MyFrameUOC

Disseny i implementació d'un framework de persistència

Jordi Garcia Casanova
Enginyeria en Informàtica

Josep Maria Camps Riba

14 de Gener del 2013

Llicència Creative Commons

Aquest treball està subjecte - excepte que s'indiqui el contrari - en una llicència de Reconeixement-NoComercial-SenseObraDerivada 2.5 Espanya de Creative Commons. Podeu copiar-lo, distribuir-lo i transmetre'l públicament sempre que citeu l'autor i l'obra, no es faci un ús comercial i no es faci còpia derivada. La llicència completa es pot consultar en http://creativecommons.org/licenses/by-nc-nd/2.5/es/deed.es.

Dedicatòria i agraïments

Dedicat als meus pares perquè sense la seva ajuda i el seu suport hauria estat impossible arribar fins aquí. Ells m'han inculcat els valors del treball, la constància i la perseverança que m'han permès lluitar sempre per aconseguir els meus propòsits.

A tots aquells que han aportat els seus coneixements, les seves idees i han resolt amb destresa la majoria dels dubtes que els he plantejat.

I especialment a la Miriam Serrano Mensa per la seva comprensió a l'hora de sofrir les hores que li he dedicat al projecte, pel sacrifici del nostre temps de lleure i per aguantar tants mals de cap durant el meu periple acadèmic en la UOC. Sense ella hauria estat molt més difícil.

Gràcies a tots.

Resum

El projecte s'engloba en l'àrea J2EE i el seu objectiu principal és dissenyar i implementar un framework de persistència que faciliti i agilitzi el desenvolupament de la capa d'integració de les aplicacions J2EE. La finalitat última és proporcionar una eina que permeti implementar la persistència de les dades d'una manera eficient, ràpida i simple.

Un dels problemes que es presenten en la implementació de la capa d'integració és el mapeig entitat-relació: la traducció del paradigma de la programació orientada a objectes cap a les bases de dades relacionals és problemàtica i, per tant, és necessari dur a terme una adaptació prèvia per a poder emmagatzemar un objecte d'una classe en una base de dades.

En conseqüència, el desenvolupador d'aplicacions ha de portar a terme una tasca llarga i feixuga d'implementació. El framework que es desenvolupa en aquest projecte té com a objectiu evitar aquesta tasca i facilitar la implementació de la persistència de les dades. És a dir, evita que el desenvolupador es preocupi dels detalls d'implementació que són necessaris per a emmagatzemar un objecte d'una classe en una base de dades i li proporciona una visió d'orientació a objectes.

Per acabar, el projecte es divideix en tres parts ben diferenciades. En primer lloc, es realitza un estudi i avaluació dels frameworks de persistència més rellevants que hi ha actualment disponibles al mercat. En segon lloc, es du a terme el desenvolupament d'un framework de persistència propi. I en tercer lloc, es desenvolupa una petita aplicació J2EE de prova que mostra l'ús i el funcionament de l'eina desenvolupada.

Paraules clau

- J2EE
- Framework de persistència
- Java
- Bases de dades
- Mapeig entitat-relació

Índex de contingut

Resum **4**

Paraules clau **4**

1. Introducció **10**

1.1. Descripció del projecte 10

1.2. Objectius generals i específics 12

1.3. Enfocament i mètode seguit 13

1.4. Pla de treball 14

2. Què és un framework? **17**

2.1. Frameworks de persistència existents 17

3. Un exemple **18**

4. EJB d'entitat **20**

4.1. Definició i característiques 20

4.2. Exemple 22

4.3. Avantatges i inconvenients 26

5. Java Data Objects (JDO) **27**

5.1. Definició i característiques 27

5.2. Exemple 30

5.3. Avantatges i inconvenients 31

6. iBATIS **32**

6.1. Definició i característiques 32

6.2. Exemple 34

6.3. Avantatges i inconvenients 36

7. JPA (Java Persistence API) **37**

7.1. Definició i característiques 37

7.2. Exemple 38

7.3. Avantatges i inconvenients 40

8. Hibernate **40**

8.1. Definició i característiques 40

8.2. Exemple 42

8.3. Avantatges i inconvenients 43

9. Anàlisis de requeriments de MyFrameUOC **44**

9.1. Requeriments bàsics 44

9.2. Requeriments avançats 46

9.3. Diagrama de casos d'ús 47

10. Consideracions i refinament **48**

10.1. Primeres decisions 48

10.2. Fitxer de configuració 48

10.3. Anotacions 49

10.4. Funcionament de MyFrameUOC 51

10.5. Claus foranes 53

10.6. Relacions d'un a un (1 a 1) 54

10.7. Relacions d'un a molts (1 a N) 56

10.8. Relacions de molts a molts (M a N) 57

10.9. Gestió de la base de dades 57

10.10. Llenguatge UOCQL 58

11. Disseny **58**

11.1. Esquema invariant 59

11.2. Diagrama de classes 59

11.2.1. Capa *Annotations* 61

11.2.2. Capa *Environment* 62

11.2.3. Capa *DataBase* 63

11.2.4. Capa *Mapping* 64

11.2.5. Capa *QueryManagement* 65

11.2.6. Capa *Exceptions* 67

11.3. Diagrames de seqüències 68

11.3.1. Obtenció d'una sessió 68

11.3.2. Registre d'una entitat 69

11.3.3. Emmagatzemament d'un objecte 70

11.3.4. Recuperació d'un objecte 70

11.3.5. Actualització d'un objecte 71

11.3.6. Eliminació d'un objecte 71

12. Implementació **72**

12.1. Eines i maquinari utilitzat 72

12.2. Estructura del directori del projecte MyFrameUOC 72

12.3. Decisions preses en la implementació 73

13. Aplicació d'exemple *FootballTeam* **74**

13.1. Anàlisi de requeriments 74

13.2. Disseny 75

13.3. Implementació 80

13.3.1. Eines i configuracions utilitzades 81

13.3.2. Estructura del directori del projecte *FootballTeam* 82

13.3.3. Implementació 82

14. Conclusions **83**

15. Millores i ampliacions **84**

Glossari **86**

Bibliografia **88**

Annex A: aplicació FootballTeam **90**

Índex de figures

Figura 1. Diagrama de GANTT del projecte 16
Figura 2. Model conceptual d'exemple 18
Figura 3. Model lògic de l'exemple 18
Figura 4. Classe Java de l'entitat *Producte* 19
Figura 5. Classe Java de l'entitat *Cataleg* 20
Figura 6. Disseny del component EJB d'entitat *Cataleg* 22
Figura 7. Interfície local del component *Cataleg* 22
Figura 8. Interfície local d'inici del component *Cataleg* 23
Figura 9. Classe del component EJB d'entitat *Cataleg* 24
Figura 10. Disseny del component EJB d'entitat *Producte* 24
Figura 11. Descriptor de desplegat 26
Figura 12. Interfícies JDO 27
Figura 13. Implementació JDO 28
Figura 14. JDO *Enhancer* 29
Figura 15. Diagrama de les classes de les interfícies bàsiques JDO 30
Figura 16. Descriptor de persistència 31
Figura 17. Exemple d'ús de JDO 31
Figura 18. Solucions d'iBATIS 33
Figura 19. Esquema conceptual de la configuració d'iBATIS 33
Figura 20. Fitxer de propietats d'iBATIS 34
Figura 21. Fitxer SqlMapConfig.xml d'iBATIS 34
Figura 22. Fitxer SqlMap pel cas del catàleg 35
Figura 23. Fitxer SqlMap pel cas del producte 35
Figura 24. Exemple d'ús d'iBATIS 36
Figura 25. Mètodes de la interfície *EntityManager* de l'API de JPA 37
Figura 26. Cicle de vida d'una entitat en JPA 38
Figura 27. Fitxer *persistence.xml* del JPA 39
Figura 28. Anotacions per a la classe *Cataleg* 39
Figura 29. Anotacions per a la classe *Producte* 39
Figura 30. Codi d'exemple amb JPA 39
Figura 31. Arquitectura general d'Hibernate 41
Figura 32. Arquitectura detallada d'Hibernate 41
Figura 33. Fitxer de configuració *Hibernate.cfg.xml* 42
Figura 34. Fitxer de mapeig *Producte.hbm.xml* 43
Figura 35. Fitxer de mapeig *Cataleg.hbm.xml* 43
Figura 36. Exemple d'ús d'Hibernate 43
Figura 37. Estructura del fitxer *MyFrameUOC.xml* 45
Figura 38. Diagrama de casos d'ús de MyFrameUOC 47
Figura 39. Clau forana 53
Figura 40. Clau forana amb *@ForeignKey* 53
Figura 41. Relació d'un a un sense taula 54
Figura 42. Relació d'un a un amb *@ForeignKey* 55
Figura 43. Relació d'un a un amb taula 55
Figura 44. Relació d'un a un amb *@OneToOne*. 55
Figura 45. Relació d'un a molts amb taula 56
Figura 46. Relació d'un a molts amb *@OneToMany* 56
Figura 47. Relació de molts a molts 57
Figura 48. Relació de molts a molts amb *@ManyToMany* 57
Figura 49. Exemple d'una consulta UOCQL 58
Figura 50. Esquema invariant de MyFrameUOC 59
Figura 51. Diagrama de classes de MyFrameUOC 60
Figura 52. Diagrama de classes de la capa *Annotations* 61
Figura 53. Diagrama de classes de la capa *Environment* 62
Figura 54. Diagrama de classes de la capa *DataBase* 63

Figura 55. Diagrama de classes de la capa *Mapping* 65
Figura 56. Diagrama de classes de la capa *QueryManagement* 66
Figura 57. Diagrama de classes de la capa *Exceptions*..... 67
Figura 58. Diagrama de seqüències de l'obtenció d'una sessió..... 68
Figura 59. Diagrama de seqüències del registre d'una entitat 69
Figura 60. Diagrama de seqüències de l'emmagatzemament 70
Figura 61. Diagrama de seqüències de la recuperació 70
Figura 62. Diagrama de seqüències de l'actualització..... 71
Figura 63. Diagrama de seqüències de l'eliminació..... 71
Figura 64. Model conceptual de l'aplicació *FootballTeam*..... 76
Figura 65. Descripció en molt alt nivell del sistema *FootballTeam*..... 76
Figura 66. Arquitectura programari del sistema *FootballTeam* 77
Figura 67. Arquitectura programari refinat del sistema *FootballTeam*..... 78
Figura 68. Diagrama de desplegament del sistema *FootballTeam*..... 78
Figura 69. Refinament del component *PJugadors*..... 79
Figura 70. Refinament del component *PEquips*..... 80
Figura 71. Refinament del component *PEntrenaments* 80
Figura 72. Millora del llenguatge UOCQL..... 84
Figura 73. Millora de les anotacions de MyFrameUOC..... 84
Figura 74. Menú principal 90
Figura 75. Gestió dels equips de futbol..... 90
Figura 76. Jugadors d'un equip..... 90
Figura 77. Gestió dels jugadors 91
Figura 78. Entrenaments d'un jugador 91
Figura 79. Gestió dels entrenaments 91

1. Introducció

En aquest apartat es realitza una introducció del projecte amb l'objectiu de justificar-lo i contextualitzar-lo. En primer lloc, es mostra la seva descripció per a conèixer el problema que s'ha de resoldre. En segon lloc, es defineixen els seus objectius per saber què s'espera obtenir i fins a on arribar. I en tercer lloc, es confecciona el pla de treball que mostra la temporització de les tasques que s'han de seguir durant l'execució del projecte.

1.1. Descripció del projecte

El projecte consisteix en dissenyar i implementar un marc de treball o framework de persistència (a partir d'ara l'anomenarem *MyFrameUOC*) que faciliti i agilitzi la implementació de la capa d'integració d'una aplicació J2EE.

Una aplicació J2EE segueix un estil arquitectònic heterogeni que combina característiques de diferents estils arquitectònics (client-servidor, multicapa o multinivell, orientada a components, orientada a serveis, etc.). De fet, segons l'estàndard J2EE ha de tenir una arquitectura d'un mínim de tres capes:

- **Capa client**: conté els components que s'executen en les màquines client com per exemple els *applets*, les aplicacions *Java stand-alone*, les pàgines HTML, etc.
- **Capa intermèdia (*middle layer*)**: conté els components que s'encarreguen de la lògica de negoci i s'executen en els contenidors d'un servidor d'aplicacions J2EE. A més, aquesta capa es pot subdividir en tres subcapes:
    - **Capa de presentació**: conté els components de la lògica de presentació o d'interacció entre l'usuari i l'aplicació, i s'executen en el contenidor web del servidor d'aplicacions J2EE.
    - **Capa de negoci**: conté els components *Enterprise JavaBean* (EJB) que implementen la lògica de negoci i s'executen dins del contenidor d'EJB.
    - **Capa d'integració**: conté els components que s'encarreguen d'integrar l'aplicació amb els sistemes amb els que ha d'interactuar, com per exemple una base de dades o un *legacy-system*.
- **Capa EIS (sistemes d'informació de l'empresa)**: conté els components que proporcionen les dades emmagatzemades o els serveis d'informació d'altres sistemes. És a dir, en aquesta capa s'ubiquen les bases de dades, els sistemes ERP, els *legacy-systems*, etc.

Pel que fa a la capa d'integració, la seva implementació es pot fer bàsicament de dues maneres:

- **Amb POJO (*Plain Old Java Object*)**: són classes de Java normals que utilitzen alguna tècnica per a accedir a les dades d'una base de dades. Per exemple, poden utilitzar les classes JDBC, els frameworks de persistència, etc.
- **Amb EJB d'entitat**: un component EJB d'entitat permet accedir a les dades d'una base de dades relacional mitjançant dos mecanismes:

    a) **Persistència gestionada pel contenidor (CMP)**. El contenidor d'EJB s'encarrega d'implementar la gestió de la persistència del component i per a fer-ho utilitza un descriptor de desplegat que el desenvolupador defineix.
    b) **Persistència gestionada pel *bean* (BMP)**. En aquest cas, el desenvolupador s'encarrega d'implementar la gestió de la persistència del component EJB d'entitat. És a dir, implementa els mètodes de *callback*[1] i per a fer-ho pot utilitzar les POJO o un framework de persistència.

En el cas dels EJB d'entitat amb CMP, el desenvolupador només s'ha de preocupar de definir el comportament persistent dels seus components mitjançant un descriptor de desplegat i s'oblida dels detalls d'implementació dels accessos a bases de dades. En canvi, en el cas de les POJO i els EJB d'entitat amb BMP, el desenvolupador s'encarrega dels detalls d'implementació de la persistència dels objectes de la lògica de negoci.

