Framework persistència J2EE

Memòria

Alumne: Sergio Maeso García

Enginyeria en Informàtica

Consultor: Josep Mª Camps Riba

18 de Juny de 2012

Llicència d'ús

Dedicatòria

Dedicat a la meva família, amics i a la meva parella per suportar tots els moments crítics alhora de realitzar aquest projecte.

Resum del projecte

Aquest projecte tracta sobre el desenvolupament d'un marc de treball o framework destinat a la capa de persistència de les aplicacions J2EE.

En una primera fase es realitza un estudi de mercat per a analitzar els frameworks que existeixen actualment. Es troba que actualment existeix un estàndard J2EE a nivell de persistència de base de dades relacionals: JPA 2.0, a més d'altres que opten per un altre enfocament, com MyBatis. Agafant com a model l'estàndard JPA, s'observa la existència de diferents implementacions que el compleixen. S'analitzen aquestes implementacions per a aconseguir que el nostre framework tingui les característiques mínimes per a poder complir-lo, o s'hagi d'adaptar lo mínim possible.

En la segona fase es realitza el disseny de framework tenint sempre present l'estàndard JPA per aconseguir que el framework de persistència o Fwkpers, pugui en un futur implementar l'estàndard de persistència J2EE. En aquesta línia es realitza el disseny de Fwkpers, realitzant test unitaris conforme s'avança en la implementació.

Per finalitza es construeix una petita i senzilla aplicació web que realitza totes les accions bàsiques de persistència alhora del tractament de registres, i de cerques, d'aquesta forma es comprova que Fwkpers funciona sense problemes.

Paraules clau: j2ee, persistència, framework, marc de treball, JPA, Hibernate, fwkpers.

Àrea: J2EE.

Índex

1 Introducció 8
1.1 Descripció del projecte 8
1.2 Objectius generals i específics 10
1.3 Planificació 10
1.4 Productes obtinguts 13
2 Estudi de mercat 13
2.1 Aplicacions multicapa 13
2.2 Capa de persistència 14
2.3 Frameworks de persistència 15
2.3.1 EJB 15
2.3.2 Java Persistence API (JPA) 17
2.3.3 Hibernate 19
2.3.4 Oracle TopLink 22
2.3.5 MyBatis 23
2.3.6 Java Data Objects (JDO) 24
2.4 Comparativa de frameworks 25
2.5 Necessitats bàsiques del framework 26
3 Disseny del framework 27
3.1 Anàlisi de funcionalitats 27
3.1.1 Casos d'ús 28
3.1.2 Diagrama de classes 30
3.2 Diagrama de seqüències 37
3.2.1 Creació d'una taula 37
3.2.2 Eliminació d'una taula 38
3.2.3 Creació d'una base de dades 39
3.2.4 Eliminació d'una base de dades 40
3.2.5 Inserció, modificació i eliminació d'un registre 40
3.2.6 Consulta d'un registre per clau primària 42
3.2.7 Consulta de registres 42

3.2.8 Cerques amb filtre 43
3.2.9 Cerques amb sentència SQL 44
3.2.10 Cerques amb llenguatge Fwkql 45
4 Implementació 46
4.1 Definició model / Anotacions 47
4.2 Connexió Base Dades 48
4.3 Mapeig i Registre d'objectes 49
4.4 Nucli 50
4.5 Utilitats Cerca 52
4.6 Excepcions 53
5 Aplicació d'exemple 54
6 Conclusions 61
7 Glossari 62
8 Bibliografia 63
9 Annex: Entorn de treball 64
10 Annex: Base de dades 66

Índex de figures

Figura 1: Arquitectura en capes. 13
Figura 2: Estructura d'un JAR EJB. 16
Figura 4: Exemple codi tractament amb *Entity Manager*. 18
Figura 5: Classes del paquet de persistència. 19
Figura 6: Arquitectura Hibernate. 20
Figura 7: Exemple d'entitat Hibernate. 21
Figura 8: Fitxer configuració hibernate.cfg.xml. 21
Figura 9: Arquitectura Oracle TopLink. 22
Figura 10: Fitxer configuració consultes MyBatis. 23
Figura 11: Arquitectura MyBatis 24
Figura 12: Classes del paquet jdo. 25
Figura 13: Casos d'ús de Fwkpers. 28
Figura 14: Diagrama de classes Fwkpers. 31
Figura 15: Paquet del model i anotacions. 32
Figura 16: Paquet de connexió base de dades. 33
Figura 17: Paquet mapeig i registre d'objectes. 34
Figura 18: Paquet nucli. 35
Figura 19: Paquet utilitats cerca. 36
Figura 20: Paquet d'Excepcions. 36
Figura 21: Diagrama seqüència creació taula 37
Figura 22: Diagrama seqüència esborrar taula. 38
Figura 23: Diagrama seqüència creació base de dades. 39
Figura 24: Diagrama seqüència creació base de dades. 40
Figura 25: Diagrama seqüència inserció, modificació i eliminació d'un registre. 41
Figura 26: Diagrama seqüència consulta d'un registre per clau primària. 42
Figura 27: Diagrama seqüència consulta de registres. 43
Figura 28: Diagrama seqüència consulta amb filtres. 44
Figura 29: Diagrama seqüència consulta sql nativa. 45
Figura 30: Diagrama seqüència consulta fwkql. 46
Figura 31: Estructura del projecte del framework. 47

Figura 32: Anotacions per mapejar les taules de base de dades. 48
Figura 33: Connexió base de dades. 48
Figura 34: Carrega del driver JDBC per Oracle a ConnexioBDOracle 49
Figura 35: Capa mapeig de Fwkpers. 49
Figura 36: Inicialització de l'objecte mapejat. 50
Figura 37: Implementació paquet del nucli. 51
Figura 38: Implementació modificació registre 52
Figura 39: Classes del paquet utilitats cerca. 52
Figura 40: Generació de la sentència condicional del filtre. 53
Figura 41: Es retorna el missatge de la excepció. 53
Figura 42: Diagrama de classes Aplicació RRHH. 54
Figura 43: Llibreria persist-work versió 1.0. 55
Figura 44: Aplicació web recursosHumansApp, part JAVA. 55
Figura 45: Classe Departament. 56
Figura 46: Classe Empleat. 56
Figura 47: Classe Projecte. 57
Figura 48: Classe Relació Projecte i Empleat. 57
Figura 49: Pàgina principal de l'aplicació 58
Figura 50: Pàgina per a crear les taules. 58
Figura 51: Pàgina per a esborrar les taules. 58
Figura 52: Creació d'un departament. 59
Figura 53: Cerca de departament amb filtres. 59
Figura 54: Carrega i edició d'un empleat. 60
Figura 55: Cerca d'empleats per cognom. 60
Figura 56: Assignació d'un empleat a un projecte. 61
Figura 57: SpringSource Tool Suite 64
Figura 58: Configuració pom.xml framework. 65
Figura 59: Configuració pom.xml aplicació web. 65
Figura 60: Accés a Application Express 66
Figura 61: MySQL Workbench. 66
Figura 62: Administració base de dades MySQL 66

1 Introducció

1.1 Descripció del projecte

En l'actualitat ens trobem que la majoria de les aplicacions són multicapa. Una aplicació multicapa es divideix en diferents unitats lògiques o capes. Al separar l'aplicació en capes es guanya en rendiment, en seguretat i en escalabilitat. Normalment les aplicacions tenen les capes següents:

- Aplicació client: s'encarrega de mostrar la informació a l'usuari. En molts casos es pot considerar que els navegadors web són l'aplicació client.

- Servidors d'aplicacions: es troba a la xarxa i és accessible per a totes les aplicacions clients. En el servidor d'aplicacions és on despleguem les aplicacions. Aquestes aplicacions poden estar implementades en diferents tecnologies (J2EE, .NET, etc).

- Servidor de base de dades: a vegades el servidor de base de dades i el d'aplicacions poden ser el mateix, però cada cop més sovint es troben separats.

Les aplicacions multicapa J2EE s'implementen amb un llenguatge de programació orientat a l'objecte. Ja no es treballa amb programes procedurals, que es limiten a accedir a la informació que necessiten, la tracten, la mostren i prou, sinó amb objectes que volen representar el món real i interaccionen entre ells. La creació d'aquests objectes s'ha de realitzar a partir de la informació emmagatzemada a les bases de dades, afegeix una complexitat que amb altres tipus de programació no es tenia. La complexitat radica principalment, en com poder consultar la informació i com transformar-la en els nostres objectes de forma senzilla.

A més d'això, ens trobem que actualment existeixen diferents tipus de bases de dades:

- Base de dades relacionals: és el tipus més extens en la majoria de les aplicacions J2EE. Es pot establir connexions entre les dades, les quals es troben guardades en taules amb atributs. Podem parlar d'entitats i de relacions.

- Base de dades orientat a l'objecte: la informació està representada per objectes. Es troba molt poc extens, però és una evolució per a poder treballar de forma integrada amb els llenguatges orientats a l'objecte (JAVA, C#, Visual Basic.NET i C++). Es tracta del següent pas natural de les bases de dades amb J2EE.

- Base de dades XML: estan basades en els documents XML. No fan servir atributs i es poden obtenir els resultats directament en documents XML. Aquesta característica pot ser important, ja que treballar amb XML pot facilitar l'administració de les dades degut a que és un estàndard per a la transferència de dades entre sistemes diferents.

- Base de dades LDAP: fa referència a un protocol que permet l'accés a un directori ordenat i distribuït per a cercar diversa informació en un entorn de xarxa. Aquest directori normalment es troba organitzat de forma lògica i jeràrquica.

En aplicacions J2EE per a accedir a les bases de dades, modelar la informació i transformar-la en objectes, es pot fer ús d'un framework (o entorns de treball) de persistència. Un entorn de treball ens proporciona uns mòduls o components per a facilitar la feina d'accedir a la base de dades i tractar-la. Aquest ens ajuda en la transformació de la informació cap a objectes del nostre sistema, aquesta transformació és important, ja que la majoria de les bases de dades fan servir el model ER (entitat-relació) i realitzar la equivalència als objectes en J2EE és complex, i si fem servir un framework estalviem en complexitat.

En aquest projecte, realitzarem un estudi dels diferents frameworks de persistència que existeixen actualment al mercat, analitzarem els diferents tipus de persistència de dades existents (base de dades relacionals, base de dades OO, XML, etc) i desenvoluparem un framework propi de persistència pensat en i pel desenvolupador.Afegirem les característiques bàsiques i imprescindibles que es necessiten per a treballar amb base de dades, identificarem de cada framework els seus punts forts per a afegir-los, i millorarem els punts febles. I per finalitzar, realitzarem una petita aplicació J2EE, a on mostrarem de forma clara i senzilla com s'ha de fer servir el nou framework de persistència.

1.2 Objectius generals i específics

L'objectiu principal d'aquest projecte és el desenvolupament d'un framework de persistència que serveixi per a aplicacions J2EE. Es realitzarà un estudi dels frameworks existents i dels diferents tipus de persistència. Aquest framework complirà els requisits següents:

- Implementació de les funcionalitats bàsiques de la persistència amb base de dades.
- Components reutilitzables i basats en tecnologies estàndards: JavaBeans, XML, etc.
- Simplificat i intuïtiu, pensat en i per al desenvolupador.
- Fàcilment escalable per a afegir noves funcionalitats sense problemes.

1.3 Planificació

Hem dividit el nostre projecte en les següents fases:

1. Planificació del projecte
    a. Elaboració del pla
    b. Presentació del pla
    c. Pla de projecte acceptat

2. Anàlisi de mercat dels frameworks de persistència
    a. Estudi dels frameworks existents
    b. Estudi comparatiu de les tecnologies
    c. Definició de les necessitats del framework
    d. Presentació del document amb l'estudi

3. Disseny del framework de persistència
    a. Anàlisi de requeriments
    b. Disseny funcional
    c. Disseny tècnic

4. Desenvolupament i implementació del framework
    a. Implementació opcions bàsiques (CRUD)
    b. Test unitari opcions CRUD
    c. Implementació opcions avançades
    d. Test unitari opcions avançades
    e. Test integrat

   f. Desenvolupament de l'aplicació de test
   g. Prova pilot aplicació test

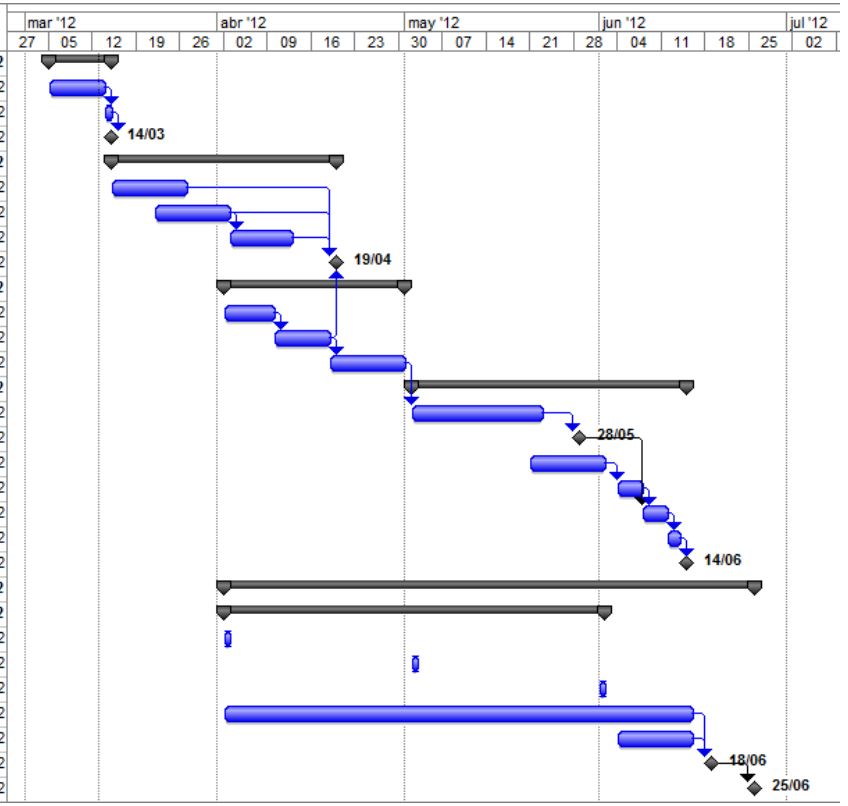
5. Documentació i seguiment del projecte
   a. Seguiment i control
   b. Elaboració de la memòria i presentació
   c. Manual de l'usuari del framework
   d. Presentació de la memòria
   e. Tancament del projecte

Tenim les següents fites:

- 14 de Març 2012: Pla de projecte acceptat (PAC1).
- 19 d'Abril 2012: Estudi de les tecnologies existents, anàlisi de requeriments i disseny funcional (PAC2).
- 28 de Maig 2012: Disseny tècnic i implementació opcions bàsiques framework (PAC3).
- 14 de Juny 2012: Framework finalitzat amb aplicació J2EE de test del framework.
- 18 de Juny 2012: Entrega de la memòria i de la presentació.
- 25 de Juny 2012: Tancament del projecte.

Hem establert tres punts de control per a avaluar l'estat del projecte, es realitzarà un anàlisi de les tasques, indicant el seu avenç i informant si hi ha algun risc que fiqui en perill les fites i el projecte. Aquests es realitzaran a principis de cada mes (2 d'Abril 2012, 2 de Maig de 2012 i 1 de Juny de 2012).

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Planificació del projecte**	**8 días**	**lun 05/03/12**	**mié 14/03/12**
2	Elaboració del pla	7 días	lun 05/03/12	mar 13/03/12
3	Presentació del pla	1 día	mié 14/03/12	mié 14/03/12
4	Pla de projecte acceptat	0 días	mié 14/03/12	mié 14/03/12
5	**Anàlisi de mercat dels frameworks de persistència**	**26 días**	**jue 15/03/12**	**jue 19/04/12**
6	Estudi dels frameworks de persistència existents	8 días	jue 15/03/12	lun 26/03/12
7	Estudi comparatiu de les diferents tecnologies	8 días	jue 22/03/12	lun 02/04/12
8	Definició de les necessitats del framework de peristència	8 días	mar 03/04/12	jue 12/04/12
9	Presentació del document amb l'estudi (PAC2)	1 día	jue 19/04/12	jue 19/04/12
10	**Disseny del framework de persistència**	**21 días**	**lun 02/04/12**	**lun 30/04/12**
11	Anàlisi de requeriments	6 días	lun 02/04/12	lun 09/04/12
12	Disseny funcional	7 días	mar 10/04/12	mié 18/04/12
13	Disseny tècnic	8 días	jue 19/04/12	lun 30/04/12
14	**Desenvolupament i implementació del framework**	**32 días**	**mié 02/05/12**	**jue 14/06/12**
15	Implementació opcions bàsiques (CRUD)	15 días	mié 02/05/12	mar 22/05/12
16	Test unitari opcions bàsiques (PAC3)	4 días	mié 23/05/12	lun 28/05/12
17	Implementació opcions avançades (consultes complexes)	10 días	lun 21/05/12	vie 01/06/12
18	Test unitari opcions avançades	4 días	lun 04/06/12	jue 07/06/12
19	Test integrat	2 días	vie 08/06/12	lun 11/06/12
20	Desenvolupament de l'aplicació de test	2 días	mar 12/06/12	mié 13/06/12
21	Prova pilot aplicació de test	1 día	jue 14/06/12	jue 14/06/12
22	**Documentació i seguiment del projecte**	**61 días**	**lun 02/04/12**	**lun 25/06/12**
23	**Seguiment i control**	**45 días**	**lun 02/04/12**	**vie 01/06/12**
24	Punt de control 1	1 día	lun 02/04/12	lun 02/04/12
25	Punt de control 2	1 día	mié 02/05/12	mié 02/05/12
26	Punt de control 3	1 día	vie 01/06/12	vie 01/06/12
27	El·laboració de la memòria i presentació	55 días	lun 02/04/12	vie 15/06/12
28	Manual d'usuari del framework	10 días	lun 04/06/12	vie 15/06/12
29	Presentació de la memòria	1 día	lun 18/06/12	lun 18/06/12
30	Tancament del projecte	1 día	lun 25/06/12	lun 25/06/12



1.4 Productes obtinguts

Un cop finalitzat el projecte els productes que s'obtindran són els següents:

- Implementació del framework en una llibreria per a un ús posterior en qualsevol aplicació J2EE.
- Javadoc del framework de persistència.
- Aplicació d'exemple que fa servir la llibreria obtinguda.

2 Estudi de mercat

2.1 Aplicacions multicapa

Una aplicació multicapa defineix interfícies entre els codis que implementen diferents conceptes o problemàtiques, això permet fer canvis en una implementació o capa sense afectar de forma significant a les altres capes. Normalment les capes es comuniquen de dalt a baix, i una capa només depèn de la que es troba per sota. Segons els sistemes, es poden definir diferents capes, però una definició típica es realitza en tres capes: capa de presentació, capa de negoci i capa de persistència.

