacionalització i connexió estan concentrats en el controlador. Aquest fet permet aplicar coherentment aquestes funcions en totes les peticions. Quan el comportament d'aquests serveis necessita modificar-se, els canvis que afecten potencialment a tota la aplicació es tenen que realitzar només en una petita àrea del programa.

El component de controlador en una aplicació MVC té vàries responsabilitats, incloses rebre dades d'un client, invocar una operació de negoci i coordinar les vistes a retornar per el client. El controlador pot realitzar també moltes d'altres funcions però aquestes són algunes de les principals.

En el cas de l'aplicació LLOG s'utilitza el *framework* **Struts**, que utilitza un Servlet com a controlador.

La següent figura ens ajuda a il·lustrar el que hem comentat:

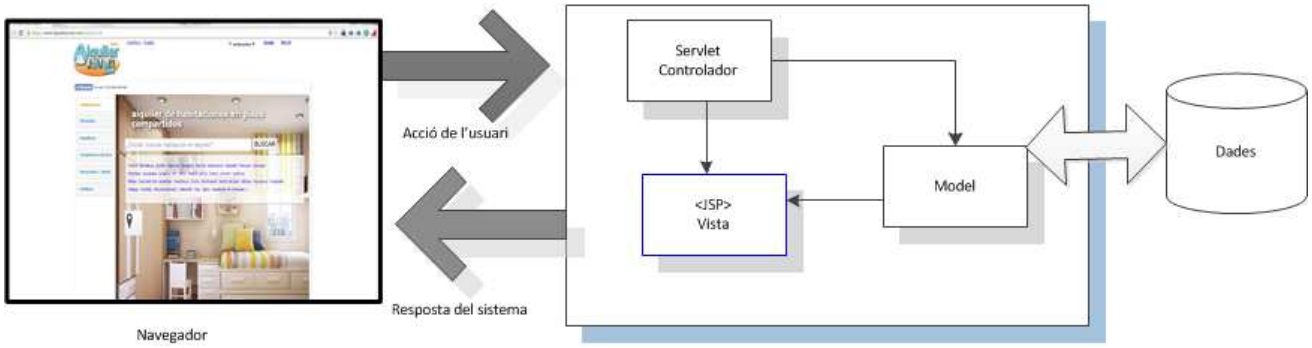


Figura 2-8 El marc de treball Struts utilitza un servlet com a controlador

Vista

Composite View i Templating (Plantilles)

Un requeriment típic en una aplicació és que les vistes de l'aplicació tenen una disposició comuna. Una plantilla (*template*) és un component específic que constitueix sub-vistes en una pàgina amb una disposició específica. Cada sub-vista, com un *banner*, un menú de navegació, o el cos del document és una component independent. Les vistes poden compartir una plantilla que tingui la mateixa disposició, ja que la plantilla controla el *layout*.

Per exemple, la següent figura ens mostra una disposició típica d'una pàgina creada a partir d'una plantilla. En aquesta podem diferenciar la part superior (per a un *banner*, per exemple), el menú de navegació lateral, el cos amb la informació que conté la pàgina i la part inferior

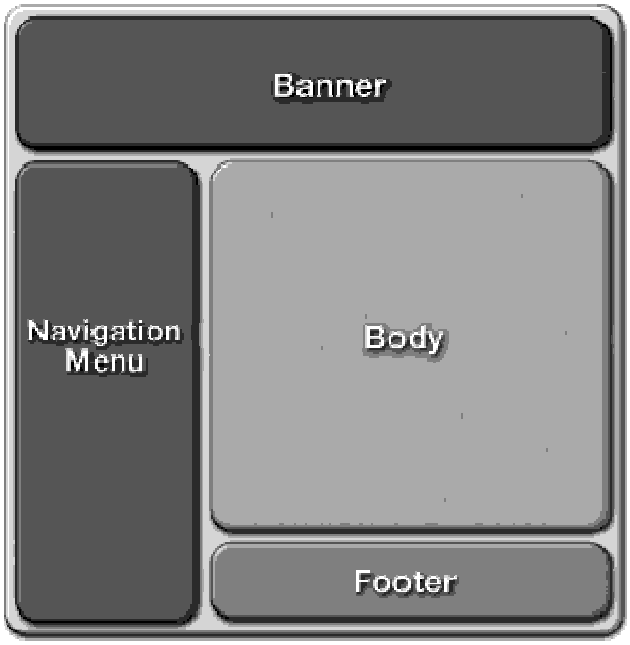


Figura 2-9 Una plantilla constitueix les diferents vistes en una disposició consistent

Utilitzar plantilles en el disseny d 'una aplicació centralitza el control sobre el total de la disposició de les pàgines, facilitant el manteniment i la

gestió. D'aquesta manera, canviant la disposició en la plantilla podem canviar la disposició de tota l'aplicació de cop. Més important si cal, les vistes individuals que s'utilitzen en la plantilla són referències en la plantilla, en comptes d' utilitzar el típic Copiar-Enganxar. De la mateixa forma, canviar una sub-vista significa canviar una únic arxiu font en comptes de canviar tots els arxius en el qual aquesta sub-vista apareix.

Per construir les pàgines aplicarem el patró de disseny anomenat CompositeView (Deepak, et al., 2003)

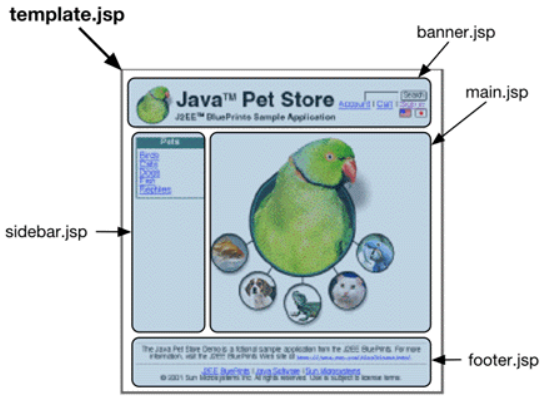


Figura 2-10 Exemple de composició de vista

Aplicarem l'estratègia *Custom Tag View Management Strategy*. Per més informació consultar
http://www.oracle.com/technetwork/java/compositeview-137722.html

En el cas de l'aplicació LLOG, s'utilitza el *framework* **Tiles** https://tiles.apache.org/. Veure l'apartat 4.4

Model

En el següent apartat es descriu el disseny dels objectes de tipus Repositori.

Repositoris: Solució Data Access Objects

Per dissenyar els repositoris de dades s'utilitza el Patró DAO (Data Access Object) (Deepak, et al., 2003). El patró de disseny Data Access Object (DAO) separa les interfícies a un recurs del sistema de l'estratègia subjacent utilitzada per accedir a aquest recurs.

Objectes d'accés a dades (DAO) de i repositoris juguen un paper molt important en DDD. L'objectiu d'un DAO és abstreure i encapsular tots els accessos a les dades i proporcionar una interfície. El DAO maneja la connexió amb la font de dades per a obtenir i emmagatzemar dades.

Una classe DAO proporciona una API abstracta per a manipular una font de dades. Aquesta API abstracta no fa referència a com està implementada la font de dades. El DAO simplement ha de saber com carregar-se ell mateix des de la font de dades basant-se en algun tipus d'informació d'identitat (una clau primària, primary key o un nom de fitxer, per exemple), com emmagatzemar-se ell mateix, etcètera.

Encapsulant les crides d'accés a les dades, els DAOs permeten a les dades adaptar-se a diferents esquemes o fins i tot a diferents tipus de bases de dades. DAOs per diferents esquemes i bases de dades poden compartir una interfície comú i permeten al assemblador de l'aplicació escollir l'objecte apropiat en temps d'assemblatge.

L'avantatge principal d'objectes d'accés a dades és que desacoblen l'usuari de l'objecte del mecanisme de programació que té accés a les dades subjacents. El DAO exposa la mateixa interfície per als seus clients, independentment de l'API que utilitza per accedir a les dades de EIS. Fins i tot els canvis en l'esquema o la funció d'especificació de l'EIS poden no afectar la interfície d'usuari de la DAO. Això vol dir que el model de programació de l'usuari no ha de canviar quan els mecanismes d'accés EIS canvien o hi ha modificacions en l'esquema de EIS.

Essencialment, el DAO actua com un adaptador entre el component i la font de dades (Patró adaptador (Gamma, Erich et al, 1995)).

Estructura

La següent figura mostra el diagrama de classes que representa les relacions per al patró DAO:

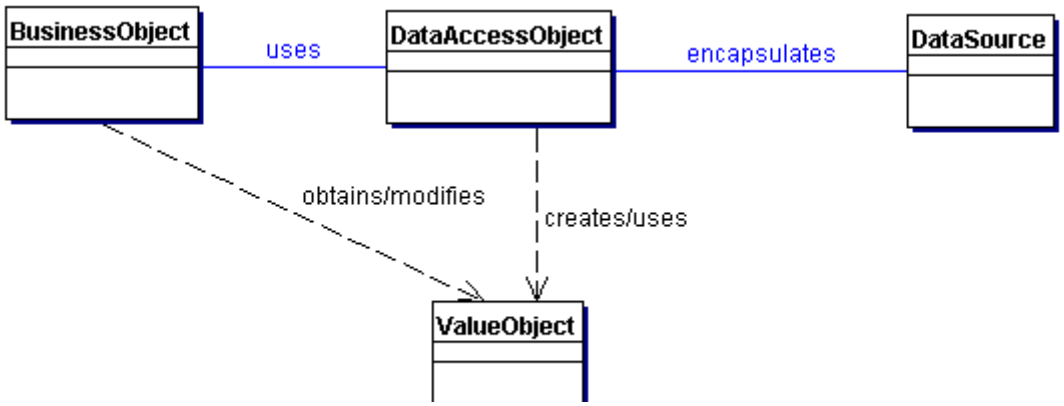


Figura 2-11 Diagrama de classes del patró DAO

Participants y Responsabilitats

La següent figura (Figura 5.4) mostra el diagrama de seqüència de la interacció entre els diferents participants en aquest patró:

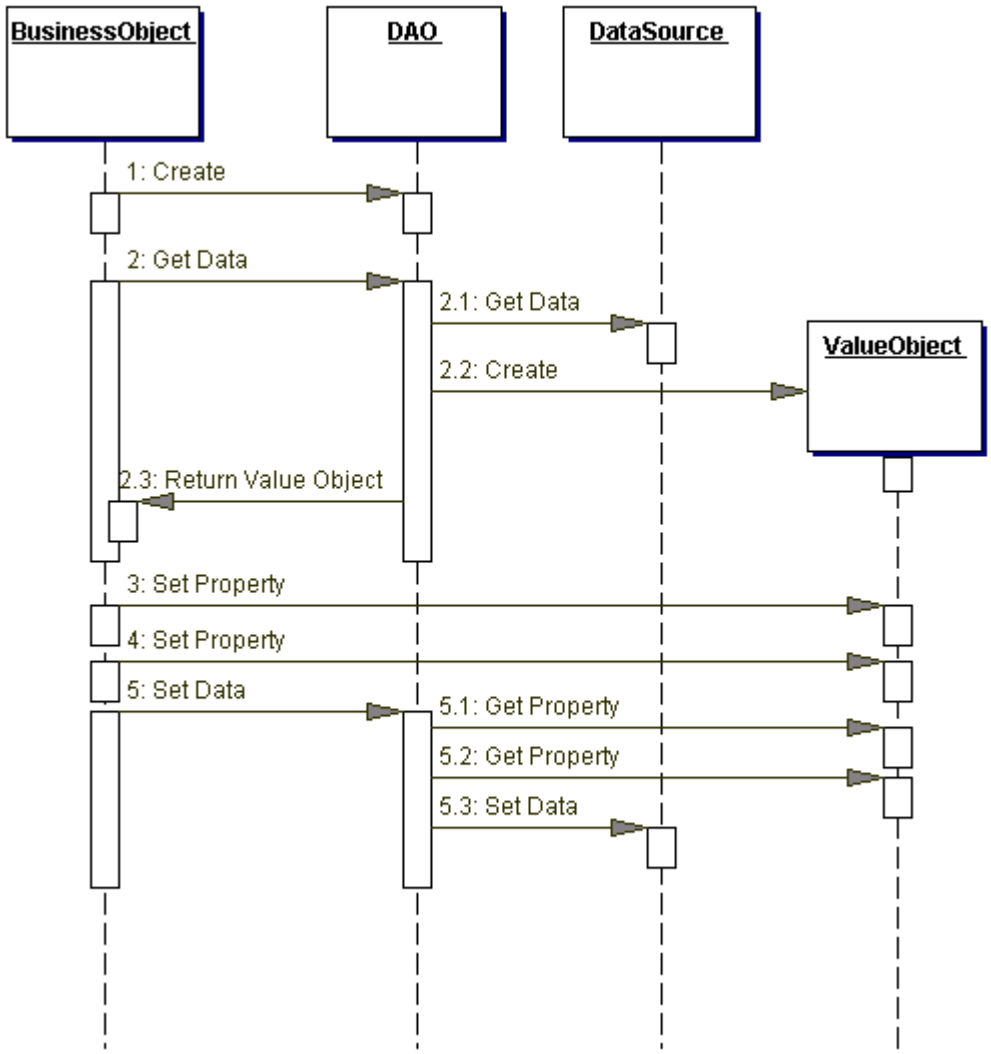


Figura 2-12 Seqüència d'interacció en el patró DAO

- BusinessObject: representa les dades del client. És l'objecte que requereix l'accés a la font de dades per a obtenir i emmagatzemar dades.

- DataAccessObject: és l'objecte principal d'aquest patró. DataAccessObject abstreu la implementació de l'accés a dades subjacent al BusinessObject per a permetre-li un accés transparent a la font de dades. El BusinessObject també delega les operacions de càrrega i emmagatzematge en el DataAccessObject.

- DataSource: Representa la implementació de la font de dades. Una font de dades podria ser una base de dades com una RDBMS, una OODBMS, un repositori XML, un fitxer pla, etc. També ho poden ser altres sistemes (mainframes/*legacy*), serveis (servei B2B o oficina de targetes de crèdit), o algun tipus de repositori (LDAP).

- Value Object o Transfer Object: Representa un objecte[1] utilitzat per al transport de dades. DataAccessObject podria utilitzar un Transfer Object per a retornar les dades al client. El DataAccessObject també podria rebre dades des del client en un Transfer Object per a actualitzar les dades en la font de dades.

Factoria per DAOs i Serveis

El patró DAO ens dóna lloc a un altre problema de disseny. Qui crea aquests DAOs? Si les crea algun objecte del domini, les responsabilitats dels objectes excedeixen la lògica pura de la aplicació i entren en altres qüestions, relacionades amb la connexió amb components de software externs (BBDD).

Si apliquem el principi de separació d'interessos (separar en mòduls o àrees diferents de manera de que cadascun tingui un propòsit cohesiu) no podem escollir un objecte del domini per crear els adaptadors (DAOs), ja que disminueix la cohesió.

Una alternativa en aquest cas, és flexibilitzar el patró DAO adaptant els patrons Abstract Factory [GoF] i Factory Method [GoF] (Gamma, Erich et al, 1995), en el que es defineix un objecte "Factoria" per crear els objectes (Larman, 2003).

Els objectes factoria tenen varies avantatges:

- Separen la responsabilitat de la creació complexa en objectes cohesius.
- Oculten lògica de creació complexa.
- Permeten introduir millores per millorar el rendiment . (objectes en memòria cau, etc..)

Factory Method

Quan l'emmagatzematge subjacent no està subjecte a canvis d'una implementació a una altra, aquesta estratègia es pot implementar utilitzant el patró Factory Method per a produir el nombre de DAOs que necessita l'aplicació. En la següent figura podem veure el diagrama de classes per a aquest cas:

---

[1] Sobre DTO: En l'aplicació LLOG no s'usen DTOs. En comptes de DTO, el controlador serveix els objectes del domini directament a la capa web.

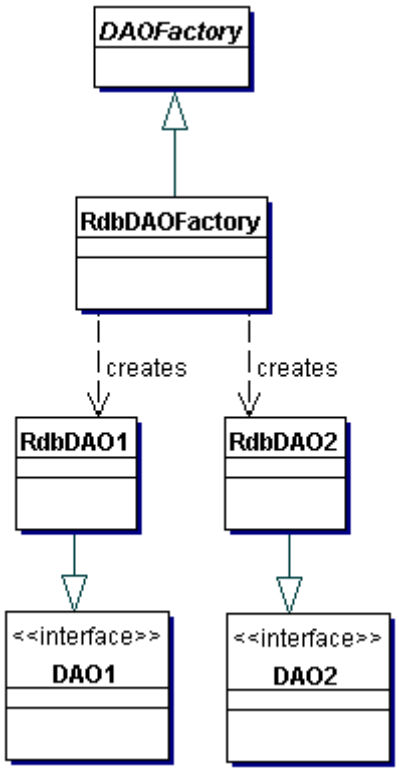


Figura 2-13 Patró Factory Method

Abstract Factory

Quan l'emmagatzematge subjacent si està subjecte a canvis d'una implementació a una altra, aquesta estratègia es podria implementar usant el patró Abstract Factory. Aquest patró al seu torn pot construir i utilitzar la implementació Factory Method. En aquest cas, aquesta estratègia proporciona un objecte factoria abstracta de DAOs (Abstract Factory) que pot construir diversos tipus de factories concretes de DAOs, cada factoria suporta un tipus diferent d'implementació de l'emmagatzematge persistent. Una vegada obtenim la factoria concreta de DAOs per a una implementació específica, la utilitzem per a produir els DAOs suportats i implementats en aquesta implementació.

En la següent figura (Figura 2-14) podem veure el diagrama de classes per a aquesta estratègia. Podem veure una factoria base de DAOs, que és una classe abstracta que estenen i implementen les diferents factories concretes de DAOs per a suportar l'accés específic a la implementació de l'emmagatzematge. El client pot obtenir una implementació de la factoria concreta del DAO com una `RdbDAOFactory` i utilitzar-la per a obtenir els DAOs concrets que funcionen en la implementació de l'emmagatzematge. Per exemple, el client pot obtenir una `RdbDAOFactory` i utilitzar-les per a

obtenir DAOs específic com `RdbCustomerDAO`, `RdbAccountDAO`, etc. Els DAOs poden estendre i implementar una classe base genèrica (mostrades com DAO1 i DAO2) que descrigui específicament els requeriments del DAO per a l'objecte de negoci que suporta. Cada DAO concret és responsable de connectar amb la font de dades i d'obtenir i manipular les dades per a l'objecte de negoci que suporta.

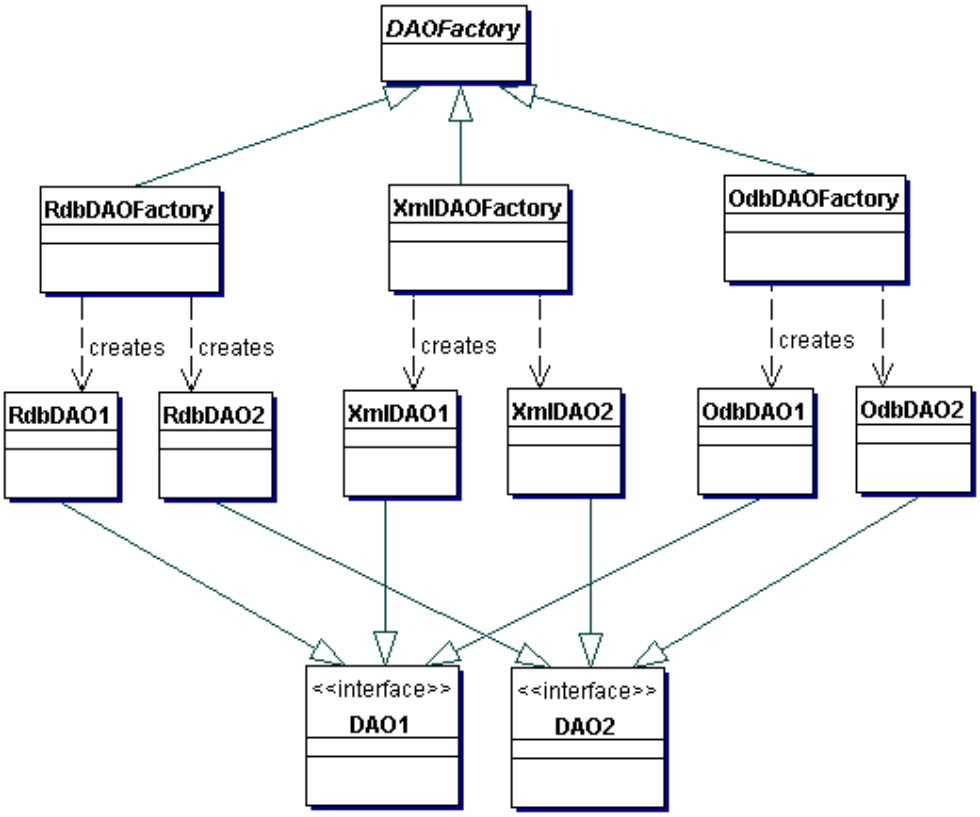


Figura 2-14 Diagrama de classes DAO per l'estratègia DAOFactory

En la següent Figura 2-15 podem veure el diagrama de seqüència per a aquesta estratègia:

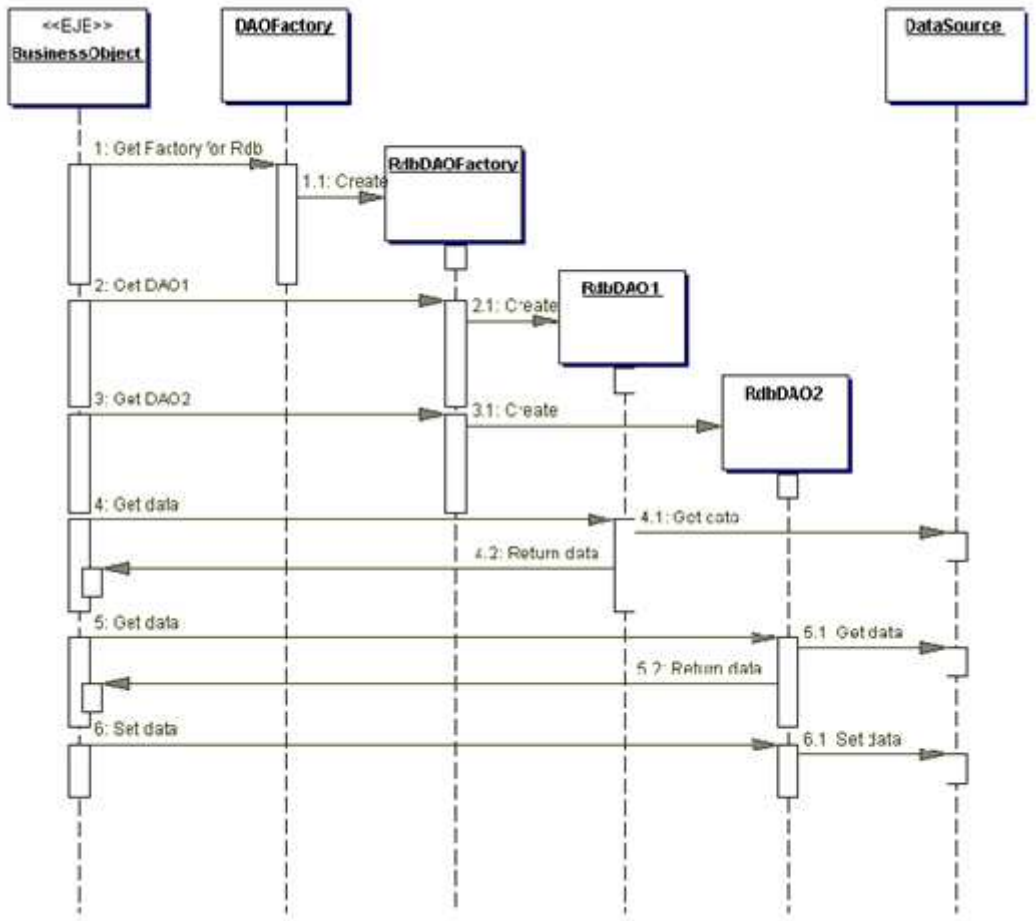


Figura 2-15 Diagrama de seqüència Abstract DAO Factory

Strategy pattern

Strategy pattern és un patró de disseny de comportament. S'utilitza quan tenim múltiples algoritmes per a una tasca específica i el client decideix en temps d'execució la implementació o estratègia actual a utilitzar.

En la secció 4.3 es descriu la implementació específica per a la aplicació LLOG d'aquests patrons de disseny.

3. Descripció del sistema

3.1 Visio General / Vista de Context

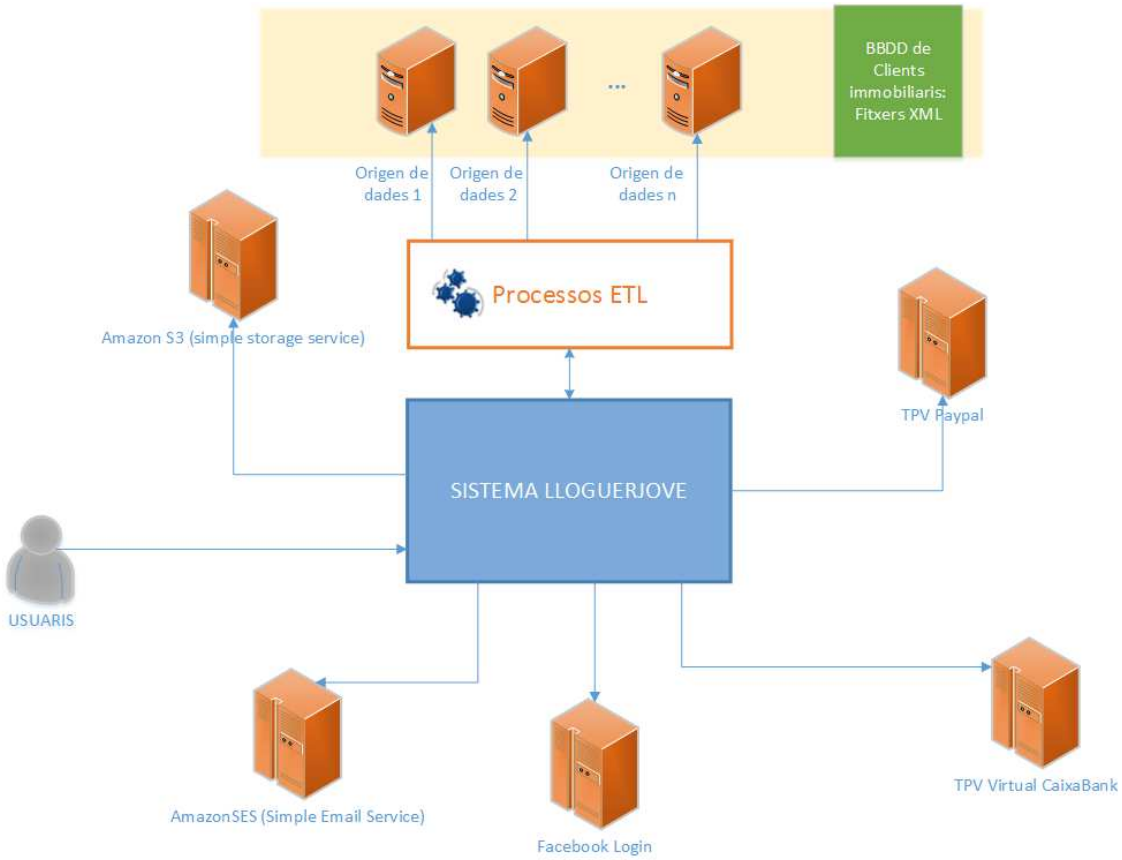


Figura 3-1 Vista de context de l'aplicació LLOG

El portal LloguerJove.com proveeix el servei de publicació i gestió d'anuncis de ofertes i demandes d'immobles i l'enviament de missatges entre els usuaris.