En aquesta implementació apareix el problema del mapeig d'entitat-relació que consisteix en resoldre les diferències que hi ha entre els objectes de la lògica de negoci i els relacionals d'una base de dades. És a dir, en el paradigma de la programació orientada a objectes les dades es representen mitjançant classes o objectes i, normalment, no tenen una correspondència directa amb les entitats (taules) que s'emmagatzemen en una base de dades relacional. Per exemple, es poden donar els següents casos:

- El nom d'una classe pot ser diferent al d'una taula.
- El nom d'un atribut o propietat d'una classe pot ser diferent al d'una columna d'una taula.
- Les relacions d'un a molts (1-N) o les relacions de molts a molts (N-M) no es representen de la mateixa manera en el món relacional que en el de l'orientació a objectes.
- L'herència i l'agregació s'han de poder representar en una base de dades relacional.
- Etc.

[1] Els mètodes de *call*-back són un conjunt de mètodes que proporciona l'estàndard J2EE per a portar a terme la gestió de la persistència dels components EJB d'entitat i que són cridats pel contenidor d'EJB. Exemples: *ejbLoad()*, *ejbStore()*, *ejbFind()*, *ejbCreate()*, *ejbRemote()*, etc.

Per tant, cada cop que es desenvolupa una aplicació cal implementar aquests detalls de la capa d'integració i solucionar les diferències que hi ha entre el paradigma de la programació orientada a objectes i les bases de dades relacionals. MyFrameUOC pretén ajudar al desenvolupador perquè pugui dur a terme aquesta tasca de manera senzilla, còmoda i ràpida. D'aquesta manera, podrà dedicar més esforços a la capa de la lògica de negoci que és la que acostuma a tenir més rellevància.

Per a desenvolupar MyFrameUOC es segueix el cicle de vida clàssic o en cascada. El seu ús implica que no es pugui passar a l'etapa següent fins que no es tingui acabada l'etapa anterior. Tot i així, es pot tornar a una etapa anterior si es troben errors. Tot seguit es descriuen les principals etapes que es portaran a terme durant l'execució del projecte.

En primer lloc, es fa un estudi i avaluació dels diferents frameworks de persistència que hi ha disponibles actualment al mercat. L'objectiu és conèixer les seves característiques i poder fer una recopilació d'aquelles més rellevants per a extrapolar-les a MyFrameUOC. A més, es pretén adquirir coneixements i idees sobre com implementar aquestes funcionalitats.

En segon lloc, es porta a terme la descripció del sistema a desenvolupar mitjançant el model de referència RM-ODP i el llenguatge UML. En aquesta fase s'obtenen un conjunt de productes que formen part de l'anàlisi i el disseny: casos d'ús, diagrames de seqüències, diagrames d'entitat-relació, diagrames de classes, etc.

En tercer lloc, mitjançant la descripció feta en el pas anterior es porta a terme la implementació de MyFrameUOC i es fan les proves necessàries per assegurar el seu correcte funcionament.

Per a finalitzar, es desenvolupa una petita aplicació J2EE que mostra l'ús i el funcionament de MyFrameUOC.

1.2. Objectius generals i específics

Per una banda, l'objectiu general del projecte és desenvolupar un framework de persistència (MyFrameUOC) que faciliti i agilitzi la implementació de la capa d'integració de les aplicacions J2EE. D'aquesta manera, el desenvolupador pot evitar començar des de zero la implementació de la capa d'integració i disminuir el cost de desenvolupament.

Per una altra banda, es pretén resoldre la problemàtica del mapeig entitat-relació que apareix a l'hora d'implementar la capa d'integració quan s'utilitza el paradigma de la programació orientada a objectes i les bases de dades relacionals. Per aconseguir-ho es porta a terme una anàlisis dels mecanismes i tècniques que utilitzen els diferents frameworks de persistència disponibles al mercat.

Per acabar, de forma general els objectius que es pretenen aconseguir amb el desenvolupament de MyFrameUOC són els següents:

- Proporcionar les **funcionalitats bàsiques (CRUD[2])** que permetin emmagatzemar un objecte de la lògica de negoci en una base dades. És a dir, MyFrameUOC ha de permetre realitzar les operacions bàsiques de creació, lectura, actualització i eliminació d'un objecte.
- Proporcionar algunes **funcionalitats més avançades** com la utilització de relacions d'un a molts, de molts a molts, etc.
- Obtenir una eina que **faciliti i agilitzi la implementació** de la capa d'integració d'una aplicació J2EE.
- **Reduir el màxim possible la corba d'aprenentatge**. MyFrameUOC ha de ser fàcil d'utilitzar perquè el desenvolupador pugui aprendre el més ràpid possible.
- **Permetre integrar una aplicació amb la majoria de sistemes de gestió de bases de dades**. L'objectiu és que el desenvolupador es despreocupi del SGBD[3] que utilitzi i es concentri en la implementació de les funcionalitats que la seva aplicació ha de proporcionar.
- **Millorar el rendiment del desenvolupador**. Si el desenvolupador dedica menys esforç a la implementació de la capa d'integració, llavors el seu rendiment millorarà perquè el temps de desenvolupament disminuirà.

1.3. Enfocament i mètode seguit

Tal i com ja s'ha esmentat, per a desenvolupar el projecte s'utilitza el cicle de vida clàssic o en cascada, en el que no es pot passar a l'etapa següent fins que no s'acabi l'anterior. La raó d'aquesta elecció és que es considera que el projecte és petit, el desenvoluparà una única persona i les etapes poden ser seqüencials.

Per tant, per al desenvolupament del projecte es segueixen les següents etapes:

- Anàlisi de requeriments
- Disseny
- Implementació
- Proves

Malgrat tot, prèviament a l'anàlisi de requeriments, es porta a terme un procés d'investigació dels frameworks existents al mercat amb l'objectiu d'estudiar el tractament que en fan de la problemàtica del mapeig d'entitat-relació.

A més, per a fer l'anàlisi i el disseny es segueix el model de referència RM-ODP en el que es fa la descripció del sistema des de diferents punts de vista mitjançant diferents diagrames (casos d'ús, diagrames de classes, diagrames de seqüències, etc.).

[2] CRUD són les sigles en anglès de *Create*, Read, Update i Delete.
[3] SGBD són les sigles de Sistemes de Gestió de Bases de Dades.

1.4. Pla de treball

Amb la finalitat de confeccionar el pla de treball es desglossa el projecte en fases i tasques de la següent manera:

1. **Inici del projecte**
2. **Planificació del projecte**
   - 2.1. Selecció de la proposta de projecte
   - 2.2. Elaboració de la descripció del projecte
   - 2.3. Definició dels objectius
   - 2.4. Elaboració del pla de treball
3. **Investigació dels frameworks existents**
   - 3.1. Cerca i recopilació d'informació
   - 3.2. Estudi i avaluació de les diferents alternatives
4. **Desenvolupament de MyFrameUOC**
   - 4.1. Anàlisi de requeriments
   - 4.2. Disseny
   - 4.3. Implementació i proves de les funcionalitats bàsiques
   - 4.4. Implementació i proves de la resta de funcionalitats
5. **Desenvolupament de l'aplicació de prova**
   - 5.1. Anàlisi de requeriments
   - 5.2. Disseny
   - 5.3. Implementació i proves
6. **Documentació**
   - 6.1. Elaboració de la memòria
   - 6.2. Acabament de la memòria
   - 6.3. Elaboració de la presentació
7. **Tancament del projecte**

Algunes consideracions a tenir en compte:

- Les fases 1 i 7 només són formals i, per tant, no suposen cap càrrega de treball.
- La tasca 2.1 consisteix en seleccionar el projecte que es realitzarà i, per tant, cal portar a terme algunes activitats com avaluar les propostes del consultor, cercar informació relacionada per Internet, analitzar quina alternativa és més atractiva o interessant, etc.
- En la tasca 4.3 fa referència a les funcionalitats bàsiques que es considerin oportunes en l'anàlisi de requeriments. En el moment de elaborar el pla de treball encara no es coneix amb exactitud l'abast del problema, però es pot intuir que aquestes seran les operacions CRUD (crear, llegir, actualitzar i esborrar).
- La tasca 6.1 es planifica de tal manera que l'elaboració de la memòria es porti a terme al llarg de tot el projecte i així no es deixi per al final.
- La tasca 6.2 consisteix en finalitzar els detalls de la memòria que puguin queda pendents i perfeccionar-la.

Les fites del projecte es fan coincidir amb les dates de lliurament de les PAC[4] i del lliurament final, i en cadascuna es lliuren els següents productes:

- **10/10/2012 (PAC 1)**: descripció del projecte, definició dels objectius i pla de treball.
- **08/11/2012 (PAC 2)**: investigació dels frameworks existents i anàlisi de requeriments de MyFrameUOC.
- **17/12/2012 (PAC 3)**: disseny de MyFrameUOC, implementació i proves de les funcionalitats bàsiques.
- **14/01/2013 (lliurament final)**: implementació i proves de la resta de funcionalitats de MyFrameUOC, desenvolupament de l'aplicació de prova, memòria i presentació.

[4] PAC són les sigles de Prova d'Avaluació Continuada, i és la denominació que se li dona al treball que es lliura durant el semestre acadèmic en la UOC que serveix per avaluar el progrés de cada estudiant.

I el diagrama de GANTT de la planificació temporal del projecte es mostra en la següent figura:

	Task Name	Duration	Start	Finish
1	Inici del projecte	0 days	Wed 19/09/12	Wed 19/09/12
2	**Planificació del projecte**	**22 days**	**Wed 19/09/12**	**Wed 10/10/12**
3	Selecció de la proposta de projecte	4 days	Wed 19/09/12	Sat 22/09/12
4	Elaboració de la descripció del projecte	4 days	Sun 23/09/12	Wed 26/09/12
5	Definició dels objectius	4 days	Thu 27/09/12	Sun 30/09/12
6	Elaboració del pla de treball	10 days	Mon 01/10/12	Wed 10/10/12
7	**Investigació dels frameworks existents**	**20 days**	**Thu 11/10/12**	**Tue 30/10/12**
8	Cerca i recopilació d'informació	9 days	Thu 11/10/12	Fri 19/10/12
9	Estudi i avaluació de les diferents alternatives	11 days	Sat 20/10/12	Tue 30/10/12
10	**Desenvolupament de MyFrameUOC**	**56 days**	**Wed 31/10/12**	**Tue 25/12/12**
11	Anàlisi de requeriments	9 days	Wed 31/10/12	Thu 08/11/12
12	Disseny	16 days	Fri 09/11/12	Sat 24/11/12
13	Implementació i proves de les funcionalitats bàsiques	23 days	Sun 25/11/12	Mon 17/12/12
14	Implementació i proves de la resta de funcionalitats	8 days	Tue 18/12/12	Tue 25/12/12
15	**Desenvolupament de l'aplicació de prova**	**13 days**	**Wed 26/12/12**	**Mon 07/01/13**
16	Anàlisi de requeriments	2 days	Wed 26/12/12	Thu 27/12/12
17	Disseny	4 days	Fri 28/12/12	Mon 31/12/12
18	Implementació i proves	7 days	Tue 01/01/13	Mon 07/01/13
19	**Documentació**	**118 days**	**Wed 19/09/12**	**Mon 14/01/13**
20	Elaboració de la memòria	111 days	Wed 19/09/12	Mon 07/01/13
21	Acabament de la memòria	3 days	Tue 08/01/13	Thu 10/01/13
22	Elaboració de la presentació	4 days	Fri 11/01/13	Mon 14/01/13
23	Tancament del projecte	0 days	Mon 14/01/13	Mon 14/01/13

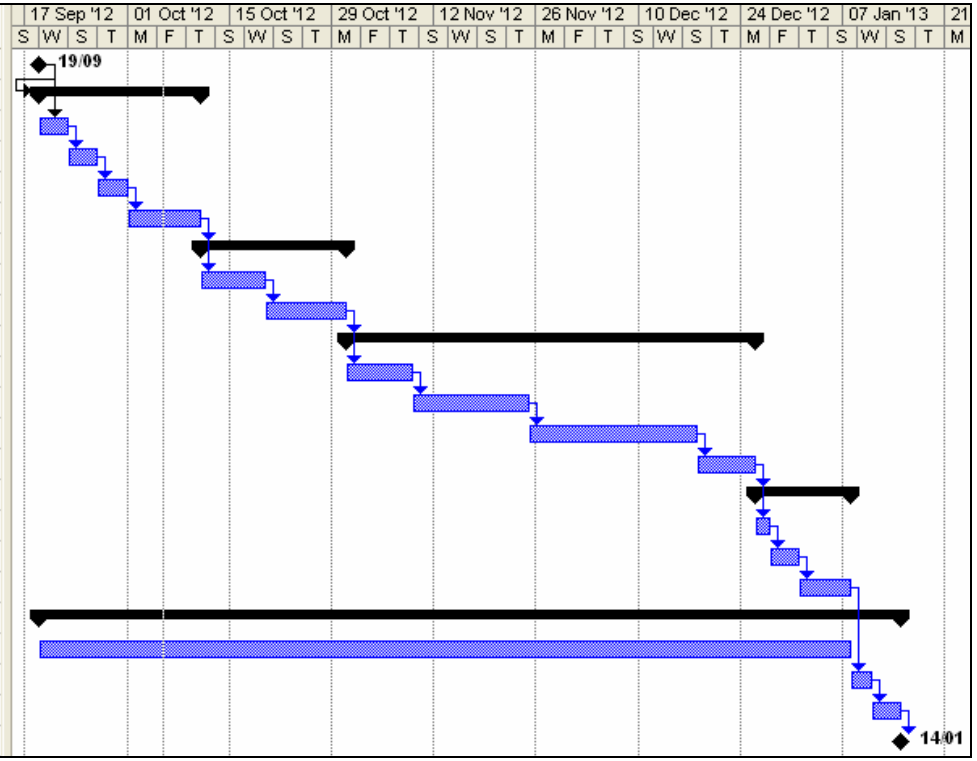

Figura 1. Diagrama de GANTT del projecte

2. Què és un framework?

Per començar, es definirà el concepte de *framework* per a entendre què es el que es pretén solucionar en aquest projecte. Per un costat, es farà una breu exposició de les definicions extretes d'alguns autors o articles i, per un altre costat, s'explica el cas concret d'un framework de persistència.