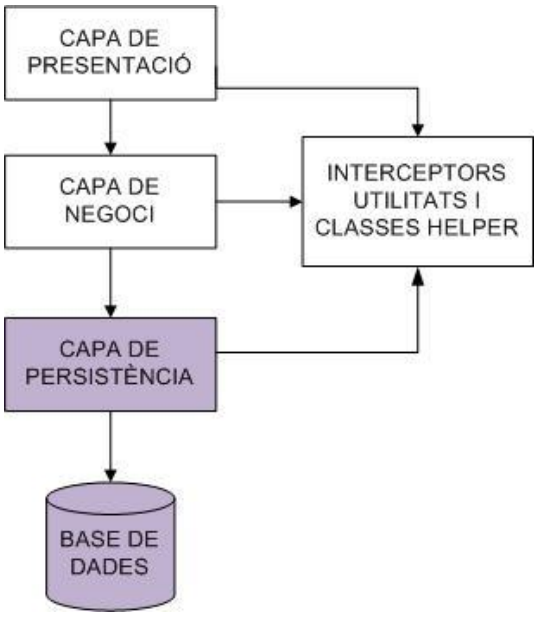


Figura 1: Arquitectura en capes.

2.2 Capa de persistència

Quasi totes les aplicacions necessiten persistir dades. Entenem amb persistència l'acció de preservar la informació d'un objecte de forma permanent, així com de poder recuperar la seva informació posteriorment. En cas que un sistema no guardés dades, un cop s'apagués o es deixés de treballar amb ell, la informació es perdria i el sistema no seria gaire útil.

La persistència ens permet emmagatzemar, transferir i recuperar l'estat dels objectes. Podem trobar diferents tècniques de persistència:

- Serialització: es converteix l'objecte en un arxiu o buffer de memòria amb la finalitat de realitzar una transmissió per una xarxa, també es pot transformar en un format més llegible com XML o JSON. És un mecanisme molt ampli i s'utilitza per a persistir un objecte a una base de dades amb format XML o per transmetre aquest per una xarxa i poder fer una rèplica o clon de l'objecte.

- Motors de persistència de base de dades relacionals: els motors de persistència o motors de mapeig objecte relacional (ORM), s'encarreguen de guardar els objectes a les bases de dades relacionals. Amb aquests motors es poden fer servir les característiques pròpies de la programació orientada a l'objecte, principalment herència i polimorfisme. El problema radica en fer la transformació entre els objectes i les taules normalitzades de les bases de dades. El mapeig objecte-relacional, ORM a partir d'ara, s'utilitza per a facilitar aquesta tasca de transformació entre taules i objectes. Molts d'aquests motors de persistència són anomenats frameworks o marcs de treball.

- Base de dades orientades a objectes: es tracta d'un nou tipus de base de dades, on la informació es representa mitjançant els objectes presents a la programació orientada a l'objecte. Quan es realitza aquesta integració, llavors podem parlar d'un sistema gestor de base de dades orientada a l'objecte (OODBMS). Aquestes bases de dades es dissenyen per a treballar conjuntament amb un llenguatge de programació orientat a l'objecte, com per exemple JAVA, C#, Visual Basic .NET, etc. Aquest tipus de base de dades no és molt popular, ja que implica perdre la independència de les dades, per que deixa de ser independent del sistema que el vulgui fer

servir, per exemple, una aplicació no desenvolupada orientada a l'objecte no la podrà fer servir, en canvi si fos una base de dades relacional, llavors sí.

Normalment, quan es parla sobre la persistència en JAVA, es tracta sobre guardar dades a una base de dades relacional usant SQL o mitjançant els motors de persistència. En els apartats següents veurem diferents frameworks de persistència.

2.3 Frameworks de persistència

2.3.1 EJB

Els Enterprise JavaBeans (EJB) formen part de l'estàndard de construcció d'aplicacions J2EE d'Oracle. La seva especificació detalla com els servidor d'aplicacions proveeixen objectes des del costat del servidor. L'objectiu dels EJBs es aconseguir que el programador es pugui abstreure dels problemes generals d'una aplicació empresarial (concurrència, transaccions, persistència, seguretat, etc).

Amb la versió J2EE 1.4, tenim la versió 2.1 dels EJBs i trobem que existeixen tres tipus diferents:

- EJB d'entitat: són els responsables de la persistència. Tenim dos tipus diferenciats:
  - Persistència gestionada pel contenidor (*CMP*): el contenidor s'encarrega d'emmagatzemar i recuperar les dades de l'objecte d'entitat mitjançant un mapeig dels camps a una taula de la base de dades.
  - Persistència gestionada pel Bean (*BMP*): el propi objecte s'encarrega, mitjançant una base de dades o un altre mecanisme, d'emmagatzemar i recuperar les dades. És el propi programador el responsable d'implementar els mecanismes.

  A la darrera documentació J2EE 5.0, amb la versió 3.0, aquests tipus d'EJB són reemplaçats per la *Java Persistence Api o JPA*.

- EJB de sessió: gestionen el flux de la informació en el servidor.
  - Amb estat (*stateful*). En un *bean* amb estat, les variables de la instància guarden dades específiques obtingudes durant la connexió amb el client.

  - Sense estat (*stateless*). No tenen estat associat per lo que es poden executar de forma concurrent.

- EJB dirigits per missatge: són els únics que funcionen de forma asíncrona. Fan servir *Java Messaging System* (*JMS*) per a la seva comunicació.

Per a desenvolupar un EJB, es necessita proporcionar els següents fitxers:

- Descriptor de desplegament: un fitxer XML que especifica informació sobre el bean, com el seu tipus de persistència, els seus atributs de transacció, etc.
- Fitxer classe de l'EJB: fa la implementació dels mètodes definits a les interfícies.
- Interfícies: les interfícies *remote* i *home* són necessàries per a poder accedir de forma remota. Per a accedir localment, es necessiten la interfície *local* i *local home*.
- Classes d'utilitats: altres classes que necessita l'EJB, com classes per al tractament d'excepcions o utilitats.

Després es realitza l'empaquetament en un fitxer JAR d'EJB, es tracta d'un mòdul que guarda l'EJB. Aquest JAR és portable i es pot usar en diferents aplicacions. Per a realitzar una aplicació J2EE, es poden paquetitzar un o més mòduls a dintre d'un fitxer EAR, quan aquest es desplega al servidor, també es desplega l'EJB.

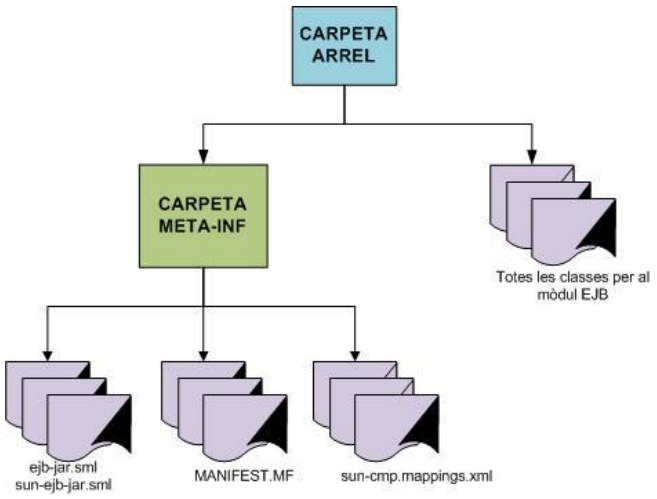


Figura 2: Estructura d'un JAR EJB.

2.3.2 Java Persistence API (JPA)

A partir de J2EE 5.0 es desenvolupa la versió 3.0 dels EJBs. En aquesta versió els *Beans* d'entitat desapareixen, i es crea la *Java Persistence API* o JPA. JPA té les següents característiques:

- **Beans simples (persistència POJO).** Es simplifiquen els beans d'entitat i s'afegeix com a estàndard el model de persistència amb POJOs (*Plain Old Java Object)* o classes simples. Els beans són classes simples i no requereixen cap interfície. Aquestes classes suporten polimorfisme i herència, a més de tindre diferent tipus de relacions entre si.

- ***Entity Manager API.*** S'introdueix *l'API Entity Manager*, que serveix per a crear, cercar, eliminar o modificar entitats. Introdueix un nou concepte de separació i fusió de les instàncies dels beans similar al patró *Value Object (*objectes VO*)*. Una instància pot ser modificada pel client, i després s'envia a *l'Entity Manager* per a que realitzi les modificacions.

- **Metadades amb anotacions.** Afegir metadades amb anotacions[1] facilita enormement el desenvolupament de les entitats, eliminant la necessitat de tindre descriptors per als desplegaments. Amb una anotació d'entitat s'especifica que una classe és un bean d'entitat.

- **Millora en el llenguatge de consultes.** EJB 3.0 afegeix un nou llenguatge per a fer consultes sobre les entitats. Concretament *Java Persitence Query Language* o JPQL. JPQL millora EJB-QL, proporciona operacions addicionals, com modificacions i eliminacions massives, operacions amb combinacions, projecció, subconsultes, etc.

La capa de persistència de les aplicacions EJB ara són anomenades entitats JPA. Aquestes adopten un model de persistència lleuger, dissenyada per a treballar perfectament amb Oracle TopLink i Hibernate.

[1] Una anotació és un modificador de metadades que s'incorporen a un fitxer font JAVA. Les anotacions es poden especificar en les classes, mètodes, paràmetres, variables locals, constructors, enumeracions i paquets. Es poden fer servir per a generar codi, per a documentar codi, o per a prestar serveis durant el temps d'execució.

```
@Entity

@Table(name = "EMP")

public class Employee implements java.io.Serializable

{
   private int empNo;
   private String eName;
   private double sal;
   @Id
   @Column(name="EMPNO", primaryKey=true)
   public int getEmpNo()
   {
     return empNo;
   }
   public void setEmpNo(int empNo)
   {
     this.empNo = empNo;
   }
   public double getSal()
   {
     return sal;
   }
```

Figura 3: Exemple entitat JPA.

```
EntityManagerFactory factory =
Persistence.createEntityManagerFactory(null);
// get an EntityManager from the factory
EntityManager em = factory.createEntityManager();
// Begin a transaction
em.getTransaction().begin();
// query for all employees who work in our research division
// and put in over 40 hours a week average
Query query = em.createQuery("SELECT e " +
" FROM Employee e " +
" WHERE e.division.name = 'Research' " +
" AND e.avgHours > 40");
List results = query.getResultList();
// give all those hard-working employees a raise
for (Object res : results) {
Employee emp = (Employee) res;
emp.setSalary(emp.getSalary() * 1.1);
}
// commit will detect all updated entities and save them in
database
em.getTransaction().commit();
// free the resources
```

Figura 4: Exemple codi tractament amb *Entity Manager*.

JPA s'ha convertit en un nou estàndard, del qual destaquem dos qüestions molts importants:

- Els motors de JPA han de poder ser intercanviables. S'ha de poder substituir un motor per un altre en qualsevol moment.
- Els motors de JPA han de poder treballar fora d'un entorn EJB 3.0 sense problemes.

A conseqüència d'això, ens trobem amb moltes opcions per als desenvolupadors i arquitectes, provocant més competició i una millora en els productes. És important remarcar que JPA és una especificació i no una implementació.

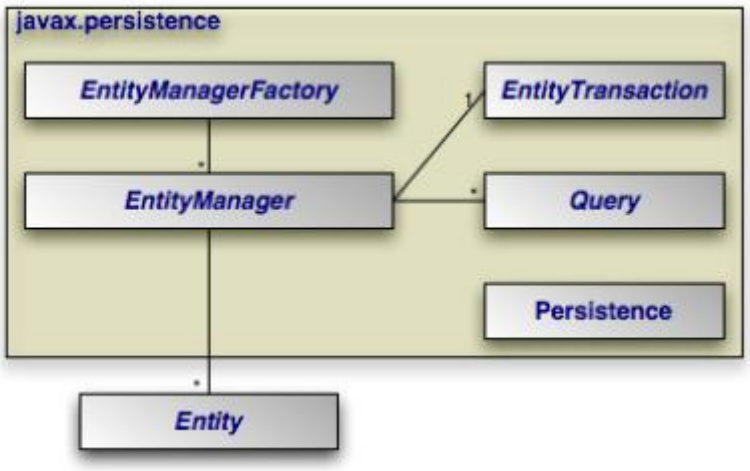


Figura 5: Classes del paquet de persistència.

2.3.3 Hibernate

Hibernate és un eina que serveix per a mapejar objectes i relacions (ORM). Podem considerar-lo com el més important que existeix actualment, i bastant responsable de la creació d'un nou estàndard en el camp de la persistència.

EJB 2.1 va rebre moltes crítiques per part dels desenvolupadors, especialment en el cas dels beans d'entitats i de persistència. Degut a això, Sun va començar una nova especificació per a simplificar-los. Desenvolupadors de l'equip d'Hibernate van entrar molt al principi i van ajudar en la nova especificació: EJB 3.0.

La versió actual de Hibernate és la 4.1 i les seves característiques són les següents:

- Llenguatge de consultes propi anomenat HQL, molt semblant al JPQL, i que es bassa també en SQL.
- Funciona en qualsevol servidor d'aplicacions J2EE, aplicacions SWING, i aplicacions simples J2EE.
- Configuració dels mapejos amb fitxers XML.
- Inclusió i ús d'anotacions. Aquestes anotacions implementen l'estàndard JPA, a més de tindre altres anotacions per a funcionalitats més avançades. Es poden fer servir com a reemplaçament dels fitxers XML o com una adició a aquestos.
- *Hibernate Entity Manager*. La especificació de JPA defineix interfícies de programació, regles de cercle de vida per als objectes de persistència, i característiques per a les consultes. La implementació que realitza Hibernate per a aquesta part s'anomena *Hibernate Entity Manager*.

A partir de la versió 3.x d'Hibernate, es pot treballar amb interfícies estandarditzades. Això comporta el benefici de poder fer servir la capa de persistència en qualsevol servidor d'aplicacions que compleixi l'estàndard EJB 3.0.

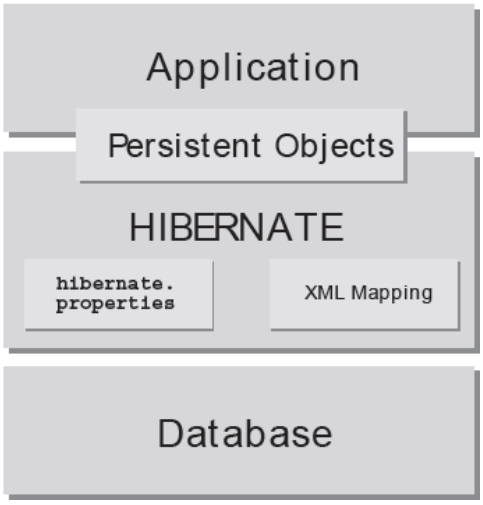


Figura 6: Arquitectura Hibernate.

```
package eg;

@Entity
@Table(name="cats") @Inheritance(strategy=SINGLE_TABLE)
@DiscriminatorValue("C") @DiscriminatorColumn(name="subclass",
discriminatorType=CHAR)
public class Cat {
        @Id @GeneratedValue
        public Integer getId() { return id; }
        public void setId(Integer id) { this.id = id; }
        private Integer id;

        public BigDecimal getWeight() { return weight; }
        public void setWeight(BigDecimal weight) { this.weight = weight; }
        private BigDecimal weight;

        @Temporal(DATE) @NotNull @Column(updatable=false)
        public Date getBirthdate() { return birthdate; }
        public void setBirthdate(Date birthdate) { this.birthdate = birthdate; }
        private Date birthdate;

        @org.hibernate.annotations.Type(type="eg.types.ColorUserType")
        @NotNull @Column(updatable=false)
        public ColorType getColor() { return color; }
        public void setColor(ColorType color) { this.color = color; }
        private ColorType color;

        @NotNull @Column(updatable=false)
        public Integer getLitterId() { return litterId; }
        public void setLitterId(Integer litterId) { this.litterId = litterId; }
        private Integer litterId;

        @ManyToOne @JoinColumn(name="mother_id", updatable=false)
        public Cat getMother() { return mother; }
```

Figura 7: Exemple d'entitat Hibernate.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<!-- a SessionFactory instance listed as /jndi/name -->
<session-factory
name="java:hibernate/SessionFactory">
<!-- properties -->
<property name="connection.datasource">java:/comp/env/jdbc/MyDB</property>
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="show_sql">false</property>
<property name="transaction.factory_class">
org.hibernate.transaction.JTATransactionFactory
</property>
<property name="jta.UserTransaction">java:comp/UserTransaction</property>
<!-- mapping files -->
<mapping resource="org/hibernate/auction/Item.hbm.xml"/>
<mapping resource="org/hibernate/auction/Bid.hbm.xml"/>
<!-- cache settings -->
<class-cache class="org.hibernate.auction.Item" usage="read-write"/>
<class-cache class="org.hibernate.auction.Bid" usage="read-only"/>
```

Figura 8: Fitxer configuració hibernate.cfg.xml.

2.3.4 Oracle TopLink

Oracle TopLink s'encarrega de la persistència d'objectes en base de dades relacionals, implementant l'estàndard JPA de forma semblant a Hibernate, i també realitza transformacions d'objectes a elements XML.

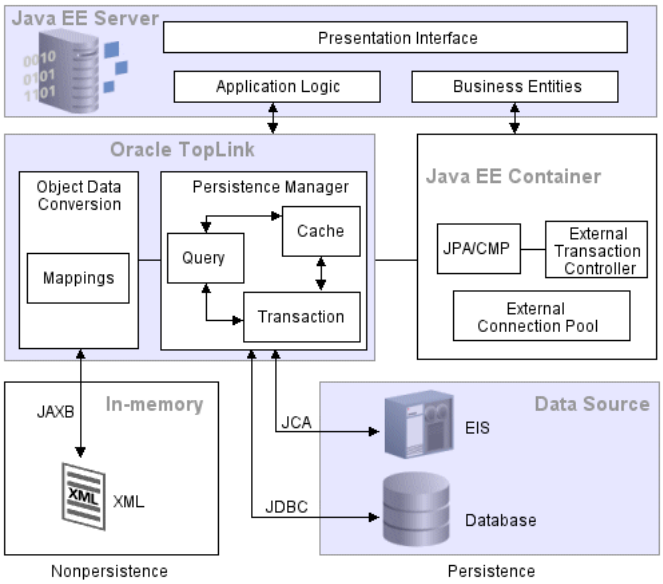


Figura 9: Arquitectura Oracle TopLink.

Oracle TopLink es pot fer servir per a un ampli rang d'aplicacions i arquitectures J2EE. Serveix per a dissenyar, implementar, desplegar, a més d'optimitzar persistència d'objectes i per a la capa de transformació d'objectes que suporta una gran varietat de fonts de dades:

- Persistència transaccional d'objectes relacionals en una base de dades relacionals, principalment fent servir JDBC.
- *Enterprise information system (EIS)*: persistència transaccional d'objectes Java cap a una font de dades no relacionals, fent servir arquitectura Java EE Connector Adapter (JCA), i qualsevol tipus de registre EIS, incloent indexat, mapejat o XML.
- XML no transaccional i no persistent, només en memòria, i conversió entre objectes JAVA i documents XML fent servir JAXB.

2.3.5 MyBatis

MyBatis és un framework de persistència que mapeja els objectes mitjançant sentències SQL, aquestes es guarden en fitxers XML. Existeixen diferents implementacions per a una gran varietat de llenguatges, incloent Java, Ruby, .NET, etc.

Una diferència important en relació a altres frameworks de persistència, és que no té un llenguatge propi per a realitzar les consultes, sinó que es recolza només en SQL, això permet una gran simplificació. Les característiques d'aquest framework són les següents:

- Divisió de la capa de persistència en tres subcapes:
  - Capa d'abstracció: es comunica amb la capa del negoci. S'implementa mitjançant el patró *Data Access DAO (DAO)*.
  - Capa de persistència: interfície amb el gestor de la base de dades mitjançant un API. En comptes de fer servir JDBC, MyBatis fa servir SQL-MAP.
  - Capa del driver: s'ocupa de la comunicació amb la base de dades fent servir un driver específic.
- Únicament fa ús de SQL per a realitzar les consultes.

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
  insert into Author (id,username,password,email,bio)
  values (#{id},#{username},#{password},#{email},#{bio})
</insert>

<update id="updateAuthor" parameterType="domain.blog.Author">
  update Author set
    username = #{username},
    password = #{password},
    email = #{email},
    bio = #{bio}
  where id = #{id}
</update>

<delete id="deleteAuthor" parameterType="int">
  delete from Author where id = #{id}
</delete>
```

Figura 10: Fitxer configuració consultes MyBatis.