L'aplicació de Lloguerjove.com també proveeix el servei de ETL (extracció, transformació i càrrega) d' anuncis d'immobles d'agències i professionals (en format XML) i consolidar-los en la base de dades unificada de LloguerJove.com de forma automàtica.

L'aplicació de LloguerJove.com es connecta amb la passarel.la de pagament de CaixaBank per tal de processar el pagament dels diferents productes.

La metodologia utilitzada per al disseny i modelatge de l'aplicació s'ha basat en la tecnologia UML (*Unified Modelling Language*). Gràcies a aquesta desenvoluparem els diagrames més comuns i útils per a la posterior implementació del sistema.

3.2 Requeriments

Els requeriments són capacitats i condicions amb les quals ha de ser conforme el sistema, i més àmpliament, el projecte (Jacobson, I. Booch, G., and Rumbaugh, J., 1999). El principal objectiu dels requisits és trobar, comunicar i registrar el que es necessita de manera que tingui un significat clar tan per el client (en cas de realitzar un projecte per a un tercer) com per als membres de l'equip de desenvolupament.

En el cas de LLOG s'ha dividit els requeriments en dos tipus:

- funcionals: característiques de comportament del sistema.
- no funcionals: tota la resta de requisits (rendiment, seguretat, arquitectura, etc...).

Requeriments funcionals

L'aplicació de LLOG ha de complir amb els següents requisits:

-Registre: L' usuari ha de poder inscriure's al sistema. Un cop registrat rebrà les credencials per email i podrà accedir al sistema utilitzant aquestes credencials.

-Cercar i visualitzar anuncis: L' usuari podrà cercar anuncis de l'aplicació a través d'un cercador i visualitzar la fitxa dels anuncis.

-Un usuari que ha iniciat sessió amb el sistema podrà realitzar les següents funcionalitats:

- Gestió del compte.

- Gestió de missatges.
- Gestionar anuncis.
- Serveis de pagament.
- Gestió de reserves.

En l'apartat dels casos d'ús es descriu de manera més detallada cadascuna d'aquestes funcionalitats

Requeriments no funcionals

Especificació de l'arquitectura i l'entorn tecnològic

Tal i com s'ha descrit en l'apartat anterior , s'ha escollit un enfocament *web-centric* per implementar l'arquitectura de LLOG.

La decisió d'utilitzar un model *web-centric* es basa en que la funcionalitat continguda en el contenidor web per al control de concurrència, seguretat i gestió de sessions és completament suficient per complir amb els requeriments no funcionals.

Entorn tecnològic:

- Java Virtual Machine: Java 1.7
- L'aplicació LLOG es desplegarà en un Servlet Container (Tomcat 7)
- Base de dades relacional : MySQL 5.6

Transaccionalitat

L'aplicació LLOG es un sistema transaccional. Per tal de que l'aplicació sigui transaccional, ha de satisfer 4 característiques clau per les quals el sistema pot considerar-se segur:

- Atòmic: una transacció atòmica s'ha d'executar completament o bé no s'ha d'executar. Cada tasca dins de transacció atòmica s'ha d'executar sense error. Si alguna de les tasques falla, tota la transacció s'aborta i es desfan els canvis. Si totes les tasques s'executen correctament, la transacció és "commitejada", que significa que els canvis a les dades esdevenen permanents o durables.
- Consistent: la consistència fa referència la integritat del magatzem de dades subjacent. Imposa que s'escriguin només dades vàlides seguint les regles i restriccions d'integritat de la base dades.

- Aïllat: significa que s'ha de assegurar que una transacció s'executa sense interferència d'altres processos o transaccions.

- Durable: significa que tots els canvis a les dades realitzats durant el transcurs d'una transacció han de ser guardats en algun tipus de emmagatzematge físic abans que la transacció hagi completat, i un cop l'usuari ha estat notificat de que la transacció ha finalitzat correctament , aquesta persistirà, i es podrà recuperar l'estat del sistema en cas d'una eventual fallada.

El model transaccional del *framework* **Hibernate** serà reutilitzat intensivament.

Seguretat

El sistema ha d'estar assegurat, de manera que un client pugui fer pagaments en línia.

L'aplicació ha d'implementar comportaments de seguretat bàsiques:

- Autenticació: Identificar-se utilitzant almenys un nom d'usuari i una contrasenya

- Autorització: d'acord al seu perfil, el client en ha d'estar habilitat per realitzar o no algunes accions específiques .

- Per a l'accés a Internet, els següents requisits són obligatoris

- Confidencialitat: (pagaments amb targeta de crèdit) les dades sensibles han d'estar xifrades.

- Integritat de les dades: Les dades enviades a través de la xarxa no poden ser modificat per una capa o nivell.

- Auditoria: Cada acció sensible pot ser registrada.

- SSL *Segure socket Layer*, LLOG utilitza SSL amb Certificat de servidor utilitza la criptografia per xifrar el flux d'informació entre el client i el servidor.

Persistència

La persistència de dades s'abordarà fent d'ús d'una base de dades relacional.

Internationalització (i18n)

LLOG ha de ser capaç de gestionar diversos idiomes (almenys espanyol, català i anglès).

La capa de presentació ha de ser capaç de suportar i18n. Altres capes han de ser prou genèriques per treballar amb qualsevol context d'internacionalització.

Loading time / performance

El temps de càrrega de la pagina web haurà de minimitzar-se el màxim possible.

El criteri a seguit per al temps de resposta acceptable per a l'usuari de les peticions és el següent (Agileblaze, 2010):

- Els usuaris no es donen compte d'un retard de menys de 0,1 segons.
- Un retard de menys d'un segon no interromp el treball d'un usuari.
- Els usuaris esperaran la resposta si el retard és menor de 10 segons.
- Després de 10 segons, els usuaris perden l'atenció i comencen a realitzar alguna altra cosa.

Per tal de aconseguir els paràmetres anteriors i minimitzar el temps de càrrega dels documents HTML caldrà tenir en compte els següents paràmetres:

- Latència: servidors propers als usuaris (ping <80 ms).
- Temps de processament: el temps de generació del codi HTML ha ser mínim.
- Temps de càrrega de recursos addicionals: Minimització y compactació de javascripts i CSS.
- Minimitzar el nombre de crides HTTP. (màxim 45 objectes). Veure Figura 3-2 Diagrama en cascada del temps de càrrega de la página webFigura 3-2.

- CSS *spriting*: Compactació de arxius d'imatges png i gif en sprites globals.

Figura 3-2 Diagrama en cascada del temps de càrrega de la página web

Cerca Faced (Faceted search)

Faceted search proporciona una forma efectiva per permetre als usuaris refinar els resultats de cerca, desglossant contínuament cap avall fins que es trobin els elements desitjats. Els principals avantatges de les cerques *facet* es descriuen a continuació:

- *Retorn Superior*: els usuaris poden veure a simple vista un resum dels resultats de cerca i com aquests resultats es descomponen en diferents criteris.

- No hi ha sorpreses o carrerons sense sortida: els usuaris saben quants resultats coincideixen abans de fer clic. Valors amb zero coincidències es retiren normalment per reduir el soroll visual i eliminar la possibilitat de que l'usuari seleccionant accidentalment una restricció que conduiria a un resultat amb zero coincidències.

- No s'imposa una jerarquia de selecció: els usuaris són lliures d'afegir o eliminar les restriccions en qualsevol ordre.

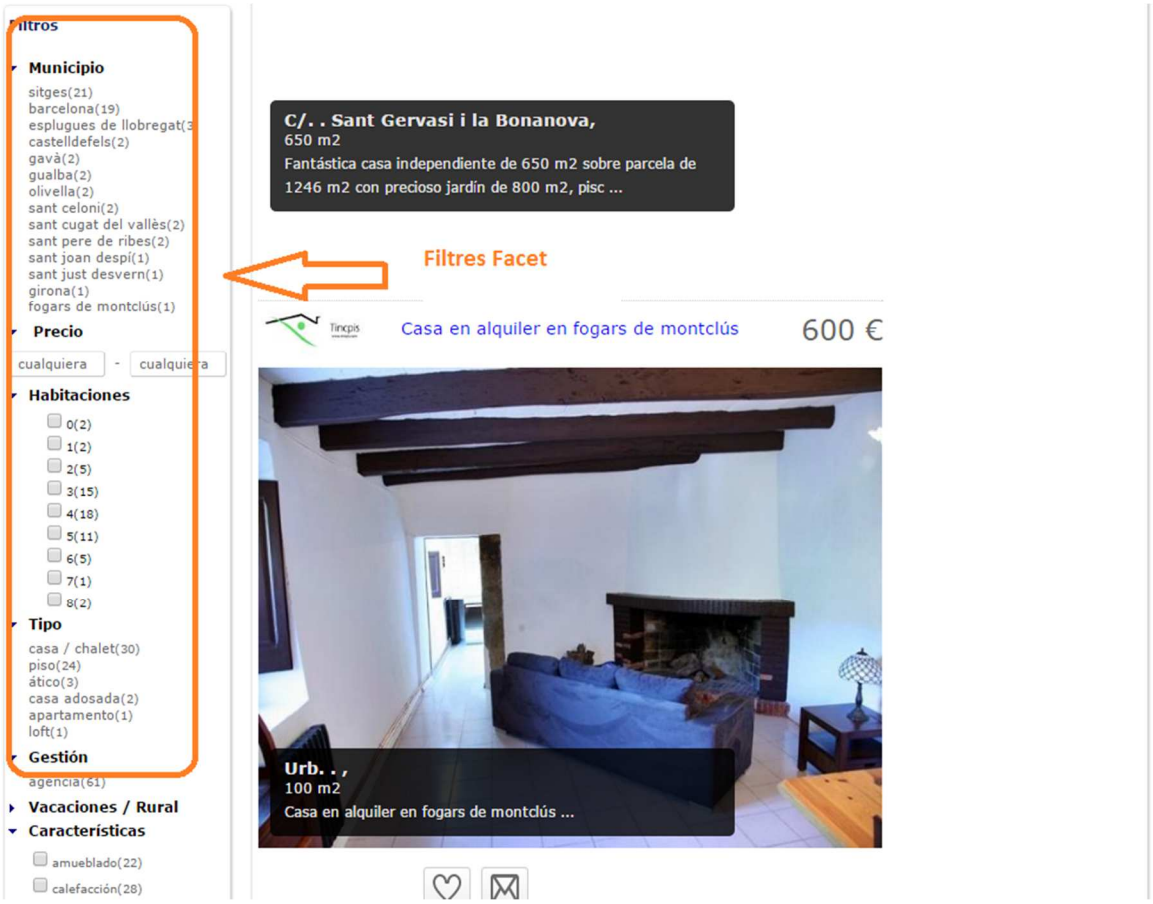


Figura 3-3 Filtres facet

SEO

SEO és el procés de millora de la visibilitat de l'aplicació en els resultats orgànics.

Friendy URLs

En l'àmbit del SEO és molt comú utilitzar la reescriptura de URL per tal de generar URL més curtes o més *amigables* per tal de millorar les posicions en els resultats de cerca dels cercadors d'internet (Google, Bing, Yahoo, etc... (Jones, 2007) (Powell, et al.), (Sharpened Productions)

URL *not friendy*:

http://localhost:80/es/SearchAction.do?reqCode=rentFlats&type=10&q=Barcelona&id_municipality=2265

Friendly URL:

http://localhost:80/es/pisos-alquiler-Barcelona-2265

Integracions

- Facebook Login: L'usuari es podrà autentificar a l'aplicació a través de Facebook utilitzant OAUTH2.

- Amazon S3: Tots els arxius de fotografies s'emmagatzemaran al servei d'emmagatzematge de dades d'Amazon S3, una infraestructura molt escalable I de costos molt baixos.

- Amazon SES: L'aplicació es connectarà al servei d' Amazon SES (Simple Email Service) per l'enviament d'emails de notificacions i d'alertes.

Definició dels usuaris / actors

Un actor és qualsevol cosa amb comportament, incloent el propi sistema que s'està estudiant, quan sol·licita els serveis d'altres sistemes (Larman, 2003). Els actors no són solament rols, també organitzacions, software i màquines. En l'aplicació LLOG s'han definit els següents actors:

Administrador

És l usuari que té permís sobre els altres usuaris, podrà afegir o esborrar altes de usuaris autentificats i dels anuncis publicats.

Usuari no registrat

Son els tipus de usuaris que accedeixen a la web per realitzar una consulta. Poden registrar-se al sistema (sign in).

Usuari registrat

Son els tipus de usuaris que es registren a la pàgina web amb les seves dades personals. Poden iniciar sessió o inscriure's (*log in*) amb les credencials que han rebut a través d'email.

Usuari autentificat

Són els usuaris registrats que han iniciat una sessió (*log in*). Poden realitzat tota una sèrie de operacions restringides dins la aplicació (publicar anuncis, gestionar reserves, etc...)

Sistema

Es tracta d' un algoritme que se encarregarà de realitzar cert tipus de funcions periòdiques como són enviar emails d'alertes als usuaris.

Diagrames de casos d'ús

El diagrama de casos d'ús descriu el llistat d'accions que pot realitzar cada tipus d'usuari. En les següents figures es mostren els casos d'ús per totes les funcionalitats que es desitja que puguin realitzar els usuaris juntament amb la seva relació d'actors.

Com podem observar en la Figura 3-4 l'actor usuari autentificat "estén" els casos d'ús de l'actor usuari registrat, i aquest a la vegada, "estén" els casos d'ús de l'actor usuari no registrat. S'utilitza aquesta notació per indicar que cada subactor "hereta" els casos d'ús del seu superactor.

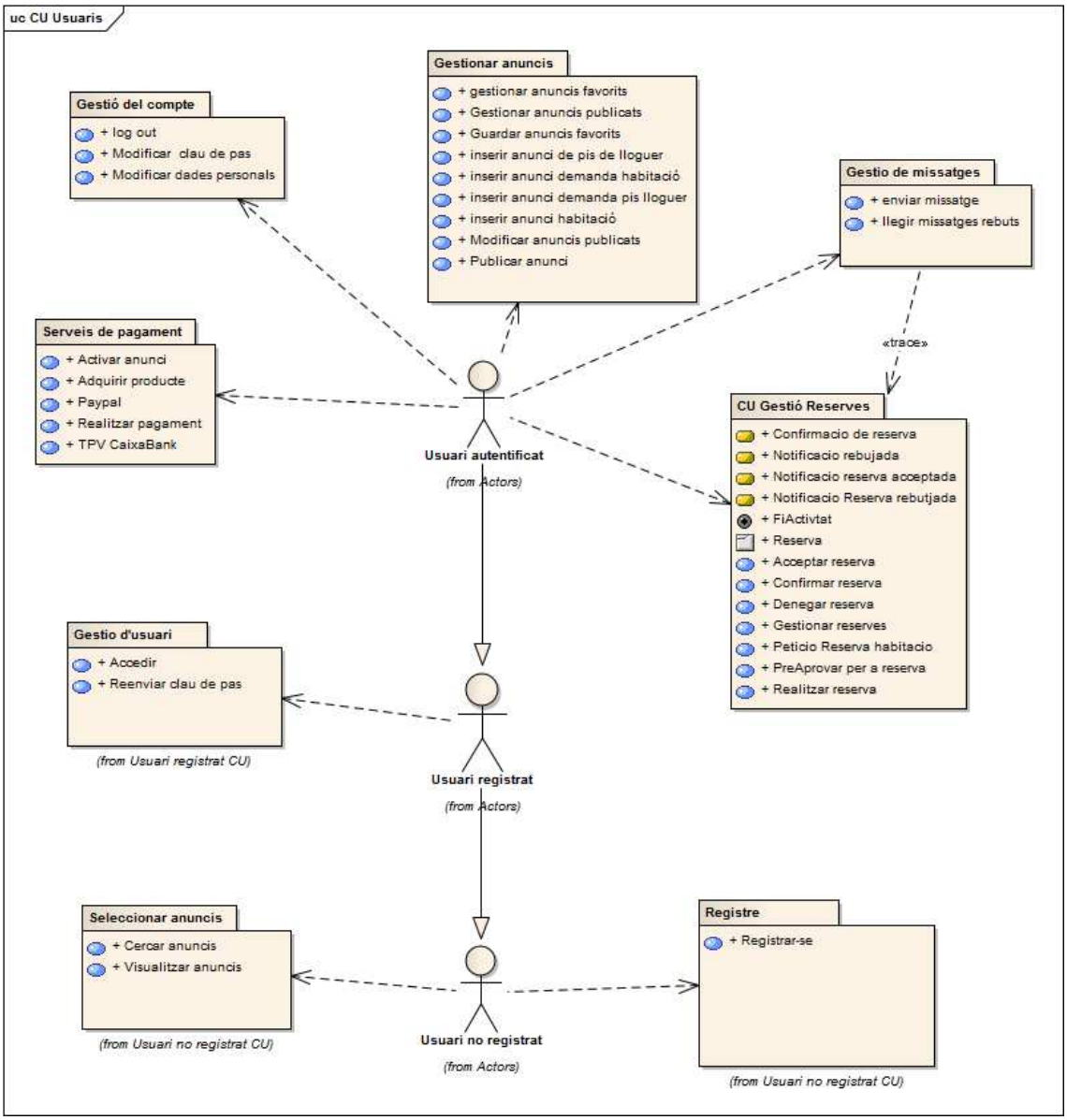


Figura 3-4 Relació d'actors i casos d'ús

Casos d'ús de l'usuari no registrat

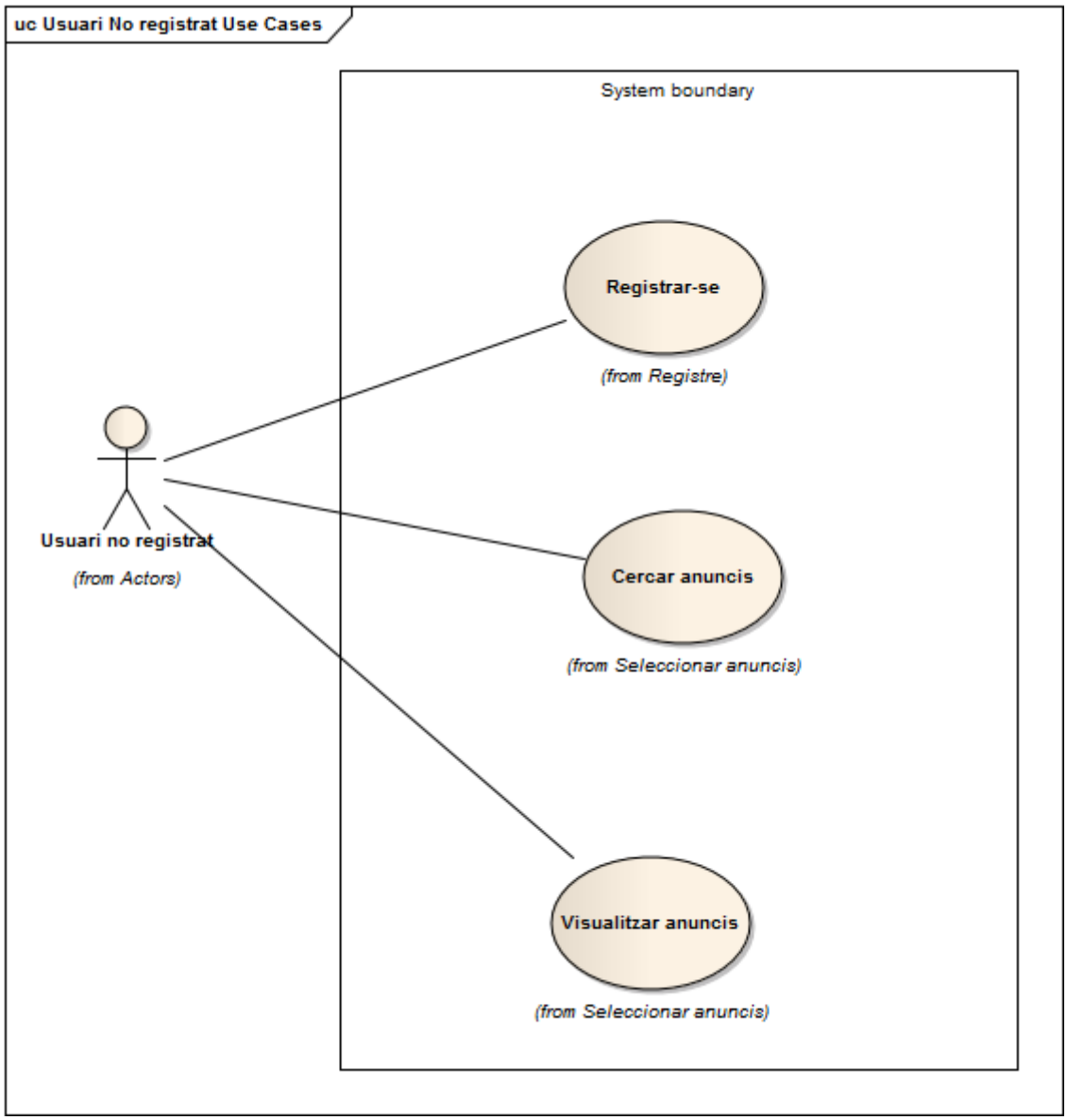


Figura 3-5 Casos d'ús de l'usuari no registrat

L'usuari no registrat pot registrar-se a la aplicació a través d'un formulari. També podrà cercar anuncis i visualitzar-los. Al finalitzar el procés de registre (usuari registrat) l'usuari rebrà un email amb les seves credencials, amb les quals, l'usuari podrà inscriure's (*sign in*) al sistema.

Casos d'ús de l'usuari registrat

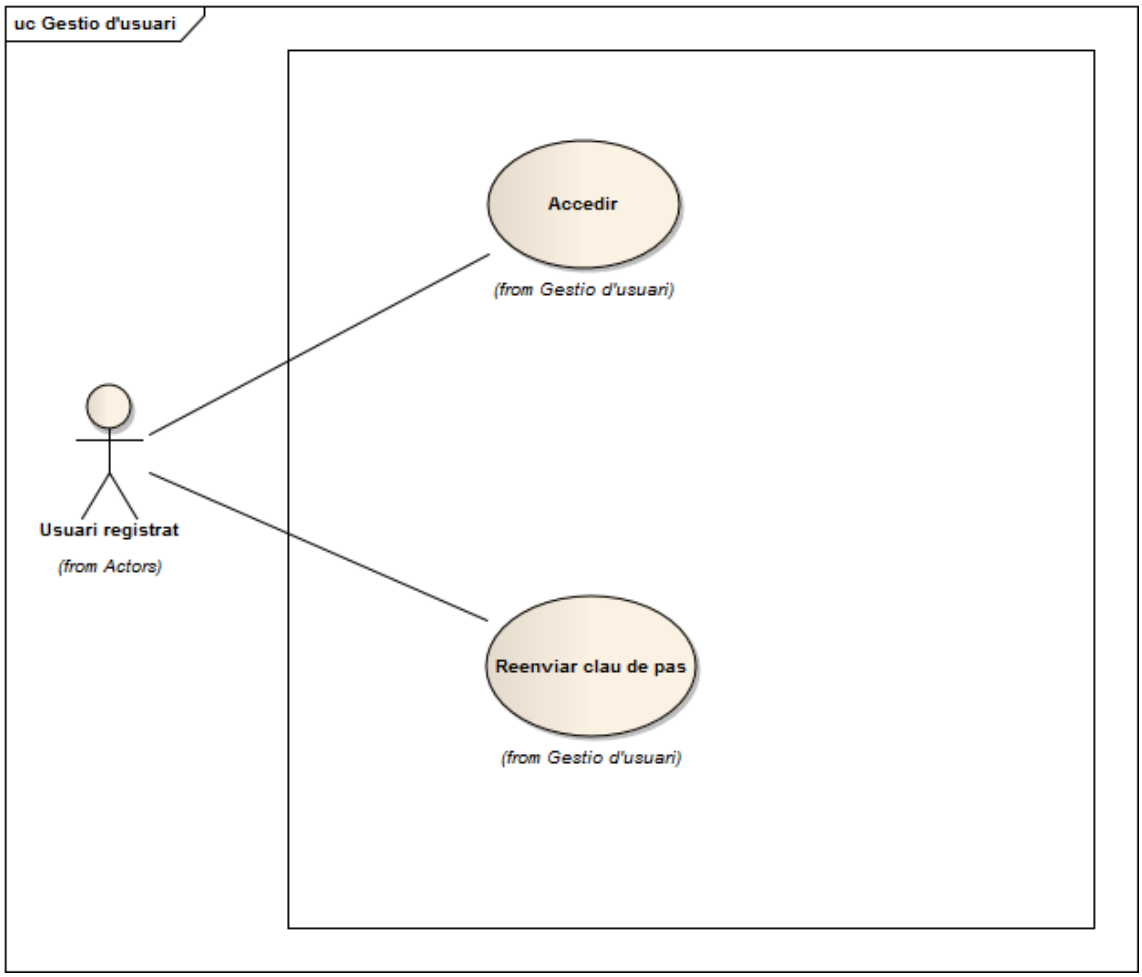


Figura 3-6 Cas d'ús d'usuari registrat

L'usuari registrat: Un usuari registrat al sistema podrà demanar un reenviament (via email) de les credencials introduint el seu email de registre. L'usuari registrat podrà accedir al sistema amb les seves credencials.