Un framework és un conjunt de components personalitzables i intercanviables que s'utilitzen per a desenvolupar aplicacions. Així, es pot considerar com una aplicació genèrica, incompleta i configurable a la que se li han d'afegir les últimes peces per a construir una aplicació concreta. Els objectius generals són millorar el procés de desenvolupament, reutilitzar codi ja existent i promoure bones pràctiques com l'ús de patrons.[5]

Un framework és una aplicació reutilitzable i semi-completa que pot ser especialitzada per a produir aplicacions concretes i específiques. El framework descriu els objectes que componen el sistema, com aquests interactuen i quines són les seves responsabilitats.[6]

I un framework de persistència és un framework que s'utilitza per a facilitar i agilitzar la implementació de la persistència de les dades que es tracten en la lògica de negoci d'una aplicació. És a dir, és un conjunt de components i classes que permeten implementar la capa d'integració abstraient els detalls d'implementació dels accessos a les bases de dades. Normalment, utilitzen el mapeig entitat-relació (ORM) que és una tècnica de programació per a vincular els objectes de la programació orientada a objectes amb les dades d'una base de dades relacional.

Aquest projecte tracta del disseny i la implementació d'un framework de persistència que s'anomena MyFrameUOC, i que permet al desenvolupador implementar la capa d'administració de dades d'una forma simple i ràpida.

2.1. Frameworks de persistència existents

Els frameworks de persistència per a Java que hi ha disponibles al mercat i que es consideren transcendents per a definir els requeriments de MyFrameUOC són els següents:

- EJB d'entitat
- JDO (Java Data Objects)
- iBatis
- JPA (Java Persistence API)
- Hibernate

En els següents apartats s'analitzen amb detall i s'exposen les característiques més importants. L'objectiu és conèixer quins conceptes,

[5] Javier J. Gutiérrez, *¿Qué es un framework web?*, pàg. 1.
[6] R. Johnson, B. Foote (1988), *Designing Reusable Classes*, vol.1, núm. 2, pàgs. 22-35.

tècniques i mecanismes utilitzen per a facilitar i agilitzar la implementació de la capa d'integració de les aplicacions J2EE.

Però, abans es presenta un petit sistema d'informació a mode d'exemple que s'implementarà mitjançant l'ús de cadascun dels frameworks. L'objectiu es descobrir el funcionament i les tècniques que utilitzen d'una manera general.

3. Un exemple

Per tal d'entendre el funcionament dels diferents frameworks de persistència es seguirà el següent exemple. Es tracta d'un sistema que permet emmagatzemar els catàlegs de productes d'una botiga virtual, en el que cada catàleg inclou cap o diversos productes, i un producte ha d'estar inclòs en un, i només un, catàleg. El diagrama d'entitat-relació (E/R) o esquema invariant amb les dades que gestiona el sistema és el següent:

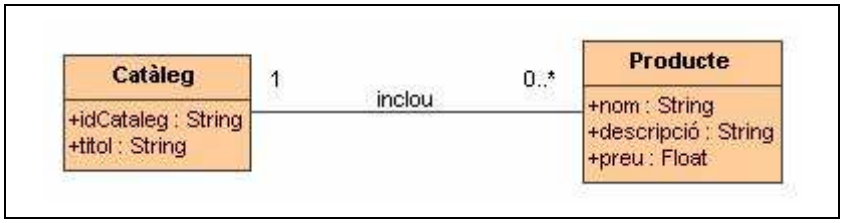

Figura 2. Model conceptual d'exemple

El disseny lògic (model relacional) corresponent a aquest model conceptual seria el següent:

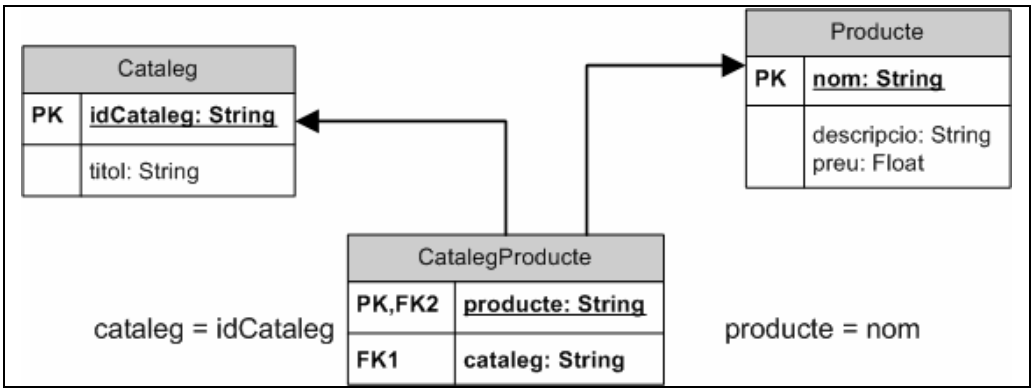

Figura 3. Model lògic de l'exemple

I les classes Java que s'utilitzarien en el paradigma de la programació orientada a objectes per a representar els catàlegs i els productes es mostren a continuació (figures 4 i 5).

```
public class Producte{

    private String nom                = null;
    private String descripcio         = null;
    private Float preu                = null;

    public Producte(){
    }

    public Producte(String nom, String descripcio, BigDecimal preu){
        this.nom = nom;
        this.descripcio = descripcio;
        this.preu = preu;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getDescripcio() {
        return descripcio;
    }

    public void setDescripcio(String descripcio) {
        this.descripcio = descripcio;
    }

    public Float getPreu() {
        return preu;
    }

    public void setPreu(Float preu) {
        this.preu = preu;
    }
}
```

Figura 4. Classe Java de l'entitat *Producte*

```
public class Cataleg{
    private String idCataleg        = null;
    private String titol            = null;
    private Collection productes    = null;

    public Cataleg(){
        productes = new Collection();
    }

    public Cataleg(String idCataleg, String titol){
        this.idCataleg = idCataleg;
        this.titol = titol;
    }

    public String getIdCataleg() {
        return idCataleg;
    }

    public void setIdCataleg(String idCataleg) {
        this.idCataleg = idCataleg;
    }

    public String getTitol() {
        return titol;
    }

    public void setTitol(String titol) {
        this.titol = titol;
    }

    public Collection getProductes() {
```

```
        return productes;
    }

    public void setProductes(Collection productes) {
        this.productes = productes;
    }

    public void afegirProducte(Producte producte){
        this.productes.add(producte);
    }
}
```

Figura 5. Classe Java de l'entitat *Cataleg*

Es pot observar com la relació que hi ha entre l'entitat *Cataleg* i l'entitat *Producte* es representa mitjançant l'atribut *productes* de la classe *Cataleg*. Aquest és una *Collection* que contindrà tots els productes que s'inclouen en un catàleg determinat.

4. EJB d'entitat

En aquest apartat es fa un estudi sobre els *Enterprise Java Beans* (EJB) d'entitat, es mostra un exemple i es presenten els seus avantatges i inconvenients més importants.

4.1. Definició i característiques

Els EJB d'entitat són els components que s'ubiquen en la capa d'integració d'una aplicació J2EE, dins del contenidor d'EJB del servidor d'aplicacions, i serveixen per a fer persistents els objectes o components de la capa de la lògica de negoci.

Un component EJB d'entitat es pot veure com un objecte Java que representa una dada que està emmagatzemada en algun lloc (normalment, una base de dades relacional). Per tant, un EJB d'entitat dura exactament el que dura la dada a la qual representa. De fet, es diu que un EJB d'entitat dura més que un servidor d'aplicacions J2EE, malgrat que això no sigui exactament així perquè el que realment dura més són les dades.

Els EJB d'entitat són objectes intel·ligents perquè ells mateixos poden gestionar la seva pròpia persistència. És a dir, fan el mapeig dels seus atributs amb els camps de les taules de les bases de dades, de tal manera que quan es modifica algun d'aquests atributs són capaços d'actualitzar els camps en les taules.

El contenidor on es desplega un EJB d'entitat proporciona un conjunt de serveis que implementen algunes funcionalitats referents a la persistència (transaccionalitat, concurrència, etc.), de tal manera que el desenvolupador del component EJB es pot despreocupar dels detalls d'implementació d'aquestes.

En canvi, sí que s'ha d'implementar la gestió de la persistència que consisteix en mantenir consistents o fer correspondre les dades que hi ha en un EJB d'entitat amb les dades que hi ha en el magatzem de dades. De la implementació d'aquesta gestió s'encarrega el contenidor d'EJB cridant al mètodes de *callback* cada cop que es crea, es llegeix, s'actualitza o s'esborra la dada que representa l'EJB. Llavors, hi ha dues maneres d'implementar la gestió de la persistència (aquests mètodes de *callback*) d'un EJB d'entitat:

- **Persistència gestionada pel *bean* (BMP)**. El propi desenvolupador s'encarrega d'implementar els mètodes de *callback*.
- **Persistència gestionada pel contenidor (CMP)**. El contenidor implementa els mètodes de *callback* mitjançant un descriptor de desplegat (fitxer XML) que el desenvolupador configura o defineix.

Per exemple, el contenidor d'EJB crida al mètode de *callback* anomenat *ejbLoad()* perquè el component EJB d'entitat carregui les dades de la font de dades, o crida al mètode *ejbStore()* perquè emmagatzemi en la base de dades els canvis que s'hagin produït en els seus atributs.

En la persistència gestionada pel *bean* (BMP) el desenvolupador disposa de dues opcions per a implementar els mètodes de *callback*:

- Utilitzar les **POJO** i alguna tècnica (com les llibreries JDBC) per a emmagatzemar les dades de l'EJB en la base de dades.
- Utilitzar un **framework de persistència**.

En canvi, en la persistència gestionada pel contenidor (CMP) és el propi contenidor l'encarregat d'implementar els mètodes de *callback*. I per a fer-ho utilitza un descriptor de desplegat (fitxer XML) en el qual el desenvolupador defineix quina és la vinculació que hi ha entre els atributs de l'EJB i els camps de la font de dades.

Un altre aspecte que cal tenir en compte és la gestió de les relacions. Els EJB d'entitat poden representar dades que estiguin relacionades entre elles. Si s'utilitza BMP, llavors el desenvolupador s'ha d'encarregar d'implementar les relacions (relacions gestionades pel Bean o BMR). Per contra, si s'utilitza CMP, llavors és el contenidor qui s'ha d'encarregar d'implementar-les mitjançant el descriptor de desplegat (relacions gestionades pel contenidor o CMR). Per tant, el desenvolupador també ha de definir i configurar aquests camps de relació en el descriptor de desplegat.

A més a més, també es pot utilitzar un llenguatge de consultes, anomenat EJB QL, que permet definir, declarar i configurar el comportament dels mètodes de cerca que es poden definir per a cada component.

En conclusió, els EJB d'entitat amb CMP i CMR permeten implementar la capa d'integració mitjançant un descriptor de desplegat. Aquest consisteix en un fitxer XML en el qual es defineix el comportament persistent de l'EJB:

- La taula que està associada a l'EJB.
- Les vinculacions entre els atributs de l'EJB i els camps de la taula.
- Les relacions en les que intervé l'EJB.

4.2. Exemple

Per a implementar l'exemple de la botiga virtual, exposat en el capítol anterior, es necessita un component EJB d'entitat per a representar els catàlegs i un altre per als productes. I es considera que l'accés a aquests components EJB d'entitat és local, és a dir, que els components de la lògica de negoci i els EJB d'entitat s'ubiquen en la mateixa màquina virtual.

Per un costat, el disseny del component *Cataleg* és el següent:

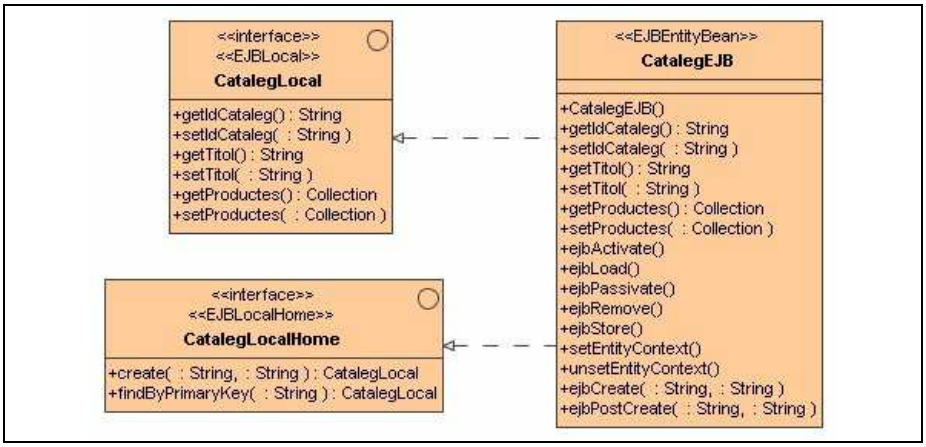

Figura 6. Disseny del component EJB d'entitat *Cataleg*

La classe Java que implementa la interfície *CatalegLocal* es pot veure en la figura 7. Aquesta interfície especifica els mètodes que els clients (els components de la lògica de negoci) poden cridar.

```
public interface CatalegLocal extends EJBLocalObject
{
    //Getters i Setters pels camps CMP.
    public String getIdCataleg();
    public void setIdCataleg(String idCataleg);
    public String getTitol();
    public void setTitol(String titol);

    //Getters i Setters pel camp CMR.
    public Collection getProductes();
    public void setProductes(Collection productes);
}
```

Figura 7. Interfície local del component *Cataleg*

La classe Java que implementa la interfície *CatalegLocalHome* es pot veure en la figura 8. Aquesta interfície especifica els mètodes relacionats amb la gestió del cicle de vida del component EJB d'entitat i que els clients podran utilitzar.

```
public interface CatalegLocalHome extends EJBLocalHome
{
    public CatalegLocal create(String idCataleg, String titol) throws CreateException;
    public CatalegLocal findByPrimaryKey(String idCataleg) throws FinderException;
}
```

Figura 8. Interfície local d'inici del component *Cataleg*

Els mètodes de les interfícies *CatalegLocal* i *CatalegLocalHome* s'implementen en la classe *CatalegEJB* que és la classe del component d'EJB d'entitat corresponent al catàleg. La seva implementació és la següent:

```
public abstract class CatalegEJB implements EntityBean
{
    //Constructor.
    public CatalegEJB()
    {
        super();
    }

    //Mètodes de callback.
    public void ejbActivate() throws EJBException, RemoteException
    {
    }

    public void ejbLoad() throws EJBException, RemoteException
    {
    }

    public void ejbPassivate() throws EJBException, RemoteException
    {
    }

    public void ejbRemove() throws RemoveException, EJBException,
            RemoteException
    {

    }

    public void ejbStore() throws EJBException, RemoteException
    {
    }

    public void setEntityContext(EntityContext arg0) throws EJBException,
            RemoteException
    {
    }

    public void unsetEntityContext() throws EJBException, RemoteException
    {
    }

    public    CatalegLocal    ejbCreate(String    idCataleg,    String    titol)    throws
CreateException
    {
        try
        {
            this.setIdCataleg(idCataleg);
            this.setTitol(titol);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
```

```
        return this.getProductID();
    }

    public void ejbPostCreate(String idCataleg, String titol) throws CreateException
    {
    }

    //Getters i Setters dels camps CMP.
    public abstract String getIdCataleg();
    public abstract void setIdCataleg(String idCataleg);
    public abstract String getTitol();
    public abstract void setTitol(String titol);

    //Getters i Setters del camp CMR.
    public abstract Collection getProductes();
    public abstract void setProductes(Collection productes);
}
```

Figura 9. Classe del component EJB d'entitat *Cataleg*

Es pot observar com el desenvolupador només ha d'especificar la firma dels *getters* i *setters* dels atributs que han de ser persistents. El contenidor s'encarregarà de la seva implementació mitjançant el descriptor de desplegat que es mostra més endavant en la figura 11.