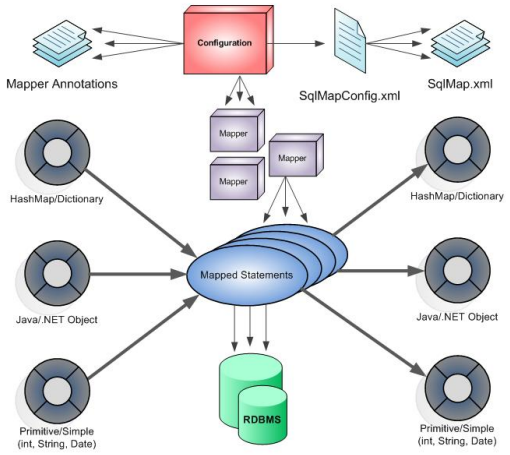


Figura 11: Arquitectura MyBatis

2.3.6 Java Data Objects (JDO)

L'API Java Data Objectes (JDO) és una especificació estàndard per a models JAVA de persistència. Es tracta d'una alternativa a JPA i les seves principals característiques són les següents:

- Fàcil d'usar: els programadors es poden centrar en el seu domini d'objectes i deixar els detalls de la persistència a la implementació JDO.

- Portable: les aplicacions que utilitzen JDO API es poden executar en diferents implementacions sense tornar a compilar o modificar codi font.

- Independent de la base de dades: les implementacions de JDO suporten diferents formes de guardar dades, incloent base de dades relacionals, orientades a l'objecte, XML, fitxers, i altres.

- Alt rendiment: els programadors delegen els detalls de la persistència a JDO, que pot optimitzar el patró d'accés a dades per a un rendiment òptim.

- Integració amb EJB: les aplicacions poden aprofitar les característiques dels EJB, com el processament dels missatges a distància, coordinació automàtica de transaccions distribuïdes, seguretat,etc.

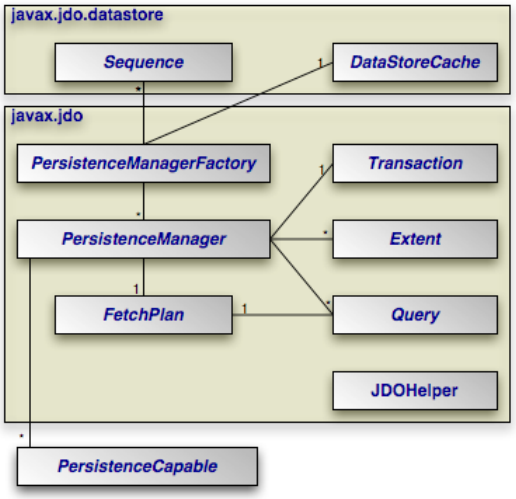


Figura 12: Classes del paquet jdo.

2.4 Comparativa de frameworks

Realitzarem una taula comparativa dels principals frameworks i analitzarem els conceptes més importants.

	EJB 2.1	JPA (Hibernate i TopLink)	MyBatis	JDO
Orientació objecte	Té orientació a l'objecte, però hi ha conceptes avançats que no els suporta.	Orientació a l'objecte amb conceptes avançats suportats.	Orientació a l'objecte amb conceptes avançats suportats.	Orientació a l'objecte amb conceptes avançats suportats.
Tipus de base de dades	Base de dades relacionals i no relacionals.	Només base de dades relacionals.	Només base de dades relacionals.	Qualsevol tipus de base de dades: relacionals, OO, XML.
Llenguatge de consultes	Llenguatge propi de consultes EJB-QL.	Llenguatge propi de consultes (JQL, HQL, etc)	No en té. Fa servir SQL.	Llenguatge propi de consultes JDOQL.
Estàndard i portabilitat	Es tracta d'un estàndard de persistència per a	Es tracta de l'estàndard de persistència a	No és estàndard.	Es tracta d'un estàndard de persistència de

	J2EE 1.4	partir de J2EE 5.0		diferents bases de dades.
Simplicitat	És bastant complex desenvolupar la capa de persistència.	Molt simple, es basa en POJO's.	Bastant simple, fa servir directament sentències SQL.	Molt simple, es basa en POJO's.
Anotacions	No suporta anotacions.	Suporta anotacions a les classes.	No suporta anotacions.	Suporta anotacions a les classes.
Configuració	Bastant complexa mitjançant fitxers XML que descriuen com es realitza el desplegament dels EJB.	Bastant simple, amb un únic fitxer XML o amb fitxer properties.	Configuració amb fitxers XML.	Configuració amb fitxer persistence.xml

2.5 Necessitats bàsiques del framework

Les necessitats bàsiques que tindrà el nostre framework de persistència són les següents:

- Configuració senzilla i intuïtiva: optarem per una configuració mínima de mapeig amb XML, per lo que únicament tindrem un fitxer de configuració, a on s'haurà de configurar l'accés a la base de dades, el tipus de base de dades, i el paquet principal a on es trobaran les nostres entitats. Aquest fitxer serà un fitxer de propietats (conf-fwkpers.properties). També es deixarà la possibilitat de personalitzar-lo.

- Mapeig d'entitats amb anotacions: per continuar amb la simplificació, farem servir anotacions per a indicar les entitats. Aquestes anotacions seran compatibles amb l'estàndard de JPA, d'aquesta manera el nostre framework complirà amb la especificació de persistència de J2EE, intentant que estigui al mateix nivell que Hibernate, ApacheJPA, TopLink, etc.

- Llenguatge propi per a realitzar les consultes (FWKQL), semblant al JQL i HQL. Possibilitat de fer servir SQL, JQL o HQL. Es desenvoluparà un adaptador que permetrà decidir quin llenguatge es vol utilitzar per a realitzar les consultes.

- Suport per a realitzar cerques sense tindre coneixements de cap llenguatge *SQL. Es crearà unes classes Helper que donaran funcionalitats i utilitats per a realitzar cerques afegint criteris mitjançant mètodes. D'aquesta forma es podrà utilitzar sense tindre coneixements d'SQL o amb uns coneixements mínims.

3 Disseny del framework

3.1 Anàlisi de funcionalitats

El que es vol aconseguir amb el framework de persistència, a partir d'ara Fwkpers, és que sigui senzill i que solucioni la problemàtica del mapeig entitat-relació alhora de treballar amb objectes. A més d'això, amb molt poques classes es pot realitzar totes les funcionalitats bàsiques i complexes que es troben a les aplicacions amb base de dades.

Per a agrupar les funcionalitats, Fwkpers es troba dividit en tres mòduls:

- **Mòdul d'administració de base de dades.** Aquest mòdul dona funcionalitats al programador per a que realitzi totes les operacions relacionades amb la creació de la base de dades, la creació de les taules, les modificacions, etc. Està pensat per a ser utilitzat de forma puntual, ja que no és freqüent fer servir un framework de persistència per a crear la base de dades o per esborrar-la.

- **Mòdul d'administració de dades.** En aquest mòdul es proporciona les eines necessàries per a que el programador pugui manegar les dades: afegir, consultar, modificar i esborrar registres. S'encarrega de les accions CRUD *(**C**reate, **R**ead, **U**pdate, **D**elete)*[2].

- **Mòdul de consultes complexes i FWKQL.** Aquest mòdul s'encarregarà de facilitar al programador les cerques complexes d'entitats, així com del llenguatge propi que definirem per al nostre framework. També donarà la possibilitat de fer servir SQL directament. El resultat de les consultes sempre seran objectes.

[2] Crear, consultar, modificar i esborrar.

Encara que existeixin tres mòduls diferenciats, aquests no són independents entre si, sinó que en alguns casos interactuen entre ells, només es tracta d'una forma de classificar les funcionalitats proporcionades per Fwkpers.

3.1.1 Casos d'ús

Una vegada identificat els tres grans grups de funcionalitats, farem un anàlisi dels casos d'ús. D'aquesta manera podem observar les diferents funcionalitats que implementarem.

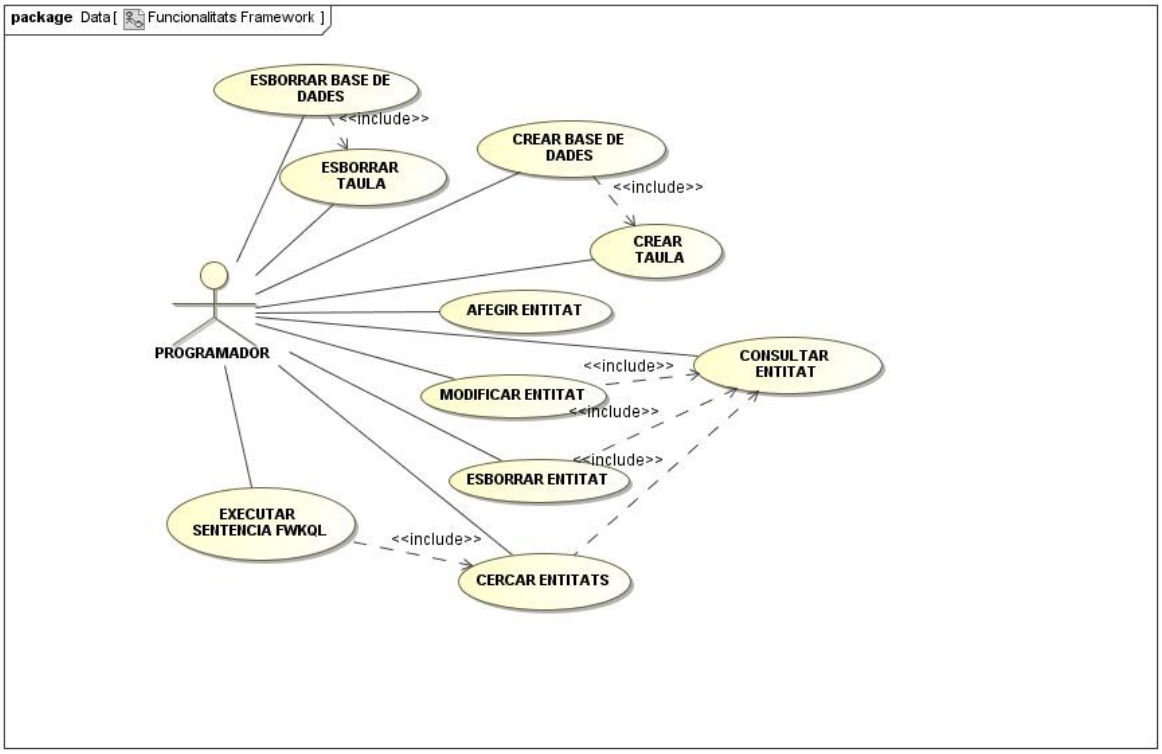


Figura 13: Casos d'ús de Fwkpers.

- Crear base de dades: es donarà la possibilitat de treballar amb una base de dades existent, o de crear-la des de programa. En aquest cas d'ús es crearan les taules i les relacions entre elles, sempre a partir de les classes que estiguin a la configuració del context.

- Crear taula: en aquest cas d'ús, el programador podrà crear la taula a la base de dades. No sempre es vol crear de zero una base de dades, es pot donar el cas que només tinguem la necessitat d'afegir una taula nova al nostre sistema, d'aquesta manera donem la possibilitat de crear-la des de programa.

- Esborrar taula: com que tenim la possibilitat de crear una taula amb el framework, llavors també donem la possibilitat d'esborrar-la. De moment, degut a la seva complexitat, no es contempla la possibilitat de fer modificacions a una taula ja existent.

- Esborrar base de dades: aquest cas ús permet mitjançant la configuració de fitxer de propietats, esborrar tota la base de dades. També s'esborraran les relacions que hagi entre elles. Fa servir el mateix fitxer de propietats que el cas d'us de la creació de la base de dades.

- Afegir entitat: a partir d'un objecte o entitat, el programador podrà afegir o donar d'alta aquesta entitat (registre) a la base de dades. Equivaldrà a la sentència "INSERT" de SQL.

- Consultar entitat/s: a partir d'un identificador d'entitat, o clau primària, es carregaran les dades a un objecte JAVA. A part de la clau primària, també es podrà fer servir altres camps per a seleccionar el registre que es vol carregar, en aquest cas podrà retornar diferents registres. Equivalent a fer un "SELECT" de SQL.

- Modificar entitat: un cop s'ha carregat la entitat a memòria, es pot modificar sense problemes. En aquest cas d'ús, un cop fetes les modificacions, s'ha de fer servir el manegador d'entitats per a que les faci efectives cap a la nostra base de dades. Es tracta de l'equivalent a fer un "UPDATE" de SQL.

- Esborrar entitat: en cas que es vulgui esborrar una entitat de la base de dades (registre), no caldrà carregar-la primerament en memòria, es pot esborrar sense carregar-la, només informant de la clau o claus primàries. De totes formes, també es pot eliminar mitjançant la entitat que s'ha carregat prèviament. Equival a fer un "DELETE" de SQL.

- Cercar entitats: si es vol realitzar una consulta a la base de dades, la qual retorna diferents entitats, llavors es pot fer servir cerques d'entitats. Aquestes cerques es poden fer de diferents formes, es dona la possibilitat de fer cerques exactes o amb operacions (igual que, menor que, etc). Aquest cas d'ús retorna un llistat d'entitats segons els criteris de cerca.

- Executar sentència Fwkql: De la mateixa manera que altres frameworks de persistència han definit un llenguatge propi diferent al SQL i adaptat a treballar amb objectes, Fwkql també defineix un llenguatge propi per a simplificar les cerques o les operacions a realitzar sobre la nostra base de dades. Aquest llenguatge en la primera versió, opta per catalanitzar les sentències d' SQL.

3.1.2 Diagrama de classes

Fwkpers incideix en tres punts claus:

- Simplificació: és important que l'ús sigui senzill, per lo que s'ha optat per la creació d'anotacions, les quals un cop afegides a classes simples de JAVA, ja defineixen el model de dades sense problemes. La informació d'una taula es defineix a la creació de les classes. Es pot canviar el nom de la taula, o que coincideixi amb el nom de la classe, de forma semblant amb els camps, els tipus de camps s'agafen del tipus d'atribut de la classe.

- Estandardització: Fwkpers treballa en la línia de ser estàndard JPA. És per això que s'ha dissenyat de tal manera, que amb poques modificacions i/o adaptacions pugui complir l'estàndard amb JPA. Les anotacions creades són molts semblants, així com els objectes que s'encarreguen de realitzar les operacions amb la base de dades.

- Escalabilitat: un punt important d'un framework es poder afegir funcionalitats sense problemes. De la mateixa forma que el nostre disseny treballa per a poder complir l'estàndard JPA, també es troba preparat per a afegir noves funcionalitats sense problemes: suport per a més motors de base de dades, afegir nous tipus de camps de base de dades (blob, bfiles, clob, etc).

A continuació, podem veure el diagrama de classes de Fwkpers:

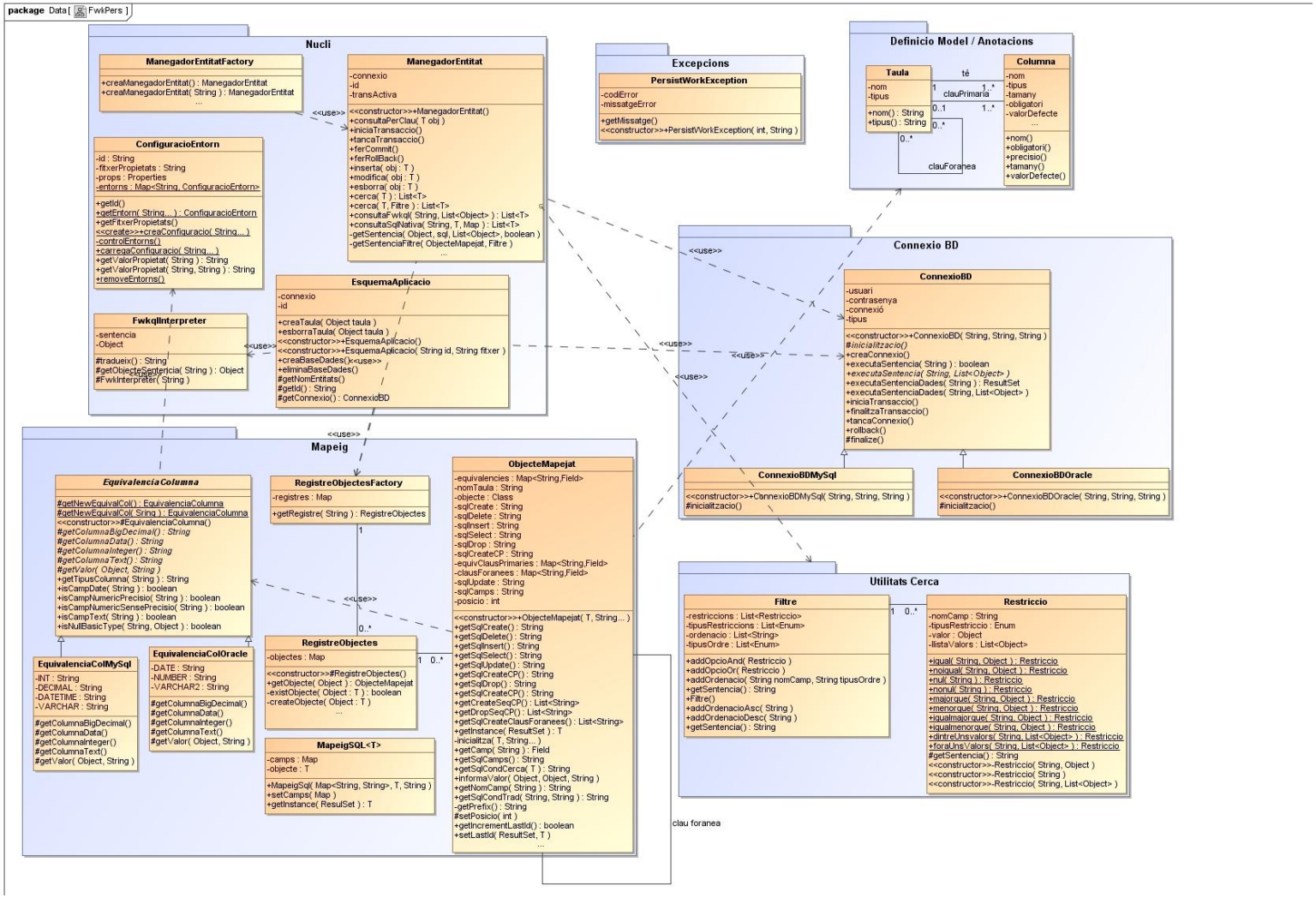


Figura 14: Diagrama de classes Fwkpers.