Casos d'us de l'usuari autentificat

Gestió de missatges

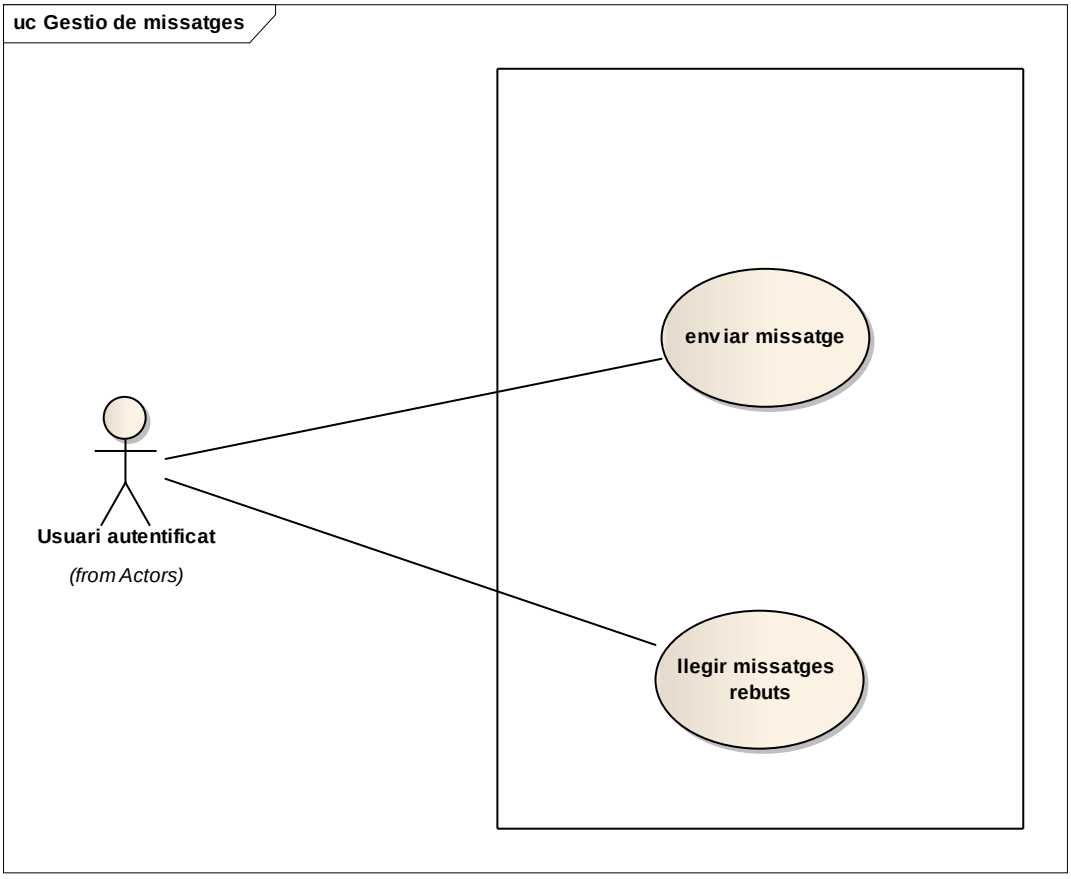


Figura 3-7 Cas d'ús de gestió de missatges

L'usuari pot enviar missatges a altres usuaris i també llegir els missatges rebuts.

Gestionar anuncis

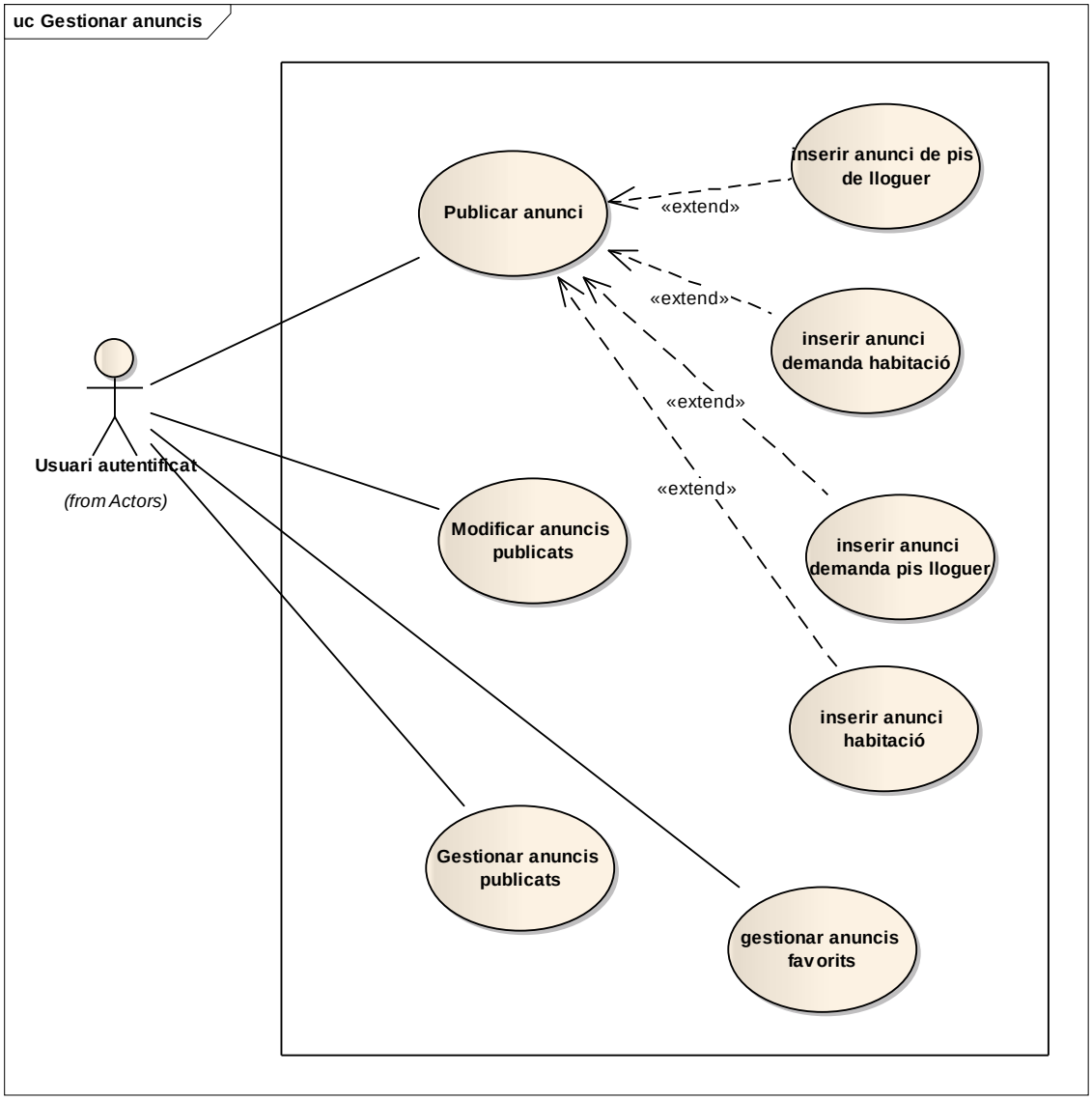


Figura 3-8 Casos d'ús de l'usuari autentificat: gestió d'anuncis

L'usuari autentificat podrà publicar anuncis (de diversos tipus: pis de lloguer, habitació en pis compartit, demanda de habitació o demanda de pis de lloguer).

L'usuari autentificat podrà llistar els anuncis publicats en el sistema i modificar-los a través d'un formulari d'edició.

També podrà accedir a un llistat dels anuncis afegits a "Favorits", i accedir a la fitxa d'aquests anuncis (Figura 3-8).

Gestió del compte

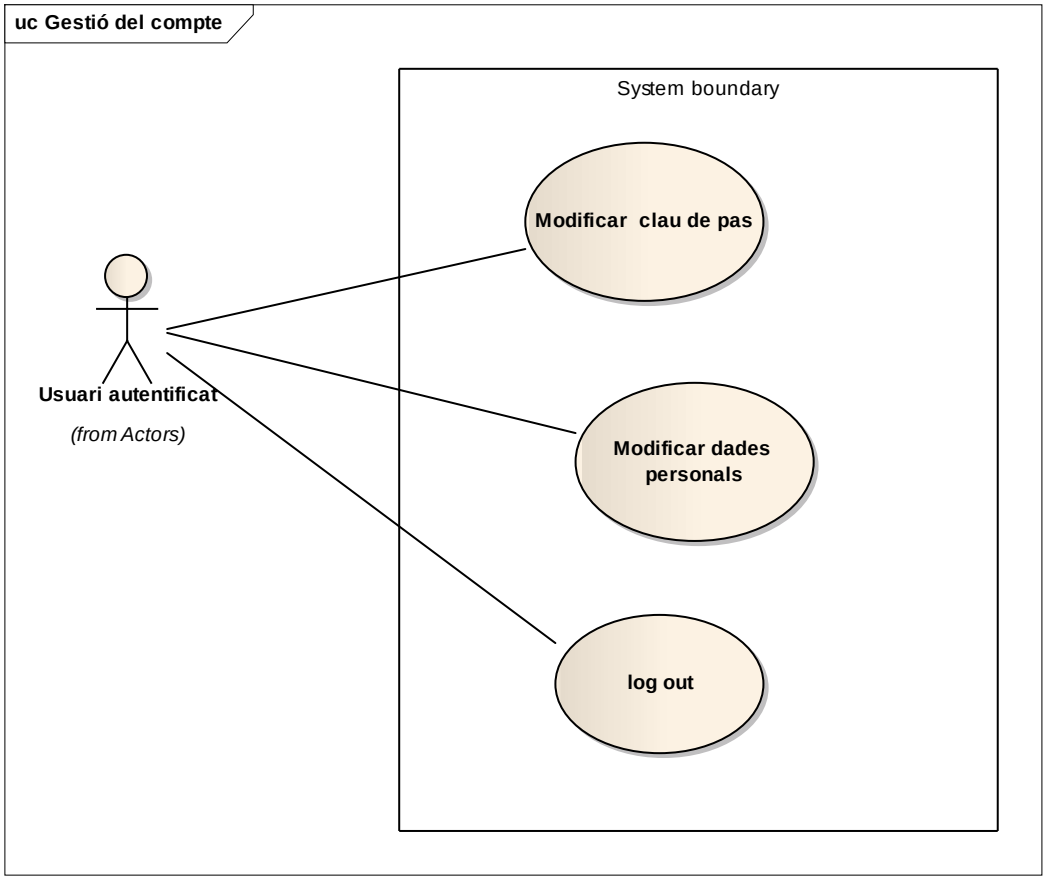


Figura 3-9 Casos d'ús de l'usuari autentificat: gestió del compte

L'usuari autentificat podrà modificar la clau de pas a través d'un formulari, com també modificar les seves dades personals.

L'usuari autentificat podrà canviar el seu email de registre i d'autentificació al sistema sempre i quan no n'hi hagi un d'igual registrat al sistema.

L'usuari autentificat podrà tancar la sessió al sistema.

Serveis de pagament

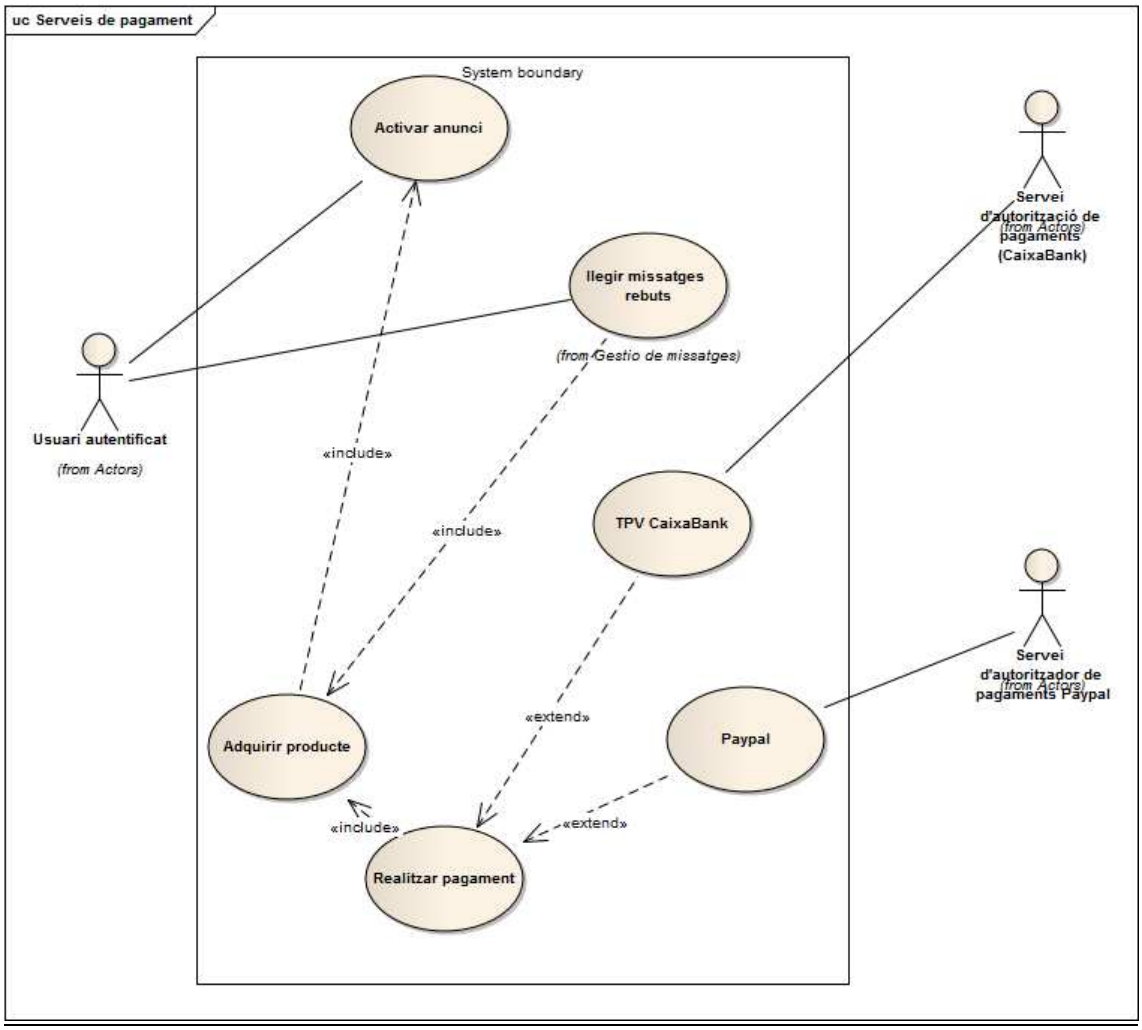


Figura 3-10 Casos d'ús usuari autentificat: Serveis de pagament

L'usuari podrà "comprar" determinats productes (activació d'anuncis, actualització de la data de publicació, *boosting* o millora del posicionament en llistats dels resultats de cerca, etc...) a través de la passarel·la de pagament de CaixaBank i Paypal.

Gestionar reserves

El sistema de LLOG permet als usuaris realitzar reserves, acceptar una reserva o denegar-la, i fins i tot pre-aprovar una reserva.

Per tal d'entendre el funcionament de les reserves, cal tenir en compte el següent diagrama de transició d'estats de la reserva:

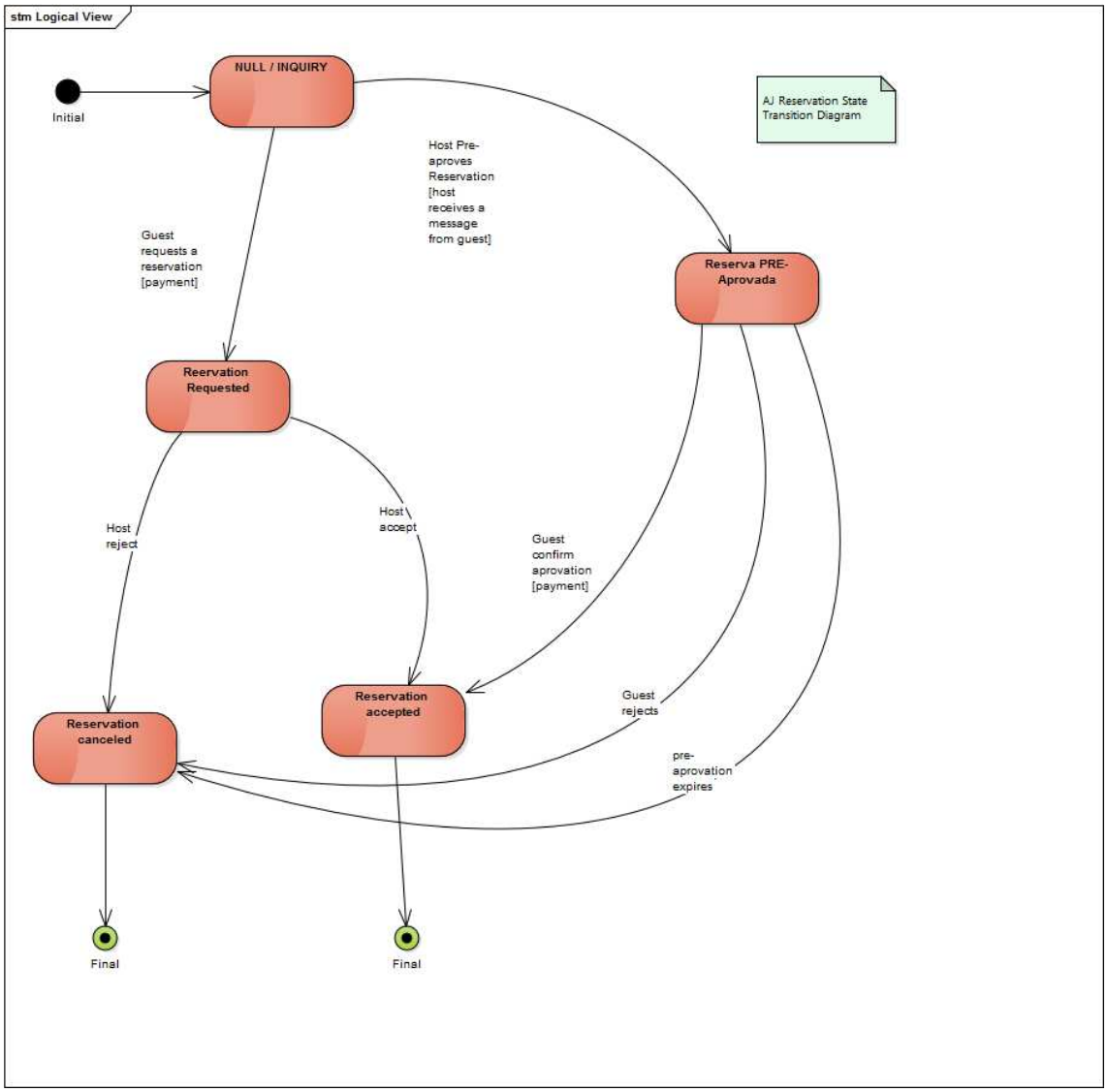


Figura 3-11 Diagrama de transició d'estats de la reserva

La reserva pot prendre 4 estats:

- *Requested*: quan un usuari ha realitzat una petició de reserva.
- *Pre-aproved*: quan un usuari pre-aprova un altre per reservar l'habitatge.
- *Canceled*: La petició de reserva ha estat denegada.
- *Accepted*: la petició de reserva ha estat acceptada.

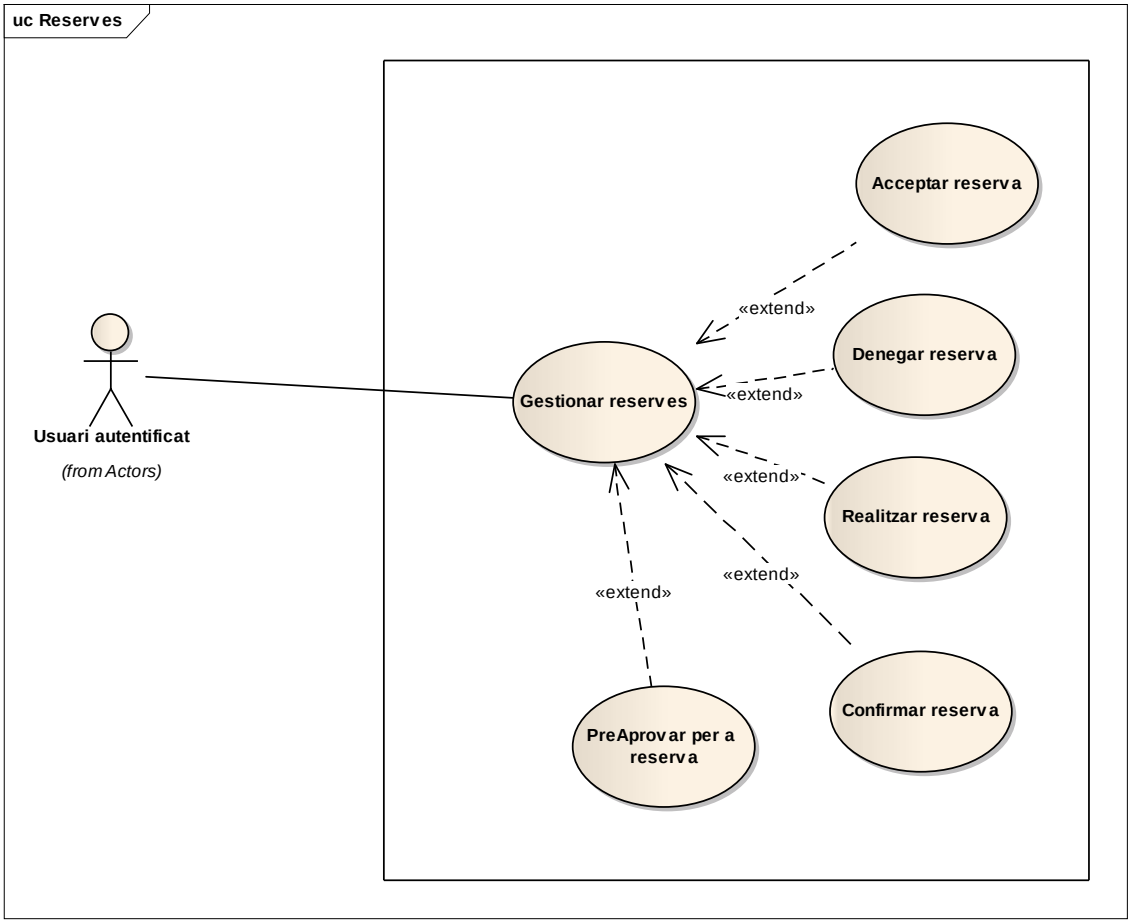


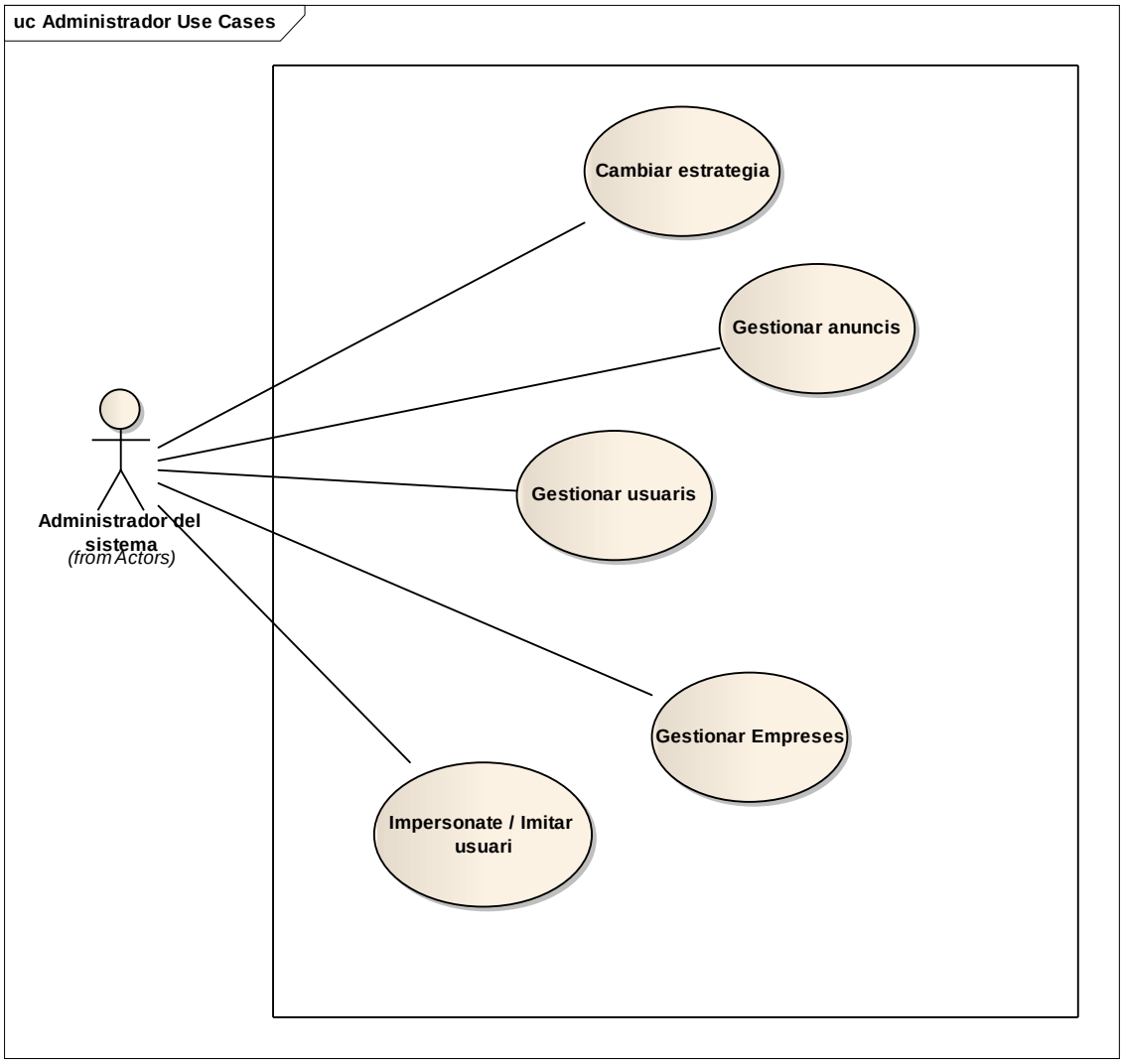
Figura 3-12 Casos d'ús usuari autentificat: Gestió de reserves

Els usuaris poden realitzar reserves sobre determinats anuncis (anuncis d'apartaments en lloguer a curt termini).

Si s'efectua una reserva, l'usuari amfitrió o propietari de l'anunci de lloguer a curt termini rebrà una notificació de petició de reserva. A través del portal LLOG podrà aprovar o denegar aquesta sol·licitud de reserva. En el cas de acceptar la reserva, el sistema carregarà a la targeta de crèdit de l'usuari *hoste* l'import de la reserva. El dia posterior del *checkout* el sistema transferirà l'import de la reserva al compte de l'usuari amfitrió.