Per un altre costat, el disseny del component *Producte* seria el següent:

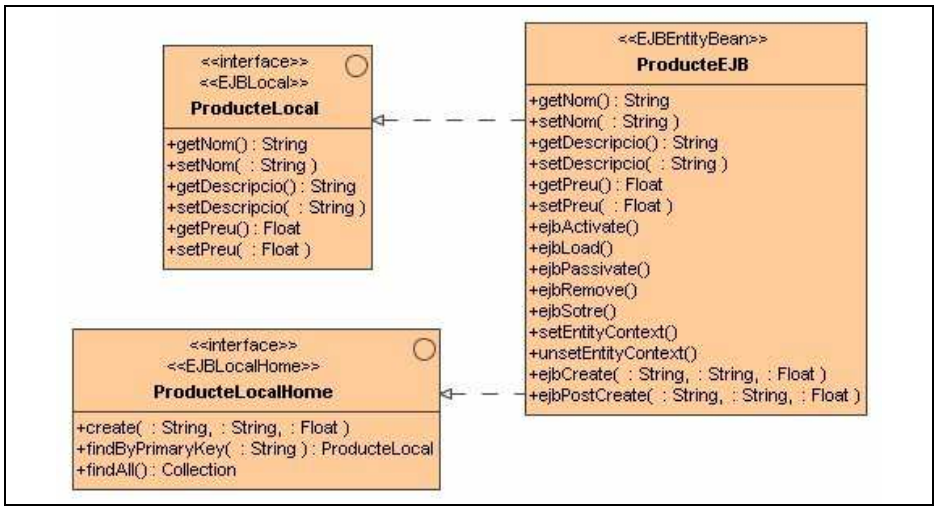

Figura 10. Disseny del component EJB d'entitat *Producte*

La implementació de P*roducteLocal*, *ProducteLocalHome* i *ProducteEJB* és molt similar a la que s'ha exposat en el cas del catàleg. Per tant, es considera innecessari mostrar-les ja que l'objectiu principal és estudiar com els EJB d'entitat permeten facilitar la implementació de la capa d'integració.

A més a més, s'ha d'especificar el comportament que es vol que tinguin els components EJB d'entitat perquè el contenidor pugui implementar els detalls de persistència. I això es fa mitjançant un fitxer XML, anomenat descriptor de desplegat, i que en el nostre cas és el següent:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar >
   <description><![CDATA[Exemple catàleg botiga virutal]]></description>
   <display-name>Exemple</display-name>

   <enterprise-beans>
        <!-- Session Beans -->
        <!-- Aquí aniria la descripció dels components EJB de sessió de la lògica de
negoci.

        <!-- Entity Beans -->
        <entity>
            <description><![CDATA[EJB dels catàlegs]]></description>
            <display-name>EJB Cataleg</display-name>
            <ejb-name>Cataleg</ejb-name>
            <local-home>uoc.exemple.integracio.interfaces.CatalegLocalHome</local-home>
            <local>uoc.exemple.integracio.interfaces.CatalegLocal</local>
            <ejb-class>uoc.exemple.integracio.ejb.CatalegEJB</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>False</reentrant>
            <cmp-version>2.x</cmp-version>
            <abstract-schema-name>Cataleg</abstract-schema-name>
            <cmp-field>
                    <description><![CDATA[Getter pel CMP idCataleg]]></description>
                    <field-name>idCataleg</field-name>
            </cmp-field>
            <cmp-field>
                    <description><![CDATA[Getter pel CMP titol]]></description>
                    <field-name>titol</field-name>
            </cmp-field>
            <primkey-field>idCataleg</primkey-field>
        </entity>

        <entity>
            <description><![CDATA[EJB dels productes]]></description>
            <display-name>EJB Producte</display-name>
            <ejb-name>Producte</ejb-name>
            <local-home>uoc.exemple.integracio.interfaces.ProducteLocalHome</local-
home>
            <local>uoc.exemple.integracio.interfaces.ProducteLocal</local>
            <ejb-class>uoc.exemple.integracio.ejb.ProducteEJB</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>False</reentrant>
            <cmp-version>2.x</cmp-version>
            <abstract-schema-name>Producte</abstract-schema-name>
            <cmp-field>
                    <description><![CDATA[Getter pel CMP nom]]></description>
                    <field-name>nom</field-name>
            </cmp-field>
            <cmp-field>
                    <description><![CDATA[Getter pel CMP descripcio]]></description>
                    <field-name>descripcio</field-name>
            </cmp-field>
            <cmp-field>
                    <description><![CDATA[Getter pel CMP preu]]></description>
                    <field-name>preu</field-name>
            </cmp-field>
            <primkey-field>nom</primkey-field>

            <query>
                <query-method>
                    <method-name>findAll</method-name>
                    <method-params></method-params>
                </query-method>
                <ejb-ql>
                    <![CDATA[SELECT OBJECT(a) FROM Producte AS a WHERE a.nom IS NOT
NULL]]>
                </ejb-ql>
            </query>
       </entity>
   </enterprise-beans>
```

```
   <!-- Relationships -->
   <relationships >
        <ejb-relation>
            <ejb-relation-name>Cataleg-Producte</ejb-relation-name>
            <ejb-relationship-role>
                <ejb-relationship-role-name>Cataleg-Category</ejb-relationship-role-
name>
                <multiplicity>Many</multiplicity>
                <relationship-role-source>
                    <ejb-name>Cataleg</ejb-name>
                </relationship-role-source>
                <cmr-field>
                    <cmr-field-name>productes</cmr-field-name>
                    <cmr-field-type>java.util.Collection</cmr-field-type>
                </cmr-field>
            </ejb-relationship-role>
            <ejb-relationship-role>
                <ejb-relationship-role-name>Producte-Cataleg</ejb-relationship-role-
name>
                <multiplicity>One</multiplicity>
                <relationship-role-source>
                    <ejb-name>Producte</ejb-name>
                </relationship-role-source>
            </ejb-relationship-role>
        </ejb-relation>
   </relationships>
</ejb-jar>
```

Figura 11. Descriptor de desplegat

En aquest descriptor es pot observar com s'utilitza el llenguatge EJB QL per a definir el comportament del mètode *findeAll()* de la interfície *ProducteLocalHome* que serveix per a obtenir tots els productes que hi ha emmagatzemats en la base de dades.

En conclusió, els components EJB d'entitat faciliten la implementació de la capa d'integració mitjançant l'ús d'un descriptor de desplegat que el propi desenvolupador defineix i configura. A més, permet utilitzar un llenguatge de consultes propi, anomenat EJB SQL, que permet definir el comportament dels mètodes de cerca (*finders*).

4.3. Avantatges i inconvenients

Els avantatges dels EJB d'entitat són els següents:

- Independència de la plataforma. Suportat per l'estàndard J2EE i, per tant, tots els servidors d'aplicacions J2EE que compleixin amb la norma suportaran aquesta tecnologia.
- Escalabilitat.
- Allibera el desenvolupador d'aspectes com la transaccionalitat, la seguretat, la concurrència, les connexions a les bases de dades, *pooling*, *multi-thread*, etc.

I els inconvenients són els següents:

- La corba d'aprenentatge pot ser gran. Es necessita temps i experiència per a dominar la tecnologia EJB.

- Necessiten un contenidor d'EJB d'entitat.
- Les classes que s'han de fer persistents han d'implementar les interfícies EJB (en l'exemple que s'ha vist són: *CatalegLocal, CatalegLocalHome, ProducteLocal* i *ProducteLocalHome*).
- S'ha de definir un descriptor de desplegat que pot resultar ser tediós.
- No suporta herència entre classes.
- Tot i que és un estàndard, cada proveïdor té masses extensions i, a la pràctica, no resulta tan fàcil moure una aplicació d'un servidor d'aplicacions J2EE a un altre.
- Les proves son difícils i lentes.

5. Java Data Objects (JDO)

En aquest apartat es fa un estudi sobre l'estàndard JDO, es mostra un exemple i es presenten els seus avantatges i inconvenients més importants.

5.1. Definició i característiques

Java Data Objects (JDO) és un estàndard que especifica un conjunt d'interfícies per a fer transparent i genèric la persistència dels objectes de Java. JDO descriu quins mètodes s'han d'implementar per a emmagatzemar, consultar i recuperar objectes d'una base de dades i proporcionar al desenvolupador una visió basada en la programació orientada a objectes. És a dir, el desenvolupador pot utilitzar aquestes interfícies per a proporcionar persistència als seus objectes de forma transparent.

Així, l'especificació JDO defineix un conjunt d'interfícies (JDO API) que el desenvolupador pot utilitzar per a proporcionar persistència a les seves classes i també defineix un altre conjunt d'interfícies (JDO SPI) que els proveïdors han d'implementar per a obtenir una implementació específica de JDO. En la figura següent es pot veure una representació gràfica:

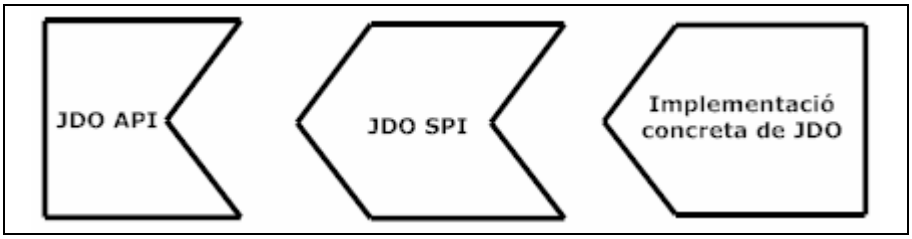

Figura 12. Interfícies JDO

Algunes de les característiques de l'estàndard JDO són les següents:

- JDO és transparent als detalls d'implementació del mapeig entre les instàncies JDO i les entitats del magatzem de dades.

- JDO és transparent als objectes de Java que es fan persistents. És a dir, el desenvolupador no ha de modificar les seves classes. A més, tampoc ha de modificar la visibilitat dels atributs. De fet, els atributs sense *getters* i *setters* no són cap problema per a JDO.
- JDO es pot utilitzar en diferents tipus de bases de dades com les relacionals, les orientades a objectes, sistemes de fitxers, XML, etc.
- JDO és transparent al magatzem de dades. Això permet que les aplicacions funcionin encara que es canviï el tipus de magatzem de dades sempre i quan hi hagi una implementació JDO disponible per al magatzem de dades en qüestió. O dit d'una altra manera, l'aplicació és portable a diferents implementacions de l'especificació JDO.
- Si una aplicació utilitza una referència a un objecte que és persistent i el modifica, llavors la implementació JDO actualitzarà implícitament la entitat de la font de dades (quan es faci el *commit* de la transacció activa). Això permet alliberar al programador de la repetició de codi per a emmagatzemar l'objecte que modifica.

JDO necessita que es defineixi la correspondència entre els objectes Java i les dades de la font de dades. Aquesta vinculació entre els atributs persistents d'una classe i les dades d'una base de dades es fa mitjançant un descriptor de persistència (fitxer XML).

Per tant, la implementació concreta de JDO s'encarrega dels detalls d'implementació de la persistència dels objectes Java i permet que el desenvolupador s'oblidi d'ells. A més, les aplicacions que utilitzen la JDO API es poden executar en diferents implementacions JDO sense haver de modificar o compilar el codi font, tal i com es pot veure en la següent figura:

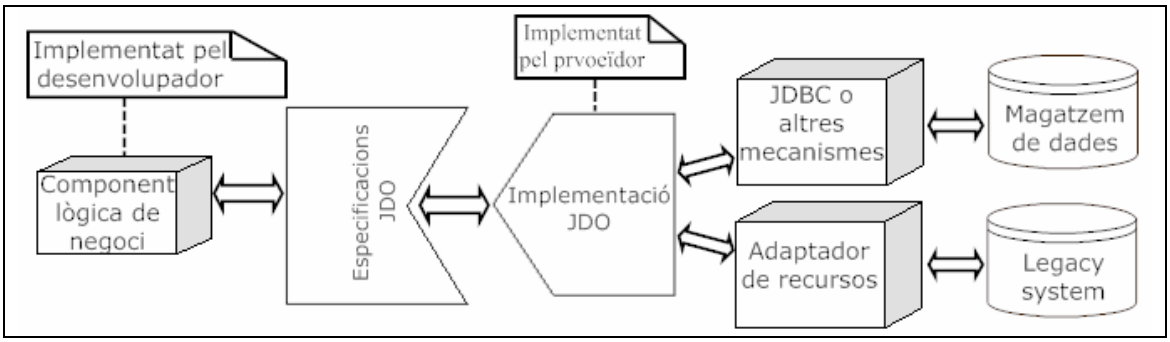

Figura 13. Implementació JDO

La majoria de proveïdors d'implementacions de JDO modifiquen el *bytecode* font (codi interpretat per la màquina virtual de Java) per tal d'evitar la modificació del codi font Java que sol ser confús i permetre que els detalls d'implementació de la persistència quedin ocults al programador. En la figura 14 es mostra un esquema d'aquesta millora del *bytecode*.

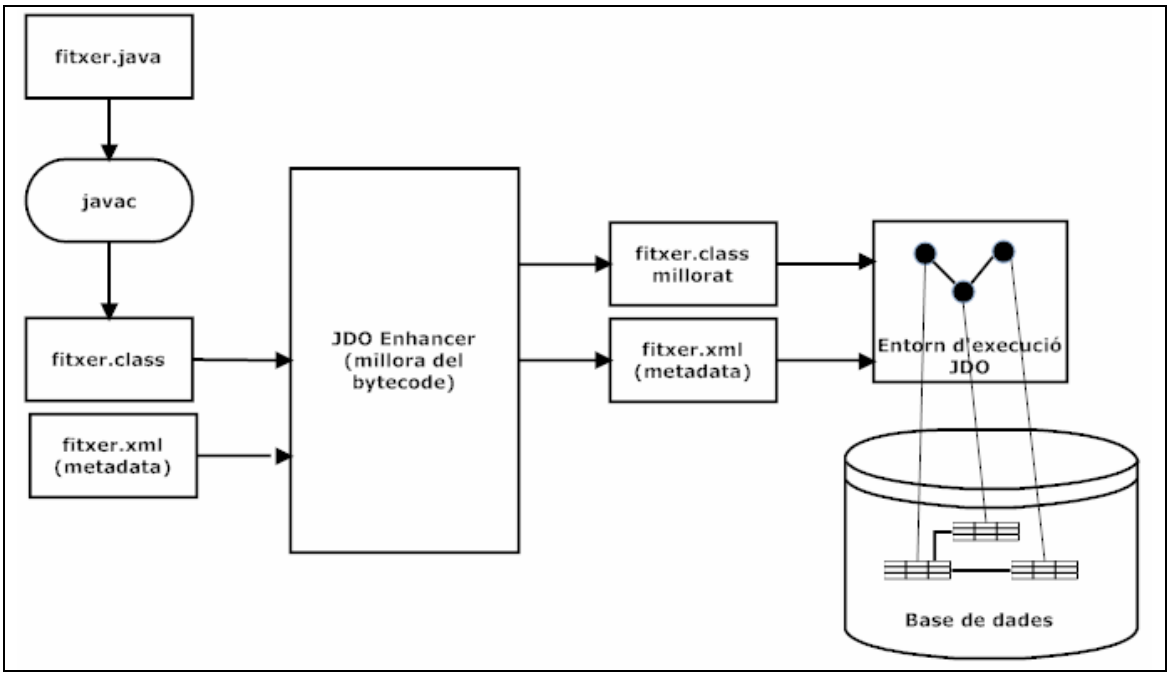

Figura 14. JDO *Enhancer*

L'especificació JDO defineix moltes interfícies i classes, però hi ha cinc classes bàsiques que el desenvolupador ha de conèixer:

- **PersistenceManagerFactory**. S'utilitza per a obtenir una instància de *PersistenceManager*.
- **PersistenceManager**. Encapsula la connexió entre el magatzem de dades i la memòria on es troben els objectes persistents. Aquesta interfície és el principal medi entre els objectes persistents en memòria i la font de dades. Una aplicació pot obtenir instàncies de *Extent, Query* i *Transaction* mitjançant aquesta interfície.
- **Query**. Permet cercar objectes persistents de l'aplicació mitjançant els valors dels seus atributs.
- **Extent**. Representa totes les instàncies d'una determinada classe (i opcionalment, de les seves subclasses) que estan emmagatzemades en una base de dades.
- **Transaction**. Permet controlar les transaccions. Una instància de *PersistenceManagerFactory* només pot crear una instància de *Transaction*. Una transacció s'ha d'iniciar explícitament i s'ha de finalitzar amb un *commit* o *rollback*.

En el següent diagrama de classes es mostra com es relacionen aquestes classes:

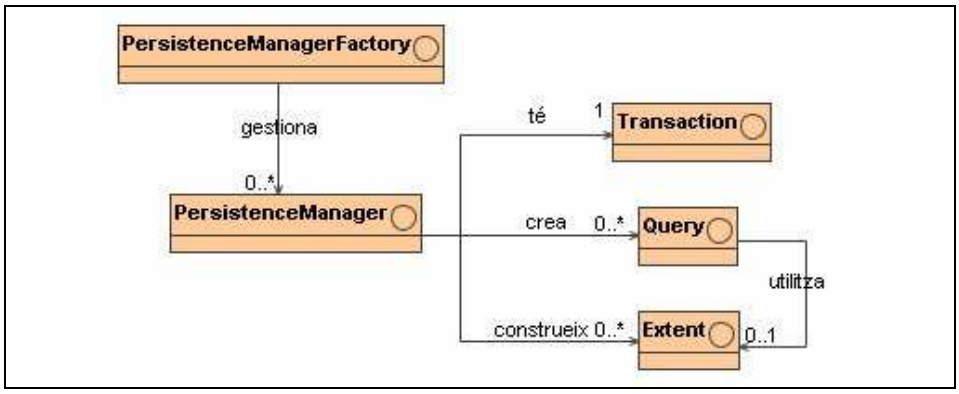

Figura 15. Diagrama de les classes de les interfícies bàsiques JDO

JDO també proporciona un llenguatge de consultes, anomenat JDOQL, que permet realitzar consultes sobre els objectes persistents mitjançant una sintaxis propera a Java.

Alguns exemples d'implementació de les especificacions JDO són els següents: enJin, FastObjects, FrontierSuite, IntelliBO, JDO Genie, JRelay, Kodo JDO, LiDO, OpenFusion JDO, Orient, PE:J The Productivity Environment, etc.

5.2. Exemple

Segons l'exemple que s'ha presentant anteriorment es farà la implementació mitjançant l'ús de JDO.

Les classes dels productes i dels catàlegs quedarien exactament com les que es mostren en les figures 4 i 5. És a dir, que no sofreixen cap modificació.

Ara bé, JDO necessita un descriptor de persistència en format XML que defineixi la correspondència entre els objectes Java i els objectes relacionals de la base de dades:

```
<jdo>
    <package name="uoc.exemple.integracio">
        <class name="Cataleg" identity-type="application" table="Cataleg">
            <field name="idCataleg" primary-key="true">
                <column name="idCataleg" jdbc-type="VARCHAR" allows-null="false"/>
            </field>
            <field name="titol">
                <column name="titol" jdbc-type="VARCHAR"/>
            </field>
            <field>
                <collection element-type="uoc.exemple.integracio.Producte"/>
            </field>
        </class>

        <class name="Producte" identity-type="application" table="Producte">
            <field name="nom" primary-key="true">
```

```
                <column name="nom" jdbc-type="VARCHAR" allows-null="false"/>
            </field>
            <field name="descripcio">
                <column name="descripcio" jdbc-type="VARCHAR"/>
            </field>
            <field name="preu">
                <column name="preu" jdbc-type="DECIMAL"/>
            </field>
        </class>
    </package>
</jdo>
```

Figura 16. Descriptor de persistència

El codi següent d'exemple mostra com es pot utilitzar el JDO:

```
public class MakeConnectionExample {
    public static void main(String[] args) {
        Properties properties = new Properties();

        properties.put(javax.jdo.PersistenceManagerFactoryClass, XXX);
        properties.put(javax.jdo.option.ConnectionURL, XXX);
        properties.put(javax.jdo.option.ConnectionUserName, XXX);
        properties.put(javax.jdo.option.ConnectionPassword, XXX);

        PersistenceManagerFactory pmf =
JDOHelper.getPersistenceManagerFactory(properties);
        PersistenceManager pm = pmf.getPersistenceManager();

        Transaction tx  = pm.currentTransaction();
        tx.begin();

        Producte producte = new Producte ("Ratolí", "Ratolí òptic per a PC.", 35.5);
        pm.makePersistence(producte);

        tx.commit();

        /* Do something interesting */
        pm.close();
    }
}
```

Figura 17. Exemple d'ús de JDO

5.3. Avantatges i inconvenients

Els avantatges de JDO són els següents:

- Proporciona portabilitat perquè es pot canviar la implementació concreta de l'especificació JDO sense modificar el codi de l'aplicació.
- S'elimina la frontera entre la base de dades i els objectes Java que hi ha en memòria.
- JDO evita la utilització d'altres API per a fer persistents els objectes Java en diferents tipus de bases de dades. JDO oculta els detalls d'implementació de l'emmagatzematge dels objectes Java en cadascun dels diferents tipus de bases de dades. El desenvolupador només utilitza una API (JDO API) i es pot oblidar de la resta perquè la seva aplicació podrà funcionar encara que es canviï el SGBD.
- Proporciona un bon rendiment segons la implementació.
- Es pot integrar amb els EJB d'entitat amb persistència gestionada pel *Bean* (BMP). En aquest cas, els objectes de Java adquiriran les

funcionalitats que el contenidor d'EJB proporciona als components (transaccionalitat, seguretat, gestió de connexions, etc.).
- Millora del bytecode: JDO transforma la classe persistent de tal manera que li afegeix codi per a permetre la seva sincronització amb la seva representació en la base de dades. I tot això ho fa de forma transparent al desenvolupador.
- JDO és fàcil d'utilitzar i d'aprendre.
- La programació amb JDO requereix menys línies de codi que amb altres tecnologies.

I els inconvenients són:

- En les aplicacions J2EE es perd la capacitat que ofereix JDO per actualitzar en la base de dades els canvis que es produeixen en un objecte. Un objecte es pot passar a la capa de presentació i aquesta pot modificar-lo. Però, si els components de la capa de presentació i els de la lògica de negoci són remots, aquest objecte s'ha de serialitzar per a poder-lo enviar a la capa de presentació. Si en aquest moment, la capa de presentació modifica l'objecte, fins que no l'enviï a la lògica de negoci es perd la capacitat de persistència de les actualitzacions de JDO. És a dir, l'objecte es modifica en la capa de presentació però aquests canvis no es veuen reflectits en el magatzem de dades fins que l'objecte de la capa de presentació no es passi a la capa de la lògica de negoci.
- Només admet els tipus de col·leccions estàndard (*Collection, Set* i *HashSet*).
- JDO afegeix un pas més a la compilació (per a afegir codi a la classe per tal de poder transformar-la i sincronitzar-la amb la representació de la base de dades).

6. iBATIS

En aquest apartat s'analitza el framework de persistència iBATIS, es mostra un exemple i es presenten els seus avantatges i inconvenients més importants.

6.1. Definició i característiques

iBATIS és un framework de persistència híbrid que serveix per a implementar la capa de persistència d'una aplicació i que adquireix les millors característiques i idees de les solucions més populars per a accedir a les bases de dades relacionals.[7]

En la següent figura es pot veure una representació de com iBATIS recull i combina totes aquestes solucions per a simplificar el procés de desenvolupament:

[7] Clinton Begin, Brandon Goodin, Larry Meadors, *iBATIS in Action*, pàg. 5.

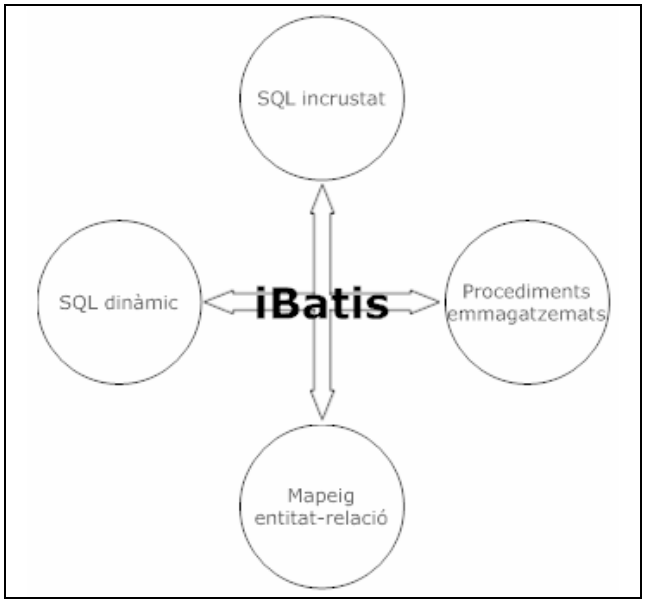

Figura 18. Solucions d'iBATIS

iBATIS necessita dos fitxers XML:

1. **SqlMapConfig.xml**. Permet configurar les opcions generals d'iBATIS i referenciar els *SqlMap.xml* individuals.
2. **SqlMap.xml**. Permet definir les consultes de mapeig que s'utilitzaran conjuntament amb les dades d'entrada que proporciona l'aplicació i que serveixen per a interactuar amb la base de dades.

En la següent figura es mostra un esquema conceptual de la configuració d'iBATIS:

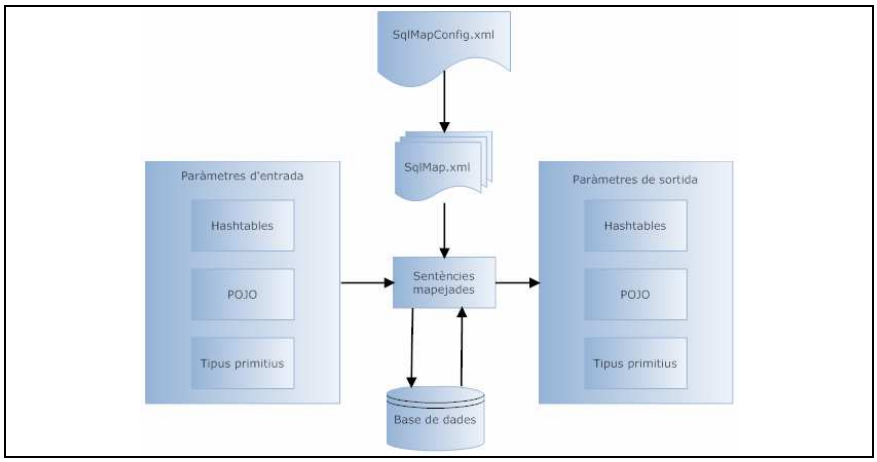

Figura 19. Esquema conceptual de la configuració d'iBATIS

6.2. Exemple

Suposem que s'utilitza el següent fitxer *properties*:

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost/test
user=root
pword=mypass
```

Figura 20. Fitxer de propietats d'iBATIS

El fitxer *SqlMapConfig.xml* per al nostre cas seria el següent:

```
<sqlMapConfig>
    <properties resource="properties" />
    <settings
        useStatementNamespaces="false"
        cacheModelsEnabled="true"
        enhancementEnabled="true"
        lazyLoadingEnabled="true"
        maxRequests="32"
        maxSessions="10"
        maxTransactions="5"
    />

    <transactionManager type="JDBC" >
        <dataSource type="SIMPLE">
            <property name="JDBC.Driver" value="${driver}"/>
            <property name="JDBC.ConnectionURL" value="${url}"/>
            <property name="JDBC.Username" value="${user}"/>
            <property name="JDBC.Password" value="${pword}"/>
        </dataSource>
    </transactionManager>

    <sqlMap resource="Cataleg.xml" />
    <sqlMap resource="Producte.xml" />
</sqlMapConfig>
```