En els apartats següents, explicarem els diferents paquets de classes que conté Fwkpers.

Definició model / Anotacions

En aquest paquet es troben les anotacions que serviran per a crear el model de dades. Aquestes anotacions poden descriure la relació de la nostra classe amb una taula de base de dades, indiquen els camps de la taula, el seu format, si són claus primàries, claus foranies, el tipus de clau primària, etc. Les relacions de clauForanea i clauPrimaria, alhora de fer la implementació, es converteixen en anotacions.

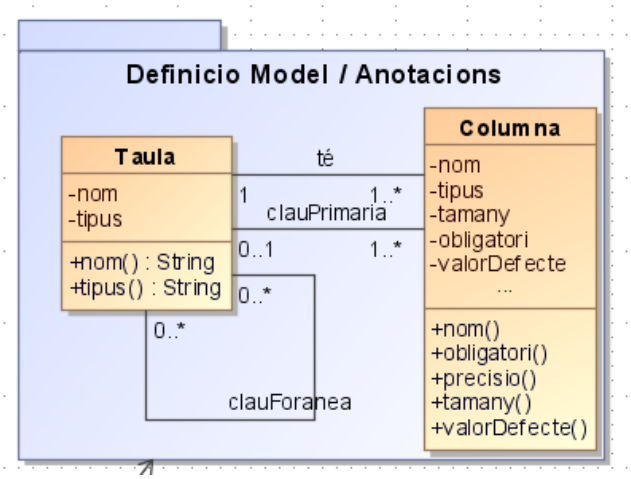


Figura 15: Paquet del model i anotacions.

Connexió Base Dades

Aquest paquet s'encarrega de la connexió que es realitza amb la base de dades. Proporciona suport per a treballar amb bases de dades Oracle i MySql. Es pot ampliar fàcilment per a suportar nous motors de base de dades.

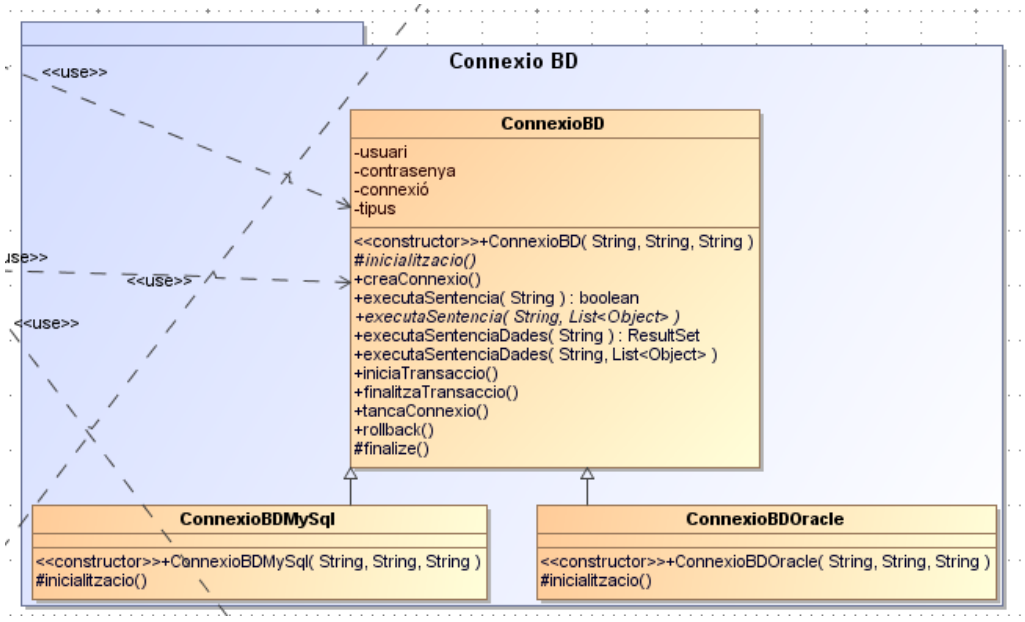


Figura 16: Paquet de connexió base de dades.

Mapeig i Registre d'objectes

Aquest paquet s'encarrega del mapeig dels objectes, i del seu registre. S'encarrega de generar les diferents sentències necessàries per a realitzar totes les operacions amb la base de dades. Aquestes són generades a partir de les anotacions afegides a les classes que defineixen el model de base de dades.

A més d'això, tenim un registre d'objectes, amb aquest registre un cop es té la informació d'un objecte específic, no cal tornar-lo a generar, doncs ja es troba registrat i es pot tornar a fer servir sense cap problema.

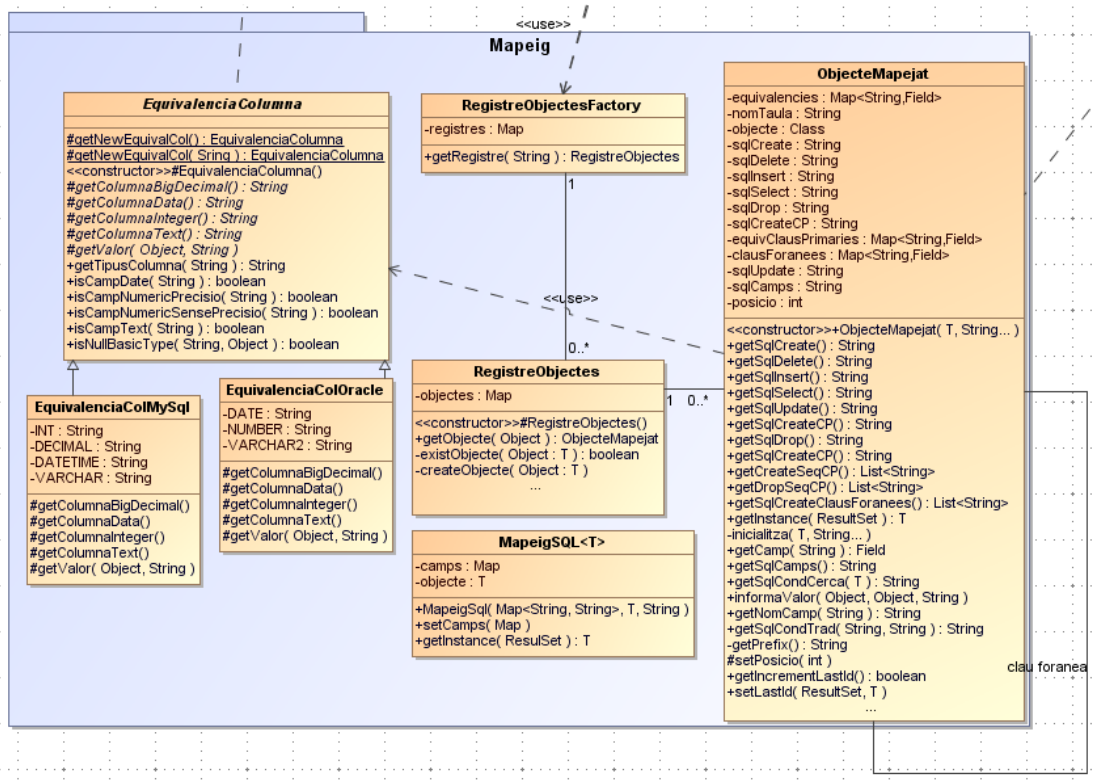


Figura 17: Paquet mapeig i registre d'objectes.

Nucli

El nucli s'encarrega de configurar l'entorn amb el fitxer de propietats. En aquest fitxer s'indica el motor de base de dades, la cadena de connexió, el usuari, la seva contrasenya, el paquet base de les classes mapejades, així com una llista de totes les classes que té el model de dades. En versions posteriors es preveu afegir suport per treballar amb *datasources*[3] o orígens de dades. A més a més existeix la possibilitat de fer servir diferents entorns a la vegada, per lo que Fwkpers té configuració multi-entorn. Si no s'indica res, s'agafa la configuració d'un fitxer propietats per defecte: META-INF/persist-work-conf.properties.

En aquest paquet també es troben les classes que es faran de forma directa. Aquestes classes són les següents:

- EsquemaAplicacio.java: en aquesta classe es troben les funcionalitats per a crear taules, esborrar-les, crear una base de dades i eliminar-la. Concretament totes les funcionalitats relacionades amb el mòdul d'administració de base de dades. Hem optat per separar aquestes

[3]Datasource o origen de dades s'encarrega de controlar totes les connexions amb la base de dades alhora de crear-les, eliminar-les, etc.

accions en una classe a part, ja que normalment tot lo relacionat amb administració de base de dades es realitza de forma puntual.

- ManegadorEntitat.java: aquesta classe centralitza totes les accions relacionades amb les dades, així com les cerques complexes i el llenguatge Fwkql. Les accions dels mòduls d'administració de dades i de les accions complexes es troben proporcionades per aquesta classe. D'aquesta manera amb una classe es pot fer totes les accions sense problemes, i es guanya en simplificació.

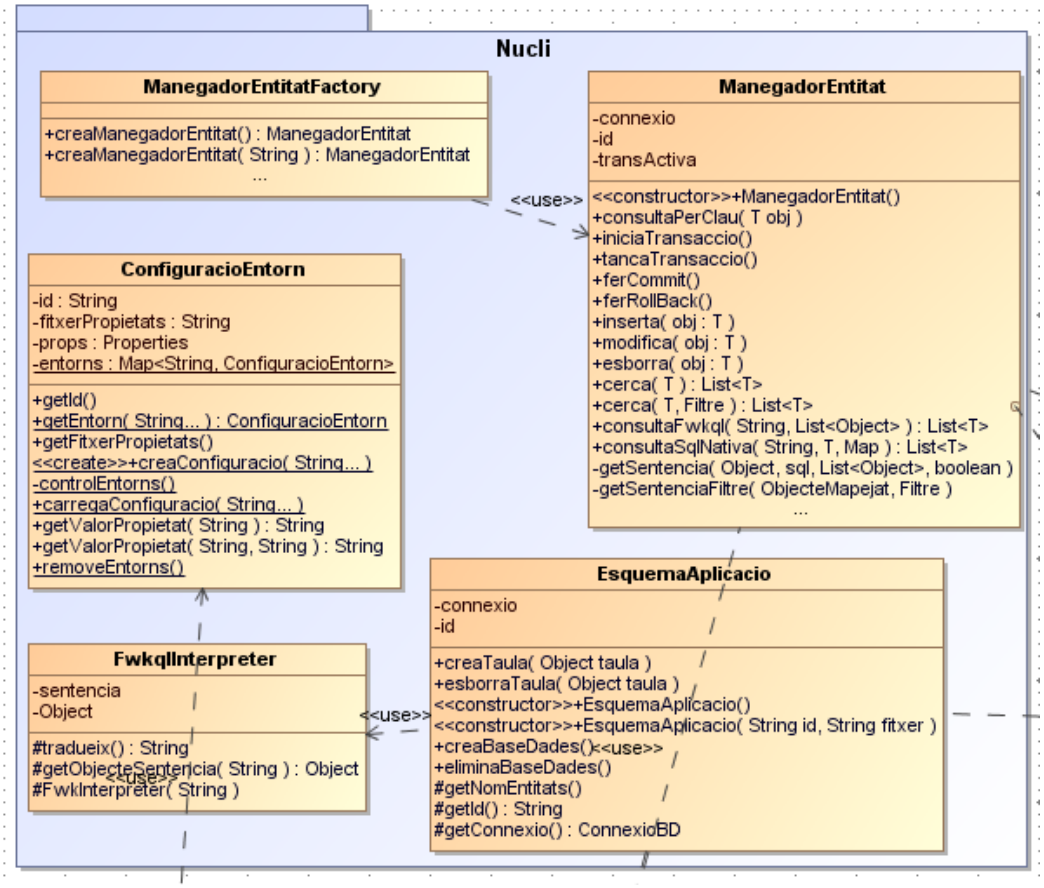


Figura 18: Paquet nucli.

Utilitats Cerca

Per a realitzar cerques de forma senzilla, hem creat dues classes que permet definir diferents tipus de restricció. Aquestes classes es troben dins el paquet d'utilitats cerca. Proporcionen funcionalitats per a crear condicions de cerca,

relacionades amb els camps, i a la seva vegada aquestes es poden afegir a un filtre. Aquest filtre és el que es fa servir quan es realitza la cerca.

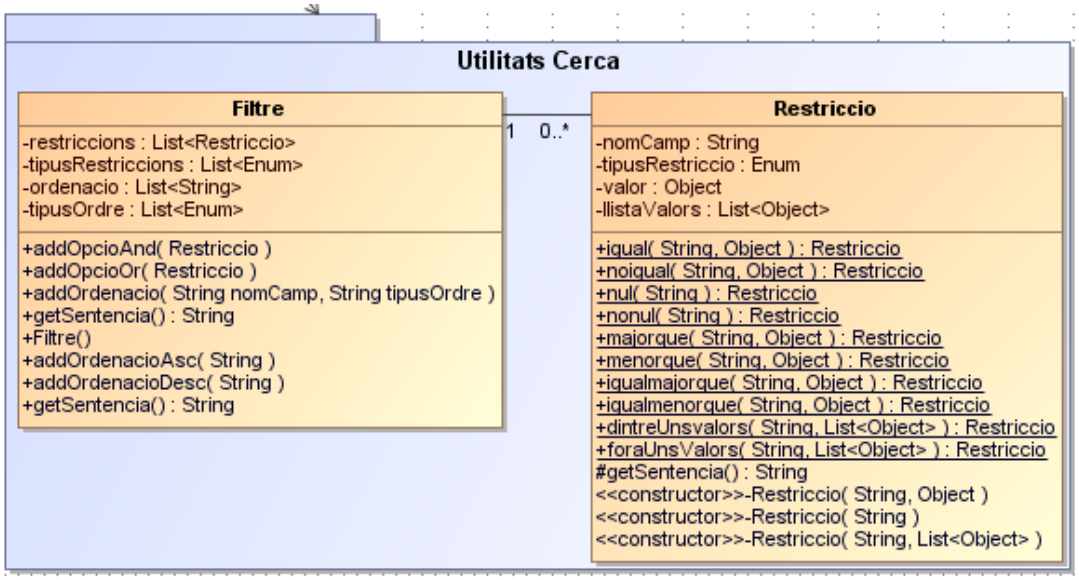


Figura 19: Paquet utilitats cerca.

Excepcions

Hem creat una classe per tractar totes les possibles excepcions que es puguin produir als altres paquets. S'han codificat els possibles errors, realitzant una classificació per paquet, afegint també la descripció de l'error. D'aquesta manera es pot saber de forma ràpida i senzilla on s'ha produït l'error i que cal fer per solucionar-lo.

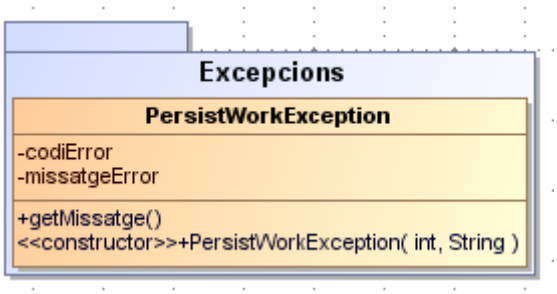


Figura 20: Paquet d'Excepcions.

3.2 Diagrama de seqüències

En aquest apartat s'inclouen els diferents diagrames de seqüències, on poder veure les interaccions dels diferents objectes del Fwkpers.

3.2.1 Creació d'una taula

La creació d'una taula es realitza mitjançant la classe EsquemaAplicació. Es necessita un objecte que tingui les anotacions, a partir d'aquest es crea l'objecte mapejat i es registra per al seu ús posterior. Mitjançant l'objecte mapejat creat, l'EsquemaAplicació obté les sentències, i les executa a la base de dades per a crear la nova taula.

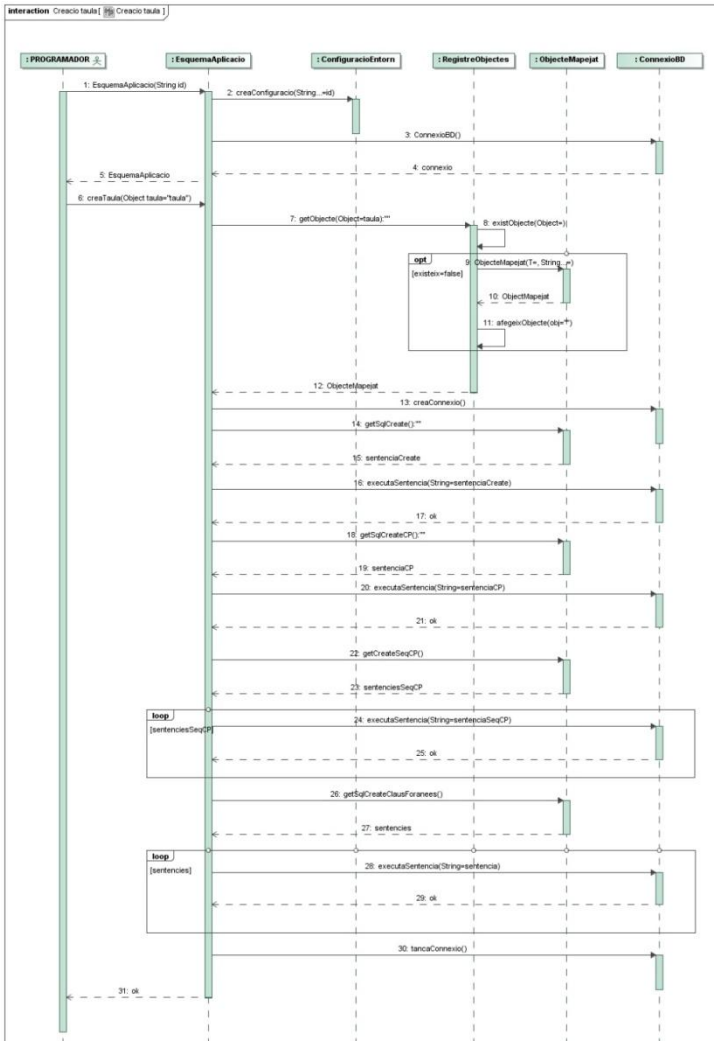


Figura 21: Diagrama seqüència creació taula

3.2.2 Eliminació d'una taula

La eliminació d'una taula funciona de forma semblant que la creació de la taula. Mitjançant l'objecte mapejat, la classe EquemaAplicacio executa les sentències a la base de dades per a esborrar la taula indicada.