L'usuari amfitrió també podrà pre-aprovar a un usuari que li hagi realitzat una consulta sobre una de els seves propietats. En aquest cas, si l'usuari host realitza la reserva, la sol·licitud s'aprovarà automàticament, i el sistema carregarà a la targeta de crèdit de l'usuari *hoste* l'import de la reserva. De la mateixa forma que en l'anterior procés, es transferirà l'import de la reserva el dia posterior del *checkout*.

Casos d'ús de l'administrador



L'usuari administrador podrà realitzar diverses tasques d'administració del sistema, tals com gestionar usuaris, gestionar empreses, gestionar anuncis i canviar l' estratègia en la publicació d'anuncis. L'estratègia en la publicació d'anuncis es refereix a la capacitat de canviar en temps real l'estratègia comercial del sistema:

- En determinats períodes de temps es pot canviar el model de negoci o ventes, habilitant o des-habilitant la publicació i activació d'anuncis sense requerir pagament, o l'ocultament de les dades de contacte dels propietaris dels anuncis.

Casos d'ús de l'actor processos

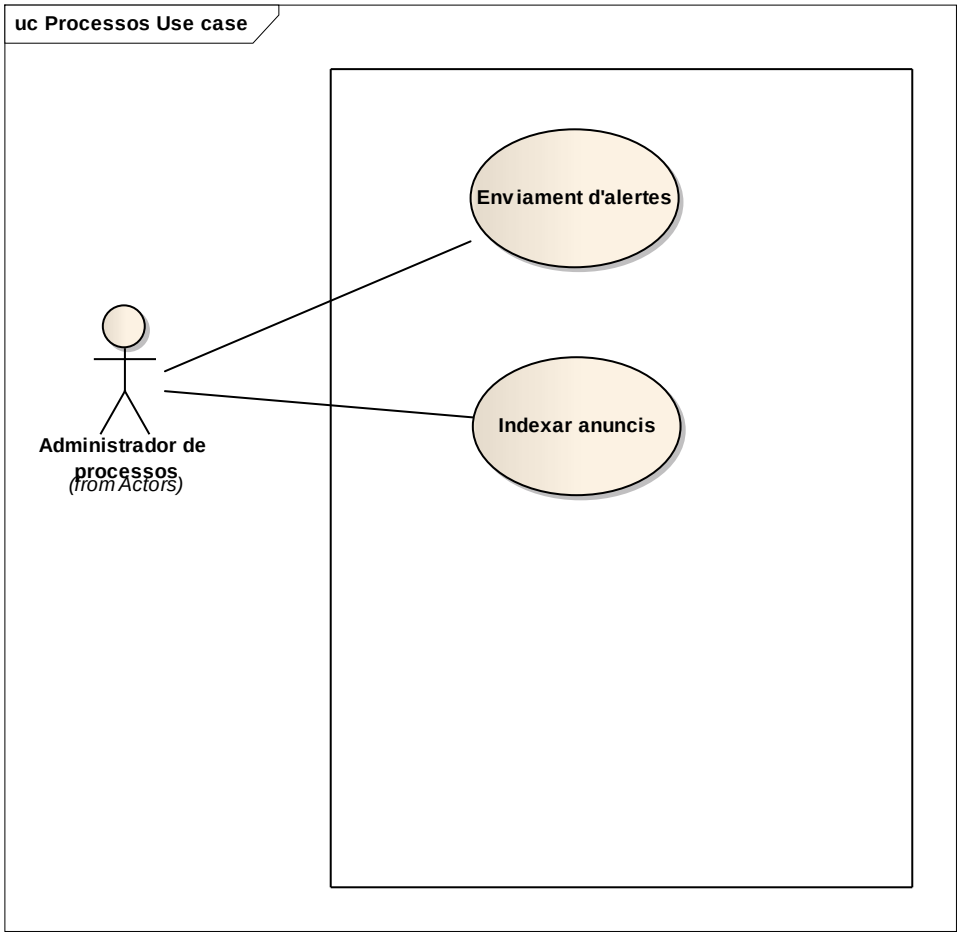


Figura 3-13 Casos d'ús de l'actor processos

L'actor administrador de processos representa l'administrador de processos programats (cron). Aquest sistema realitza de forma periòdica l'execució dels processos d'indexació dels anuncis en el motor **Lucene** a través de **Solr** i l'execució del procés d'enviament d'alertes i notificacions als usuaris.

3.3 Model conceptual

En el desenvolupament de software, el terme 'model' s'utilitza tan per la representació lògica de les entitats del món real, com la creació física de classes e interfícies que poden utilitzar els programes. Tot i així, el primer pas sempre hauria de portar a terme un anàlisis exhaustiu de l'àmbit del problema. Una cop es completen els casos d'ús, el següent pas és desenvolupar un model conceptual.

En el model conceptual o model del domini trobem la representació dels conceptes (objectes) significatius que trobem en el domini del projecte.

El model del domini mostra una vista parcial o abstracció, *no components de software o objectes de software amb responsabilitats*.

Principalment, el model conceptual ens mostra les principals objectes que hi participen, les associacions entre aquests i també s'hi mostren els atributs més significatius que tenen.

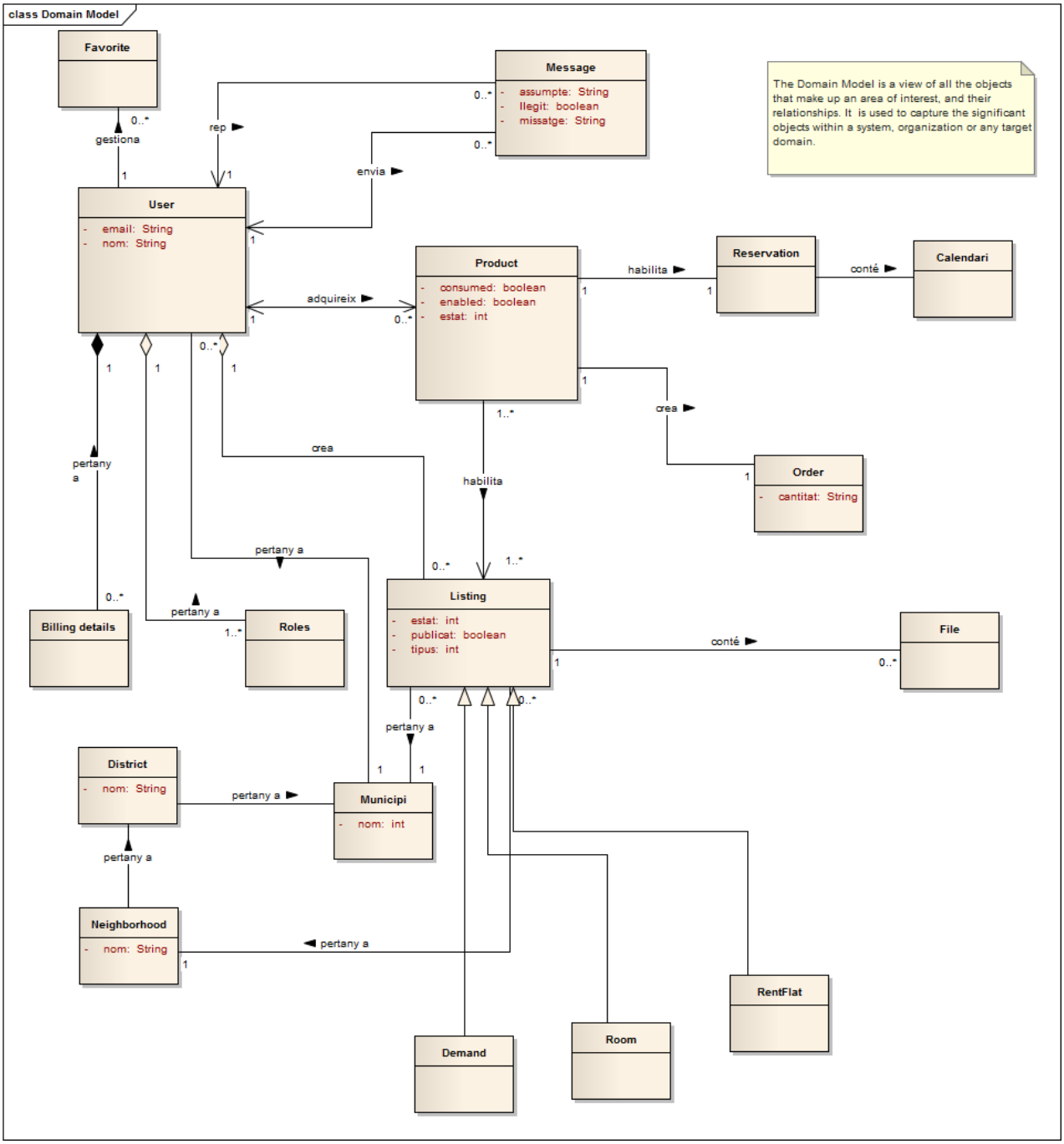


Figura 3-14 El model conceptual de LLOG

4. Disseny i implementació

4.1 Disseny del sistema

Tal i com s'ha descrit en l'apartat 2.7 l'aplicació LLOG fa ús del patró d'arquitectura MVC. A continuació es descriuen les interfícies entre els seus components interns i una descripció de les tecnologies i *frameworks* utilitzats en cada una de les capes (Model-Vista-Controlador).

4.2 Components de Controlador

Marc de treball Struts

El marc de treball (o *framework*) de codi obert **Struts** es va crear per permetre que els desenvolupadors poguessin crear aplicacions web basant-se en les tecnologies Java Servlet i JavaServer Pages(JSP). **Struts** proporciona als desenvolupadors una infrastructura unificada sobre la qual es poden basar les aplicacions de internet. Utilitzar **Struts** com la base, permet als desenvolupadors centrar-se en la creació de la aplicació de negocis en lloc de la infraestructura.

El marc de treball **Struts** va ser creat per **Craig R.McClanahan** i donat a la fundació **Apache Software Fundation** (ASF) a l'any 2000. L'entorn de treball **Struts** és un dels projectes Apache més coneguts i de més èxit.

El marc de treball **Struts** utilitza un *servlet* per processar peticions entrants. De totes formes, es basa en molts d'altres components que són part del domini del controlador per a poder portar a terme les seves responsabilitats.

A continuació s'exposa un breu resum dels components de controlador que utilitza **Struts** i a continuació es mostraran les classes específiques dissenyades per a aquesta aplicació.

El mecanisme del controlador

En l'entorn de treball **Struts**, varis components són els responsables de les obligacions del controlador. La figura següent n'és un diagrama de classe d'aquests components que comparteixen part de la responsabilitat del controlador.

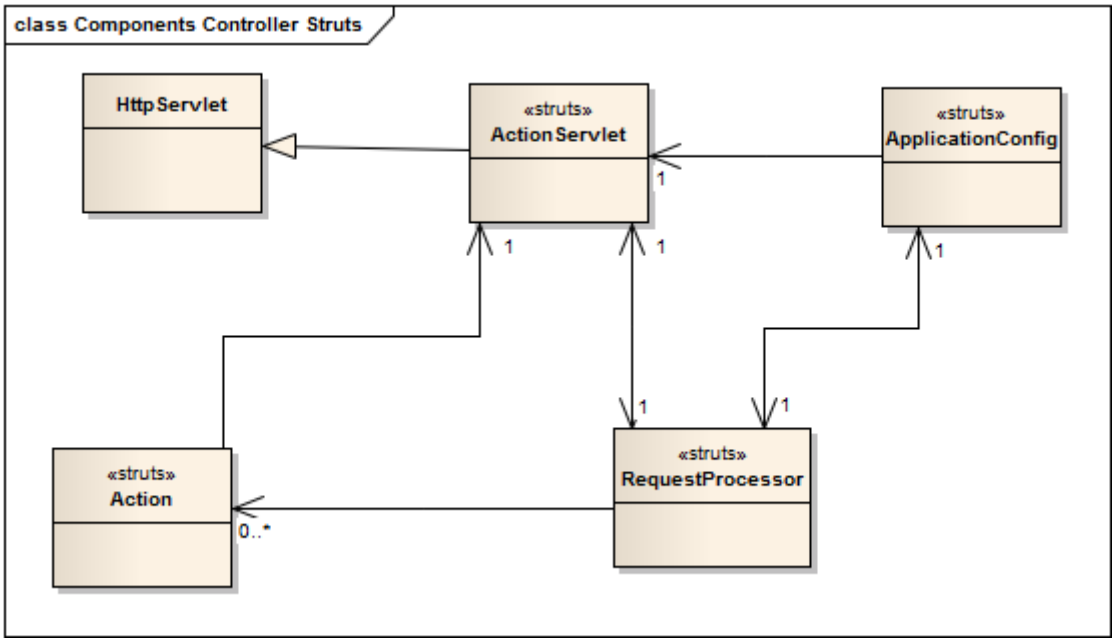


Figura 4-1 Components del controlador Struts

La classe ActionServlet

La classe `org.apache.struts.action.ActionServlet` actua com un interceptor per a una aplicació **Struts**. Totes les peticions del nivell de client han de passar per la `ActionServlet` abans de continuar a algun altre lloc en la aplicació. Quan una instància de `ActionServlet` rep una `HttpRequest`, tant a través del mètode `doGet` o `doPost`, s'invoca el mètode `process()` per a gestionar la petició.

La classe RequestProcessor

La classe `org.apache.struts.RequestProcessor` es va afegir al marc per a permetre que els desenvolupadors personalitzessin el comportament de gestió de petició per a una aplicació. La classe `RequestProcessor` conté

molts mètodes que es poden anul·lar si cal modificar la funcionalitat per defecte.

La classe Action

La classe `org.apache.struts.action.Action` és el cor del marc. És el pont entre una petició del client i una operació de negoci. Tota classe `Action` està dissenyada per a portar a terme una sola operació de negoci en nom de un client. Això no vol dir que `Action` només pugui duu a terme una sola tasca, ben al contrari, la tasca que realitza ha d' estar centrada i agrupada entorn a una única funcionalitat.

La classe ActionForward

La classe `ActionForward` representa una abstracció lògica d'un recurs web. Aquest recurs és normalment una pàgina JSP o un servlet de Java.

`ActionFoward` és un embolcall per a un recurs, de manera que hi hagi el menor acoblament possible de la aplicació al recurs físic.

El recurs físic s'especifica només en l'arxiu de configuració (com els atributs name, path i redirect de l'element forward).

En l'aplicació LLOG s'utilitza la classe `CustomRequestProcessor` (Figura 4-2), que és una classe que estén el comportament de la classe `RequestProcessor` per tal de que es pugui realitzar la validació a nivell de mètode de les classes de tipus DispatchAction[2].

Veure Annex: Paràmetres de inicialització i configuració del servlet de LLOG.

[2] Una Action abstracta que redirecciona a un mètode públic anomenat de la mateixa manera que un paràmetre del request, el nom del qual està especificat per la propietat del paràmetre de la ActionMapping corresponent. Aquesta Action és útil per combinar moltes accions similars en una sola classe d'acció, amb la finalitat de simplificar el disseny de l'aplicació.

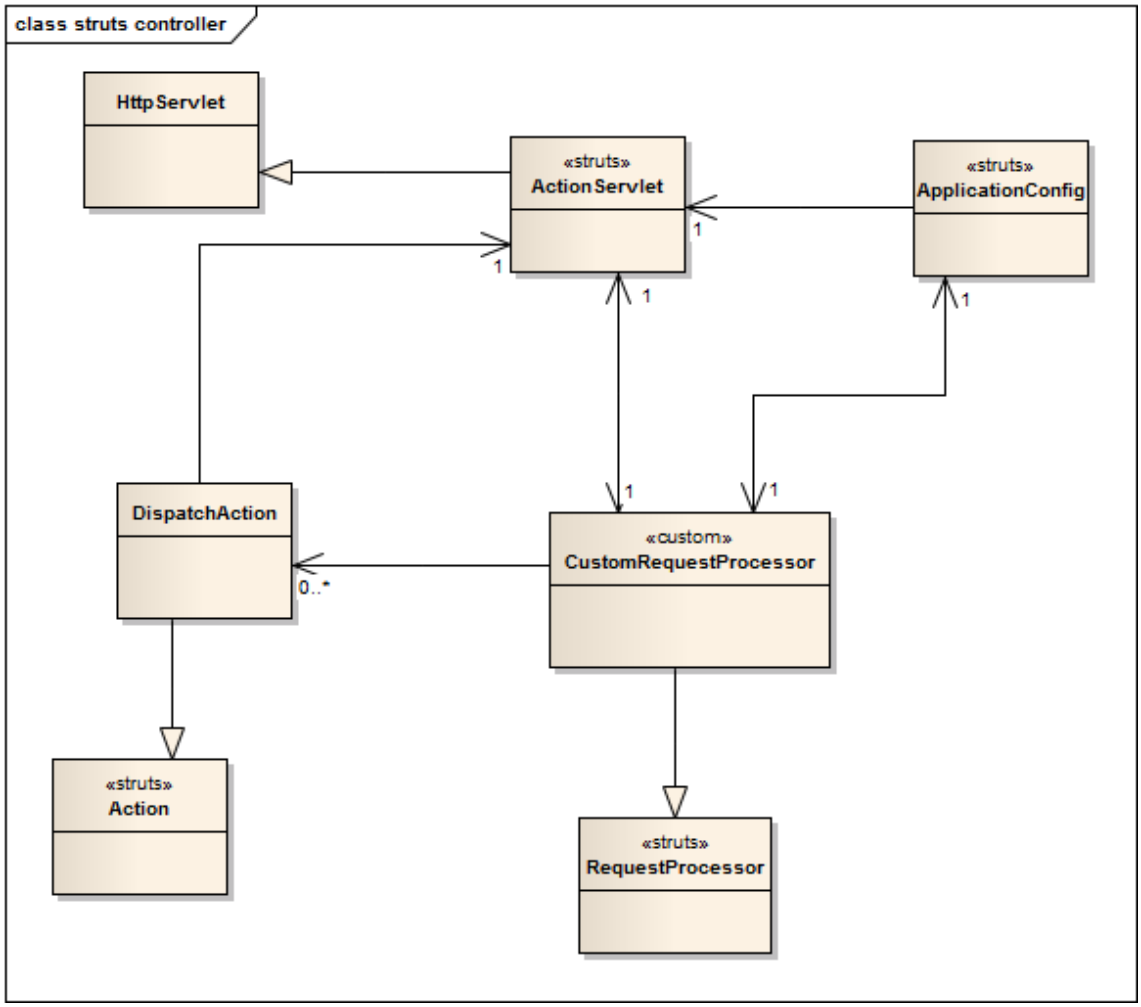


Figura 4-2 Components del Controlador de LLOG

Tractament de les peticions web

El tractament d'una petició des d'un navegador inclou els següents passos:

1. El servlet container consulta la variable 'url-pattern' en el fitxer de desplegament. En aquesta variable s'especifiquen els patrons de tractament de peticions i quins són els seus gestors.

2. En rebre una petició amb el patró final '*.do' (patró utilitzat per **Struts**), detecta que el gestor associat és el `ActionServlet` i li cedeix la petició.
3. Fitxer de *mapping* de **Struts**. Aquest fitxer serà utilitzat per determinar on anirà cada petició (una classe 'Action'), quins *beans* de formulari utilitzar i on redirigir la petició una vegada s'hagi resolt.

4. El 'ActionServlet' de Struts, segons el fitxer de *mapping* definit, cedeix la petició al 'processorClass' configurat.

```
<controller>
<set-property property="processorClass"
value="com.framework.web.struts.CustomRequestProcessor"/>
   <set-property property="maxFileSize" value="10M"/>
   <set-property property="nocache" value="true"/>
</controller>
```

Cal recordar que **Struts** permet l'extensió del seu comportament mitjançant l'extensió del bàsic 'RequestProcessor'.

5. La classe CustomRequestProcessor consulta el fitxer 'struts-config.xml' quina és la classe 'Action' que tractarà la petició.

```
<action className="com.framework.web.struts.DispatchActionMapping"
   input="producte.attachForm" name="productForm" parameter="reqCode"
   path="/ProductAction" type="com.webpage.action.ProductAction"
validate="true">
   <set-property property="validateActions" value="attach"/>
   <forward name="successAddNewProduct"         path="user.addedNewProduct"/>
   <forward name="successActivateForm"          path="producte.activateForm"
         redirect="false"/>
   <forward name="successConsumeProductForm"  path="producte.consumeForm"
            redirect="false"/>
   <forward name="successConsume"                      path="producte.consumed"
     redirect="false"/>
   <forward name="successActivate"                     path="producte.activated"
     redirect="false"/>
   <forward name="successAttachForm"           path="producte.attachForm"
     redirect="false"/>
   <forward name="successAttach"                       path="producte.attach"
      redirect="false"/>
   <forward name="successConfirmForm"          path="producte.confirmForm"
         redirect="false"/>
   <forward name="tilesSuccessConfirmForm"
  path="tiles.producte.confirmForm"           redirect="false"/>
 </action>
```

Servei de validació

En una aplicació existeixen dos tipus de validacions a considerar:

- Validació de dades
- Validació funcional

La validació de dades inclou el tractament de l'entrada concreta de tipus de dades, en la que no influeix l'estat de l'aplicació. Aquesta validació, ha de tenir en compte, entre d'altres:

1. Si un camp és obligatori
2. Si un camp és numèric, cadena, etc...
3. Si un camp té un mínim o màxim de caràcters.
4. Si un camp compleix una expressió regular (format de NIF, codi postal, etc.).

La validació funcional depèn de l'estat de l'aplicació i de les dades de negoci, i queda fora de l'àmbit d'aquest apartat, perquè el seu tractament es realitzarà per mitjà del llançament i tractament d'excepcions de negoci (veure Servei d'excepcions).

Així com en **Struts** la validació es realitza a nivell de presentació, en realitat el que s'està fent és anticipar la validació dels nostres objectes de negoci corresponents (un camp és requerit en presentació perquè ho és en negoci, etc.).

En la aplicació utilitzem el marc de treball Validator (https://commons.apache.org/proper/commons-validator/), que permet configurar regles de validació declarativament mitjançant arxius xml. Això significa que s'especifiquen externament a la font de l'aplicació. Existeixen dos arxius de configuració importants per al marc de treball **Validator**: `validator-rules.xml` i `validation.xml`.

L'arxiu validator-rules.xml

L'arxiu de configuració `validation-rules.xml` conté un conjunt global de regles de validació que la aplicació pot utilitzar directament. Aquest arxiu no depèn de l'aplicació i el pot utilitzar qualsevol aplicació **Struts**. En el cas de que sigui necessari ampliar o modificar el conjunt de regles per defecte, caldrà modificar aquest arxiu.

Validator-rules_1_1.dtd descriu la sintaxis de l'arxiu `validation-rules.xml`. L'element arrel és l'element `form-validation`, que requereix un o més elements `global`:

```
<!ELEMENT form-validator (global+)>
<!ELEMENT global (validator+)>
```

Cada element `validator` descriu una regla de validació única. El següent fragment de l'arxiu `validation-rules.xml` és la definició de la regla de validació `required`:

```
<validator name="required"
          classname="org.apache.struts.validator.FieldChecks"
            method="validateRequired"
      methodParams="java.lang.Object,
                    org.apache.commons.validator.ValidatorAction,
                    org.apache.commons.validator.Field,
                    org.apache.struts.action.ActionMessages,
                    org.apache.commons.validator.Validator,
                    javax.servlet.http.HttpServletRequest"
               msg="errors.required"/>
```

l'element `validator` suporta set atributs, com es mostra aquí:

```
<!ATTLIST validator name CDATA #REQUIRED
classname CDATA #REQUIRED
```

```
method CDATA #REQUIRED
methodParams CDATA #REQUIRED
msg CDATA #REQUIRED
depends CDATA #IMPLIED
jsFuncionName CDATA #IMPLIED>
```

l'atribut `name` assigna un nom lògic a la regla de validació, i aquest nom ha de ser únic, els atributs `classname` i `method` defineixen la classe i mètodes que contenen la lògica per la regla de validació, l'atribut `msg` és una clau del paquet de recursos (i18n), **Validator** utilitza aquest valor per cercar un missatge del paquet de recursos **Struts** quan es dóna un error de validació.

L'aplicació conté regles de validació personalitzades utilitzant el sistema descrit anteriorment, el següent fragment és una regla personalitzada per a la validació de l'extensió d'un arxiu:

```
<validator name="imagefile"
    classname="com.webpage.validator.ImageFileValidator"
    method="validateImageFile"
    methodParams="java.lang.Object,
                  org.apache.commons.validator.ValidatorAction,
                  org.apache.commons.validator.Field,
                  org.apache.struts.action.ActionMessages,
                  org.apache.commons.validator.Validator,
                  javax.servlet.http.HttpServletRequest"
    msg="errors.fileimage"/>
```

L'atribut `msg` conté el missatge que es mostra en el formulari de presentació quan hi ha un error de validació.

```
errors.fileimage={0} es un format de imatge invàlid (només .jpeg)
```

i els atributs `method`, `methodParams` i `classname` defineixen la classe, el nom del mètode i els paràmetres d'entrada:

L'arxiu validation.xml

Aquest arxiu és específic de l'aplicació; descriu quines regles de validació de l'arxiu `validation-rules.xml` utilitza un `ActionForm` determinat. La lògica de validació està associada amb una o més classes `ActionForm` per mitjà d'aquest arxiu extern.

El següent fragment mostra les regles de validació d'un formulari de l'aplicació:

```
<form name="messageForm">
  <field depends="required,maxlength,contactData,existsEmail" property="text">
   <arg key="comuns.text" position="0"/>
   <arg key="${var:maxlength}" name="maxlength" position="1" resource="false"/>
   <var>
    <var-name>maxlength</var-name>
    <var-value>500</var-value>
   </var>
  </field>
 </form>
```

Veure Annex: Implementació classe Validador del tipus d'arxiu imatge.

L'element `form` defineix un conjunt de camps a validar. `name` es correspon a l'identificador que l'aplicació assigna al formulari. En el cas del framework de **Struts**, és l'atribut `name` de l'apartat `form-bean`.

```
<form-bean dynamic="true" name="messageForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="text"          type="java.lang.String"/>
   <form-property name="reqCode"
   type="java.lang.String"/>
```

```
    <form-property name="id_listing"
    type="java.lang.String"/>
    <form-property name="parent_message_id"
    type="java.lang.String"/>
    <form-property name="id_user_to"
    type="java.lang.String"/>
  </form-bean>
```

L'element `field` es correspon a una propietat específica de un JavaBean que s'ha de validar. En una aplicació Struts, aquest JavaBean és un `ActionForm`.

No es pot utilitzar la classe `ActionForm` estàndard de Struts amb el marc **Validator**. En el seu lloc, s'ha d'utilitzar una subclasse de la classe ActionForm que està específicament dissenyada per a funcionar amb Validator.