Figura 21. Fitxer SqlMapConfig.xml d'iBATIS

I el fitxer *Cataleg.xml* seria el següent:

```
<sqlMap namespace="Cataleg">
    <cacheModel id="catalegCache" type="LRU">
        <flushInterval hours="24" />
        <flushOnExecute statement="insertCataleg" />
        <flushOnExecute statement="updateCataleg" />
        <flushOnExecute statement="deleteCataleg" />

        <property name="size" value="1000" />
    </cacheModel>

    <resultMap id="CatalegResultMap" class="uoc.exemple.integration.Cataleg">
        <constructor>
            <idArg column="idCataleg" javaType="String"/>
        </constructor>
        <result property="titol" column="titol"/>
        <collection property="productes" ofType="uoc.exemple.integration.Producte">
            <id property="nom" column="nom"/>
            <result property="descripcio" column="descripcio"/>
            <result property="preu" column="preu"/>
        </collection>
    </resultMap>

    <select id="getCataleg" parameterType="String" resultMap="catalegResultMap">
        SELECT C.idCataleg AS idCataleg, C.titol AS titol, P.nom AS nom, P.descripcio
As descripcio, P.preu As preu
        FROM Cataleg C, Producte P, CatalegProducte CP WHERE C.idCataleg = #idCataleg#
```

```
AND CP.cataleg = C.idCataleg AND CP.producte = P.nom
    </select>

    <insert id="insertCataleg" parameterClass="uoc.exemple.integrations.Cataleg">
        INSERT INTO Cataleg (idCataleg, titol)
        VALUES (#idCataleg#, #titol#)
    </insert>

    <insert                                                    id="addProducteToCataleg"
parameterClass="uoc.exemple.integration.CatalegProducte">
        INSERT INTO CatalegProducte(cataleg, producte)
        VALUES (#cataleg#, #producte#)
    </insert>
    <update id="updateCataleg" parameterClass="uoc.exemple.integration.Cataleg">
        UPDATE Cataleg
        SET titol = #titol# WHERE idCataleg = #idCataleg#
    </update>

    <delete id="deleteProductesOfCataleg" parameterType="String">
        DELETE CatalegProducte WHERE cataleg = #idCataleg#
    </delete>

    <delete id="deleteCataleg" parameterClass="uoc.exemple.integration.Cataleg">
        DELETE Cataleg WHERE idCataleg = #idCataleg#
    </delete>
</sqlMap>
```

Figura 22. Fitxer SqlMap pel cas del catàleg

I el fitxer *Producte.xml* del producte seria el següent:

```
<sqlMap namespace="Producte">
    <cacheModel id="producteCache" type="LRU">
        <flushInterval hours="24" />
        <flushOnExecute statement="insertProducte" />
        <flushOnExecute statement="updateProducte" />
        <flushOnExecute statement="deleteProducte" />
        <property name="size" value="1000" />
    </cacheModel>

    <select id="getProducte" resultClass="uoc.exemple.integration.Producte">
        SELECT nom, descripcio, preu
        FROM Producte WHERE nom = #nom#
    </select>

    <insert id="insertProducte" parameterClass="uoc.exemple.integrations.Producte">
        INSERT INTO Producte (nom, descripcio, preu)
        VALUES (#nom#, #descripcio#, #preu#)
    </insert>

    <update id="updateProducte" parameterClass="uoc.exemple.integration.Producte">
        UPDATE Cataleg
        SET descripcio = #descripcio#, preu = #preu# WHERE nom = #nom#
    </update>

    <delete id="deleteProducte" parameterClass="uoc.exemple.integration.Producte">
        DELETE Producte WHERE nom = #nom#
    </delete>
</sqlMap>
```

Figura 23. Fitxer SqlMap pel cas del producte

Finalment, en l'exemple de la figura 24 es crea un catàleg, es crea un producte i s'inclou el producte en el catàleg.

```
public class CatalegProducte(){
    private String cataleg;
    private String producte;

    public CatalegProducte(String cataleg, String producte){
        this.cataleg = cataleg;
        this.producte = producte;
    }

    public String getCataleg(){
        return (this.cataleg);
    }

    public void setCataleg(String cataleg){
        this.cataleg = cataleg;
    }

    public String getProducte(){
        return (this.producte);
    }

    public void setProducte(String producte){
        this.producte = producte;
    }
}

...

Reader reader = Resources.getResourceAsReader("sqlMapConfig.xml");
sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);

sqlMap.startTransaction();
Producte producte = new Producte ("Ratolí", "Ratolí òptic per a PC.", 35.5);
sqlMap.insert("insertProducte", producte);
Cataleg cataleg = new Cataleg("CTLG1", "Temporada d'hivern");
sqlMap.insert("insertCataleg", cataleg);
ProducteCataleg = new ProducteCataleg(cataleg.getIdCataleg(), producte.getNom());
sqlMap.insert("addProductoToCataleg", ProducteCataleg);
sqlMap.commitTransaction();
sqlMap.endTransaction ();
```

Figura 24. Exemple d'ús d'iBATIS

6.3. Avantatges i inconvenients

Alguns dels avantatges d'iBATIS són els següents:

- Tot i que cal conèixer l'SQL és fàcil d'utilitzar.
- La corba d'aprenentatge és petita ja que és molt intuïtiu.
- Proporciona flexibilitat perquè permet utilitzar consultes SQL dinàmiques.
- Es redueix la quantitat de codi Java que es necessitaria per a accedir a una base de dades relacional mitjançant JDBC.
- Utilitza la millora del *bytecode* per a millorar el codi en temps d'execució mitjançant la configuració o altres regles que es poden definir. D'aquesta manera, permet el *lazy loading* que consisteix en no carregar tot a memòria, només allò que es necessiti. Si es pensa en les comandes fetes per un client, no cal tenir-les totes en memòria, sinó només aquelles que es requereixin.
- Utilitza una *pool* de connexions a bases de dades que s'obren i es comparteixen de tal manera que tots els usuaris de l'aplicació les

poden utilitzar. Cal tenir en compte que crear noves connexions sota demanda pot consumir molt de temps.

I els inconvenients són:

- No és una eina ORM (mapeig d'entitat-relació).
- El desenvolupador ha de tenir bons coneixements del llenguatge SQL.
- Si es canvia de sistema de gestió de bases de dades caldrà reescriure les consultes SQL dels fitxers XML de configuració.

7. JPA (Java Persistence API)

En aquest apartat es fa un estudi sobre l'estàndard *Java Persistence API* (JPA), es mostra un exemple i es presenten els seus avantatges i inconvenients més importants.

7.1. Definició i característiques

JPA (*Java Persistence API*) és una especificació que forma part de l'estàndard J2EE5 (EJB 3.0) i que simplifica la persistència d'objectes Java. Proporciona un mapeig d'entitat-relació que permet definir de forma declarativa el mapeig entre els objectes Java i les taules d'una base de dades relacional.

El comportament de les classes persistents (anomenades entitats) es configura mitjançant codi simple i expressiu que s'afegeix a les classes i que rep el nom d'*anotació*. Es diu que les classes Java es decoren mitjançant anotacions.

A més, JPA també permet definir el comportament de les classes persistents mitjançant un fitxer XML.

La interfície *EntityManager* gestiona el cicle de vida de les entitats i actua de pont entre el paradigma de la programació orientada a objectes i el món relacional. És a dir, és la que permet emmagatzemar un objecte de Java en una base de dades relacional. Alguns dels mètodes que proporciona (i que el proveïdor ha d'implementar) són els següents:

```
public void persist(Object entity);
public<T> T merge(T entity);
public void remove(Object entity);
public<T> T find(Class<T> entityClass, Object primaryKey);
public void flush();
public void setFlushMode(FlushModeType flushMode);
public FlushModeType getFlushMode();
public void refresh(Object entity);
public Query createQuery(String jpqlString);
public Query createNamedQuery(String name);
```

Figura 25. Mètodes de la interfície *EntityManager* de l'API de JPA

Per a fer persistent una classe, a més d'incloure l'anotació *@Entity*, s'ha d'associar l'entitat a una *EntityManager*. La idea és que el seu temps de gestió sigui el mínim possible.

Es diu que una entitat està en *managed* o *attached* quan està associada a una *EntityManager*. Està en *detached* quan la *EntityManager* termina d'administrar-la. I està en *transient* quan no passa per una *EntityManager*.

Per tant, quan una entitat es crea amb el constructor *new*, es diu que està en *transient* o *new*. Aquesta entitat passa a ser *managed* o *attached* quan es passa com a paràmetre als mètodes *persist, merge* o *refresh*. A més, també pot estar en *managed* quan es rep com a resultat del mètode *find*. Quan una entitat perd el seu vincle amb l'*EntityManager* està en *detached* i pot passar que el seu estat no estigui sincronitzat amb el de la base de dades.

El cicle de vida d'una entitat es pot representar gràficament de la següent manera:

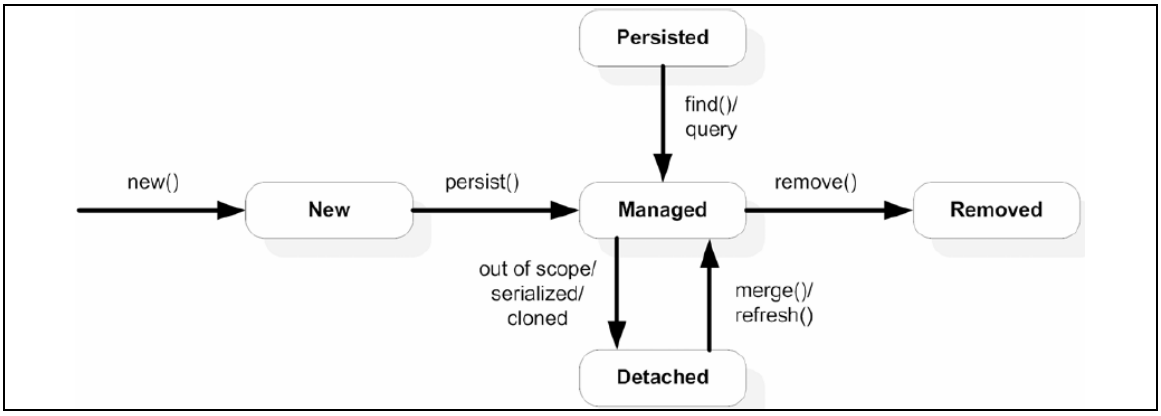

Figura 26. Cicle de vida d'una entitat en JPA

Per a finalitzar, alguns exemples d'implementació de JPA són els següents: TopLink Essentials, OpenJPA, Hibernate, etc.

7.2. Exemple

En primer lloc, cal definir el fitxer *persistence.xml* per a definir la unitat de persistència que s'utilitzarà:

```
<persistence>
    <persistence-unit name="BotigaManager" transaction-type="RESOURCE_LOCAL">
        <class>uoc.exemple.integration.Cataleg</class>
        <class>uoc.exemple.integration.Producte</class>
        <properties>
            <property                                name="javax.persistence.jdbc.driver"
value="org.apache.derby.jdbc.ClientDriver"/>
            <property                                  name="javax.persistence.jdbc.url"
value="jdbc:derby://localhost:1527/EmpServDB;create=true"/>
            <property name="javax.persistence.jdbc.user" value="XXX"/>
            <property name="javax.persistence.jdbc.password" value="XXX"/>
        </properties>
```

```
    </persistence-unit>
</persistence>
```

Figura 27. Fitxer *persistence.xml* del JPA

En segon lloc, s'afegeixen les anotacions en les classes *Cataleg* i *Producte*:

```
@Entity
@Table (name="Cataleg")
public class Cataleg{
    @Id
    private String idCataleg        = null;
    @Column (name="titol")
    private String titol            = null;
    @OneToMany
    @JoinTable(
        name="CatalegProducte",
        joinColumns = { @JoinColumn( name="cataleg") },
        inverseJoinColumns = {@JoinColumn( name="producte")}
    )
    private Collection<Producte> productes    = null;

    //...
}
```

Figura 28. Anotacions per a la classe *Cataleg*

```
@Entity
@Table (name="Producte")
public class Producte{
    @Id
    private String nom          = null;
    @Column (name="descripcio")
    private String descripcio   = null;
    @Column (name="preu")
    private Float preu          = null;

    //...
}
```

Figura 29. Anotacions per a la classe *Producte*

Finalment, per a manipular els objectes i fer-los persistents s'utilitza el següent codi:

```
EntityManagerFactory emf    = Persistence.createEntityManagerFactory("BotigaManager");
EntityManager em            = emf.createEntityManager();
Producte producte           = new Producte ("Ratolí, "Ratolí òptic per a PC.", 35.5);
Cataleg cataleg             = new Cataleg("CTLG1", "Temporada d'hivern");

em.getTransaction().begin();

em.persist(producte);
cataleg.afegirProducte(producte);
emp.persist(cataleg);
em.getTransaction().commit();

em.close();
```

Figura 30. Codi d'exemple amb JPA

7.3. Avantatges i inconvenients

Els avantatges de JPA són els següents:

- Redueix la complexitat perquè s'evita la definició d'extensos i complicats fitxers XML.
- Funciona tant en J2EE com en J2SE.
- La definició o configuració del mapeig entitat-relació es fa mitjançant anotacions, tot i que es pot utilitzar un descriptor XML.
- Permet emmagatzemar l'herència i el polimorfisme.
- Utilitza el llenguatge de consulta JPQL (*Java Persistence Query Language*) que aporta millores respecte altres llenguatges com l'EJBQL.
- La corba d'aprenentatge és petita perquè és fàcil d'utilitzar.

I els seus inconvenients són:

- El codi queda barrejat amb les anotacions i, per tant, entre el codi es reflecteix una descripció de la base de dades.
- L'estàndard JPA és part de l'especificació EJB 3.0 de Java EE 5, per tant, es necessita tenir Java EE 5 per a utilitzar-lo.

8. Hibernate

En aquest apartat es fa un estudi del framework Hibernate, es mostra un exemple i es presenten els seus avantatges i inconvenients més importants.

8.1. Definició i característiques

Hibernate és una eina de mapeig d'entitat-relació que facilita l'emmagatzematge d'objectes Java en bases de dades relacionals i permet la consulta de bases de dades per a obtenir objectes Java.

Per un costat, implementa l'estàndard JPA i, per tant, permet l'ús d'anotacions per a definir la vinculació entre una classe Java i la base de dades.