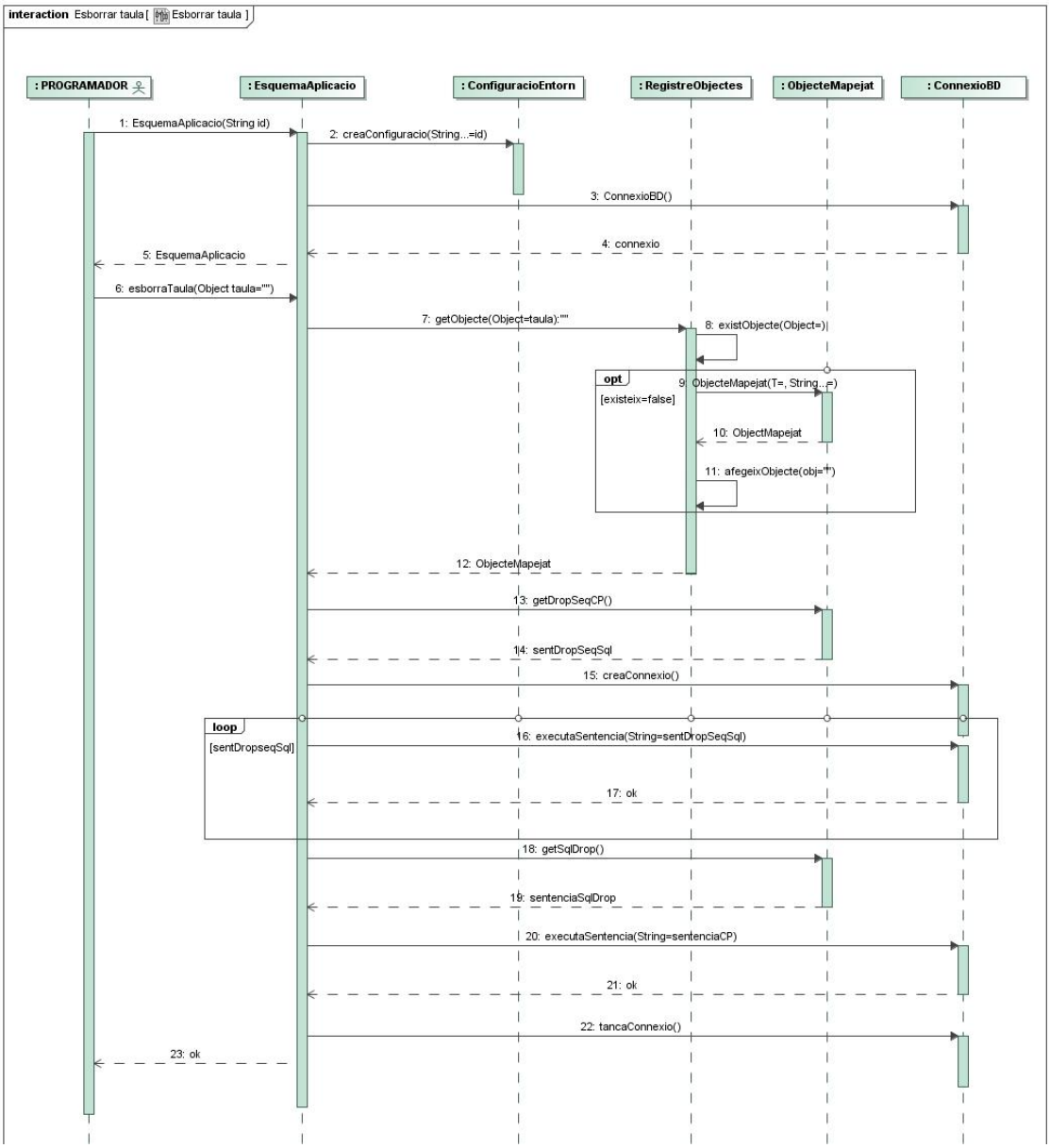


Figura 22: Diagrama seqüència esborrar taula.

3.2.3 Creació d'una base de dades

La creació de la base de dades també es realitzada per la classe EsquemaAplicació. S'obté els noms de totes les entitats que modelen la base de dades i es creen totes les taules. Els noms de les entitats es troben configurats al fitxer de propietats, indicant el package i el nom de les entitats. És important que estiguin en ordre, per a que no hi hagi problemes de restriccions[4] alhora de crear les taules.

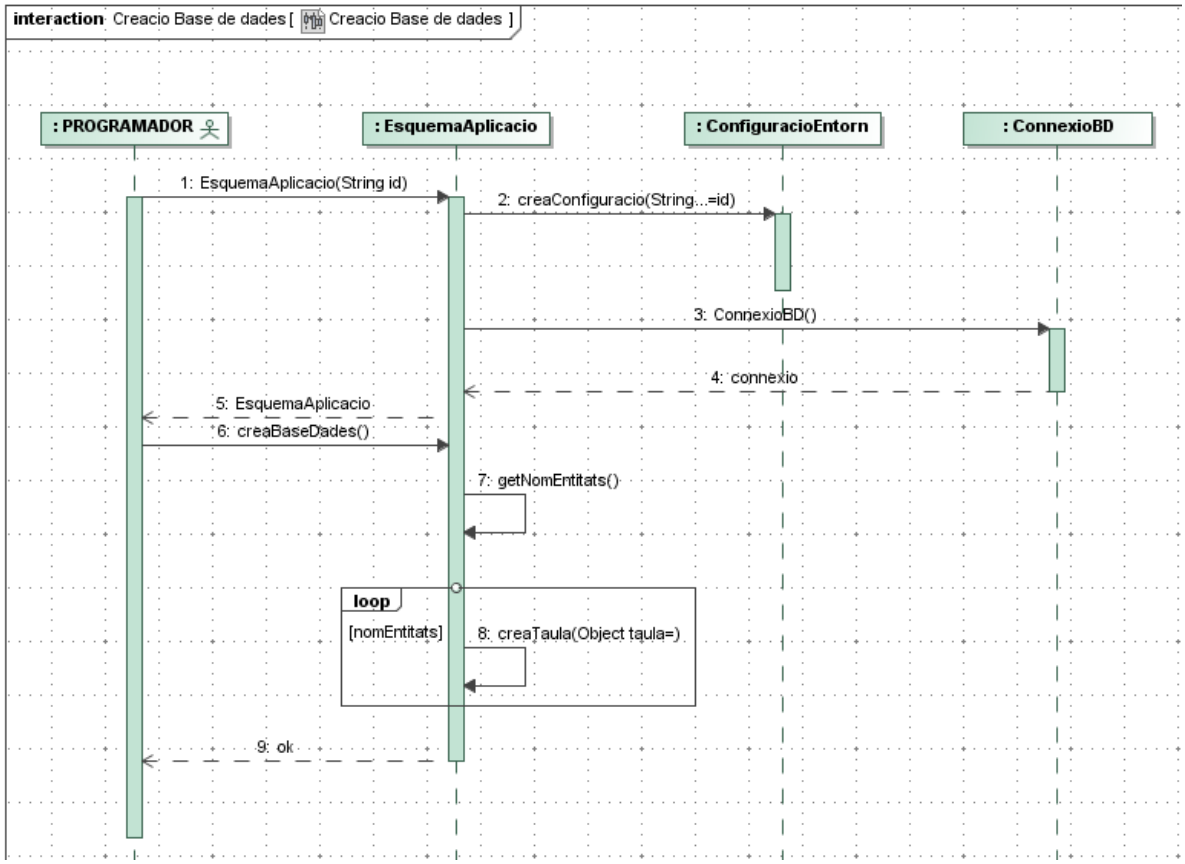


Figura 23: Diagrama seqüència creació base de dades.

[4] Una taula que tingui una clau forana s'ha de crear després que la taula de la que depèn.

3.2.4 Eliminació d'una base de dades

La eliminació de la base de dades funciona de forma similar a la creació. Es fa servir el fitxer de propietats i en comptes de crear les taules, s'esborren. Per a que no hi hagi problemes, les taules s'esborren en el ordre contrari de com es van crear, així no hi ha problemes de restriccions.

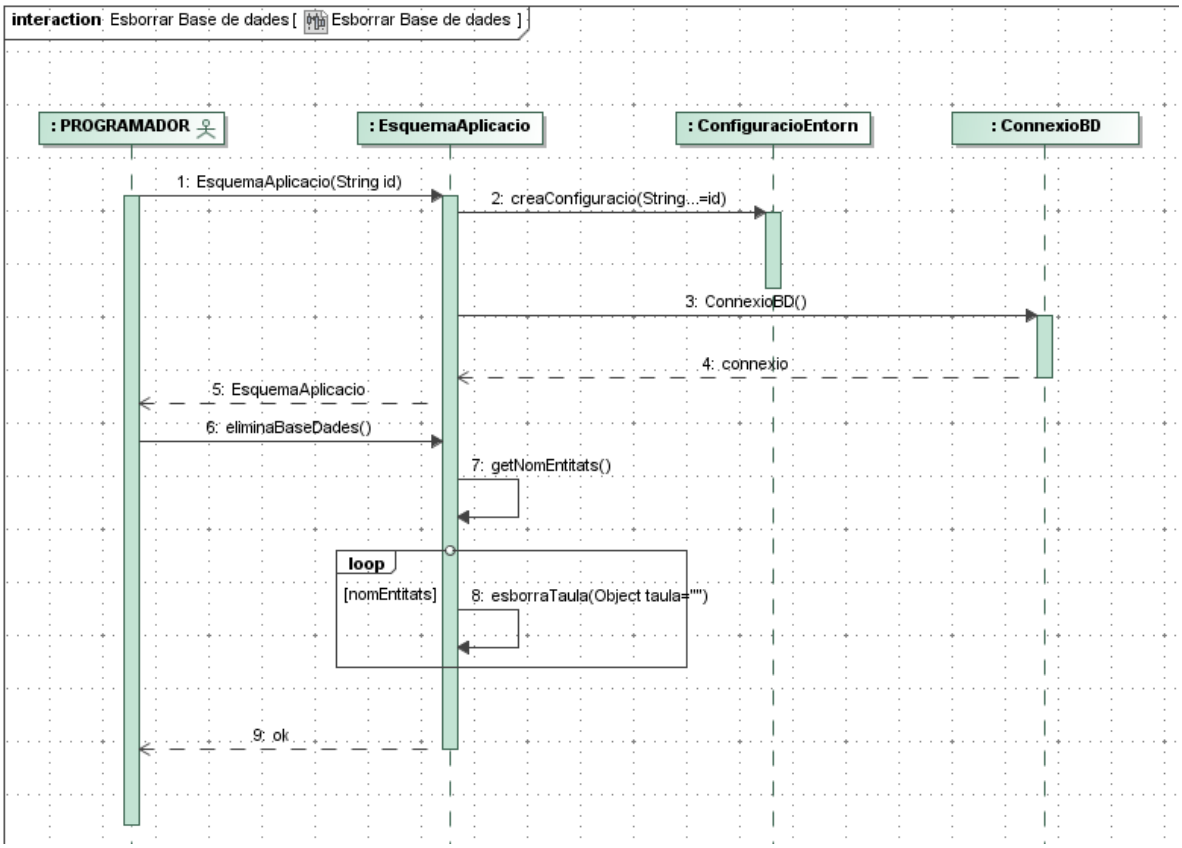


Figura 24: Diagrama seqüència creació base de dades.

3.2.5 Inserció, modificació i eliminació d'un registre

Per a simplificar els diagrames hem creat només un per a explicar la inserció, la modificació i la eliminació d'un registre. Com que la interacció entres els objectes és pràcticament la mateixa, només hem indicat el què hi ha de diferent segons l'acció que es vulgui realitzar.

Aquestes accions ja les realitza el manegador d'entitats (ManegadorEntitat), aquest demana al registre d'objectes l'objecte mapejat per a poder obtindre les sentències a realitzar. En aquestes accions es dona la possibilitat de realitzar control de transaccions.

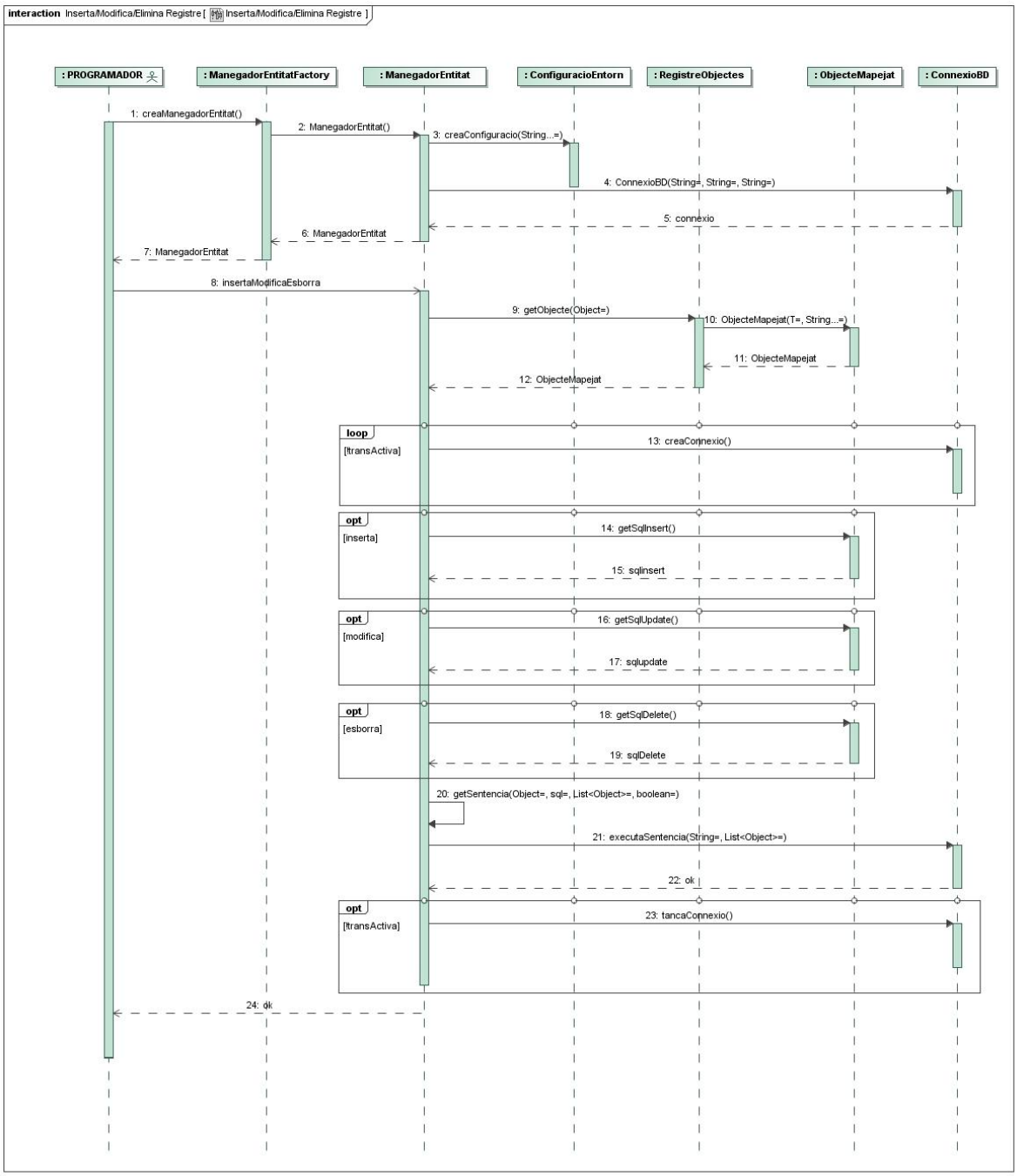


Figura 25: Diagrama seqüència inserció, modificació i eliminació d'un registre.

3.2.6 Consulta d'un registre per clau primària

Aquesta acció retornarà només un objecte. El manegador d'entitats realitza una consulta a la base de dades, mitjançant els valors de la clau fent servir una instància de l'objecte.

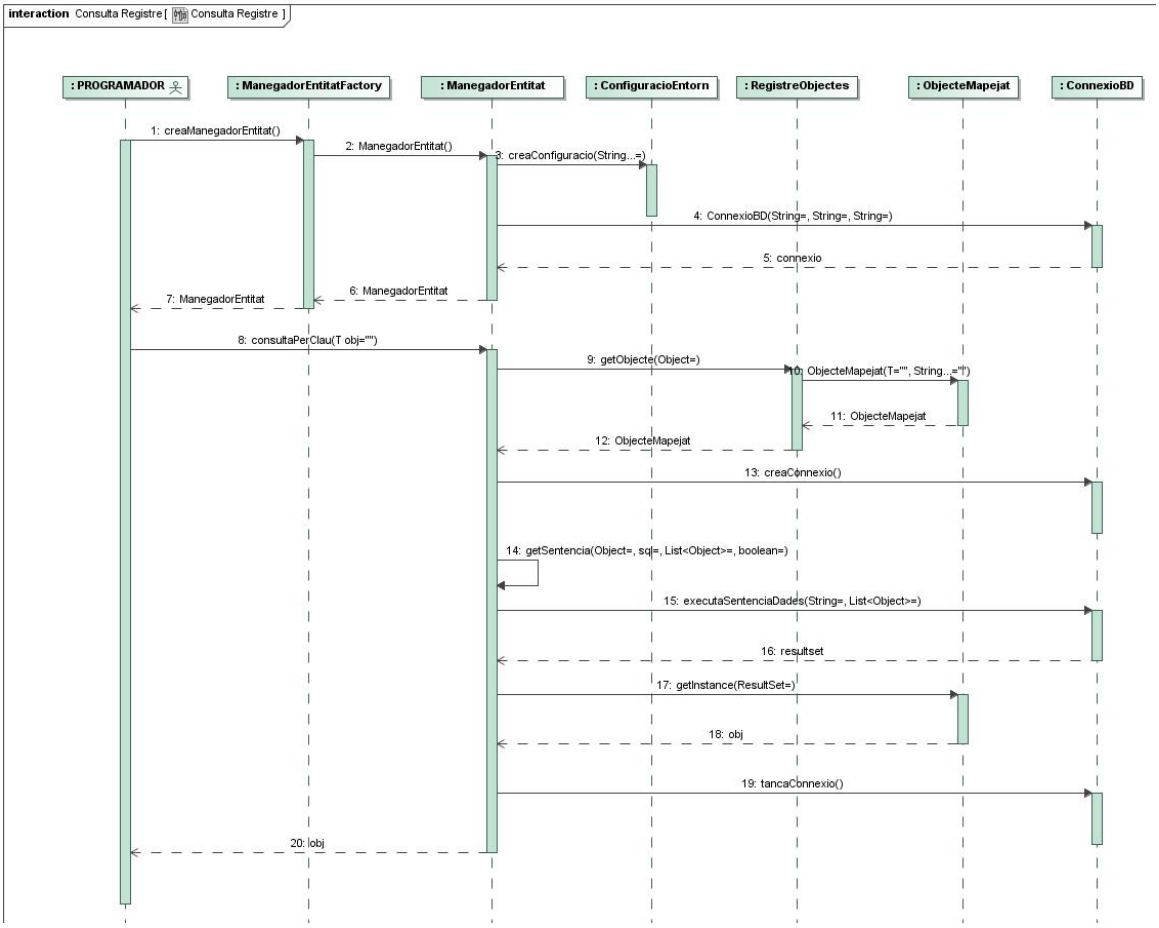


Figura 26: Diagrama seqüència consulta d'un registre per clau primària.