En el cas de l'aplicació, s'utilitzen ActionForm dinàmiques (`org.apache.struts.validator.DynaValidatorForm`) , de manera eliminem complexitat innecessària en el codi.

Validació en AJAX

Les validacions descrites anteriorment s'efectuen a nivell de servidor, si bé existeix la possibilitat d'afegir un nivell de validació per part del client basada en la presencia de l'atribut `javascript` dins l'element `validator`. La validació a nivell de client pot ser útil, però és molt insegura, degut a que qualsevol programa o robot pot enviar una petició POST i saltar-se la validació.

En el nostre cas, hem dissenyat un framework que connecta la validació a nivell de servidor utilitzant API's javascrit al costat del client. D'aquesta forma, podem obtenir una interacció més suau i moderna a nivell de interfície d'usuari per la validació dels formularis

Per realitzar aquesta validació hem utilitzat la llibreria DWR (Direct Web Remoting) (Walker, David Marginian & Joe) , Prototype Javascript Framework (Prototype Core Team) i Commons-Validator (Com).

DWR, o Web Remoting Direct, és una biblioteca Java de codi obert que ajuda als desenvolupadors a escriure pàgines web que inclouen la tecnologia Ajax. Permet utilitzar codi interpretable per un navegador d'internet de manera que possibilita la utilització de funcions Java executant-se en un servidor web talment com si aquestes funcions es trobessin en el navegador.

Consisteix de dues parts principals:

- Una part de codi que permet Javascript recuperar dades d'un servidor web basat en Servlet utilitzant els principis d'AJAX.
- Una llibreria Javascript que fa que sigui més senzill per als desenvolupadors actualitzar dinàmicament la pàgina web amb les dades recuperades.

Prototype Javascript Framework és un marc de desenvolupament de JavaScript que té com a objectiu facilitar el desenvolupament d'aplicacions web dinàmiques.

Veure Annex: Servei de validació (fragment de codi javascript de l'arxiu ajaxtags-validation.js, mostra la classe principal que executa la validació)

4.3 Components del Model

Tal i com hem descrit anteriorment, l'aplicació LLOG utilitza l'enfoc del disseny guiat pel domini (2.9) per estructurar les classes del codi en la capa de model.

En el disseny del model s'han utilitzat els conceptes de Entitat, Servei i Repositori per estructurar i agrupar les classes.

Objectes de domini

La implementació de les classes dels objectes del domini és completament anàloga al model conceptual d'objectes.

Veure el diagrama de domini: Figura 3-14

Disseny de la base de dades

L'aplicació de LLOG utilitza el *framework* de ORM **Hibernate**. ORM és un model per automatitzar la persistència d'objectes d'aplicacions Java a taules en bases de dades relacionals, utilitzant *metadata* que descriu el mapeig entre els objectes i la base de dades (Christian Bauer,Gavin King, 2005).

Abans d'establir el mapeig entre els objectes i les respectives taules, anem a descriure quines són els estratègies per *mapeja*r una relació d'herència.

Hi ha 3 models diferents per representar la jerarquia de herència en un mapeig (Ambler, 2002) (Christian Bauer,Gavin King, 2005):

- **Taula per classe concreta**: Descarta polimorfisme i relacions d'herència del model relacional.
- **Taula per jerarquia de classe**: habilita el polimorfisme ***desnormalitzant*** el model relacional i utilitzant una columna de tipus "discriminador" per mantenir la informació de tipus.
- **Taula per subclasse**: representa les relacions d' herència amb relacions de claus foranies.

Per dissenyar la base de dades fem servir un enfoc de dalt a baix (Top Down *approach*): és a dir, comencem amb un model del domini i derivem l'estructura de la base de dades.

Per dissenyar el mapeig ORM amb la base de dades hem fet servir l'enfoc de *taula per subclasse*:

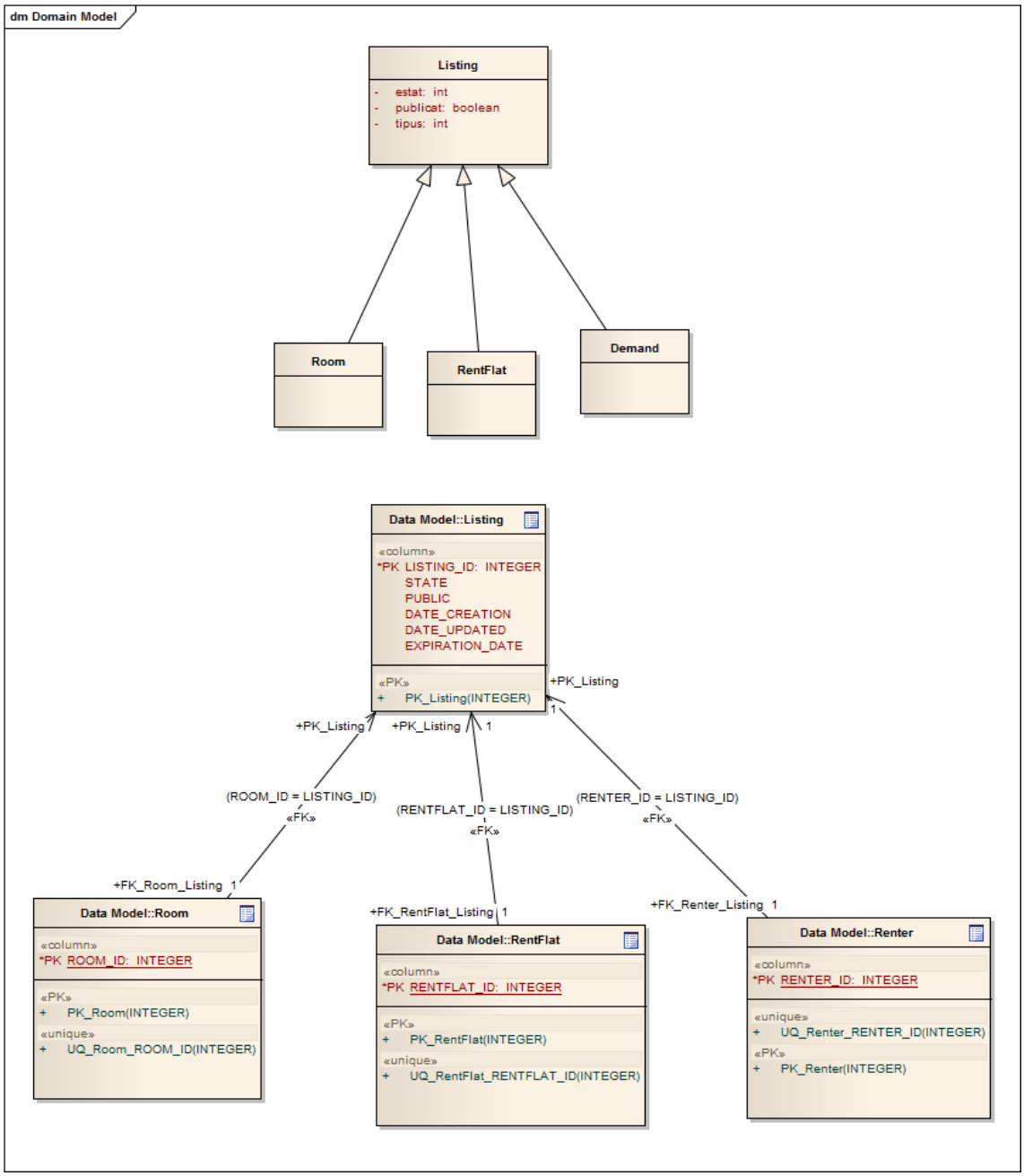


Figura 4-3 Mappejat per taula per subclasse

Com que hi ha moltes relacions i atributs entre les diferents taules a la Figura 2-2 s'ha decidit només mostrar les relacions i els atributs més importants entre la taules que descriuen el model de mapeig de jerarquia.

Factories de DAO's

Tal i com s'ha descrit anteriorment, en l'aplicació LLOG s'utilitzen classes anomenades "Repositoris" implementades amb el patró DAO per recuperar els objectes *persistits* a la base de dades. Es fa ús d'una factoria per recuperar instàncies de tots els objectes del domini.

La factoria de DAO's fa ús del patró Abstract Factory.

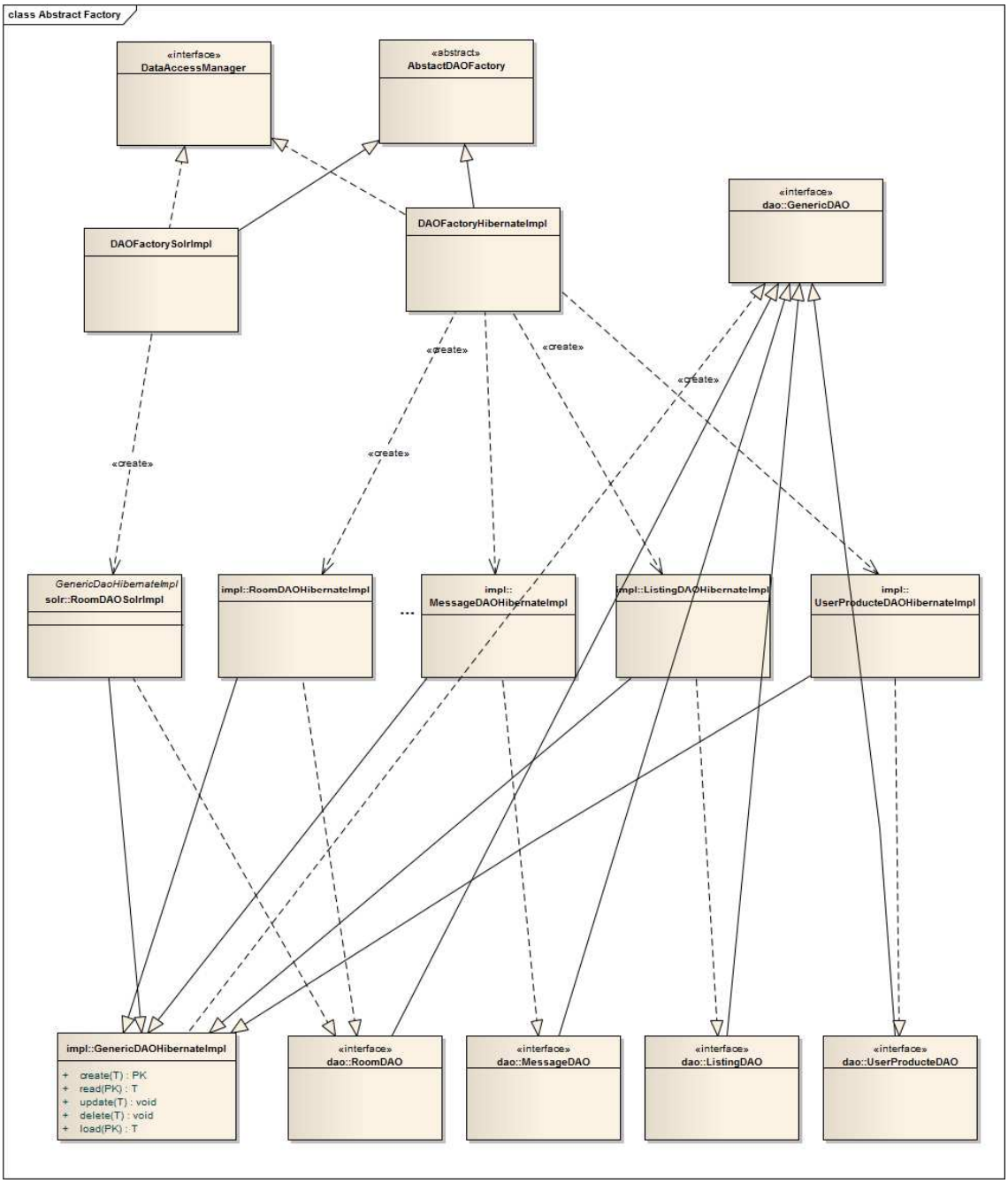


Figura 4-4 Factoria de DAO's

Veure Annex: Implementació AbstractDAO Factory

Capa de servei (negoci)

La capa de servei utilitza una factoria per instanciar els diferents serveis relacionats amb les activitats i accions que no pertanyen conceptualment als objectes del domini.

En particular es fa ús del patró *Factory Method*.

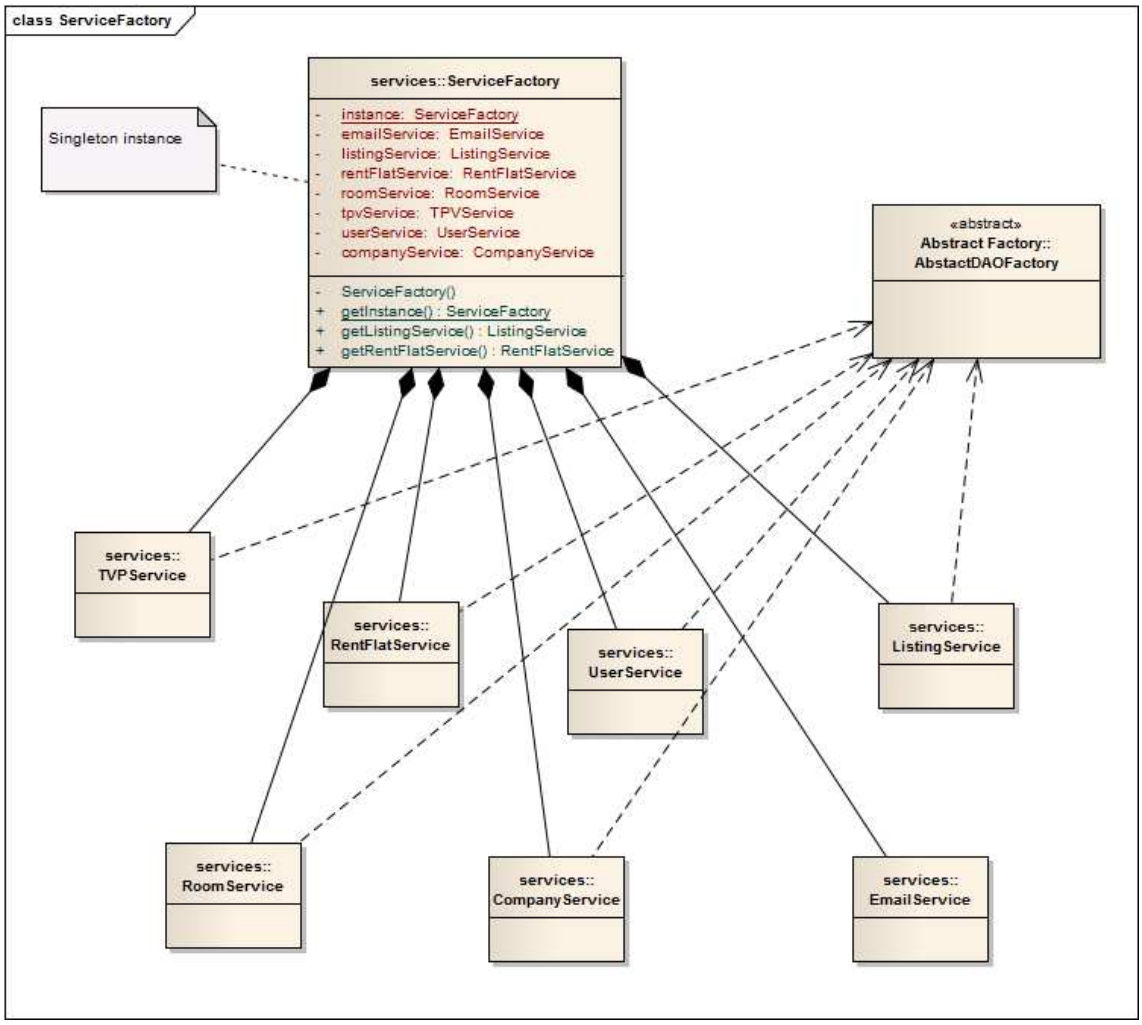


Figura 4-5 Classe Service Factory: Factoria de serveis

Servei de cerca full text i faceted search

L'aplicació de LLOG fa servir el servidor *open source* de classe empresarial **Solr** (http://lucene.apache.org/solr/). **Solr** és una aplicació escrita en java i utilitzada per multitud de lloc públics d'internet. Internament utilitza **Apache Lucene**, un motor de cerca per text d'alt rendiment i també *open source*. **Solr** proveeix una interfície HTTP de processament per indexar i consultar documents.

Solr també proveeix una interfície de consulta i filtratge per "faceting" i de creació d'un índex de documents a partir d'una connexió JDBC a la base de dades, a través del seu *framework* `DataImportHandler` (DIH).

Servei de mail

Per tal d'enviar emails, l'aplicació lloguerjove.com utilitza el servei d'enviament de email de Amazon Simple Email Service (Amazon SES).

Amazon SES permet enviar i rebre correu electrònic utilitzant una plataforma de correu electrònic fiable i escalable.

Hi ha diverses maneres d' enviar un correu electrònic mitjançant l'ús d'Amazon SES. En el cas particular de l'aplicació Lloguerjove.com s'utilitza una interfície SMTP proporcionada pel servei Amazon SES.

Per tal de connectar amb la interfície SMTP de Amazon SES per a l'enviament d'emails, l'aplicació de LloguerJoVe.com utilitza la API **JavaMail**.

L'API JavaMail és un conjunt d'APIs abstractes que modelen un sistema de correu. L'API proporciona una plataforma independent i un *framework* independent del protocol per construir aplicacions de client de correu electrònic basats en la tecnologia Java. L'API **JavaMai**l proporciona recursos per llegir i enviar correu electrònic. Els proveïdors de serveis implementen protocols particulars. Diversos proveïdors de serveis estan inclosos en el paquet de l'API **JavaMail**; altres estan disponibles per separat. L'API **JavaMail** s'implementa com un paquet opcional de Java que es pot utilitzar en JDK 1.4 i posterior en qualsevol sistema operatiu. L'API **JavaMail** és també una part necessària de la plataforma Java Enterprise Edition (Java EE).

Lloguerjove.com utilitza per el correu d'entrada el servei de correu empresarial de Zimbra.(https://www.zimbra.com/) el qual proveeix una interfície IMAP que permet l'accés als missatges de correu electrònic emmagatzemats en un servidor a internet.

Veure Annex: Mètode per a l'enviament d'email utilitzant javax.mail api

Servei de integració

El servei d'integració proveeix les següents integracions:

- Integració amb Facebook mitjançat OAuth
- Integració amb passarel.la de pagament Paypal
- Integració amb passarel.la de pagament CaixaBank.

El següent diagrama il·lustra el procés de pagament a través de la passarel.la de pagament de CaixaBank:

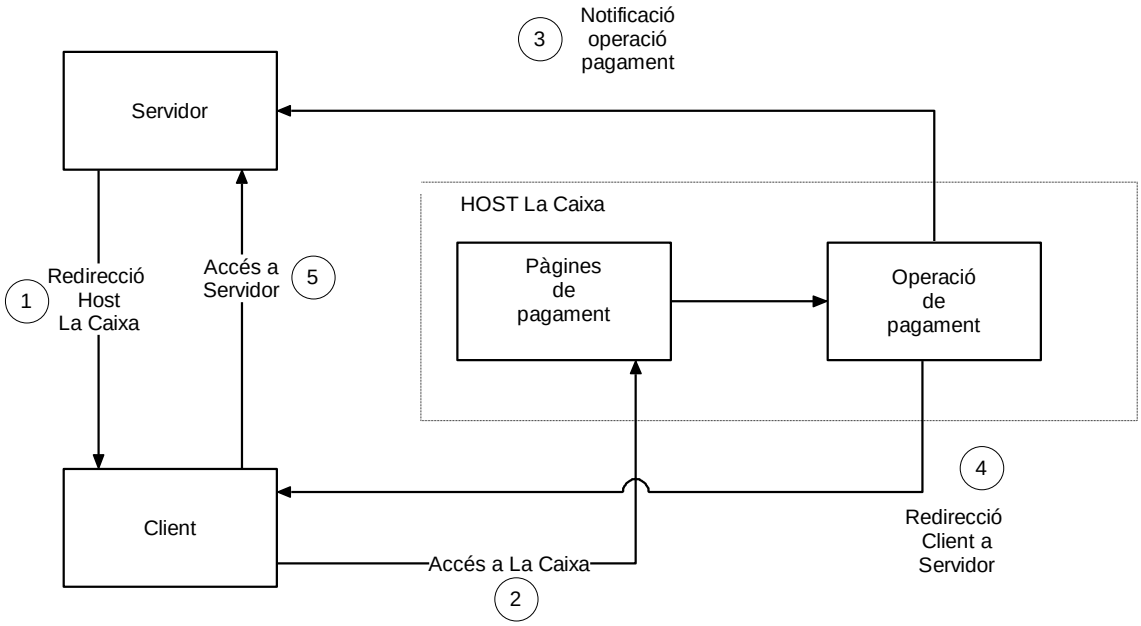


Figura 4-6 Operació de pagament mitjançant passarel.la

La resta d'integracions queden fora l'abast d'aquest projecte.

Servei d'Estratègia

El servei d'estratègia consta dels següents components:

- **L'estratègia**, és una interfície que defineix la funcionalitat a realitzar. En el cas de l'aplicació LLOG, l'estratègia defineix un mètode anomenat `getPublishState()` i `getState()`.

Classe ApplicationStrategy

```
package com.alquilerjoven.core.strategy;

public interface ApplicationStrategy {

    public boolean getPublishState();

    public Integer getState();
}
```

- **Les estratègies concretes** proporcionen diferents implementacions de l'estratègia. En el cas de l'aplicació LLOG, hi ha una `ActiveStrategy`, `InactiveStrategy`, i `LimitedStrategy`: totes retornen el mateix tipus de dates, però diferents combinacions de valors.

```
package com.alquilerjoven.core.strategy;

import com.alquilerjoven.core.LloguerJoveConstant;

public class ActiveStrategy implements ApplicationStrategy  {

    @Override
    public boolean getPublishState() {
        // TODO Auto-generated method stub
        return true;

    }

    @Override
    public Integer getState() {
        // TODO Auto-generated method stub
        return LloguerJoveConstant.ACTIVE_STATE;
    }

}
```

- **El context** és una classe que conté una estratègia concreta, referenciada a través de la seva interfície d'estratègia. Hi pot haver un context que conté l'estratègia `ActiveStrategy` i un context que conté l'estratègia , `InactiveStrategy`.

```
      package com.webpage.action;

      import java.io.PrintWriter;

      import javax.servlet.http.HttpServletRequest;
      import javax.servlet.http.HttpServletResponse;

      import org.apache.shiro.authz.annotation.RequiresRoles;
      import org.apache.struts.action.ActionForm;
      import org.apache.struts.action.ActionForward;
      import org.apache.struts.action.ActionMapping;

      import com.webpage.util.ParamUtil;

      public class StrategyAction extends LLoguerJoveDispatchAction {

          @RequiresRoles("ADMINISTRATOR")
          public ActionForward unspecified(ActionMapping mapping,
ActionForm form,
                  HttpServletRequest request, HttpServletResponse response)
                  throws Exception {

              PrintWriter out = response.getWriter();
              ParamUtil.getParameter(request,"strategy",true);
              String newStrategry=request.getParameter("strategy");

               System.setProperty("estrategiaapp.class.name",
newStrategry);
              out.print("OK");

              return null;

          }

      }
```

.

El client és una classe en la aplicació que invoca l'estratègia a través de la classe de context (`ApplicationStrategyFactory` en el cas de LLOG):

```
      ApplicationStrategy strategy=
ApplicationStrategyFactory.getInstance().getApplicationStrategy();
              roomBean.setPublishing(strategy.getPublishState());
              roomBean.setState(strategy.getState());
```

4.4 Components de Vista

Concretament treballarem amb la implementació de **Tiles de Apache** (http://tiles.apache.org).

Els components de vista de LLOG estan formats per la conjunció entre diferents elements:

- JSP (Java Server Pages) per representar les pàgines.
- JSTL, Struts Tags (html, logic, bean, tiles) .
- Per a que des dels nostres Action puguem accedir a l'objecte "populat", es fa ús de formularis del tipus `org.apache.struts.validator.DynaValidatorForm`. L'ús d'aquest tipus, enlloc del tradicional `ActionForm` de **Struts**, amplia de la classe `ActionForm` i permet definir dinàmicament les propietats del formulari HTML, i alhora permet connectar el marc de treball **Validator**.

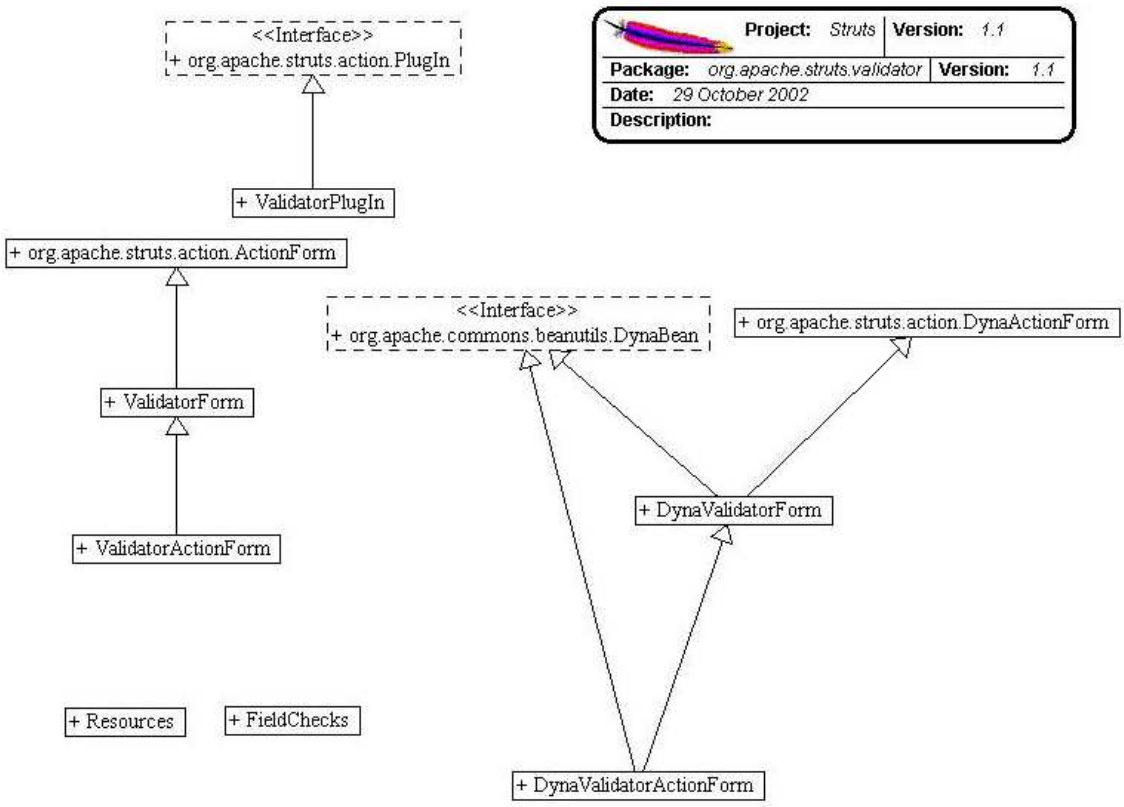


Figura 4-7 DynaValidatorForm

Declaració d'un `DynaValidatoForm` en l'arxiu de struts-config.xml:

```
....
<form-bean name="postForm"
type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="nom"                    type="java.lang.String"/>
   <form-property name="id_thread"              type="java.lang.String"/>
   <form-property name="assumpte"               type="java.lang.String"/>
   <form-property name="reqCode"                type="java.lang.String"/>
   <form-property name="j_captcha_response"
type="java.lang.String"/>
   <form-property name="id_tema"                type="java.lang.String"/>
   <form-property name="comentari"              type="java.lang.String"/>
   <form-property name="id_reply"               type="java.lang.String"/>
  </form-bean>
....
```

Servei de llistats

El servei de llistats permet la presentació de llistats HTML en el que es permet a l'usuari:

- Paginació dels resultats i navegació per pàgines mitjançant "avanç" i "retrocés" (primera, última, següent, anterior) i per accés directe a un número de pàgina.
- Presentar un número determinat de resultats per pàgina.
- Acotar el número màxim de resultats a mostrar.
- Ordenació de resultats mitjançant un *Dropdown* menú que permet a l'usuari escollir un valor sobre una llista predeterminada.