Per un altre costat, també permet definir aquest mapeig mitjançant fitxers XML de manera semblant a com es fa en el cas dels EJB d'entitat o en JDO. En aquest cas, s'han de definir els següents fitxers:

- **Hibernate.cfg.xml**. Serveix per a configurar el gestor de bases de dades i les connexions.
- **NomClasse.hbm.xml**. En aquest fitxer XML és on es defineix com es relacionen els objectes de Java amb les taules de la base de dades relacional.

També proporciona el llenguatge de consultes HQL que és molt semblant al JQL (del JPA).

L'arquitectura general d'Hibernate es pot representar gràficament de la següent manera:

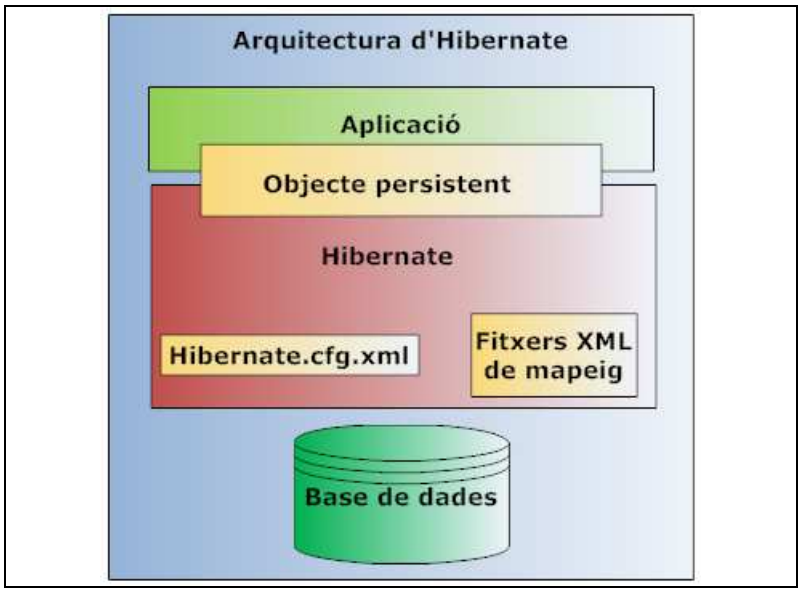

Figura 31. Arquitectura general d'Hibernate

I l'arquitectura més detallada es pot representar de la següent manera:

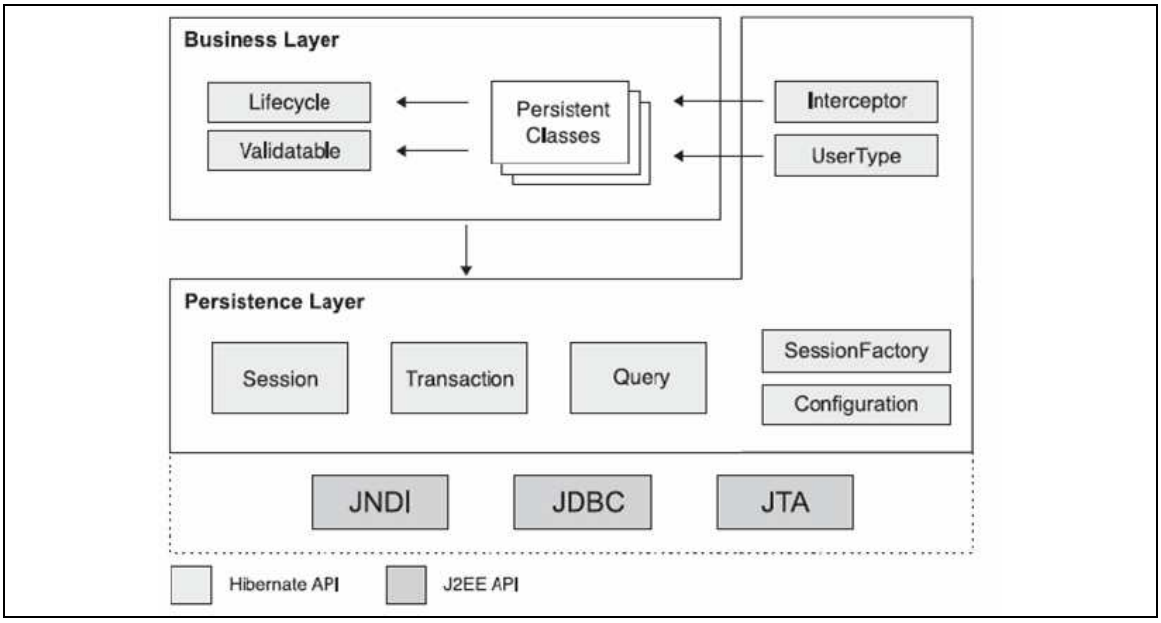

Figura 32. Arquitectura detallada d'Hibernate

Tal i com es pot observar, l'arquitectura d'Hibernate conté els següents tipus d'interfícies:

- **Session, Transaction** i **Query**. Interfícies que utilitza l'aplicació per a realitzar consultes bàsiques i dur a terme les operacions CRUD[8].
- **Configuration**. Interfícies per a configurar Hibernate.
- **Interceptor, Lifecycle** i **Validatable**. Interfícies que utilitza l'aplicació per a reaccionar davant dels esdeveniments que es produeixen dins d'Hibernate.
- **UserType, CompositeUserType** i **IdentifierGenerator**. Aquestes interfícies permeten un mapeig més avançat.

8.2. Exemple

La configuració del mapeig entre els objectes de Java i les taules de la base de dades es farà mitjançant l'ús dels fitxers XML. L'ús de les anotacions seria similar al que s'ha presentat en el cas del JPA.

En primer lloc, si suposem que s'utilitza el gestor de base de dades *MySQL*, llavors el fitxer de configuració *Hibernate.cfg.xml* seria el següent:

```
<hibernate-configuration>
    <session-factory>
        <!-- Configuraci□□ la connexi□□la base de dades -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost</property>
        <property name="connection.username">sa</property>
        <property name="connection.password">xxx</property>

        <!-- Dialecte SQL utilitzat -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="show_sql">false</property>
        <property name="hbm2ddl.auto">create</property>

        <!-- Fitxers de mapeig -->
        <mapping resource="uoc/exemple/integracio/Cataleg.hbm.xml" />
        <mapping resource="uoc/exemple/integracio/Producte.hbm.xml" />

    </session-factory>
</hibernate-configuration>
```

Figura 33. Fitxer de configuració *Hibernate.cfg.xml*

En segon lloc, la definició del mapeig entre l'objecte *Producte* de Java i la base de dades es configura mitjançant el fitxer *Producte.hbm.xml*:

```
<hibernate-mapping>
    <class name="uoc.exemple.integracio.Producte" table="Producte">
        <id name="nom" type="string" column="nom" unsaved-value="null">
            <generator class="assigned"/>
        </id>
        <property  name="descripcio"  column="descripcio"  type="string"  unique="false"
not-null="false"/>
        <property   name="preu"   column="preu"   type="float"   unique="false"   not-
null="true"/>
```

[8] CRUD són les sigles en anglès de *Create, Read, Update* i *Delete*. Són les operacions bàsiques que es poden dur a terme en una base de dades.

```
    </class>
</hibernate-mapping>
```

Figura 34. Fitxer de mapeig *Producte.hbm.xml*

En tercer lloc, la definició del mapeig entre l'objecte *Cataleg* i la base de dades es fa mitjançant el fitxer *Cataleg.hbm.xml*:

```
<hibernate-mapping>
    <class name="uoc.exemple.integracio.Cataleg" table="Cataleg">
        <id name="idCataleg" type="string" column="idCataleg" unsaved-value="null">
            <generator class="assigned"/>
        </id>
        <property   name="titol"   column="titol"   type="string"   unique="false"   not-
null="true"/>
        <set name="productes" table="CatalegProducte">
            <key column="cataleg"/>
            <many-to-many class="uoc.exemple.integracio.Producte" column="producte"/>
        </set>
    </class>
</hibernate-mapping>
```

Figura 35. Fitxer de mapeig *Cataleg.hbm.xml*

Per a finalitzar, un exemple d'ús d'Hibernate seria el següent:

```
Producte producte           = new Producte ("Ratolí, "Ratolí òptic per a PC.", 35.5);
Cataleg cataleg             = new Cataleg("CTLG1", "Temporada d'hivern");
SessionFactory session      = new Configuration().configure(new
File("Hibernate.cfg.xml")).buildSessionFactory();

cataleg.afegirProducte(producte);
session.beginTransaction();
session.save(producte);
session.save(cataleg);
session.getTransaction().commit();
```

Figura 36. Exemple d'ús d'Hibernate

8.3. Avantatges i inconvenients

Els avantatges d'Hibernate són els següents:

- Funciona en aplicacions J2EE i J2SE.
- La configuració del mapeig entre els objectes de Java i les taules de la base de dades es pot fer mitjançant les anotacions o els fitxers XML.
- Proporciona un llenguatge potent de consultes (HQL).
- Existeix molta documentació (ajudes, fòrums, etc.).

I els inconvenients són els següents:

- Disminueix el rendiment perquè es necessiten més recursos que, per exemple, amb les llibreries JDBC.
- Es necessita una gran quantitat de fitxers de configuració.

- No és recomanable per a aplicacions que gestionin una gran quantitat de dades.

9. Anàlisis de requeriments de MyFrameUOC

Tal i com ja s'ha comentat en els primers capítols, aquest projecte consisteix en desenvolupar un framework de persistència que s'anomena *MyFrameUOC*. El seu objectiu és facilitar i agilitzar la implementació de la capa de persistència de les aplicacions J2EE.

Segons l'estudi realitzat en els capítols anteriors i el temps disponible per a desenvolupar el projecte es decideix dividir els requeriments funcionals de MyFrameUOC en dos grups:

- **Requeriments bàsics**. En aquest grup s'inclouen els requeriments necessaris i imprescindibles per a obtenir un framework de persistència que permeti dur a terme les operacions bàsiques CRUD[9].
- **Requeriments avançats**. En aquest grup s'inclouen els requeriments més avançats com, per exemple, la implementació de les relacions 1 a N (un a molts) o de les relacions N a M (molts a molts) o l'ús d'un llenguatge de consultes propi, etc.

L'objectiu d'aquesta descomposició de requeriments és la possibilitat de poder desenvolupar el projecte per passos incrementals i assegurar-ne que MyFrameUOC compleixi amb uns requisits mínims.

Per tant, la finalitat principal d'aquest projecte és implementar els requeriments bàsics i algun dels requeriments avançats. Malgrat tot, s'exposaran tots i a mesura que s'avanci en el desenvolupament del projecte es prendrà la decisió d'implementar-ne uns o altres.

9.1. Requeriments bàsics

MyFrameUOC ha de ser un framework de persistència que faciliti i agilitzi la implementació de la capa de persistència de les aplicacions J2EE. L'objectiu principal és incrementar la productivitat i el rendiment del desenvolupador.

Els requeriments bàsics són aquells que MyFrameUOC ha de proporcionar obligatòriament i que serveixen per a poder dur a terme les operacions bàsiques CRUD.

La idea és que el desenvolupador només s'hagi de preocupar d'implementar les seves classes i configurar-les perquè MyFrameUOC s'encarregui del seu emmagatzematge en una base de dades. Així, la perspectiva amb què treballarà serà la de la programació orientada a objectes i s'abstraurà dels detalls d'implementació de la visió relacional d'una base de dades.

[9] CRUD són les sigles en anglès de *Create, Read, Update* i *Delete*. Són les operacions bàsiques que es poden dur a terme en una base de dades.

En primer lloc, MyFrameUOC estarà preparat per a treballar amb la majoria de sistemes de gestió de bases de dades relacionals. Per aconseguir-ho s'utilitzarà les llibreries JDBC i un fitxer de configuració en format XML que s'anomenarà *MyFrameUOC.xml* i que tindrà la següent estructura:

```
<MyFrameUOC>
    <properties>
        <driver>org.apache.derby.jdbc.ClientDriver</driver>
        <url>jdbc:derby://localhost:1527/EmpServDB</url>
        <user>sa</user>
        <password>xxx</password>
    <properties>
</MyFrameUOC>
```

Figura 37. Estructura del fitxer *MyFrameUOC.xml*

En segon lloc, la definició del mapeig entre les classes Java i els objectes de la base de dades es farà mitjançant anotacions. Aquesta decisió es basa en el fet que es considera que és més fàcil i intuïtiu que l'ús de fitxers XML.

Una classe persistent l'anomenarem *entitat* i per a definir-la s'utilitzaran les següents etiquetes dins del seu codi:

- **@Entity**. Serveix per a indicar que una classe es vol fer persistent. Tindrà les següents propietats:
    - **Name**. Nom de la taula amb la qual s'associa la classe.
- **@PrimaryKey**. Permet indicar que un atribut de la classe forma part de la clau primària.
- **@Column**. Indica que un atribut es vincula amb un camp de la taula. Tindrà les següents propietats:
    - **Field**. Nom del camp de la taula amb el qual està associat l'atribut de l'entitat.
    - **Unique**. Indica que no es pot repetir.
    - **Required**. Indica que és obligatori.
- **@ForeignKey**. Indica que l'atribut forma part d'una clau forana. Per tant, aquesta etiqueta permet definir relacions un a un (1 a 1). Tindrà les següents propietats:
    - **Field**. Nom del camp de la taula amb el qual està associat l'atribut de l'entitat.
    - **Name**. Nom de la clau forana de la qual forma part l'atribut.

Aquestes són les etiquetes i les propietats que es considera que són necessàries per a implementar les operacions bàsiques CRUD. Però, en les fases de disseny i d'implementació es podran adaptar segons les necessitats.

Per acabar, MyFrameUOC ha de permetre dur a terme les operacions que es detallen a continuació:

- **Afegir una entitat.** Permet emmagatzemar una instància d'una entitat en la base de dades.

- **Consultar una entitat**. Permet obtenir una instància d'una entitat que està emmagatzemada en la base de dades.
- **Modificar una entitat**. Permet emmagatzemar en la base de dades els canvis que s'hagin fet sobre una instància d'una entitat.
- **Esborrar una entitat**. Permet eliminar una instància d'una entitat de la base de dades.

9.2. Requeriments avançats

A més dels requeriments bàsics que s'acaben de descriure, també es proposa implementar una sèrie de requeriments que es consideren que són més avançats. Tot i que en l'abast d'aquest projecte només s'inclou la implementació dels requeriments bàsics i algun dels requeriments més avançats, en aquest apartat es presenten tots. Segons el temps disponible en les fases més avançades del projecte es prendrà la decisió sobre quins d'ells es podran implementar.

Per un costat, MyFrameUOC ha de permetre implementar les relacions d'associació entre entitats: claus foranes, relacions d'un a un, relacions d'un a molts i relacions de molts a molts.