3.2.7 Consulta de registres

De forma semblant que amb la cerca per clau primària, es pot informar els valors pels quals es vol cercar a la base de dades, es fa servir una instància de l'objecte. La limitació d'aquesta cerca és que els valors informats als camps són restrictius i només es retornaran els objectes que compleixin aquests

mateixos valors. Per a fer cerques amb criteris més flexibles, cal utilitzar altres funcionalitats.

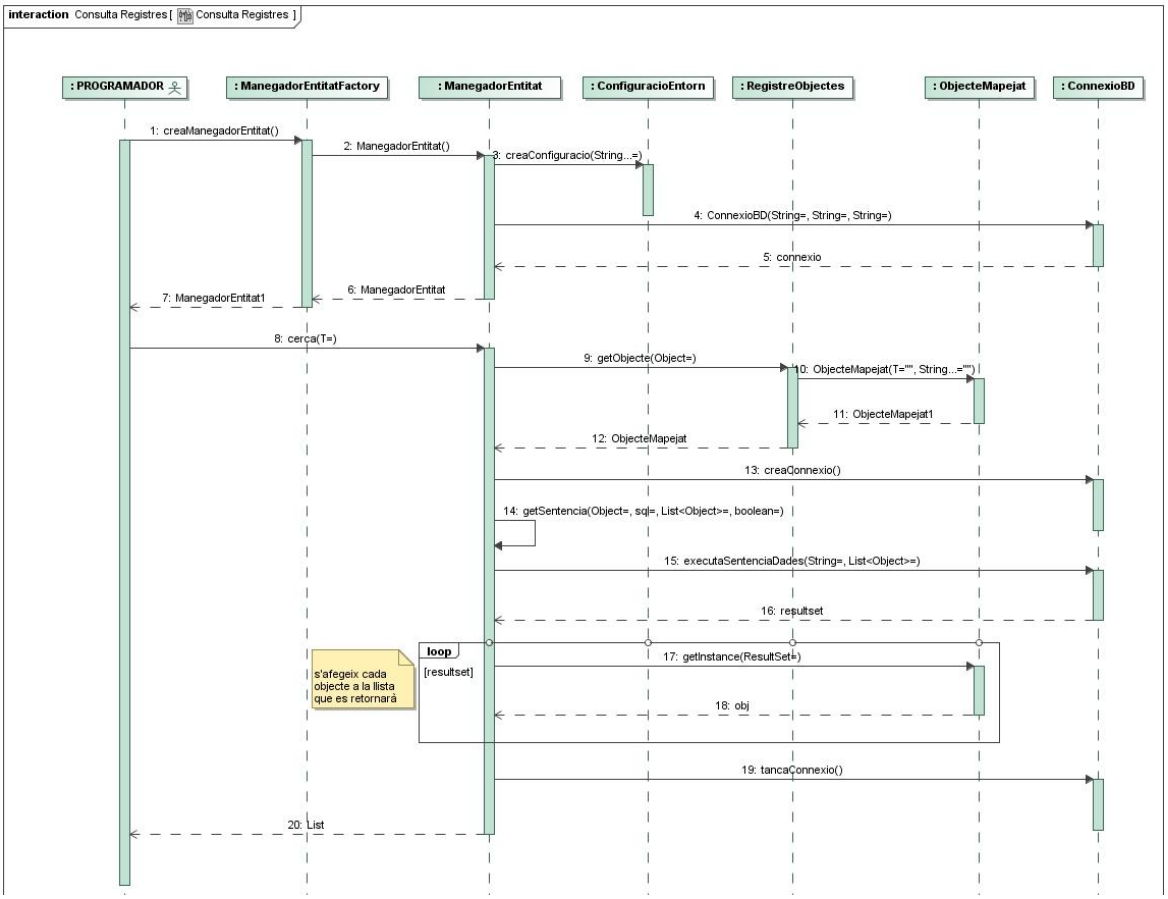


Figura 27: Diagrama seqüència consulta de registres.

3.2.8 Cerques amb filtre

Per a la realització de cerques sense fer servir llenguatge sql o amb coneixements mínims, s'ha proporcionat una sèrie de classes que permeten al programador definir restriccions per a realitzar cerques personalitzades. Aquestes restriccions permeten fer cerques de diferents tipus (iguals, no iguals, majors que, valors nuls, etc.). Les restriccions s'afegeixen al filtre, el qual també permet decidir l'ordre de la consulta, i ja es pot fer la consulta sense problemes. Aquestes consultes han de fer servir sempre una classe mapejada amb anotacions.

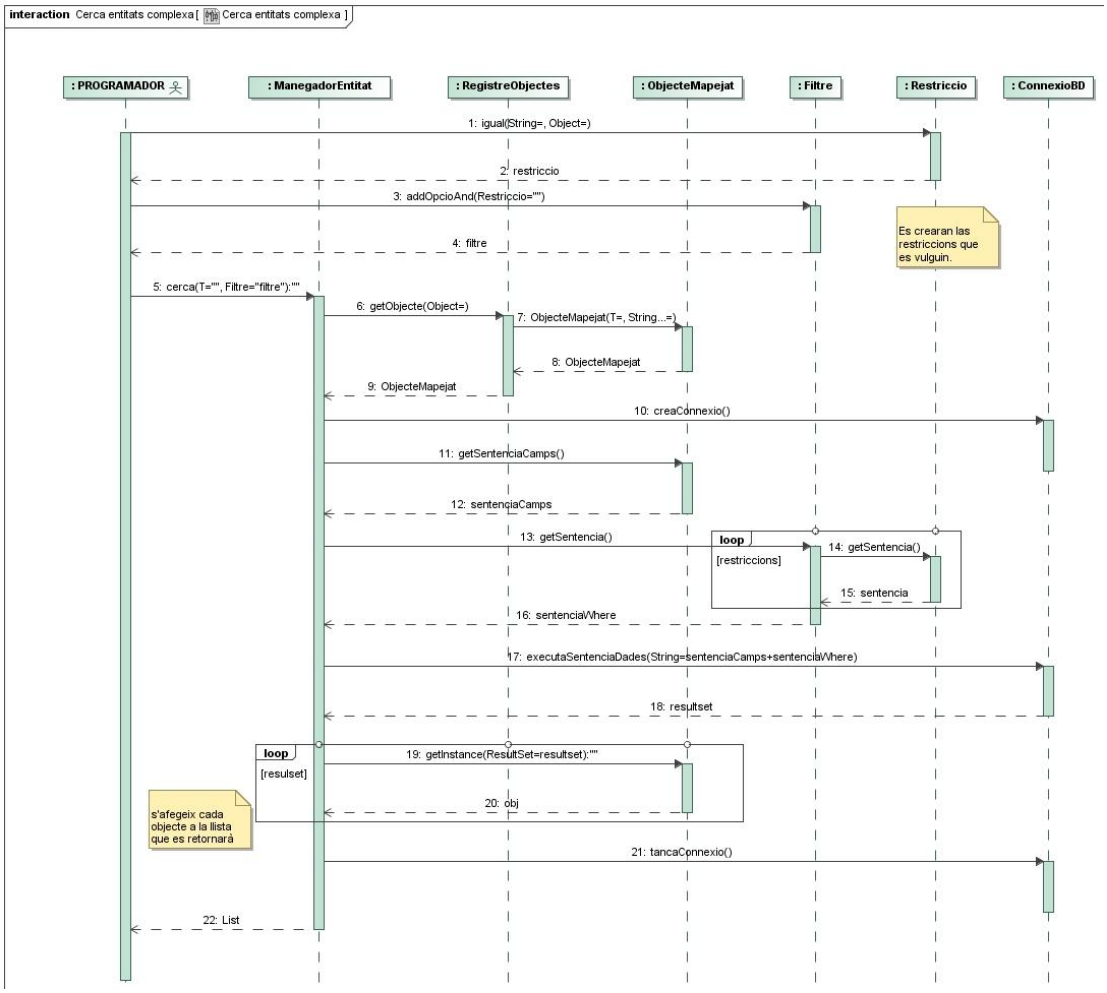


Figura 28: Diagrama seqüència consulta amb filtres.

3.2.9 Cerques amb sentència SQL

Per a realitzar cerques amb sentències sql natives, s'ha proporcionat una funcionalitat concreta. Aquesta espera una consulta sencera, no es poden parametritzar els valors a cercar. També és obligatori que el resultat de la consulta es retorni en un objecte JAVA, és per això que tenim una classe MapeigSQL a on s'ha d'indicar a quin atribut de l'objecte es vol ficar cada camp que retorna la consulta SQL, però aquesta classe pot no estar mapejada. D'aquesta manera continuem treballant amb resultats amb objectes, inclòs quan fem servir sql natiu.

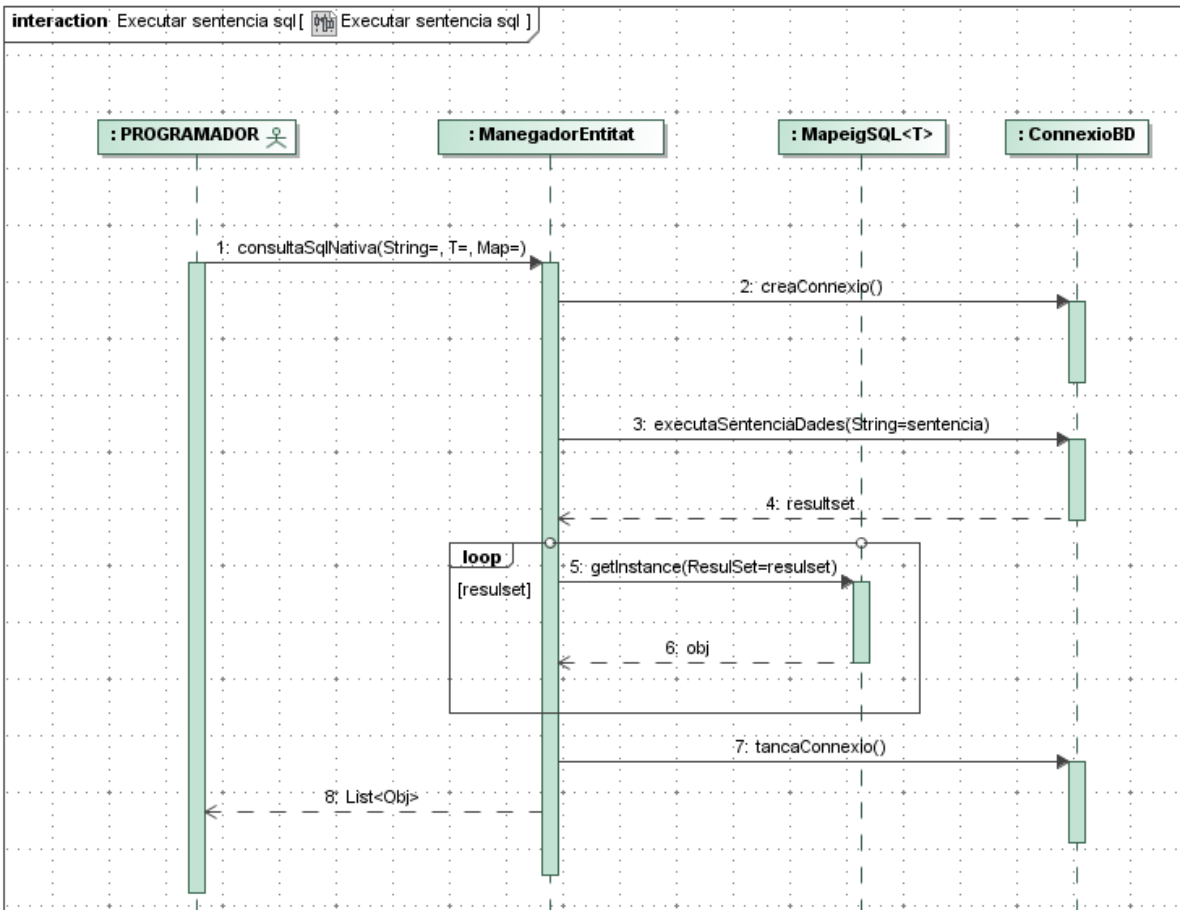


Figura 29: Diagrama seqüència consulta sql nativa.

3.2.10 Cerques amb llenguatge Fwkql

Hem definit un llenguatge semblant al SQL per a poder realitzar cerques a la base de dades. Aquest llenguatge en aquesta primera versió opta per modificar mínimament el llenguatge sql. D'aquesta manera una sentència en Fwkql pot ser la següent: "SELECT NOMTAULA ON :CAMP=? I :CAMP2!=? O CAMP=?. És important que el nom de la taula estigui configurat al fitxer de propietats, així com ficar ":" davant de cada camp que es vol utilitzar per a filtrar. Els camps fan referència als atributs de l'objecte mapejat i no als camps de la base de dades.

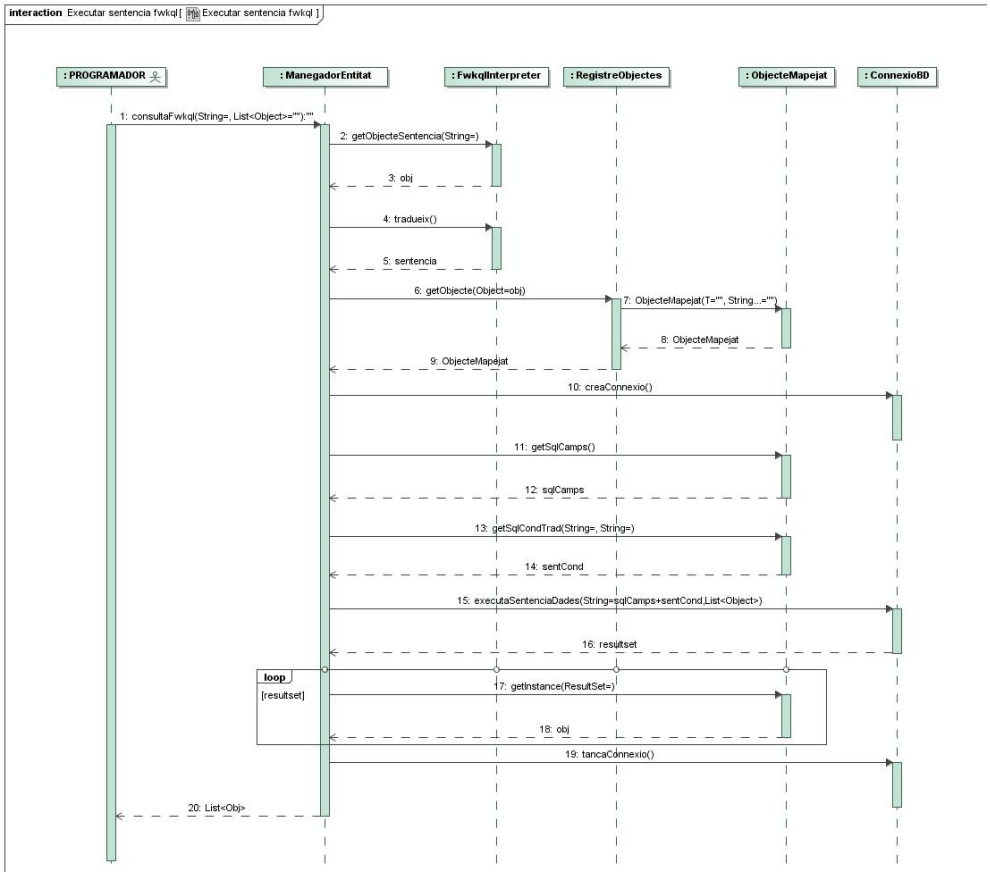


Figura 30: Diagrama seqüència consulta fwkql.

4 Implementació

Per a implementar la llibreria Fwkpers s'ha utilitzat Maven. Maven et permet tindre un repositori comú centralitzat amb totes les llibreries que necessiti la teva aplicació, així com crear-ne i afegir-ne de noves. Un cop s'ha implementat tota la funcionalitat, s'han executat els diferents tests amb Junit i s'ha realitzat la instal·lació al repositori Maven per a poder fer servir la llibreria posteriorment a l'aplicació d'exemple.

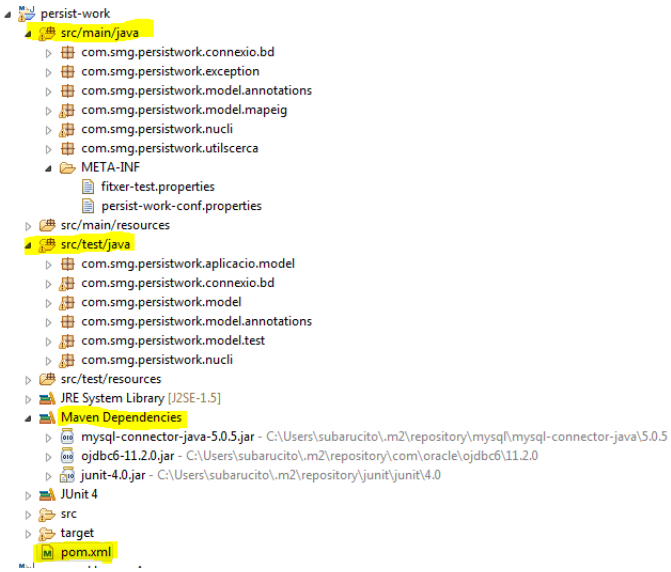


Figura 31: Estructura del projecte del framework.

A continuació es mostren les diferents classes implementades per paquet i s'explica les decisions més importants realitzades a nivell d'implementació de Fwkpers. No es realitza una explicació exhaustiva, només es destaca aquells punts més importants de cada paquet del framework.

4.1 Definició model / Anotacions

Aquest paquet de l'arquitectura defineix les anotacions que s'utilitzen per a mapejar les taules de base de dades en objectes JAVA. S'ha definit les anotacions bàsiques per a poder mapejar una taula, els seus camps, la clau primària i les relacions entre taules (claus foranies).