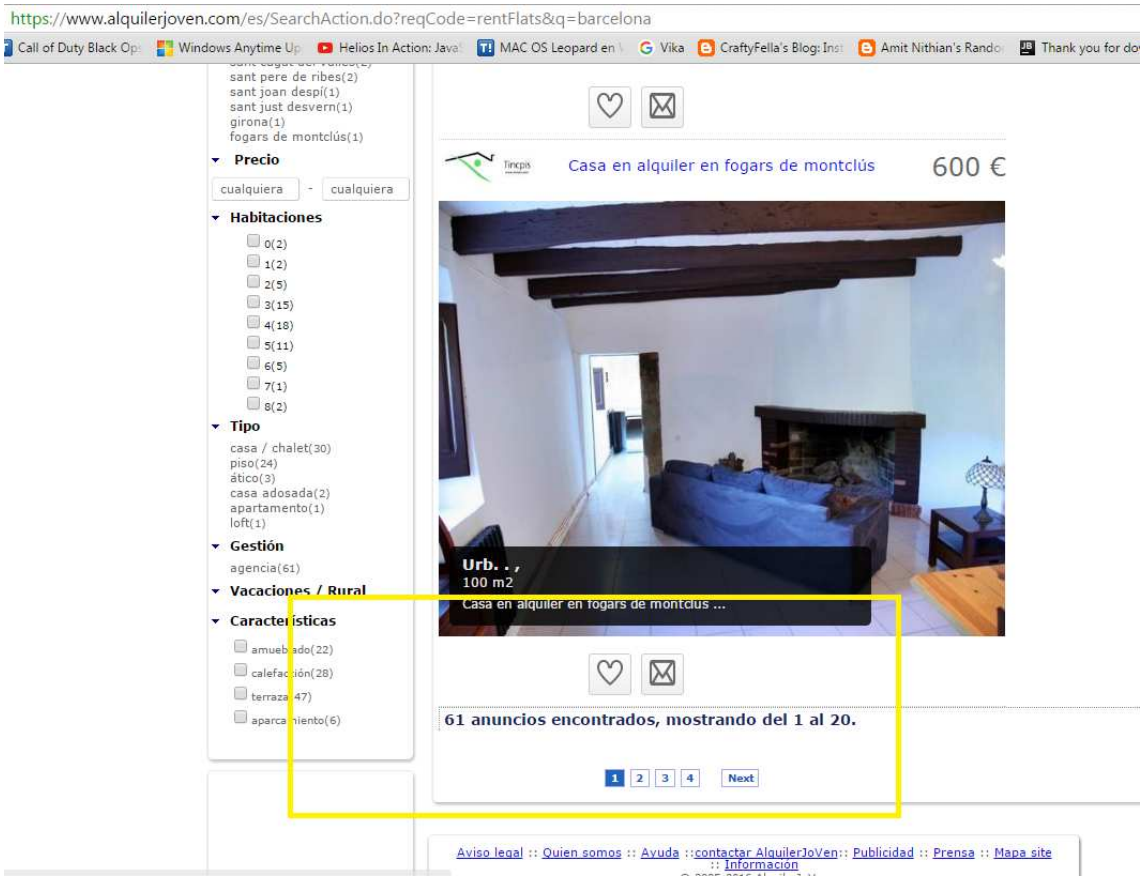


Figura 4-8 Paginació del servei de llistats

El servei de llistats s'implementa amb el *framework* **Apache Freemarker**(http://freemarker.org/).

Apache FreeMarker és un motor de plantilles: una biblioteca de Java per a generar text (pàgines web HTML, correus electrònics, arxius de configuració, codi font, etc.) sobre la base de les plantilles i de les dades. Les plantilles estan escrites en el llenguatge de plantilla **FreeMarker** (FTL), que és un llenguatge especialitzat senzill i potent, que permet blocs condicionals, iteracions, assignacions, macros i funcions, inclusions d'altres plantilles, etc...

Exemple de plantilla de paginació de LLOG utilitzant **Freemarker** amb macros, funcions i importacions de plantilles:

Plantilla paginator.ftl

```
<#import "/WEB-INF/jsp/freemarker/order-links.ftl" as orderlinks />
<#--
 * Outputs the first page link
-->
<#macro first>
   <#if (data.pageNumber < 1)>
     <#local classAttr = "class=\"disabled\"" />
   <#else>
     <#local classAttr = "" />
   </#if>
   <#local text>
     <@spring.messageText "pagination.first", "« First" />
   </#local>
   <@page 0, text, classAttr/>
</#macro>
...
<#function max x y>
   <#if (x<y)><#return y><#else><#return x></#if>
</#function>
<#function min x y>
   <#if (x<y)><#return x><#else><#return y></#if>
</#function>
<#macro pages totalPages p url reqCode>
<#assign window = 8>
   <#assign size = totalPages?size>
   <#if (p<(window-window/2)+1)> <#-- p among first 8 pages -->
      <#assign interval = 1..(min(8,size))>
   <#elseif ((size-p)<(window-window/2)-1)> <#-- p among last 5
pages -->
      <#assign interval = (max(1,(size-window)))..size >
   <#else>
      <#assign interval = (p-window/2)..(p+(window/2-1))>
   </#if>
  <div class="pagination" style="margin-left: 15em;">
    <#if !(interval?seq_contains(1))>
    <span class="pagelinks">
     <a href="${url}?<@orderlinks.pagerlinks 1
reqCode/>">First</a><#lt>
      </span>
     </#if>
    <#if !(interval?seq_contains(1))>
    <#if (p>8)>
    <span class="pagelinks">
    <a href="${url}?<@orderlinks.pagerlinks p-1
reqCode/>">Prev</a><#lt>
      </span>
```

```
    </#if>
    </#if>

    <#if (p>1)>
    <#if (p<(size))>
     <span class="pagelinks">
     <a href="${url}?<@orderlinks.pagerlinks p-1
reqCode/>">Prev</a><#lt>
       </span>
     </#if>
    </#if>
     <span class="pagelinks">
    <#list interval as page>
     <#if page=p>
       <span class="current">${page}</span> <#t>
       <#else>
        <a href="${url}?<@orderlinks.pagerlinks page reqCode/>"
title="Go to page ${page}">${page}</a>
       <#t>
       </#if>
    </#list>
    </span>
     <#if (p>1)>
     <#if (p==size)>
     <span class="pagelinks">
     <a href="${url}?<@orderlinks.pagerlinks p-1
reqCode/>">Prev</a><#lt>
       </span>
     </#if>
      </#if>
    <#if (p<size)>
     <span class="pagelinks">
     <a href="${url}?<@orderlinks.pagerlinks p+1
reqCode/>">Next</a><#lt>
       </span>
     </#if>
      <#if !(interval?seq_contains(size))>
       <span class="pagelinks">
      <a href="${url}?<@orderlinks.pagerlinks size
reqCode/>">Last</a><#lt>
       </span>
     </#if>
    </div>
</#macro>
```

Exemple d'inserció d'una plantilla de **Freemarker** en una pàgina jsp:

list.jsp

```
...
<div id="actions">
<div id="sortFilter">
<label>Ordenar por</label><tiles:insertTemplate template="/WEB-
INF/jsp/freemarker/dropdown.ftl" />
</div>
</div>
<tiles:insertTemplate template="/WEB-INF/jsp/room/list.ftl"/>
...
```

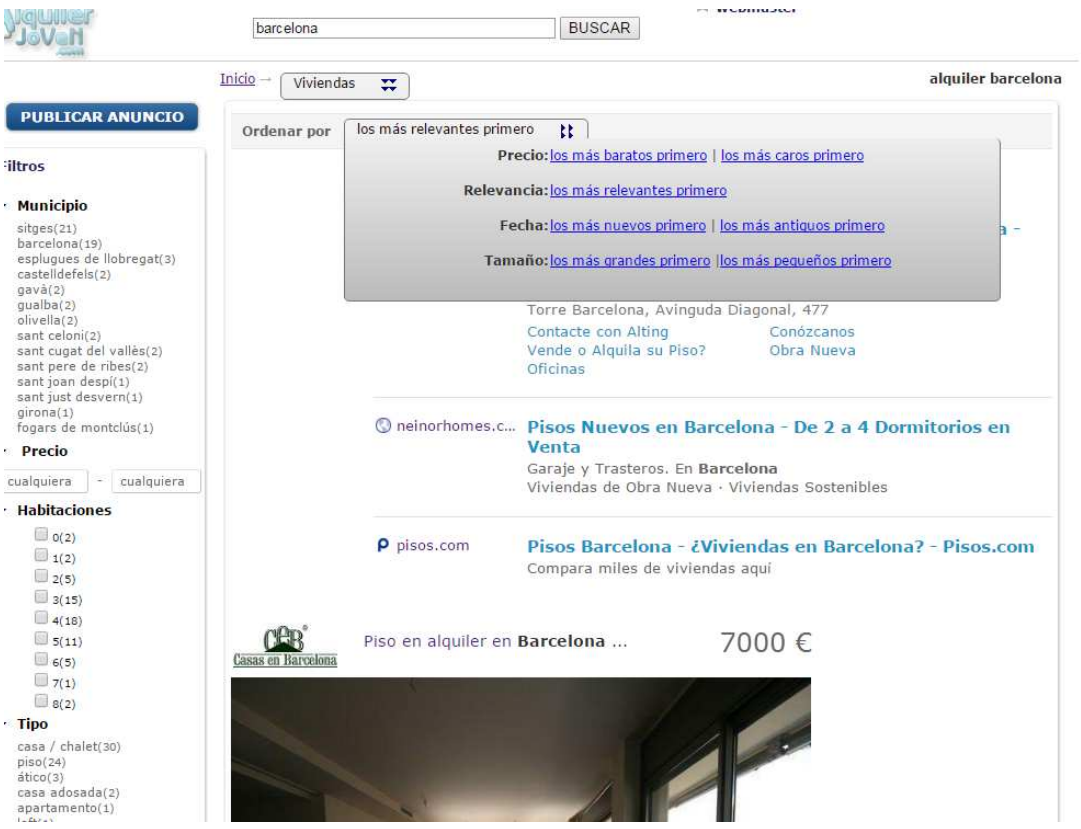


Figura 4-9 *Dropdown* menú de ordenació de resultats

4.5 Serveis transversals

Anomenen serveis transversals aquells serveis que no són específics d'una sola capa de aplicació, sinó que intervenen en tots els nivells.

Servei de logging

El servei de *logging* o traces té com a objectiu poder detectar i localitzar amb més facilitat errors de l'aplicació, entrada de dades segons la lògica del sistema i fins hi tot realitzar el seguiment de qui ha fet una determinada operació per a portar a terme una auditoria.

Per a què això pugui ser portat a terme, el servei ofereix la possibilitat de:

- Definir nivells de traces (informació, fatal, errors, etc...)
- Definir en quines sortides es generarà la traça: consola, fitxers, base de dades, correu electrònic, etc.
- Canviar en qualsevol moment quin és el mínim nivell de traces que volem mostrar, sense haver d'afectar a les classes que generen les traces.
- Definir el format de sortida de les nostres traces: incorporar l'hora, el número de línia del codi on s'ha produït, la classe, etc.
- Incorporar informació de context a la traça: usuari, ip del client, etc.

Treballarem amb la llibreria **Slf4j** (Simple Logging Facade for Java, http://www.slf4j.org/). Aquesta llibreria és la *facade* per tot el sistema de log.

Slf4j actua com una *facade* o façana, és a dir una abstracció per a diversos *frameworks* de *logging*, de manera que permet connectar (*plug-in*) el *framework* de traces desitjat en temps de desplegament (d*eployment)*.

El *framework* escollit en l'aplicació per a la inserció de declaracions (*statements*) de *log* és el **Log4j** (https://logging.apache.org/log4j/1.2/). **Log4j** és un dels frameworks de traces més estès i es basa en l'ús d'*appenders*, *categories* i *layouts*.

La següent taula defineix unes regles que haurà de complir totes les classes obligatòriament:

- La secció import de la classe importarà les classes `Logger` i `LoggerFactory`:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

- Tota classe tindrà una variable membre:

```
private static final Logger logger_ =
LoggerFactory.getLogger(MyClass.class);
```

- Cap classe tindrà línies de *log* a nivell *debug* excepte en la fase desenvolupament, i les línies sempre estaran dintre d'un *if*:

```
if(logger_.isDebugEnabled()){
logger_.debug("Text Logging");
}
if(logger_.isInfoEnabled(){
logger_.info("Info Text");
}
```

Servei de transaccionalitat

Per la capa de persistència hem utilitzat el framework **Hibernate** (http://hibernate.org/). **Hibernate** és un framework de mapeig objecte-relacional (ORM) que facilita el mapeig d'atributs entre una base de dades relacional i el model d'objectes d'una aplicació, mitjançant arxius declaratius (XML) o anotacions en els *beans* de les entitats que permeten establir aquestes relacions.

Transaccionalitat

Degut a que tota la lògica de negoci es realitza en les `Action`, es interessant centralitzar d'alguna forma la gestió de les transaccions.

Per tal d'evitar problemes amb l' "inicialització gandula" (*lazy initialitzation)* quan accedim a un objecte dins el context d'una pàgina jsp, si la sessió d'Hibernate ja ha estat tancada, no podrem accedir a les

associacions "gandules" (*lazy associations*) que no han estat capturades (*unfetched*) anteriorment.

Per tal de solucionar ambdós problemes, hem escollit el patró de disseny *thread-local session* per implementar les classes que gestionen les transaccions.

Patró thread-local session

Una sessió *thread-local* és una sola instancia de la sessió associada a una petició en particular (*request*). Tots els components cridats en el mateix *request* compartiran la mateixa sessió i el context de persistència.

És especialment útil afegir les pàgines JSP dins del context de persistència. La pàgina JSP extrau informació del model de domini navegant pel *graph* d'objectes començant en algun objecte persistent dins l'ambit de sessió o petició (*request*), no obstant, el graph de l'objecte pot incloure associacions no inicialitzades (*proxies* o *collections*) que han de ser travessades (i inicialitzades) mentre es *renderitza* la vista.

El patró sessió per *thread-local* permet tenir una sola instancia de la sessió d'Hibernate per *request*, que abasta la vista i potencialment múltiples `executes()` de les Action. El patró sessió per *thread-local* combina un `ThreadLocal` amb un interceptor o un filtre servlet que tanca la sessió quan finalitza el *reques*t, després de que la vista hagi estat renderitzada i just abans d'enviar la resposta al client.

Primerament, la nostra aplicació necessita una forma senzilla d'obtenir instancies de la `Session` d'**Hibernate**. Per això hem implementat la classe *helper* `HibernateSessionFactory`. En comptes d'obrir una nova `Session` cada vegada, quan es crida el mètode `currentSession()`, retorna la `Session` associada al *thread* actual, que la classe manté en una variable `ThreadLocal`.

Veure Annex: Implementacio thread local session

Servei de i18n

Java proporciona un conjunt important de característiques **i18n** en la llibreria central. En aquest apartat s'explica breument algunes d'aquestes

característiques, ja que el suport i18n en el marc de **Struts** es basa en aquests components.

La classe Locale

La classe `java.util.Locale` és la classe i18n més important en la llibreria Java. Quasi tot el suport per la internacionalització es basa en aquesta classe.

La classe Locale proporciona a Java les instancies del concepte de ubicació. Una instancia determinada de `Locale` representa un idioma i regió únics.

Quan es crea un objecte `Locale`, normalment especifica el codi d'idioma i país. El següent fragment de codi il·lustra la creació d'un objecte `Locale` per a Espanya en català.

```
Locale catLocale=new Locale("ca", "ES");
```

El primer argument del constructor és el codi de l'idioma, mentre que el segon argument és el codi del país. El codi d'idioma consta de dos lletres minúscules i cal que s'adapti a l'especificació ISO-639, el codi de país consta de dos lletres majúscules i cal que s'adapti a l'especificació ISO-3166.

Determinar la ubicació de l'usuari

El contenidor web utilitzarà normalment la ubicació per defecte per al seu entorn local, mentre que utilitzarà la que s'ha passat des del client en `HttpServletRequest` per mostrar informació sensible a la ubicació a l'usuari final.

Tot i que la informació d'ubicació d'usuari es pot enviar per a cada petició del client, **Struts** per defecte, recupera la informació una sola vegada i l'emmagatzema a la sessió de l'usuari. L'objecte `Locale`, si s'emmagatzema a la sessió, s'emmagatzema amb la clau de `Globals.LOCALE_KEY`.

Paquet de recursos java

La classe `java.util.ResourceBundle` proporciona la possibilitat d'agrupar un conjunt de recursos per a una ubicació específica. Els recursos són normalment elements textuals tals com etiquetes de camp i boto, missatges d'estat, missatges d'error, títols de pàgina, etc…

Struts no utilitza la classe `ResourceBundle`. En el seu lloc, proporciona una funcionalitat similar amb les classes dins del seu marc. La classe `org.apache.struts.util.MessageResources` i la seva única subclasse `org.apache.struts.util.PropertyMessageResources`, s'utilitzen per a realitzar la funcionalitat paral·lela a la de la jerarquia `ResourceBundle`

Internacionalització d'aplicacions en Struts

El suport en internacionalització que proporciona el *framework* **Struts** en centra quasi en exclusiva en la presentació de text i imatges per a l'aplicació.

Com ja hem vist, en funció dels paràmetres de configuració de **Struts**, el *framework* pot detectar la ubicació preferida de un usuari i emmagatzemar-la en la sessió de l'usuari. Un cop s'ha determinat la ubicació de l'usuari, **Struts** pot utilitzar-la per cerca text i altres recursos del paquet de recursos. Els paquets de recursos són components essencials dins l'entorn de treball de **Struts**.

Els paquets de recursos han de seguir les convencions de la classe `PropertyResourceBundle` de la llibreria central de java, és a dir, cal crear un arxiu de text per al paquet de recursos amb una extensió `.properties` i que segueix les directrius per la classe java.util.Properties. La directriu més important té que el format dels missatges dins l'arxiu és:

clau=valor

La figura següent mostra un resum d'un arxiu de propietats `Messages_Bundle_ca_ES.properties` que el *framework* **Struts** pot carregar:

```
meta.descripcio=Habitacions en pisos compartits i pisos de lloguer a  Barcelona, Hospitalet, Terrassa, Sabadell, Girona, Catalunya
meta.descripcio_lloguerjove=Pisos de lloguer i habitacions en pisos compartits a Barcelona, Hospitalet, Terrassa, Sabadell, Girona, Vic, Reus, Tarragona, Lleida, Catalunya

meta.autor=Jordi Marti Carreras
meta.organitzation=Jordi Marti S.L.
meta.locality= España
meta.lang=CA
meta.keywords=pisos lloguer, lloguer jove, habitatge jove, compartir pis, pis compartit
meta.origen=AlquilerJoVen
```

```
meta.origen.lloguerjove=LloguerJoVe
globalnav.login=nom
globalnav.password=clau
globalnav.usuaris_registrats=usuaris registrats
globalnav.pagina_inici=pagina inici
globalnav.envia_mail=Enviali un email
globalnav.button.sortir=SORTIR
globalnav.button.entrar=ENTRAR
errors.required={0} és obligatori.
errors.minlength={0} no pot tenir menys de {1} caracters.
errors.maxlength={0} no pot tenir més de {1} caracters.
errors.invalid={0} es invàlid.
```

Figura 4-10 Arxiu de propietats ResourceBundle

Directrius de nomenament del paquet de recursos

El nomenament dels paquets de recursos és crític per al bon funcionament. Tots els paquets de recursos tenen un nom base que selecciona. Per la aplicació, el nom `MessagesBundle` s'ha utilitzat com a nom base. Per als paquets de recursos localitzats addicionals, per exemple per l'idioma anglès i el país Estats Units, hem creat un arxiu de propietats anomenat `messagesBundle_en_US` amb els recursos de ubicació apropiats.

Exemple del contingut del fitxer `messagesBundle_en_US`:

```
meta.descripcio=Offers and demands of rooms in shared fats and flats for rent for
young people in Spain
meta.descripcio_lloguerjove=Offers and demands of rooms in shared fats and flats
for rent for young people in Barcelona, Catalunya

meta.autor=Jordi Marti Carreras
meta.organitzation=AlquilerJoVen Corporation
meta.locality=Spain
meta.lang=EN
meta.keywords=alquiler joven, rent flats, rooms for rent, shared flats, roommates
meta.origen=AlquilerJoVen
meta.origen.lloguerjove=RoomMateSpain
globalnav.login=name
globalnav.password=password
globalnav.usuaris_registrats=Registered Users
```

```
globalnav.pagina_inici=Start Page
globalnav.envia_mail=Send an email
globalnav.button.sortir=LOGOUT
globalnav.button.entrar=LOGIN
errors.required={0} is required.
errors.minlength={0} can not be less than {1} characters.
errors.maxlength={0} can not be greater than {1} characters.
errors.invalid={0} is invalid.
```

Accedir al paquet de recursos

Els paquets de recursos per una aplicació **Struts** es carreguen quan s'inicia i cada paquet es representa en memòria per una instancia de la classe `org.apache.struts.util.MessageResources`.

El següent fragment de codi il·lustra l'obtenció en l'aplicació d' un valor per una clau determinada d'un recurs localitzat.

```
MessageResources mis =
MessageResources.getMessageResources("i18n.MessagesBundle");
HttpSession session = request.getSession(false);
Locale locale = (Locale) session.getAttribute(Globals.LOCALE_KEY);
String origen= mis.getMessage(locale, "meta.origen");
```

Localització /Internacionalització dins de LLOG

La localització dels usuaris dins l'aplicació LLOG es gestiona d'una forma especifica.

Les condicions especials d'una aplicació desenvolupada per a conviure dins l'entorn de la web obliguen a complir una sèrie de requeriments. Aquests requeriments són importants per tal de aconseguir que els documents HTML que genera l'aplicació siguin indexats correctament en els cercadors d'internet (Google, Bing, Yahoo) i els resultats de la cerca es corresponguin correctament a les paraules clau que introdueixen els usuaris en els cercadors. El tràfic generat per un cercador es una font molt important de visitants per al portal, i generar documents que siguin indexats correctament pot ser clau per l'èxit o fracàs d'una aplicació web.

En el següent enllaç podem veure una sèrie de directrius recomanades per Google a l'hora de facilitar la indexació dels documents en el seu cercador.

http://support.google.com/webmasters/bin/answer.py?hl=en&answer=35769.

Aquests requeriments han determinat per l'aplicació LLOG la forma en que es gestiona la localització dels usuaris i com a conseqüència la renderització dels documents i les URL que enllacen els documents.

LLOG conté paquets de recursos en tres llengües: català, castellà i anglès.

Generació de URL localitzades

Cada document HTML que representa, un anunci d'una habitació, un pis de lloguer o un anunci de demanda, està enllaçat mitjançat una URL localitzada.

Les URL localitzades es formen de la següent manera:

Figura 4-11 URLs localitzades

L'aplicació LLOG conté un filtre Servlet que comprova el valor de la clau i18n en l'esquema de la URL i selecciona el llenguatge a mostrar per l'aplicació.

S'ha emprat el criteri que segons el qual la selecció per part de l'usuari del llenguatge és més important que l'assignació automàtica de llenguatge segons els paràmetres del navegador, de manera que en la pàgina principal d'entrada, l'usuari pot escollir quina llengua desitja. Un cop la llengua ha estat seleccionada, es guarda en una *cookie* la clau amb el valor del llenguatge en el navegador, i a partir d'aquest moment totes les peticions són redireccionades a recursos amb la clau i18n corresponent a l'idioma seleccionat. D'aquesta forma sempre que l'usuari connecti amb LLOG es mostraran tots els recursos web en el llenguatge seleccionat.

Figura 4-12 Pàgina d'inici i selecció de llenguatge

Un altre motiu, es deu a que gran quantitat de visites són enviades per cercadors, inclòs per usuaris que ja han visitat l'aplicació anteriorment.

Quan un cercador mostra els resultats d'una cerca, usualment la URL de destí es correspon a pàgines indexades en un idioma en particular, relacionat amb l'idioma de la paraula clau de cerca. Si un usuari retorna a l'aplicació a una URL amb una clau i18n diferent a la del llenguatge escollit (clau present en la *cookie* del seu navegador), el filtre Servlet redireccionarà l'usuari al recurs localitzat amb l'idioma que havia estat seleccionat prèviament per l'usuari.

Veure Annex: Classe URLFilter

Filtre URLRewritter

El filtre **URLRewriter** (http://www.tuckey.org/urlrewrite/) és un filtre web en Java per a qualsevol contenidor de servlets o servidor d'aplicacions web JavaEE que permet reescriure les URL abans que arribin al codi.

Reescriure les URL és una pràctica molt comuna per al servidor web **Apache**, però la majoria de contenidors i servidors d'aplicacions java no inclouen una funcionalitat similar. En particular el **Apache Tomcat** no proveeix d'aquesta funcionalitat.