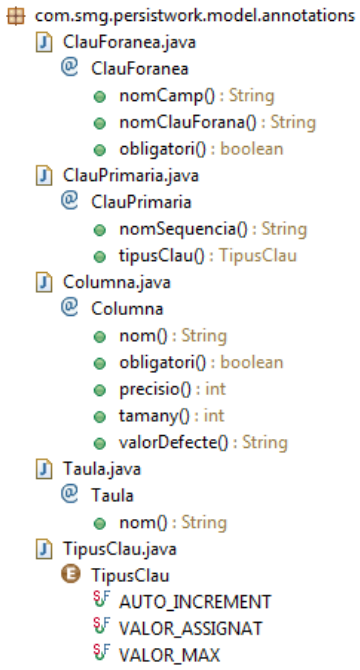


Figura 32: Anotacions per mapejar les taules de base de dades.

4.2 Connexió Base Dades

La connexió amb la base de dades es troba centralitzada en una classe abstracta, que fa servir les llibreries java.sql de J2EE. En aquesta versió només tenim dues classes per sota d'aquesta, una per a realitzar la connexió amb base de dades MySql i una altra per a la connexió amb Oracle.

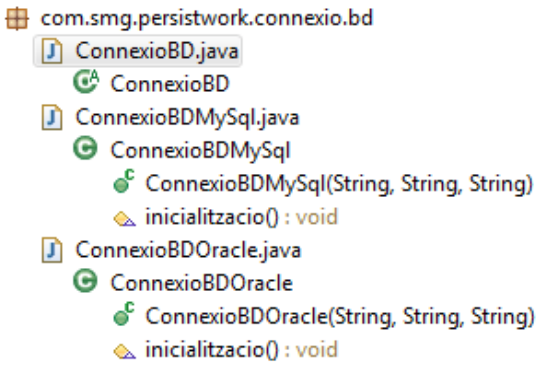


Figura 33: Connexió base de dades.

```java
/* (non-Javadoc)
 * @see com.smg.persistwork.connexio.bd.ConnexioBD#inicialitzacio()
 */
@Override
protected void inicialitzacio() throws PersistWorkException {

    try {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    } catch (Exception e) {
        throw new PersistWorkException(PersistWorkException.BD_11,
                PersistWorkException.BD11_TXT);

    }
}
```

Figura 34: Carrega del driver JDBC per Oracle a ConnexioBDOracle

4.3 Mapeig i Registre d'objectes

A partir de les anotacions, un cop es vol realitzar alguna acció amb un objecte, el que es fa es crear un ObjecteMapejat, aquest s'encarrega de realitzar totes les equivalències entre els camps de la base de dades i els atributs de l'objecte, i quan es vol realitzar una sentència la genera i la retorna cap al manegador d'entitats.

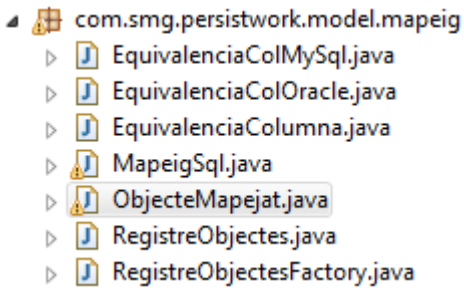


Figura 35: Capa mapeig de Fwkpers.

Quan es crea l'objecte mapejat, es crida el mètode inicialitza, a on es recorren tots els camps declarats per l'objecte per veure si estan anotats. En cas que estiguin anotats, es carreguen totes les propietats que després es fan servir per a generar les sentències i per a crear les instàncies dels objectes que es consulten a la base de dades.

```
private <T> void inicialitza(T objecte, String... id)
        throws PersistWorkException {
    this.objecte = objecte.getClass();

    if (id.length == 0)
        equiv = EquivalenciaColumna.getNewEquivalCol();
    else
        equiv = EquivalenciaColumna.getNewEquivalCol(id[0]);

    if (!objecte.getClass().isAnnotationPresent(Taula.class)) {
        throw new PersistWorkException(PersistWorkException.MAP_9, PersistWor
    }

    Taula t = this.objecte.getAnnotation(Taula.class);

    if (!"".equals(t.nom())) {
        this.nomTaula = t.nom().toUpperCase();
    } else {
        this.nomTaula = objecte.getClass().getSimpleName().toUpperCase();
    }

    for (Field f : objecte.getClass().getDeclaredFields()) {

        if (f.isAnnotationPresent(Columna.class)) {

            Columna c = f.getAnnotation(Columna.class);
            String nom = !"".equals(c.nom()) ? c.nom().toUpperCase() : f
                    .getName().toUpperCase();
            equivalencies.put(nom, f);
            equivcamps.put(f.getName().toUpperCase(), nom);
        }

        if (f.isAnnotationPresent(ClauPrimaria.class)) {
            Columna c = f.getAnnotation(Columna.class);

            String nom = !"".equals(c.nom()) ? c.nom().toUpperCase() : f
                    .getName().toUpperCase();
            equivClausPrimaries.put(nom, f);

        }

        if (f.isAnnotationPresent(ClauForanea.class)) {

            ClauForanea cf = f.getAnnotation(ClauForanea.class);
            String nom = !"".equals(cf.nomCamp()) ? cf.nomCamp()
                    .toUpperCase() : f.getName().toUpperCase();
            equivalencies.put(nom, f);
```

Figura 36: Inicialització de l'objecte mapejat.

4.4 Nucli

Al nucli trobem les classes que s'encarreguen de la configuració de l'entorn. La carrega de les propietats d'un entorn es realitza amb un fitxer propietats per defecte (META-INF/persist-work-conf.properties), o amb un propi. Les propietats que es poden configurar són les següents:

- persistwork.bd.tipus: tipus de connexió (ORACLE o MYSQL).
- persistwork.bd.usuari: usuari per a connectar a la base de dades.
- persistwork.bd.contrasenya: contrasenya per a connectar a la base de dades.
- persistwork.bd.cadenaConnexio: Cadena de connexió de la base de dades. (jdbc:oracle:thin:@localhost:1521:XE)
- persistwork.bd.package: paquet a on es troben els nostres objectes mapejats.

- persistwork.bd.entitats: entitats (nom de les classes) que s'han mapejat, és important que estiguin en ordre de dependència.

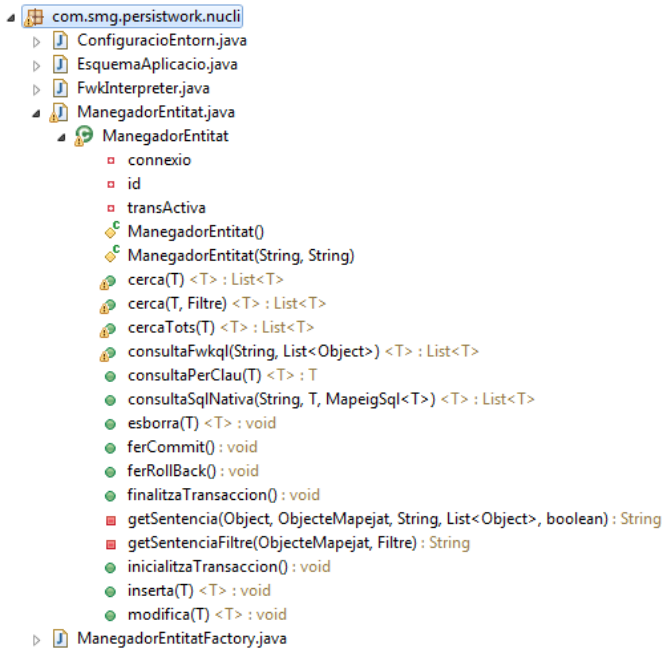


Figura 37: Implementació paquet del nucli.

A més a més al nucli es troben les dues classes més importants, amb les quals es realitzen totes les accions. La classe EsquemaAplicacio s'encarrega de les funcionalitats a nivell d'administració de base de dades, com per exemple crear i esborrar taules; i la classe ManegadorEntitat realitza totes les altres funcionalitats (consultar, cercar, afegir, modificar, esborrar, consultes sql natives, consultes amb filtres i consultes amb fwkql). Per a la implementació d'aquestes classes s'ha utilitzat la utilitat de definició de genèrics que ofereix JAVA, disponible a partir de la versió J2EE 5.0, que permet definir mètodes sense especificar el tipus d'objecte.

```
/**
 * Modifica un registre a la base de dades.
 * @param <T>
 * @param obj
 * @throws PersistWorkException
 */
public <T> void modifica(T obj) throws PersistWorkException {

    RegistreObjectes reg = RegistreObjectesFactory.getRegistre(id);
    ObjecteMapejat objmap = reg.getObjecte(obj);

    if (!transActiva) {
        connexio.creaConnexio();
    }

    List<Object> valors = new ArrayList<Object>();

    String sbModifica = getSentencia(obj, objmap, objmap.getSqlUpdate(),
            valors, false);

    connexio.executaSentencia(sbModifica, valors);
    if (!transActiva) {
        connexio.tancaConnexio();
    }

}
```

Figura 38: Implementació modificació registre .

4.5 Utilitats Cerca

Si es vol realitzar cerques amb filtres amb diferents atributs, en aquest paquet s'han implementat utilitats per a poder-les definir sense problemes. Es poden crear diferents tipus de restriccions, fixant que un atribut sigui igual a un valor, diferent, més gran, més petit, etc. A més d'això, es pot definir també la ordenació de la consulta.

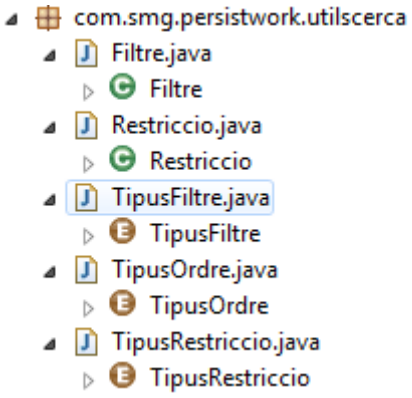


Figura 39: Classes del paquet utilitats cerca.

Totes les restriccions que es poden definir s'afegeixen a un filtre, aquest l'utilitza el nucli per a generar la part de la condició (*"where"*) de la consulta que s'executa a la base de dades.

```
/**
 * Retorna la condició per a afegir a la sentència.
 * @return sentència del filtre
 */
public String getSentencia(){

    StringBuilder st =new StringBuilder();

    if (restriccions.size()>0){
        st.append(" WHERE ");
        for(int i =0;i<restriccions.size();i++){
            if (i!=0){
                if (tipusRestriccions.get(i)==TipusFiltre.AND)
                    st.append(" AND ");
                else
                    st.append(" OR ");
            }
            st.append(restriccions.get(i).getSentencia());
        }
    }

    if (ordenacio.size()>0){
        st.append(" ORDER BY ");
        for(int i =0;i<ordenacio.size();i++){
            if (i!=0){
                st.append(",");
            }
            st.append("@#").append(ordenacio.get(i)).append("@# ");
            if (tipusOrdre.get(i) == TipusOrdre.ASCENDENT)
                st.append("ASC");
            else
                st.append("DESC");
        }
    }
    return st.toString();
}
```

Figura 40: Generació de la sentència condicional del filtre.

4.6 Excepcions

Existeix un tipus d'excepció que s'encarrega d'encapsular els diferents errors que es produeixen en els diferents paquets que formen part de Fwkpers. Per a cada paquet s'ha decidit un interval de codis, a més a més, les descripcions dels errors també es troben a la implementació de la classe.

```
/**
 * Retornarà el missatge que ha produït la excepció.
 * @return missatge d'error
 */
public String getMissatge(){

    StringBuilder st = new StringBuilder();
    st.append("Codi Error: " ).append(codiError).append(". Missatge error: ").append(missatgeError);


    return st.toString();
}
```

Figura 41: Es retorna el missatge de la excepció.

5 Aplicació d'exemple

En aquest capítol s'explica el desenvolupament d'una petita aplicació web que ens serveix per a poder comprovar el funcionament de les llibreries generades pel framework de persistència Fwkpers.

S'ha considerat una aplicació que manega dades d'un departament de recursos humans d'una empresa qualsevol. Aquesta aplicació pot donar d'alta departaments a la empresa, empleats, projectes i assignar empleats als projectes. Un empleat pertany a un departament, i pot dirigir un projecte (cap de projecte), així com estar assignat a diversos projectes.

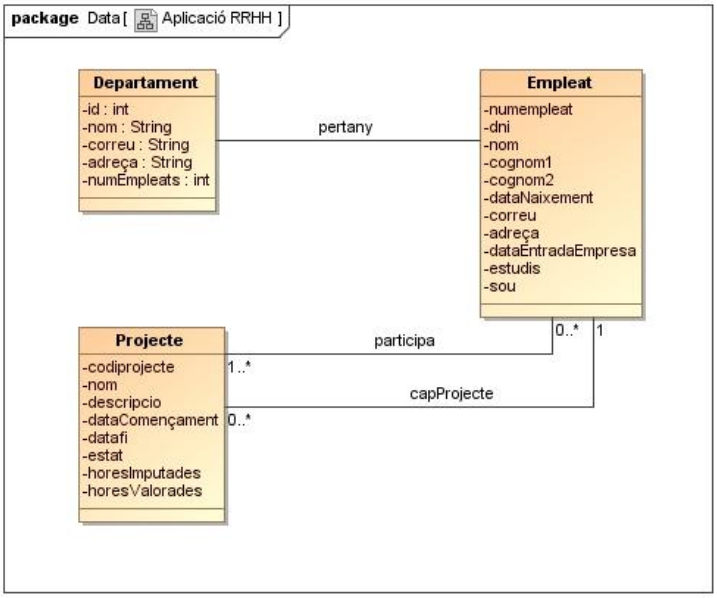


Figura 42: Diagrama de classes Aplicació RRHH.

Es tracta d'una aplicació simple web desenvolupada amb Maven. Maven et permet mitjançant fitxers de configuració afegir dependències d'un repositori comú. D'aquesta forma, hem pogut afegir la llibreria persist-work-1.0.jar sense problemes a l'aplicació web.

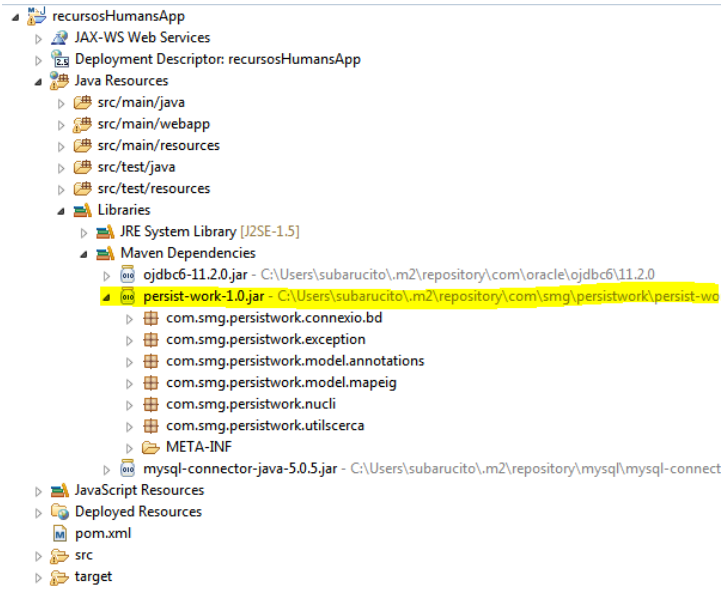


Figura 43: Llibreria persist-work versió 1.0.

A part d'això, tenim dos paquets a on hem creat les nostres classes java que modelen la nostra base de dades, un altre paquet amb les accions que farem sobre la base de dades, i per una altra banda les pàgines JSP a on mostraren els formularis i realitzarem les accions.

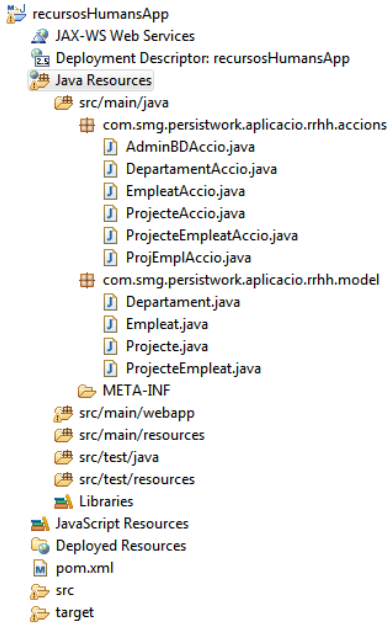


Figura 44: Aplicació web recursosHumansApp, part JAVA.

A continuació mostren les diferents classes que hem creat per a modelitzar la nostra base de dades. Tenim 4 classes diferents: Departament, Empleat, Projecte i EmpleatProjecte.

```java
package com.smg.persistwork.aplicacio.rrhh.model;

import com.smg.persistwork.model.annotations.ClauPrimaria;

/**
 * @author Sergio Maeso
 *
 */
@Taula
public class Departament {


    @Columna(nom="ID_DEPARTAMENT",obligatori=true,tamany=12)
    @ClauPrimaria(tipusClau=TipusClau.AUTO_INCREMENT,nomSequencia="SEQ_DEPARTAMENT")
    private int id;

    @Columna(obligatori=true,tamany=50)
    private String nom;

    @Columna(obligatori=true,tamany=150)
    private String correuElectronic;

    @Columna(obligatori=true,tamany=150,nom="ADRECA_POSTAL")
    private String adreca;

    @Columna(tamany=5, nom="NUMERO_EMPLEATS", valorDefecte="0")
    private int numEmpleats;
```

Figura 45: Classe Departament.

```java
@Taula(nom="EMPLEATS_EMPRESA")
public class Empleat {

    @Columna(nom="NUM_EMPLEAT",obligatori=true,tamany=12)
    @ClauPrimaria(tipusClau=TipusClau.AUTO_INCREMENT,nomSequencia="SEQ_EMPLEATS")
    private int id;

    @Columna(obligatori=true,tamany=10)
    private String dni;

    @Columna(obligatori=true,tamany=30)
    private String nom;

    @Columna(obligatori=true,nom="PRIMER_COGNOM",tamany=60)
    private String cognom1;

    @Columna(nom="SEGON_COGNOM", tamany=60)
    private String cognom2;

    @Columna
    private GregorianCalendar dataNaixement;

    @Columna(tamany=150)
    private String correuElectronic;

    @Columna(tamany=300, obligatori=true, nom="DOMICILI_EMPLEAT")
    private String adreca;

    @Columna(obligatori=true, valorDefecte="SYSDATE")
    private GregorianCalendar dataEntradaEmpresa;

    @Columna(tamany=300)
    private String estudis;

    @Columna(tamany=10,precisio=2, obligatori=true, valorDefecte="0.0")
    private double sou;

    @ClauForanea(obligatori=false,nomCamp="ID_DEPARTAMENT",nomClauForana="FK_EMP_DPT")
    private Departament pertany;
```

Figura 46: Classe Empleat.

```java
@Taula
public class Projecte {

    @Columna(nom="ID_PROJECTE", obligatori=true,tamany=12)
    @ClauPrimaria(tipusClau=TipusClau.VALOR_ASSIGNAT)
    private String id;

    @Columna(obligatori=true,tamany=30)
    private String nom;

    @Columna(obligatori=true,tamany=300)
    private String descripcio;

    @Columna(valorDefecte="SYSDATE")
    private GregorianCalendar dataInici;

    @Columna
    private GregorianCalendar dataFi;

    @Columna(nom="SITUACIO_PROJECTE",obligatori=true,valorDefecte="ESBORRANY", tamany=30)
    private String estat;

    @Columna(tamany=10)
    private long horesValorades;

    @Columna(tamany=10)
    private int horesImputades;

    @ClauForanea(nomClauForana="FK_CAPPROJECTE",obligatori=true)
    private Empleat capProjecte;
```

Figura 47: Classe Projecte.

```java
/**
 * @author Sergio Maeso
 *
 */
@Taula(nom="REL_PROJ_EMP")
public class ProjecteEmpleat {

    @Columna(obligatori=true,tamany=12)
    @ClauPrimaria(tipusClau=TipusClau.AUTO_INCREMENT,nomSequencia="SEQ_PRJEMPL")
    private int id;

    @ClauForanea(nomCamp="IDPROJ",obligatori=true,nomClauForana="FK_REL_PROJ")
    private Projecte projecte;

    @ClauForanea(nomCamp="IDEMPL",obligatori=true,nomClauForana="FK_REL_EMP")
    private Empleat empleat;

    @Columna(tamany=8,precisio=2)
    private double participacio;
```

Figura 48: Classe Relació Projecte i Empleat.