Els llocs web amb contingut dinàmic utilitzen URL que generen les pàgines des del servidor utilitzant els paràmetres del *query string*. Aquests paràmetres sovint es reescriuen per assemblar-se a les URL de les pàgines estàtiques en un lloc web amb una jerarquia de subdirectori. Per exemple, l'URL a una pàgina pot ser:

http://www.alquileroven.com/es/Room.do?reqCode=view&id_listing=45450

Però pot ésser reescrita com

http://www.alquilerjoven.com/es/room/view/45450

Reescritura de URLs

En el cas de LLOG, les URL localitzades inclouen aquesta partícula que informa l'idioma amb el qual es mostra el document.

```
http://localhost:80/es/Listing.do?reqCode=view&id_listing=45450
```

Cal eliminar aquesta partícula abans d'iniciar el processament de la petició pel framework **Struts** ja que s'interpretaria com una petició a un mòdul addicional de la petició, i no s'executaria correctament.

El filtre **URLRewriter** és l'encarregat d'eliminar aquesta partícula en les peticions que s'envien al servidor web, mitjançant la següent directiva:

```
<rule>
    <from>/([a-z]{2})/(.*)</from>
    <to>/$2</to>
  </rule>
```

Per exemple les peticions que es corresponguin a la URL:

http://www.lloguerjove.com**/es/**Room.do?reqCode=view&id_listing=45435

es redireccionaran a la següent URL:

http://www.lloguerjove.com/Room.do?reqCode=view&id_listing=45435

i seran processades correctament pel Servlet de processament del *framework* de **Struts**.

Per tal de generar correctament les URL localitzades en el document HTML que s'envia al client, cal afegir la següent regla en el filtre URLRewriter:

```
<outbound-rule>
        <from>^%{context-path}/(.*)</from>

    <to>/$1/%{session-attribute:language}/$2</to>
  </outbound-rule>
```

En aquest cas, el filtre *URLRewritter* afegirà a les URL dels documents HTML generats per l'aplicació la partícula `/lang/` corresponent a l'idioma amb el qual es presenta el document.

Per exemple, en el següent codi HTML:

```
<form id="searchFormRooms" method="GET" action="/SearchAction.do" class="searchForm" onsubmit="validate()"><input type="hidden" name="reqCode" value="rooms" id="reqCode">
</form>
```

El filtre URLRewriter reescriurà el codi HTML afegint la partícula corresponent a l'idioma en totes les referències a URLs:

```
<form id="searchFormRooms" method="GET" action="/es/SearchAction.do" class="searchForm" onsubmit="validate()"><input type="hidden" name="reqCode" value="rooms" id="reqCode">
</form>
```

Servei d'excepcions

La gestió d'excepcions permet informar que s'ha produït un error al realitzar una petició. Aquest error podrà ser tractar adequadament i en cas necessari informar a l'usuari, llençar una traça, enviar un correu, etc...

Tipus d'excepcions:

- Checked Exceptions: representen condicions invàlides a les zones fora del control immediat del programa (entrada d'usuari no vàlid, problemes de base de dades, interrupcions de la xarxa, arxius absents, etc). Hereta de `java.lang.Exception`. El llenguatge java obliga a que aquestes excepcions hagin de ser capturades mitjançant un bloc `try-catch` o bé declarar el mètode que es torna a llençar l'excepció (finalment algú l'ha de capturar) mitjançant la clàusula `throws`.

- Uncheked Exceptions: representa defectes en els programes (*bugs*). Són subclasses de `RuntimeException`, i usualment són implementades utilitzant `IllegalArgumentException`, `NullPointerException` o `IllegalStateException`.

El servei d'excepcions defineix un tipus d'excepció principal que es basa en *checked exceptions*.

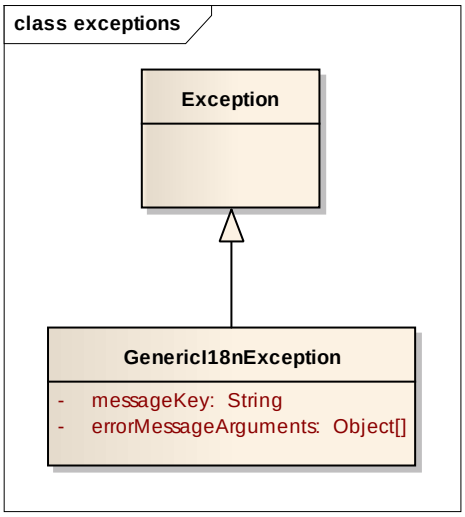


Figura 4-13 Classe principal del servei d'Excepcions

Excepció que hereta de la classe `java.lang.Exception`. Es tracta per tant d'una excepció tipus *checked*. Permet afegir un codi de error multi idioma per generar el missatge a l'usuari.

Veure Annex: Implementació GenericI18nException

Servei de seguretat

Apache Shiro (pronunciat "shee-oh", la paraula japonesa per a "castell") http://shiro.apache.org/ és un marc de treball de gran abast i fàcil d'utilitzar de seguretat de Java que porta a terme l'autenticació, l'autorització, la criptografia i gestió de sessions.

Apache Shiro és un marc integral de seguretat d'aplicacions amb moltes característiques. El següent diagrama mostra on **Shiro** centra la seva força:

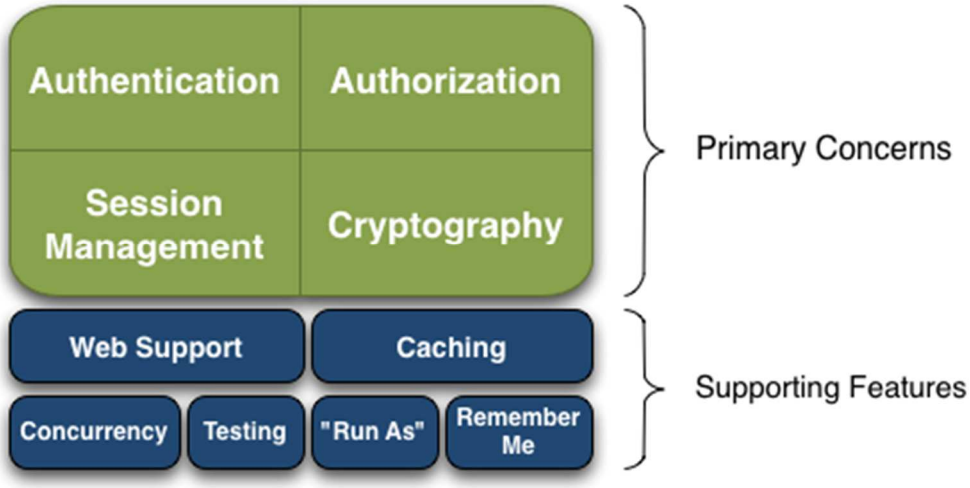


Figura 4-14 Funcions clau de Shiro

L'API de **Shiro** proveeix els següents serveis de seguretat:

- Autentificació: l'autenticació és el procés de validar la identitat d'un usuari que està intentant entrar en un sistema amb seguretat. Normalment conegut com a accés d'usuari o *login* , el sistema verifica que l'usuari actualment existeix i que ha proporcionat correctament les credencials, com la clau de pas (*password*).

- Autorització: Un cop l'usuari s'ha autentificat, aquest voldrà interactuar amb la aplicació. L'autorització implica determinar si l'usuari està autoritzat a realitzar determinada acció. En el cas de LLOG per exemple, els usuaris professionals amb ROLE=COMPANY poden gestionar les dades de la seva agència.
- Criptografia: protegir o amagar dades de manera que siguin intel·ligibles per a usuaris no autoritzats.
- Gestió de sessió: estat de cada usuari al llarg del temps.

Apache Shiro també proveeix de característiques addicionals per suportar i reforçar aquests punts en diversos entorns, especialment:

- Web Support: El suport de **Shiro** per la web ajuda a *securitzar* fàcilment aplicacions web.
- Caching: **Shiro** posa èmfasis en la memòria cau (*cache*) per tal que les operacions de seguretat siguin ràpides i eficients.
- Concurrència: **Apache Shiro** dóna suport a aplicacions amb múltiples fils (*multi-threaded)* amb les serves característiques de concurrència.
- *Testing*: Facilita el desenvolupament de tests unitaris i d'integració.
- "*Run As*": És una característica que permet als usuaris assumir la identitat d'un altre usuari (si tenen permisos), molt interessant en escenaris d'administració.
- "*Remember Me*": Recorda identitats d'usuaris entre sessions de manera que només cal autenticar-se quan és obligatori.

Arquitectura d'**Apache Shiro**:

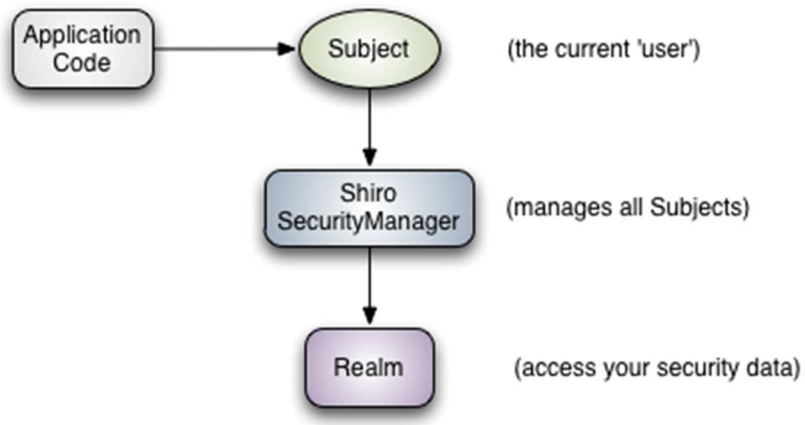


Figura 4-15 Arquitectura d'Apache Shiro

A nivell conceptual, l'arquitectura de **Shiro** presenta tres aspectes primaris principals:

- Subject: el `Subject` és una "vista" especifica de seguretat de l'usuari actual (un `Subject` pot representar un usuari en el sentit d'un ser humà, però també pot representar un servei d'una tercera part, un *daemon account*, etc... bàsicament qualsevol cosa que interaccioni amb el software. Les instàncies de `Subject` estan lligades (i requereixen) un Security Manager.
- Security Manager: El `Security Manager` és el component que coordina tots els components interns de seguretat.
- Realm: El `Realm` actua com el connector entre **Shiro** i les dades de seguretat de l'aplicació. Essencialment un `Realm` es un DAO específic per a dades relatives a seguretat. Encapsula els detalls particulars de les connexions a les fonts de dades i entrega les dades de seguretat al `Security Manager`(Veure Annex: Implementació Realm Shiro).

El `Security Manager` s'encarrega de gestionar l'ús dels diversos `Realms` i les dades d'indetificació i seguretat per tal que es representin com a instàncies de `Subject` .

Authenticació Java amb Apache Shiro

L'autentificació és el procés pel qual es verifica la identitat. Es realitza enviant els *Principals* i les credencials a **Shiro** per comprovar si coincideixen amb el que l'aplicació espera.

S'anomena *Principals* als atributs que identifiquen al `Subject`. El principal que s'utilitza en LLOG és l' email (únic per usuari en l'aplicació).

Les *credencials* son valors secrets coneguts pel `Subject` que s'usen com a evidència de que són pròpies o pertanyen a la identitat declarada. En el cas de LLOG s'utilitza una clau de pas (*password*).

En el *framework* **Shiro** el procés d'autentificació en Java es pot descompondre en diversos passos:

Pas 1 – Recollir els *principals* i les *credencials* del Subject.

```
      public class ProcessLoginAction extends
LLoguerJoveDispatchAction {

          ....

            public ActionForward unspecified(ActionMapping mapping,
ActionForm form,
              HttpServletRequest request, HttpServletResponse
response)
              throws Exception {

          ...

          String memberName = ParamUtil.getParameter(request,
"userlogin", true);
                  if(!GenericValidator.isEmail(memberNa
                      StringUtil.checkGoodName(memberNa

                  String memberPasswordSHA256 =
ParamUtil.getParameter(request, "md5pw", false);

                  UsernamePasswordToken token = new
UsernamePasswordToken( memberName,memberPasswordSHA256);

                  String
doquicklogin=request.getParameter("doquicklogin");

                  if (response != null &&  doquicklogin!=nu
doquicklogin.equals("on") ) {

                      //"Remember Me" built-in, just do this:
                      token.setRememberMe(true);
```

Recollir els *principals*

Recollir les credencials

Pas 2 – Enviar els *principals* i les credencials al sistema d'autentificació.

```
        Subject currentUser = SecurityUtils.getSubject();

        //Authenticate the subject by passing
       //the user name and password token
       //into the login method

       currentUser.login(token);
```

Pas 3 – Permetre accés, re intentar l'autentificació o bloquejar l'accés.

```
      try {
      currentUser.login(token);
      } catch ( UnknownAccountException uae ) { ...
      } catch ( IncorrectCredentialsException ice ) { ...
      } catch ( LockedAccountException lae ) { ...
      } catch ( ExcessiveAttemptsException eae ) { ...
      } ... catch your own ...
      } catch ( AuthenticationException ae ) {
      //unexpected error?
      }
      //No problems, show authenticated view…
```

Per sortir de la sessió, (*logging out*) quan l'usuari ha acabat amb la aplicació, s'utilitza una sola crida a un mètode.

```
      currentUser.logout();
      //removes all identifying information and invalidates their session
too.
```

Autorització Java amb Apache Shiro

L'autorització en Shiro es pot gestionar de 4 formes:

- Programàticament: Realitzar comprovacions de seguretat en el codi java amb estructures de l'estil *if else*.
- Anotacions JDK: Es poden vincular anotacions d'autorització en mètodes java.
- JSP/GSP TagLibs: Es pot controlar la sortida de la pagina jsp o gsp basat en rols i permisos.

En l'aplicació LLOG s'utilitzen **anotacions**.

Per tal d'utilitzar les anotacions Java, cal habilitar el suport per AOP. Per a l'aplicació LLOG s'utilitza AspectJ (https://eclipse.org/aspectj/), un llenguatge de programació orientat a aspectes construït com una extensió del llenguatge java.

Anotacions d'autorització en classe tipus `Action`:

```
public class CompanyAction extends LLoguerJoveDispatchAction{
......

    @RequiresAuthentication
    @RequiresRoles("COMPANY")
    public ActionForward edit(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
```

4.6 Desplegament en Contenidor web Tomcat

L'aplicació de LLOG es desplega en un contenidor de Servlets **Tomcat**.

Utilitzarem un diagrama de desplegament UML per descriure els diversos components de hardware, quins components de software s'executen en cada node i com es comuniquen.

En general, en un diagrama de desplegament podem veure:

- Els diferents components físics de hardware i les comunicacions entre ells.
- En quin node es desplega cada artefacte de software.
- Arquitectura tècnica del sistema.
- Configuracions de les instàncies en cada entorn particular.

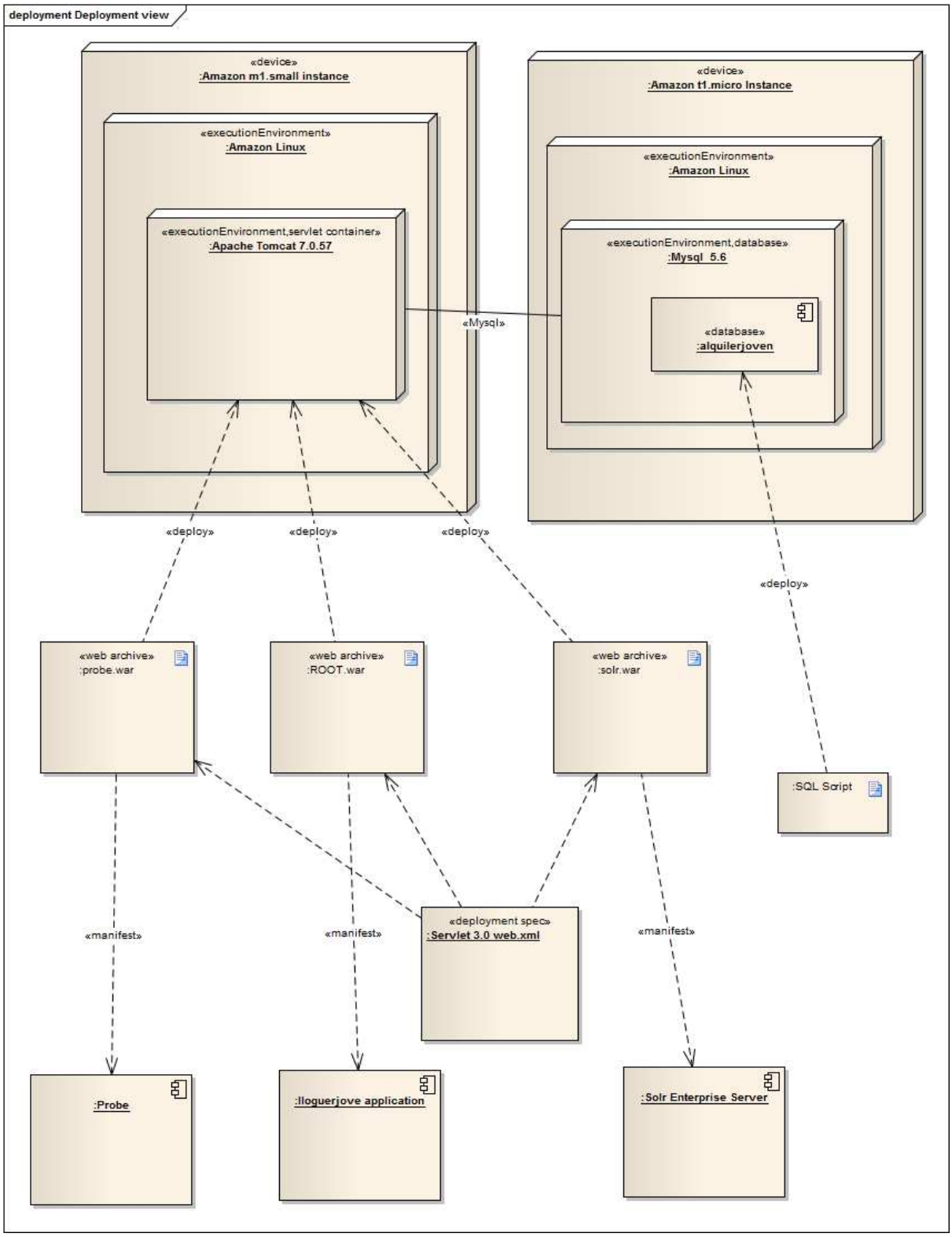


Figura 4-16 Diagrama de desplegament de l'aplicació LLOG

En l'aplicació LLOG s'utilitzen dues instàncies virtuals allotjades en el cloud d'Amazon (Amazon EC2). En una instància hi ha instal·lat el

contenidor de Servlets **Tomcat** (Amazon m1.small imatge) on es despleguen les aplicacions web (ROOT.war, solr.war), i en l'altra, hi ha instal·lada el servidor de base de dades relacional, **Mysql**.

5. Conclusions

En aquest projecte s'han descrit multitud de conceptes, sistemes i tecnologies.

Seguint aquests principis hem pogut descompondre la complexitat inicial del sistema i s'ha aconseguit desenvolupar una aplicació extensa i complexa.

Alhora la memòria del projecte esdevé un document tècnic que pot servir de guia a d'altres desenvolupadors per futures extensions o modificacions de l'aplicació.

També hem pogut demostrar l'efectivitat del disseny guiat per el domini. A partir del diagrama de objectes del domini es poden deduir els requisits del software, les funcions i el comportament.

La pròpia pervivència de la aplicació al llarg del temps és un exemple de l'efectivitat de l'aplicació d'aquests principis i tecnologies.

Experimentar amb totes aquestes interessants idees i eines i no construir un sistema de software productiu podria haver resultat en una experiència buida, inútil. Per contra, s'ha dissenyat una aplicació que es troba actualment online i en servei. Tot aquest temps que l'aplicació ha estat en online ha servit per posar de manifest la rellevància d'un bon disseny i arquitectura del sistema per facilitar el manteniment i realitzar millores o actualitzacions.

5.1 Línies futures

El futur del portal web depèn en gran mesura de la creació d'un model de negoci sostenible. Obviant les variables econòmiques, si fem una valoració estrictament tècnica, resultaria interessant proveir d'una api basada en REST(*Representational State Transfer*) per possibilitar la connexió de clients de diversa naturalesa (mòbils, tauletes, etc...) i introduir algunes de les noves tecnologies en el disseny de la infraestructura per automatitzar desplegaments i configuracions (*Docker*).

6. Annexos

6.1 Annex: Exemples de tipus MIME

Tipus de contingut/Subtipus	Extensió de l'arxiu
Arxius de text:	
text/html	html, htm
text/plain	txt, c, ec
text/richtext	rtx
text/tab-separated-values	tsv
text/x-speech	talk, spc
text/x-setext	etx
Arxius d'imatge	
image/gif	gif
image/ief	ief
image/jpeg	jpeg, jpg, jpe
image/tiff	tiff, tif
image/x-cmu-raster	ras
image/x-portable-anymap	pnm
image/x-portable-bitmap	pbm
image/x-portable-graymap	pgm
image/x-portable-pixmap	ppm
image/x-rgb	rgb
image/x-xbitmap	xbm
image/x-xpixmap	xpm

image/x-xwindowdump xwd

image/vasa mcf # Xspace para Mac

image/fif fif # Fractal imager

image/cis-cod cod # Lightning strike

Arxius d'audio

audio/basic au, snd

audio/x-midi mid, midi

audio/x-aiff aif, aiff, aifc

audio/x-wav wav

audio/voxware vox

audio/x-mod mod

audio/x-s3m s3m

audio/x-pn-realaudio ra, rv # Real audio

audio/x-pn-realaudio ram, rm

audio/x-pn-realaudio-plugin rpm

Arxius de vídeo/VRML

video/mpeg mpeg, mpg, mpe

video/quicktime qt, mov

video/x-msvideo avi

video/x-sgi-movie movie

video/vnd.vivo vivo, viv

x-world/x-vrml wrl

Arxius d'aplicació

application/oda	oda
application/pdf	pdf
application/postscript	ai, eps, ps
application/rtf	rtf
application/x-mif	mif
application/x-csh	csh
application/x-dvi	dvi
application/x-hdf	hdf
application/x-latex	latex
application/x-netcdf	nc, cdf
application/x-sh	sh
application/x-tcl	tcl
application/x-tex	tex
application/x-texinfo	texinfo, texi
application/x-troff	t, tr, roff
application/x-troff-man	man
application/x-troff-me	me
application/x-troff-ms	ms
application/x-wais-source	src
application/x-sprite	sprite, spr
application/futuresplash	spl
application/x-cdlink	vcd
application/dsptype	tsp
application/x-asap	asp
application/x-javascript	js

application/octet-stream bin, exe

application/zip zip

application/x-tar tar

application/mac-binhex40 hqx

application/x-shockwave-flash swf

application/x-director dcr

application/x-director dir

application/x-director dxr

application/x-envoy evy

application/vocaltec-media-desc vmd

application/vocaltec-media-file vmf

application/wpc wpc

application/mspowerpoint ppt, ppz, pps, pot

application/x-chat chat

application/x-streaming-audio key

6.2 Annex: Paràmetres de inicialització i configuració del servlet

Fragment de l'arxiu web.xml

```
<servlet>
 <servlet-name>action</servlet-name>
 <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
 <init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/classes/struts-config.xml</param-value>
 </init-param>

 <init-param>
  <param-name>convertNull</param-name>
  <param-value>true</param-value>
 </init-param>
 <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>action</servlet-name>
```

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

6.3 Annex: Implementacio thread local session

Hem implementat la segona part utilitzant un filtre Servlet. La tasca del filtre és tancar la Session abans que la reposta sigui enviada al client i també és el responsable de fer el commit de qualsevol transacció pendent a la base de dades.

Podem veure el mètode doFilter() d'aquest Servet en la següent figura:

Servlet

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
            throws IOException, ServletException {

             try {
            if(log.isDebugEnabled()){
                log.debug("Starting a database transaction");
            }

           hibernateSessionFactory.beginTransaction();
           chain.doFilter(request, response);
           hibernateSessionFactory.commitTransaction();
           hibernateSessionFactory.closeSession();

             } catch (StaleObjectStateException staleEx) {

                 if(log.isErrorEnabled()){
                   log.error("Error Stale",staleEx);
                 }

            try{
                hibernateSessionFactory.rollbackTransaction();

             }catch(Exception e){
```

```
            if(log.isErrorEnabled()){
                log.error(e);
            }
        }

        throw staleEx;

    } catch (Throwable ex) {

        if(log.isErrorEnabled()){
         log.error("Error Transaction",ex);
        }

        try{

            hibernateSessionFactory.rollbackTransaction();

            }catch(Exception e){
                if(log.isErrorEnabled()){
                log.error(e);
                }
            }
            throw new ServletException(ex);
    }
```

Factoria

```
package com.framework;


import javax.naming.Context;
import javax.naming.InitialContext;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.webpage.exceptions.InfrastructureException;

/**
 * Configures and provides access to Hibernate sessions, tied to the
 * current thread of execution.  Follows the Thread Local Session
 * pattern, see {@link http://hibernate.org/42.html}.
 */
public class HibernateSessionFactory {
    private static Log log =
LogFactory.getLog(HibernateSessionFactory.class);
   private static String CONFIG_FILE_LOCATION =
"/hibernate/config/hibernate.cfg.xml";

    /** Holds a single instance of Session */
   private static final ThreadLocal<Session> threadLocal = new
ThreadLocal<Session>();
```

```
   private static final ThreadLocal<Transaction>  threadTransaction = new
ThreadLocal<Transaction> ();

    /** The single instance of hibernate configuration */
   //private static final Configuration cfg = new Configuration();

    /**
     * The instance of hibernate SessionFactory
     */
   private  org.hibernate.SessionFactory sessionFactory;

    /**
     * Returns the ThreadLocal Session instance.  Lazy initialize
     * the <code>SessionFactory</code> if needed.
     *
     *  @return Session
     *  @throws HibernateException
     */
   public Session currentSession() throws HibernateException {
        Session session = (Session) threadLocal.get();
       if (session != null && ! session.isConnected())
            session = null;
        if (session == null) {
            if (this.sessionFactory == null) {
                try {
                     Context ctx = new InitialContext();
                    this.sessionFactory = (SessionFactory)
ctx.lookup("foo:/hibernate/SessionFactory");
                  if(log.isDebugEnabled()){
                      log.debug("New Session Factory");
                   }
                }
               catch (Exception e) {
                  e.printStackTrace();
                    throw new HibernateException ("%%%% Error Creating
SessionFactory %%%%");
                }
            }
            session = this.sessionFactory.openSession();

            threadLocal.set(session);
        }
        return session;
    }

    /**
     *  Close the single hibernate session instance.
     *
     *  @throws HibernateException
     */
    public  void closeSession() throws Exception {
        try{
            Session session = (Session) threadLocal.get();
            threadLocal.set(null);

            if (session != null  && session.isOpen()) {
                session.close();
            }
        }catch(HibernateException ex){
        if(log.isErrorEnabled()){
```

```
            log.error("Cannot close session");
        }
        throw new InfrastructureException(ex);
    }
    }

   static private HibernateSessionFactory instance = null;
    public static synchronized HibernateSessionFactory
getInstance(SessionFactory sessionFactory) {
        if(instance== null){
        instance=new HibernateSessionFactory(sessionFactory);
        }
         return instance;
        }
    private HibernateSessionFactory() {
    }

   private HibernateSessionFactory(SessionFactory sessionFactory) {
               setSessionFactory(sessionFactory);
            }
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
             }

   public void beginTransaction() throws Exception{

        Transaction tx = (Transaction) threadTransaction.get();
        try{
            if (tx != null && !tx.isActive()) {
                    tx = null;
                    threadTransaction.set(null);
            }
            if(tx==null){

         if (log.isDebugEnabled()) {
             log.debug("Starting new database transaction in this
thread.");
          }
          if (threadLocal.get() != null && threadLocal.get().isOpen()) {
              threadLocal.get().close();
              threadLocal.set(null);
          }
          tx = currentSession().beginTransaction();
          threadTransaction.set(tx);
        }else {
          if (log.isDebugEnabled()) {
              log.debug("Using current database transaction in this
thread.");
              log.debug("Opening new Session for this thread.");
          }
    }
    }catch(HibernateException ex){
        if(log.isErrorEnabled()){
               log.error("Cannot begin transaction");
        }
        throw new InfrastructureException(ex);
    }


    }
```

```
    public  void commitTransaction() throws Exception{
    try{

      Transaction tx = (Transaction) threadTransaction.get();
      if(tx!=null &&! tx.wasCommitted() && !tx.wasRolledBack()){
        currentSession().flush();
            tx.commit();
      }
    }catch(HibernateException ex){

        throw  ex;
    }
    }

      public  void rollbackTransaction() throws Exception {

    Transaction tx = (Transaction) threadTransaction.get();

    try{

        threadTransaction.set(null);
        if(tx!=null &&! tx.wasCommitted() && !tx.wasRolledBack()){
        if(log.isErrorEnabled()){
                log.error("Trying to rollback database transaction after
exception");
             }
        tx.rollback();
        }


    }catch(HibernateException ex){

        if(log.isErrorEnabled()){
         log.error("Could not rollback transaction after exception!");
       }

        throw new InfrastructureException(ex);

    }finally{

        closeSession();
        if(log.isErrorEnabled()){
             log.error("Close session after rollback");
            }
    }

    }

    public  void rollbackTransaction2() throws Exception {

        Transaction tx = (Transaction) threadTransaction.get();
        try{

            threadTransaction.set(null);
            if(tx!=null &&! tx.wasCommitted() && !tx.wasRolledBack()){
            if(log.isErrorEnabled()){
                    log.error("Trying to rollback database transaction
after exception");
            }
            tx.rollback();
```

```
            }


        }catch(HibernateException ex){

            if(log.isErrorEnabled()){
             log.error("Could not rollback transaction after exception!");

            }

            throw new InfrastructureException(ex);
            }
        }
}
```

6.4 Annex: Classe URLFilter

```
public class URLFilter
 {
        private static Logger log =
oggerFactory.getLogger(URLFilter.class);
    private String lang=null;
        String responseURI = null;
        String requestURI = null;
        String context = null;
        String query=null;
        HttpSession session=null;

public URLFilter(String lang,HttpServletRequest request,
HttpServletResponse response){

        this.lang=lang;
        this.requestURI = request.getRequestURI();
        this.query=request.getQueryString();
        this.context=request.getContextPath();
        session=request.getSession();
}

public static String getLanguageFromURL(String requestURI ){

        String context=removeContext(requestURI);
        String parts[]=context.split("/");
        return parts[1];
}

public static boolean isIntro(String requestURI ){
```

```
    String URI =removeContext(requestURI );
    String parts[]=URI.split("/");
    if(parts[2].equals("Intro.jsp") || parts[2].equals("Intro.do") )
        return true;
    else
return false;
}
public static String removeContext(String requestURI){

 String context=LloguerJoveConfig.getContext();
     if(!context.equals("/")){

        String stringRetorn[]=requestURI.split(context);
        String responseString="/".concat(stringRetorn[1]);
        return responseString;
     }else
     return requestURI;


    }


public  boolean checkLangByURL()  {

    String comparador="/"+lang+"/";
    boolean found=false;
    for(int i=0;i<=requestURI.length()-comparador.length();i++){
         found =
requestURI.regionMatches(true,i,comparador,0,comparador.length());
                 if (found==true)
                 {

                         return true;
                 }
        }


        return false;


        }

    public String getURL(){
        try{

            if(requestURI!=null && !requestURI.endsWith("/") ){


responseURI=requestURI.replaceFirst("/../","/"+lang+"/");

                        if(this.query!=null){
                        responseURI=responseURI+"?"+query;
                        }
                        return (responseURI);
            }else{

                return ("es/Intro.jsp");
            }
        }catch(Exception e){
            if(log.isErrorEnabled()){
```

```
                log.error(e);
            }
        return ("es/Intro.jsp");
    }

}

}
```

6.5 Annex: Implementació AbstractDAO Factory

```
package com.alquilerjoven.core.db.dao;

import com.alquilerjoven.core.solr.SolrDAOFactory;


/**
 * @author  admin
 */
public abstract class AbstactDAOFactory {


    // List of DAO types supported by the factory

      public static final int HIBERNATE = 2;
      public static final int SOLR = 3;

      // There will be a method for each DAO that can be
      // created. The concrete factories will have to
      // implement these methods.
      /**
     */
            public abstract UserTypeDAO getUserTypeDAO();


        /**
         */
        public abstract PaisDAO getPaisDAO();
        /**
         */
        public abstract ThreadDAO getThreadDAO();

        /**
         */
        public abstract CategoriaDAO getCategoriaDAO();
        /**
         */
        public abstract FileDAO getFileDAO();
```

```
/**
 */
public abstract  UserDAO getUserDAO();
/**
 */
public abstract  ListingDAO getListingDAO();
/**
 */
public abstract ProductDAO getProductDAO() ;

/**
 */
public abstract FavoriteDAO getFavoriteDAO();
/**
 */
public abstract UserProducteDAO getUserProducteDAO();
/**
 */
public abstract RenterDAO getRenterDAO() ;

/**
 */
public abstract PostDAO getPostDAO() ;
/**
 */
public abstract RoomDAO getRoomDAO();
/**
 */
public abstract RenterRoomDAO getRenterRoomDAO();
/**
 */
public abstract RentFlatDAO getRentFlatDAO() ;

/**
 */
public abstract ListingTypeDAO getListingTypeDAO();

/**
 */
public abstract FacetDAO getFacetDAO();

/**
 */
public abstract CalendarListingDAO getCalendarListingDAO();
/**
 */
public abstract MessageDAO getMessageDAO() ;
/**
 */
public abstract MunicipiDAO getMunicipiDAO();
/**
 */
public abstract EventsDAO getEventsDAO();

/**
 */
public abstract CercaDAO getCercaDAO();

/**
 */
```

```
    public abstract DistrictDAO getDistrictDAO();

    /**
     */
    public abstract NeighborhoodDAO getNeighborhoodDAO();
    /**
     */
    public abstract BidDAO getBidDAO() ;
    /**
     */
    public abstract IpDAO getIpDAO();
    /**
     */
    public abstract OrderDAO getOrderDAO();

    /**
     */
    public abstract RatingDAO getRatingDAO();
    /**
     */
    public abstract ProvinceDAO getProvinceDAO();

    /**
     */
    public abstract ReservationDAO getReservationDAO();

    /**
     */
    public abstract ComunitatDAO getComunitatDAO();

    public abstract ReviewDAO getReviewDAO();

    public abstract FacebookUserDAO getFacebookUserDAO();

    public abstract CompanyDAO getCompanyDAO();

    public abstract BankAccountDAO getBankAccountDAO();

    static private DAOFactory instance = null;

    protected void finalize() throws Throwable {
        try {
              // close open files
        } finally {
            super.finalize();
        }
    }

  public static AbstactDAOFactory getDAOFactory(
      int whichFactory) {

    switch (whichFactory) {
      case HIBERNATE:

          return DAOFactory.getInstance();
      case SOLR    :
          return SolrDAOFactory.getInstance();

      default           :
          return DAOFactory.getInstance();
```

```
        }
    }
}
```

6.6 Annex: Implementació classe Validador del tipus d'arxiu imatge

```
public class ImageFileValidator implements Serializable {

...

public static boolean validateImageFile(Object bean, ValidatorAction va,
        Field field, ActionMessages errors, Validator validator,
        HttpServletRequest request) {

    Var variable = field.getVar("format");
    String values = variable.getValue();
    String endValues[] = values.split(",");

    String value = ValidatorUtils.getValueAsString(bean, field
        .getProperty());

    if (!GenericValidator.isBlankOrNull(value)) {

        try {
        FormFile file = (FormFile) PropertyUtils.getSimpleProperty(
            bean, field.getProperty());
        String filename = file.getFileName().toLowerCase();
```

```
        for (int i = 0; i < endValues.length; i++)
            if (filename.endsWith(endValues[i])) {


            return true;
            }

        errors.add(field.getKey(), Resources.getActionMessage(request,
            va, field));

        return false;

        } catch (NoSuchMethodException ex) {
        errors.add(field.getKey(), Resources.getActionMessage(request,
            va, field));
        return false;

        } catch (InvocationTargetException ex) {
        errors.add(field.getKey(), Resources.getActionMessage(request,
            va, field));
        return false;

        } catch (IllegalAccessException ex) {
        errors.add(field.getKey(), Resources.getActionMessage(request,
            va, field));
        return false;
        }

    } else {
        return true;
```

```
    }


    }


}
```

6.7 Annex: Servei de validació

```
OpenFrameFormTag.ServerValidation = Class.create();
OpenFrameFormTag.ServerValidation.prototype=Object.extend(new
OpenFrameFormTag.Base(), {

  initialize: function(url,options) {
    this.url = url;
    this.setOptions(options);
    this.setListeners();
  },


   setOptions: function(options) {
    this.options = Object.extend({
       eventType: options.eventType ? options.eventType : "submit"
      }, options || {});
  },
    setListeners: function() {

    Event.observe($(this.options.sourceForm),
                  this.options.eventType,
                  this.execute.bindAsEventListener(this),false);

    eval("$(this.options.sourceForm).on"+this.options.eventType+" =
function(){return false;};");
    },

  // Create bean with values to send to validation service
  validateForm: function(sourceForm,validatorName) {


      dwr.engine.setErrorHandler(errorHandling);

        // Obtain all components in form
        if (sourceForm) {
            // form expected
            //list = sourceForm.elements;

            // USE this instead of above buggy IE elements function
             // use http://api.jquery.com/input-selector/
```

```
            //list =$(sourceForm).getElements();
            //list =$(sourceForm).getElements();


                bean = new Object();
                var i=0;

            jQuery("#"+sourceForm.id).find(':input').each(

                 function(index){
                     //var element = list[i];

                    var element = jQuery(this);

                    var name=element.attr('name');
                    var value=element.val();
                    var id_element=element.attr('id');
                    if(!element.is(':hidden')){
                        var idErrorSttring="#"+sourceForm.id+" #"+
jQuerySelectorEscape(id_element)+ "Error";
                        jQuery(idErrorSttring).hide();

                    }
                    element.removeClass('error');


                if (name) {



                    if (element.is(':checkbox')){

                        if( element.is(':checked') ){
                            bean[name] = value;

                            }else{
                                bean[name] = '';

                            }
                    }else   if (element.is(':radio')){

                               if(element.attr("checked")=='checked'){
                                bean[name] = value;
                                console.log(name+":"+value );
                        }else{

                            }


                    }else if (jQuery("#"+sourceForm.id).filter("
input[id=\'"+id_element+"\']").length>1  ){
                        jQuery("[id=\'"+id_element+"\']").each(function(){

                            if(jQuery(this).is(':radio')){
                                console.log("elemen radio");
                                if(
jQuery(this).attr("checked")=='checked'){
                                        bean[name] = jQuery(this).val();

                                        }else{
```

```
                                        //bean[element.name] = '';
                                    }

                            }
                            });


                        }else{
                            bean[name] = value;
                        }

                    }


                     });

            }


            if (validatorName) {


                var dataFromBrowser = new Object();
                dataFromBrowser['sourceForm'] = sourceForm;


                Demo.getValidationResults(validatorName,bean,
                {
                    callback:function(dataFromServer) {
                        callbackValidateForm(dataFromServer,dataFromBrowser);
                    },
                    timeout:5000,
                    async:false
                }
            );


            }

        },

        execute: function(e) {


            if (this.options.preFunction != null) this.options.preFunction();
            // DWR calling

           if (!validated) {

           //debugger;
               // EVC: added deny to press submit button more than once.
               $(_lastClickingElem).disabled=true;

               _disabledElem = _lastClickingElem;


this.validateForm($(this.options.sourceForm),this.options.validatorName);
```

```
            Event.stop(e);

            if($(_disabledElem))
                    $(_disabledElem).disabled=false;


        }


       if(validated){

        if (this.options.submitFunction != null) {

                  validated = false;
                  this.options.submitFunction();


        } else{
                  validated = false;

                   $(_lastClickingElem).disabled=true;



                  $(this.options.sourceForm).submit();

              }


         }else{
          if (this.options.postFunction != null)
this.options.postFunction();
          }

   }

});
```

6.8 Annex: Mètode per a l'enviament d'email utilitzant javax.mail api

```
    public static void sendMail(Collection mailStructCollection,String
contentType)

    throws MessagingException, BadInputException,
UnsupportedEncodingException {

    Session session = null;
    Transport transport = null;
```

```
    int totalEmails = mailStructCollection.size();
    int count = 0;
    int sendFailedExceptionCount = 0;
    try {
        for (Iterator iter = mailStructCollection.iterator();
iter.hasNext(); ) {
            count++;

            if ((transport == null) || (session == null)) {
                Properties props = new Properties();
                props.put("mail.smtp.starttls.enable", "true");
                props.put("mail.smtp.host", mailOption.mailServer);
                props.put("mail.smtp.port",
String.valueOf(mailOption.port));

                props.put("mail.transport.protocol", "smtp");
                props.put("mail.smtp.auth", "true");
                props.put("mail.debug", DEBUG);

                Authenticator auth = new SMTPAuthenticator();
                session = Session.getDefaultInstance(props, auth);
                transport = session.getTransport();


                    transport.connect();
                //}
            }

            MailMessageStruct mailItem = (MailMessageStruct)iter.next();

            String from = mailItem.getFrom();
            String to = mailItem.getTo();
            String cc = mailItem.getCc();
            String bcc = mailItem.getBcc();
            String subject = mailItem.getSubject();
            String message = mailItem.getMessage();

            if (from == null) from = mailOption.defaultMailFrom;

            try {
                // this will also check for email error
                checkGoodEmail(from);
                InternetAddress fromAddress = new InternetAddress(from);
                InternetAddress[] toAddress = getInternetAddressEmails(to);
                InternetAddress[] ccAddress = getInternetAddressEmails(cc);
                InternetAddress[] bccAddress =
getInternetAddressEmails(bcc);
                if ((toAddress == null) && (ccAddress == null) &&
(bccAddress == null)) {
                    throw new BadInputException("Cannot send mail since all
To, Cc, Bcc addresses are empty.");
                }

                // create a message
                Message msg = new MimeMessage(session);
                msg.setSentDate(new Date());
                msg.setFrom(fromAddress);

                if (toAddress != null) {
                    msg.setRecipients(Message.RecipientType.TO, toAddress);
```

```
                }
                if (ccAddress != null) {
                    msg.setRecipients(Message.RecipientType.CC, ccAddress);
                }
                if (bccAddress != null) {
                    msg.setRecipients(Message.RecipientType.BCC,
bccAddress);
                }
                msg.setSubject(subject);

                msg.setText(message);
                msg.setHeader("Content-Type", "text/html;charset=UTF-8");
                msg.saveChanges();

                transport.sendMessage(msg, msg.getAllRecipients());

                // now check if sent 100 emails, then close connection
(transport)
                if ((count % MAX_MESSAGES_PER_TRANSPORT) == 0) {
                    try {
                        if (transport != null) transport.close();
                    } catch (MessagingException ex) {}
                    transport = null;
                    session = null;
                }
            } catch (SendFailedException ex) {
                sendFailedExceptionCount++;
                log.error("SendFailedException has occured.", ex);
                log.warn("SendFailedException has occured. Detail info:");
                log.warn("from = " + from);
                log.warn("to = " + to);
                log.warn("cc = " + cc);
                log.warn("bcc = " + bcc);
                log.warn("subject = " + subject);
                log.info("message = " + message);
                if ((totalEmails != 1) && (sendFailedExceptionCount > 10))
{
                    throw ex;// this may look redundant, but it is not :-)
                }
            } catch (MessagingException mex) {
                log.error("MessagingException has occured.", mex);
                log.warn("MessagingException has occured. Detail info:");
                log.warn("from = " + from);
                log.warn("to = " + to);
                log.warn("cc = " + cc);
                log.warn("bcc = " + bcc);
                log.warn("subject = " + subject);
                log.info("message = " + message);
                throw mex;// this may look redundant, but it is not :-)
            }
        }
    } finally {
        try {
            if (transport != null) transport.close();
        } catch (MessagingException ex) { }
        if (totalEmails != 1) {
            log.info("sendMail: totalEmails = " + totalEmails + " sent
count = " + count);
        }
    }
```

```
}
```

6.9 Annex: Implementació GenericI18nException

```
package com.alquilerjoven.core.exceptions;

public class GenericI18nException extends Exception {

    /**
     *
     */
    public GenericI18nException(Throwable cause) {
        super(cause);
    }

    public GenericI18nException(){

    }

    public GenericI18nException(String message) {

            super(message);
    }
    private static final long serialVersionUID = 7592081843860432l7L;

    public String messageKey;

    public Object[] errorMessageArguments;

    /**
     * @return
     */
    public Object[] getErrorMessageArguments() {
        return errorMessageArguments;
    }
```

```
/**
 * @param value
 */
public void setErrorMessageArguments(Object... value) {
    this.errorMessageArguments = value;
}

/**
 * @return
 */
public String getMessageKey() {
    return messageKey;
}

/**
 * @param messageKey
 */
```

6.10 Annex: Implementació Realm Shiro

```
package com.webpage.auth;

import java.util.Collection;
import java.util.Set;

import org.apache.shiro.authc.AccountException;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authc.UnknownAccountException;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.authz.AuthorizationException;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.config.ConfigurationException;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.util.ByteSource;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.alquilerjoven.core.db.dao.AbstactDAOFactory;
import com.alquilerjoven.core.exceptions.ObjectNotFoundException;
import com.alquilerjoven.core.exceptions.PersistenceServiceException;

public class HibernateRealm extends  AuthorizingRealm {


    public enum SaltStyle {NO_SALT, CRYPT, COLUMN, EXTERNAL};
    protected SaltStyle saltStyle = SaltStyle.NO_SALT;

    protected boolean permissionsLookupEnabled = false;

    private static Logger log =
LoggerFactory.getLogger(HibernateRealm.class);

    @Override
    protected AuthenticationInfo
doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {


        // Auto-generated method stub
            UsernamePasswordToken upToken = (UsernamePasswordToken) token;
            String username = upToken.getUsername();

            // Null username is invalid
            if (username == null) {
                throw new AccountException("Null usernames are not allowed
by this realm.");
            }
```

```
            SimpleAuthenticationInfo info = null;

            try {


                String password = null;
                String salt = null;
                switch (saltStyle) {
                case NO_SALT:
                    password = getPasswordForUser(username);
                    break;
                case CRYPT:
                    // TODO: separate password and hash from
getPasswordForUser[0]
                    throw new ConfigurationException("Not implemented
yet");
                    //break;
                case COLUMN:
                     throw new ConfigurationException("Not implemented
yet");
                case EXTERNAL:
                     throw new ConfigurationException("Not implemented
yet");
                }

                if (password == null) {
                    throw new UnknownAccountException("No account found for
user [" + username + "]");
                }

                info = new SimpleAuthenticationInfo(username,
password.toCharArray(), getName());

                if (salt != null) {
                    info.setCredentialsSalt(ByteSource.Util.bytes(salt));
                }
            } catch (PersistenceServiceException e) {
                final String message = "There was an error while
authenticating user [" + username + "]";
                if (log.isErrorEnabled()) {
                    log.error(message, e);
                }

                // Rethrow any SQL errors as an authentication exception
                throw new AuthenticationException(message, e);
            } catch (ObjectNotFoundException e) {
                // TODO Auto-generated catch block
                  final String message = "There was an error while
authenticating user [" + username + "]";
                    if (log.isErrorEnabled()) {
                        log.error(message, e);
                    }

                    // Rethrow any SQL errors as an authentication
exception
                    throw new AuthenticationException(message, e);

            }
```

```
            return info;
    }



    protected String getPasswordForUser(String username) throws
PersistenceServiceException,ObjectNotFoundException{
        String passwd =
AbstactDAOFactory.getDAOFactory(AbstactDAOFactory.HIBERNATE).getUserDAO().g
etPasswordByUsername(username);
        return passwd;
    }

    protected Set<String> getRoleNamesForUser(String username)throws
PersistenceServiceException,ObjectNotFoundException{
        Set<String>
roles=AbstactDAOFactory.getDAOFactory(AbstactDAOFactory.HIBERNATE).getUserD
AO().getRoleNames(username);
        return roles;
    }



     /**
        * This implementation of the interface expects the principals
collection to return a String username keyed off of
        * this realm's {@link #getName() name}
        *
        * @see
#getAuthorizationInfo(org.apache.shiro.subject.PrincipalCollection)
        */
       @Override
       protected AuthorizationInfo
doGetAuthorizationInfo(PrincipalCollection principals) {

           //null usernames are invalid
           if (principals == null) {
               throw new AuthorizationException("PrincipalCollection
method argument cannot be null.");
           }

           String username = (String) getAvailablePrincipal(principals);
           Set<String> roleNames = null;
           Set<String> permissions = null;

           try {

               // Retrieve roles and permissions from database
               roleNames = getRoleNamesForUser( username);
               if (permissionsLookupEnabled) {
                   permissions = getPermissions(username, roleNames);
               }

           } catch (Exception e) {
               final String message = "There was an error while
authorizing user [" + username + "]";
```

```
                if (log.isErrorEnabled()) {
                    log.error(message, e);
                }

                // Rethrow any SQL errors as an authorization exception
                throw new AuthorizationException(message, e);
            } finally {

            }

            SimpleAuthorizationInfo info = new
SimpleAuthorizationInfo(roleNames);
            info.setStringPermissions(permissions);
            return info;

        }
        protected Set<String> getPermissions(String username,
Collection<String> roleNames) throws Exception {
            return null;
        }


}
```

7. Referències

Apache Shiro | Java Security Framework [En línia] / aut. The Apache Software Foundation // Apache Shiro | Java Security Framework. - http://shiro.apache.org/.

Commons Validator [En línia]. - http://commons.apache.org/validator/.

Core J2EE Patterns: Best Practices and Design Strategies (Second Edition) [Llibre] / aut. Deepak Alur, Dan Malks i John Crupi. - [s.l.] : Prentice Hall, 2003.

Design Patterns: Elements of Reusable Object-Oriented Software. [Llibre] / aut. Gamma, Erich et al. - [s.l.] : Addison Wesley, 1995.

Design Patterns:Elements of Reusable Object-Oriented Software (APC) [Llibre] / aut. Gamma,Erich; Helm , Richard;Johson Ralf;Vlissides,John. - [s.l.] : Addison-Wesley, 1994.

Direct Web Remoting [En línia] / aut. Walker, David Marginian & Joe // DWR - Easy Ajax for JAVA. - http://directwebremoting.org/dwr/index.html.

Domain-Driven Design:Tackling Complexity in the Heart of Software [Llibre] / aut. Evans Eric. - [s.l.] : Addison-Wesley, 2004.

EJB 3 in Action [Llibre] / aut. Debu Panda, Reza Rahman, Derek Lane. - [s.l.] : Manning Publications, 2007.

Enterprise Java Beans 3.0 [Llibre] / aut. Bill Burke,Richard Monson-Haefel. - 2006.

Freemarker java template engine [En línia] / aut. The Apache Software Foundation. // Apache FreeMarker. - What is Apache FreeMarker. - http://freemarker.org/.

Friendly url [En línia] / aut. Sharpened Productions // techterms.com. - http://www.techterms.com/definition/friendly_url.

Hibernate in Action [Llibre] / aut. Christian Bauer,Gavin King. - [s.l.] : Manning, 2005.

Jakarta Struts [Llibre] / aut. Cavaness Chuck. - [s.l.] : o'reilly, 2004.

Java Security Handbook [Llibre] / aut. Paul J. Perrone; Venkata S.R. Krishna Chaganti; Jamie Jaworski. - [s.l.] : Sams, 2000.

Load testing with apache jmeter [Online] / auth. agileblaze // Agileblaze. - 18 october 2010. - http://agileblaze.com/blog/load-testing-with-apache-jmeter/.

Logging Services [En línia] / aut. Apache Software Foundation // Apache Log4j. - http://logging.apache.org/log4j/1.2/.

Mapping objects to relational databases / aut. Ambler Scott. - 2002.

Pro Java EE Spring Patterns : Best practices and design strategies implementing Java EE patterns [Llibre] / aut. kayal Dhrubojyoti. - 2008.

Prototype Javascript Framework [En línia] / aut. Prototype Core Team // Prototype Javascript Framework. - http://prototypejs.org/.

SEOConsultants [Online] / auth. Powell Thomas A. and Lima Joe. - Port80 Software. - http://www.seoconsultants.com/articles/1000/urls.

Simple Logging Facade for Java [En línia] / aut. Quality Open Software // Simple Logging Facade for Java. - http://www.slf4j.org.

Sitepoint [Online] / auth. Jones Eric. - November 5, 2007. - http://www.sitepoint.com/friendly-urls/.

Sun Certified Enterprise Architect for Java EE Study Guide, 2nd Edition [Llibre] / aut. Mark Cade, Humphrey Sheil. - [s.l.] : Prentice Hall, 2010.

The Unified Software Development process [Llibre] / aut. Jacobson, I. Booch, G., and Rumbaugh, J.. - [s.l.] : Addison-Wesley, 1999.

UML y patrones [Llibre] / aut. Larman Craig. - Madrid : Pearson Educacion S.A., 2003.

What is Domain Driven Design? [En línia] / aut. Domain Language, Inc. and contributors // DDD Community. - 28 / 03 / 2007. - http://dddcommunity.org/learning-ddd/what_is_ddd/.