A continuació es mostren les diferents pantalles de l'aplicació indicant les accions que es realitzen fent servir el framework de persistència.

Figura 49: Pàgina principal de l'aplicació

Hem creat dues pantalles per ficar l'aplicació en marxa, en aquestes pantalles es podran crear o esborrar totes les taules de l'aplicació. Aquestes funcionalitats en una aplicació normal no es fan servir molt sovint. Només s'han afegit en aquesta aplicació per a poder testejar les funcionalitats d'administració de base de dades desenvolupades.

Figura 50: Pàgina per a crear les taules.

Figura 51: Pàgina per a esborrar les taules.

En la administració dels departaments es poden donar d'alta departaments i fer cerques per diferents camps, per a aquestes cerques es fan servir els filtres, ja que permeten definir restriccions més amplies i poder retornar el llistat ordenat pel camp que es vulgui.

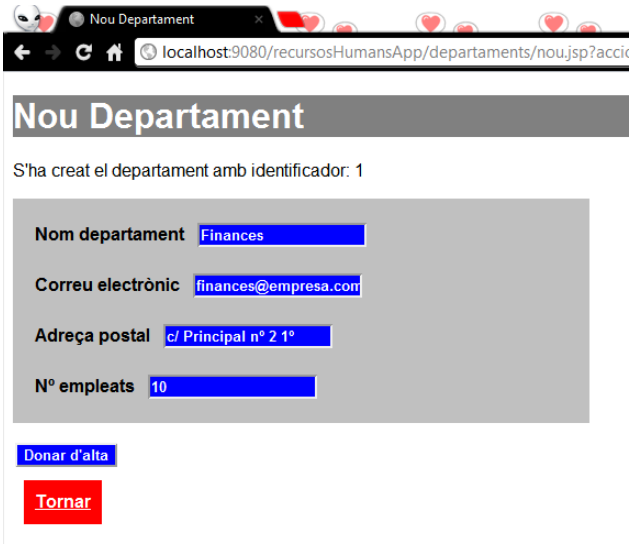


Figura 52: Creació d'un departament.

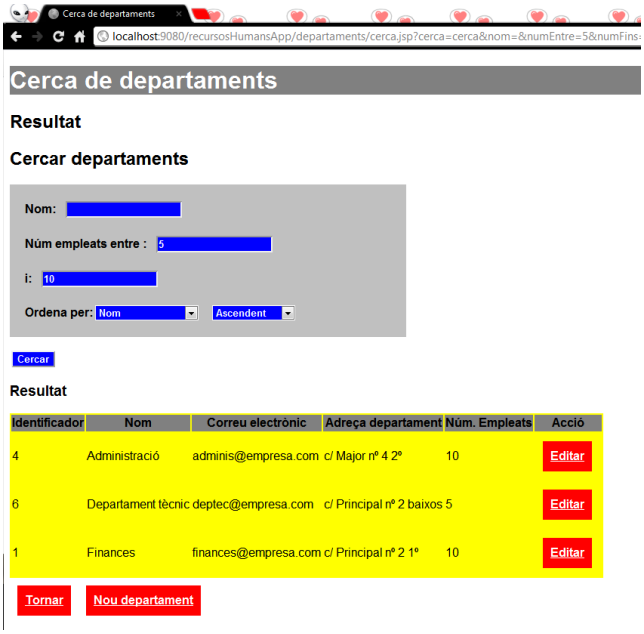


Figura 53: Cerca de departament amb filtres.

Quan es dona d'alta un empleat s'ha de seleccionar un departament per assignar-li. D'aquesta forma es crea la relació entre l'empleat i el departament. A la cerca d'empleats es fa servir la cerca a partir d'un objecte Empleat, a on s'informa els atributs pels quals es vol cercar.

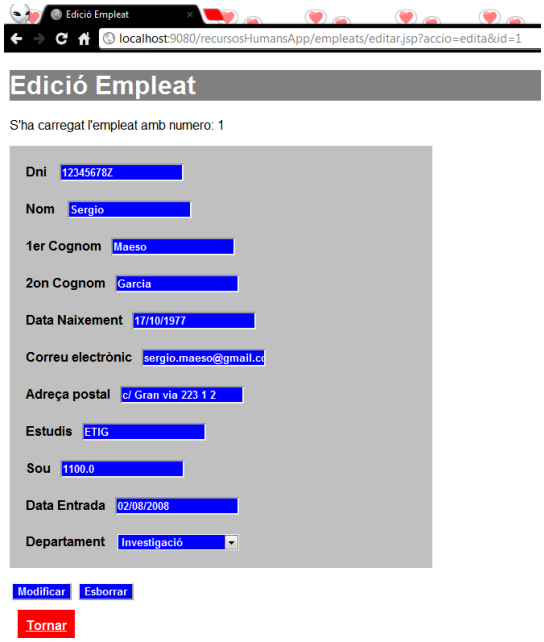


Figura 54: Carrega i edició d'un empleat.

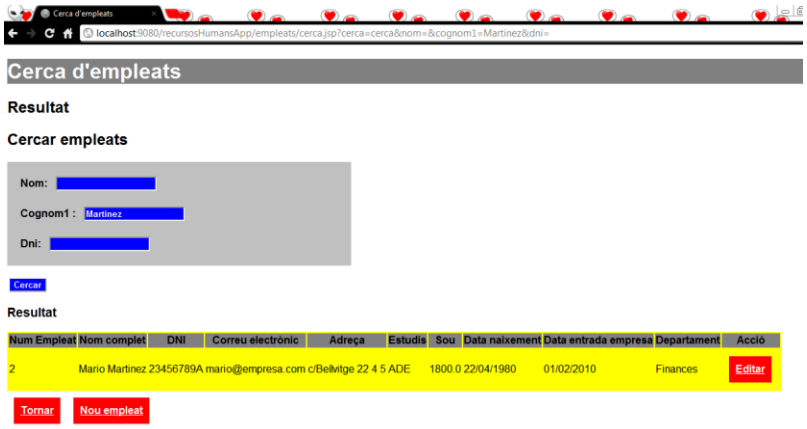


Figura 55: Cerca d'empleats per cognom.

Es poden assignar empleats als projectes creats de forma senzilla i sempre fent servir els objectes mapejats desenvolupats amb les anotacions del framework Fwkpers.

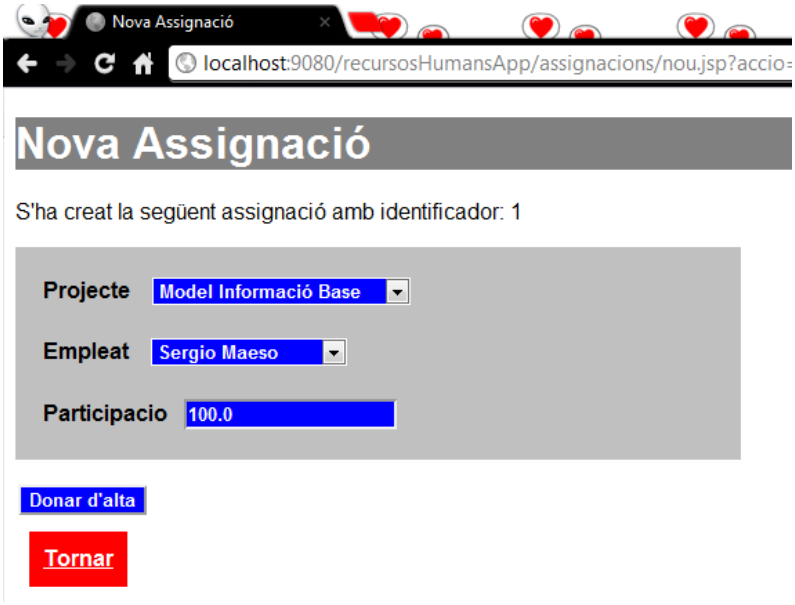


Figura 56: Assignació d'un empleat a un projecte.

6 Conclusions

Gràcies al desenvolupament d'aquest projecte podem considerar que hem assolit i aprés els següents punts:

- Creació de planificacions de projectes. S'ha definit les diferents tasques necessàries per al desenvolupament del projecte i s'ha valorat el seu cost en hores.

- Investigació de mercat. Al fer la investigació dels diferents frameworks de persistència hem obtingut experiència per a futurs projectes, en els que calgui fer un estudi de mercat per decidir la línea del projecte segons la informació obtinguda amb el estudi de mercat.

- Presa de decisions. Un cop realitzat l'estudi de mercat, s'ha realitzat la feina per decidir les diferents característiques imprescindibles que necessitava el nostre framework.

- Disseny del framework. Hem après a fer el disseny d'un framework de persistència a partir de les característiques que havíem decidit que hauria de tindre el nostre framework.

- Implementació amb anotacions. S'ha utilitzat l'ús que dona Java amb les anotacions. Programació amb genèrics i reflexió JAVA. Hem assolit molta experiència en aquest punt alhora del desenvolupament.

- Disseny d'una aplicació per testejar el framework. En base a les diferents funcionalitats que hem proporcionat al nostre framework de persistència, hem dissenyat una petita aplicació per a poder testejar el framework de forma senzilla.

7 Glossari

- Anotació: es una forma d'afegir metadades a les classes Java. Es troba disponible des de la versió 1.5 de J2EE. Es poden afegir a les classes, camps, paràmetres, variables locals i paquets.

- Bean: es tracta d'un component de programari que pot ser reutilitzable i ajuda alhora de programar. Han de complir una sèrie de requeriments:

    - Implementar la interfície Serializable
    - Atributs privats
    - Mètodes get i set per a tots els seus atributs privats
    - Constructor públic per defecte.

- Camp: es cadascuna de les columnes que forma una taula. Contenen dades de tipus diferents a la d'altres camps. Camps possibles podrien ser l'adreça, el nom, el cognom, etc.

- Clau forana: quan una taula te una referència cap a una altra taula es parla de una limitació referencial. Un o dos camps de la taula fan referència a l'altra. La referència ha de ser cap a la clau primària o cap altra clau candidata.

- Clau primària: es tracta d'un conjunt d'un o més atributs d'una taula que permeten identificar de forma unívoca un registre d'una taula.

- CRUD: les quatre accions bàsiques que es realitzen amb un registre són **c**reate, **r**ead, **u**pdate i **d**elete. Crear, llegir, modificar i eliminar. Quan es parla d'accions CRUD es fa referència a aquestes quatre accions.

- EAR: es tracta d'un format de fitxer que usa Java EE per a empaquetar un o diversos mòduls en un únic fitxer, d'aquesta manera es poden instal·lar en servidors d'aplicacions.

- POJO: es tracta d'un *Plain Old Java Object*. S'utilitza aquest nom per a descriure que es tracta d'un objecte ordinari i simple de Java, no té res d'especial. Un simple objecte amb atributs i res més.

- Registre: entenem com registre a un objecte únic de dades que es troba registrat en una taula.

- Taula: es tracta del lloc on es guarden les dades. Les taules estan compostes de dues estructures; els camps i el registre.

8 Bibliografia

- Bauer C. i King G. (2007). "*Java Persistence with Hibernate*". Greenwich: Manning.

- MyBatis Home http://www.mybatis.org/

- OpenJPA http://openjpa.apache.org/

- Oracle TopLink http://www.oracle.com/technetwork/middleware/toplink/overview/index.html

- JSR 12: Java Data Objects (JDO) Specification http://jcp.org/en/jsr/detail?id=12

- Hibernate Documentation http://www.hibernate.org/docs

- Open Source Persistence Frameworks in Java http://java-source.net/open-source/persistence
- Java Data Objectes (JDO) http://www.oracle.com/technetwork/java/index-jsp-135919.html
- Enterprise JavaBeans Documentation http://java.sun.com/products/ejb/docs.html
- Developing with EJB and JPA Components
- http://docs.oracle.com/cd/E16162_01/user.1112/e17455/dev_ejb_jpa.htm
- Migrating EJB 2.x applications to EJB 3.0 http://www.javaworld.com/javaworld/jw-08-2006/jw-0814-ejb.html?page=1

9 Annex: Entorn de treball

Per a la implementació del framework de persistència i del projecte web del departament de RRHH s'ha utilitzat el següent entorn de treball:

- SpringSourceToolSuite 2.9.1 RELEASE: es tracta d'un programa tipus Eclipse optimitzat per a treballar amb el framework de Spring. Té integrat MAVEN i JUNIT.

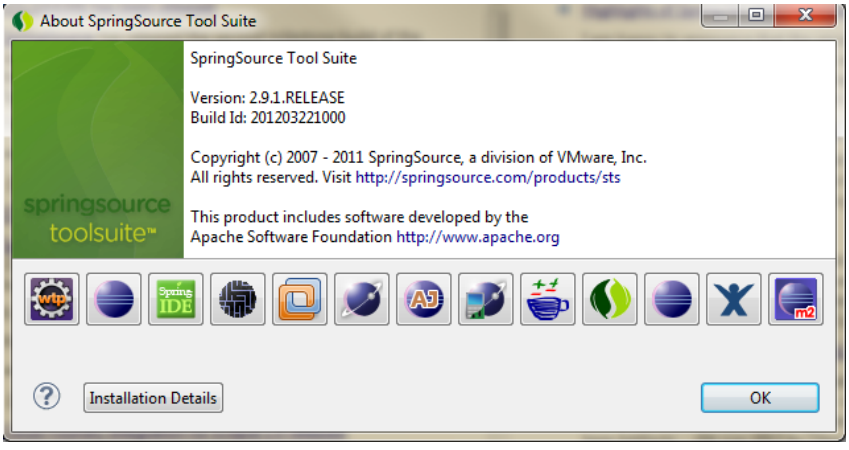


Figura 57: SpringSource Tool Suite

- JDK 1.6.0_24: s'ha compilat el framework amb la versió J2EE 6.0.
- JUnit 4: per a els tests unitaris s'ha fet servir la versió 4 JUnit. El porta integrat el programa SpringSource Tool Suite.
- MAVEN: els projectes s'han implementat utilitzant MAVEN. Maven et permet implementar projectes i decidir si es transformaran en llibreries o en aplicacions web.

Figura 58: Configuració pom.xml framework

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
  <modelVersion>4.0.0</modelVersion>
  <groupId>recursosHumansApp</groupId>
  <artifactId>recursosHumans</artifactId>
  <version>1.0</version>
  <packaging>war</packaging>
  <name>recursosHumansApp</name>

  <description>Aplicació de Recursos Humans</description>
  <dependencies>

        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>11.2.0</version>
        </dependency>

        <dependency>
            <groupId>com.smg.persistwork</groupId>
            <artifactId>persist-work</artifactId>
            <version>1.0</version>
        </dependency>
  </dependencies>
</project>
```

Figura 59: Configuració pom.xml aplicació web.

10 Annex: Base de dades

Per a les bases de dades s'ha utilitzat una base de dades Oracle 11g versió XE i una altra MySql 5.2.

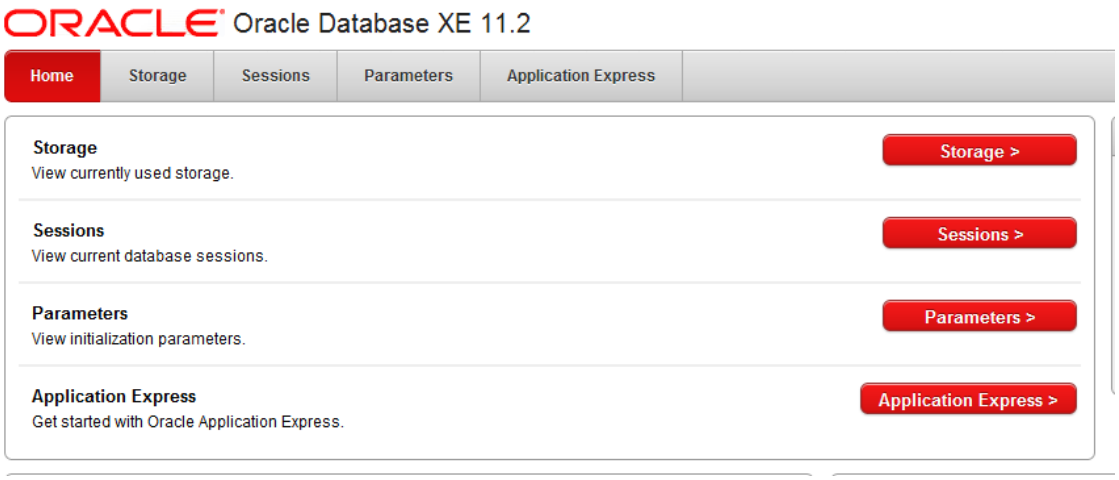


Figura 60: Accés a Application Express

Figura 61: MySQL Workbench.

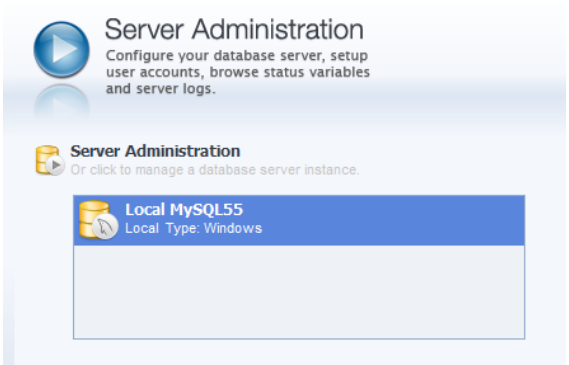


Figura 62: Administració base de dades MySQL.