Per un altre costat, MyFrameUOC ha de proporcionar un llenguatge de consultes propi que s'anomenarà *UOCQL* i que permetrà construir consultes per a obtenir instàncies d'una entitat. Una consulta UOCQL ha de tenir la sintaxis següent:

nomEntitat WHERE restricció [{AND|OR restricció}]

On *nomEntitat* és el nom de l'entitat sobre la qual es farà la cerca i *restricció* és una condició que s'ha de complir per a obtenir les instàncies de l'entitat.

Una condició té la següent sintaxis:

atribut operador valor

On *atribut* és el nom d'un dels atributs de l'entitat *nomEntitat*, *operador* és un operador de comparació (=, <, >, <>) i *valor* és un valor constant.

Per exemple, la següent consulta UOCQL serveix per a obtenir tots els productes que tenen un preu més gran que 1.5 i més petit que 2:

Producte WHERE preu > 1.5 AND preu < 2

Aquesta consulta retornarà un conjunt d'instàncies que seran les que s'obtindran del magatzem de dades segons les restriccions indicades.
La finalitat del llenguatge UOCQL és permetre i facilitar al desenvolupador la recuperació d'un conjunt d'objectes d'una entitat que estan emmagatzemades en la base de dades.

9.3. Diagrama de casos d'ús

Segons els requeriments que s'han descrit, el diagrama de casos d'ús que s'obté del sistema MyFrameUOC és el següent:

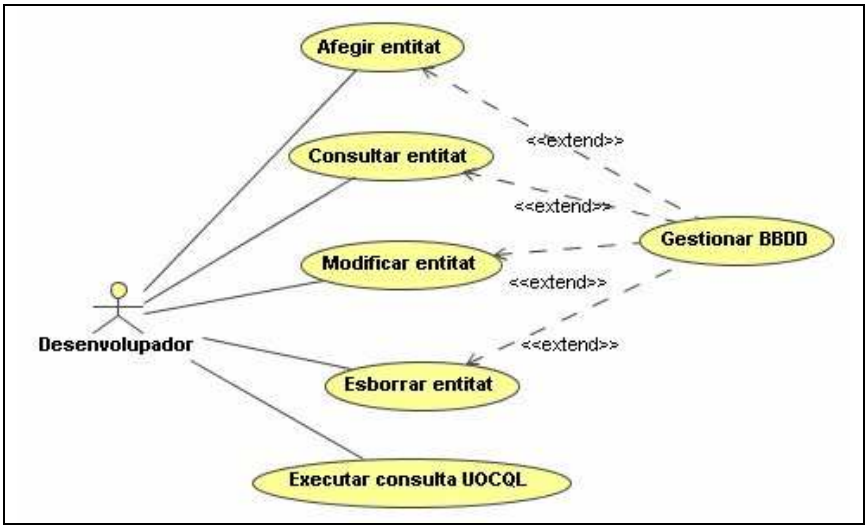

Figura 38. Diagrama de casos d'ús de MyFrameUOC

La descripció detallada de cada cas d'ús és la següent:

- **Gestionar BBDD**. Inclou les operacions de crear i eliminar una taula d'una base de dades. Aquestes funcionalitats s'utilitzaran en la resta de casos d'ús quan sigui necessari. Per exemple, si el desenvolupador vol afegir una entitat i encara no s'ha creat la seva taula, llavors es portarà a terme el procés de crear-la. En un principi, es considera innecessari que el desenvolupador pugui gestionar la base de dades directament perquè la idea és que tingui una visió de la programació orientada a objectes i es pugui despreocupar dels detalls físics de la base de dades.
- **Afegir entitat**. Permet emmagatzemar en la base de dades una nova instància d'una entitat.
- **Consultar entitat**. Permet obtenir de la base de dades una instància d'una entitat a partir de la seva clau primària.
- **Modificar entitat**. Permet actualitzar en la base de dades els canvis que s'hagin fet sobre una instància d'una entitat. És a dir, permet sincronitzar la instància d'una entitat amb les dades del magatzem de dades.
- **Esborrar entitat**. Permet esborrar de la base de dades una instància d'una entitat.
- **Executar consulta UOCQL**. Permet executar una consulta UOCQL i obtenir com a resultat un conjunt d'instàncies d'una entitat. Aquesta funcionalitat està pensada per a fer cerques d'objectes emmagatzemats en la base de dades mitjançant diferents criteris i restriccions.

10. Consideracions i refinament

La investigació i l'estudi dels frameworks de persistència que hi ha disponibles al mercat ens ha permès realitzar una primera anàlisi de requeriments del sistema MyFrameUOC. Malgrat tot, no n'hi ha prou amb això i cal entrar més en detall per a obtenir uns requeriments més precisos i poder passar a la fase de disseny.

10.1. Primeres decisions

Per un costat, MyFrameUOC s'implementarà perquè es pugui utilitzar en Oracle i en MySQL. Però, es deixarà preparat perquè la implementació per a un altre SGBD sigui fàcil, ràpida i simple.

Per un altre costat, es faran proves mitjançant els tests de JUnit per a validar i verificar el correcte funcionament de les funcionalitats que s'implementin. L'objectiu és implementar-los al mateix temps que les funcionalitats del sistema, provar el nou desenvolupament que es faci i comprovar que el desenvolupament antic funciona correctament. D'aquesta manera s'evitarà el màxim possible la introducció de nous errors.

10.2. Fitxer de configuració

El fitxer de configuració s'ha d'ubicar en el projecte Java de l'usuari i ha de tenir la següent especificació:

```
<MyFrameUOC>
    <properties>
        <SGBD>[SGBD]</SGBD>
        <url>[url]</url>
        <user>[user]</user>
        <password>[passsword]</password>
    </properties>
</MyFrameUOC>
```

On:

- **[SGBD]** tipus de sistema de gestió de bases de dades que s'utilitza. En una primera versió de MyFrameUOC es permeten dos valors:
    - **Oracle**. Per a utilitzar l'Oracle.
    - **MySQL**. Per a utilitzar MySQL.
- **[url]** és la direcció url de la màquina on es troba la base de dades.
- **[user]** és el nom d'usuari que s'utilitza per a connectar-se a la base de dades.
- **[password]** és la contrasenya que s'utilitza per a connectar-se a la base de dades.

10.3. Anotacions

Es decideix que les anotacions que MyFrameUOC ha d'oferir per a poder configurar i definir les entitats són les següents:

Nom	A quin element s'aplica?	Descripció	Paràmetres	
			Nom	Descripció
@Entity	Classe	Serveix per a definir una entitat i configurar una classe perquè sigui persistent.	Table	Nom de la taula en la qual s'emmagatzemaran els atributs de l'entitat.
@PrimaryKey	Atribut	Indica que l'atribut forma part de la clau primària de la taula de l'entitat.	Name	Nom que té la clau primària en la taula.
			Field	Nom del camp de la taula amb el qual es vincula l'atribut.
			Precision	Precisió o número de dígits de d'un número (comptant la part entera i els decimals). Aquesta propietat només té sentit amb tipus de dades numèrics.
			Scale	Escala o número de dígits de la part decimal d'un número. Aquesta propietat només té sentit amb tipus de dades numèrics (no sencers).
			Length	Mida en caràcters o bytes que té el camp en la taula. Aquesta propietat s'utilitza pel cas del tipus de dades java.lang.String o char[].
@ForeignKey	Atribut	Indica que l'atribut és clau forana cap a una entitat destí. El tipus de dades de l'atribut ha de ser una altra entitat.	Name	Nom que té la clau forana en la taula origen (la de l'entitat de l'atribut on es posa l'anotació).
			Fields	Correspondència entre els camps de la taula origen i els atributs de les claus primàries de l'entitat destí. El seu format ha de ser el següent: campOrigen1-clauprimàriaDestí1,campOrigen2-clauprimàriaDestí2,..., campOrigenN-clauprimàriaDestíN.
@Column	Atribut	Indica que l'atribut es vincula amb un camp de la taula de l'entitat.	Field	Nom del camp de la taula amb el qual es vincula l'atribut.
			Required	Pot prendre els següents valors: - **True**. Indica que el valor de l'atribut és obligatori i, per tant, no pot ser nul. - **False**. Indica que el valor de l'atribut no és obligatori i, per tant, pot ser nul. Per defecte pren el valor False.

Nom	A quin element s'aplica?	Descripció	Paràmetres	
			Nom	**Descripció**
			Unique	Pot prendre els següents valors: - **True**. Indica que el valor de l'atribut no es pot repetir. - **False**. Indica que el valor de l'atribut es pot repetir en la taula. Per defecte pren el valor False.
			Precision	Precisió o número de dígits de d'un número (comptant la part entera i els decimals). Aquesta propietat només té sentit amb tipus de dades numèrics.
			Scale	Escala o número de dígits de la part decimal d'un número. Aquesta propietat només té sentit amb tipus de dades numèrics (no sencers).
			Length	Mida en caràcters o bytes que té el camp en la taula. Aquesta propietat s'utilitza pel cas del tipus de dades java.lang.String i char[].
@OneToONe	Atribut	Indica que l'atribut implementa una relació d'un a un amb una altra entitat.	Table	Nom de la taula que s'utilitza per a implementar la relació d'un a un entre les dues entitats que hi participen.
			NameFK_To	Nom de la clau forana cap a la taula destí.
			NameFK_From	Nom de la clau forana cap a la taula origen (on es troba l'atribut).
			Fields_To	Correspondència entre els camps de la taula origen i els atributs de les claus primàries de l'entitat destí.
			Fields_From	Correspondència entre els camps de la taula destí i els atributs de les claus primàries de l'entitat origen.
@OneToMany	Atribut	Indica que l'atribut implementa una relació d'un a molts amb una altra entitat.	Table	Nom de la taula que s'utilitza per a implementar la relació d'un a molts entre les dues entitats que hi participen.
			NameFK_To	Nom de la clau forana cap a la taula destí.
			NameFK_From	Nom de la clau forana cap a la taula origen.
			Fields_To	Correspondència entre els camps de la taula origen i els atributs de les claus primàries de l'entitat destí.
			Fields_From	Correspondència entre els camps de la taula destí i els atributs de les claus primàries de l'entitat origen.

Nom	A quin element s'aplica?	Descripció	Paràmetres	
			Nom	Descripció
@ManyToMany	Atribut	Indica que l'atribut implementa una relació de molts a molts amb una altra entitat.	Table	Nom de la taula que s'utilitza per a implementar la relació de molts a molts entre les dues entitats que hi participen.
			NameFK_to	Nom de la clau forana cap a la taula destí.
			NameFK_From	Nom de la clau forana cap a la taula origen.
			Fields_To	Correspondència entre els camps de la taula origen i les claus primàries de l'entitat destí.
			Fields_From	Correspondència entre els camps de la taula destí i els atributs de les claus primàries de l'entitat origen.

10.4. Funcionament de MyFrameUOC

MyFrameUOC ha de proporcionar una sessió que permeti dur a terme totes les funcionalitats possibles. És a dir, ha de proporcionar una classe que s'anomenarà *Session* i que ha d'oferir els següents mètodes:

- ***beginTransaction()***. Permet iniciar una transacció.
- ***commit()***. Permet confirmar una transacció.
- ***rollback()***. Permet cancel·lar una transacció.
- ***insert(Object obj)***. Permet inserir l'objecte *obj* en el magatzem de dades.
- ***read(String entity, ArrayList<Object> primaryKeys)***. Permet obtenir un objecte de la entitat *entity* del magatzem de dades a partir d'una llista amb els valors de la clau primària. Observar que la clau primària pot estar formada per més d'un atribut i, per tant, es necessita una *ArrayList<Object>* per incloure tots els valors. Els valors s'han d'incloure en el mateix ordre en el que es defineixen els atributs de la clau primària dins de la classe o entitat.
- ***update(Object obj)***. Actualitza l'objecte *obj* en la base de dades. És a dir, sincronitza l'objecte *obj* que hi hagi en la base de dades amb el que el que hi hagi a memòria.
- ***delete(Object obj)***. Permet esborrar l'objecte *obj* de la base de dades.
- ***executeUOCQL(String query)***. Permet executar una consulta UOCQL.
- ***drop(String entity)***. Permet esborrar la taula d'una entitat.
- ***close()***. Permet tancar una sessió i la seva connexió amb la base de dades.
- ***closeNoEx()***. Igual que *close()* però sense llençar una excepció.

Aquesta classe també ha de proporcionar un mètode que s'anomenarà *getSession(String file)* i que permetrà obtenir una sessió segons el SGBD[10] que s'indiqui en el fitxer de configuració *file*.

Una sessió obrirà una connexió amb la base de dades i emmagatzemarà en memòria el mapeig que cal fer per a emmagatzemar un objecte d'una entitat en la base de dades. La gestió de les entitats que l'usuari defineixi la portarà a terme un gestor d'entitats (*EntityManager*).

El mapeig es farà en funció de les anotacions que l'usuari ha d'incloure en el codi de les seves classes i del SGBD que s'utilitzi. L'objectiu és emmagatzemar en memòria unes sentències SQL preparades però no definitives i que serviran per a obtenir les sentències SQL natives per a emmagatzemar, llegir, actualitzar i eliminar un objecte d'una entitat. Quan l'usuari executa una de les operacions CRUD, llavors MyFrameUOC utilitzarà aquestes consultes SQL preparades i obtindrà les sentències SQL definitives que s'acabaran enviant cap al SGBD mitjançant les llibreries JDBC[11].

Els tipus de dades dels camps de les taules es determinaran segons el SGBD i s'utilitzaran per a crear les taules quan no existeixin en la base de dades. Els tipus de dades en Java que s'han de poder utilitzar són els següents:

- **String**
- **char**
- **Character**
- **char[]**. En aquest cas és obligatori informar la mida que ha de tenir el camp en la base de dades mitjançant l'anotació.
- **int**
- **Integer**
- **float**
- **Float**
- **double**
- **Double**
- **BigDecimal**. És obligatori indicar una precisió i una escala mitjançant l'anotació.
- **java.sql.Date**
- **byte**
- **Byte**
- **short**
- **Short**
- **long**
- **Long**
- **boolean**
- **Boolean**
- **java.slq.Time**

[10] SGBD són les sigles de Sistema de gestió de bases de dades.
[11] JDBC són les sigles en anglès de *Java data base connectivity*. Són unes llibreries en Java que serveixen per accedir a bases de dades i a diferents tipus de fonts de dades (fulls de càlcul, fitxers de text, etc.).

- **BigInteger**. Aquest tipus de dades s'emmagatzema en la base de dades com una cadena de caràcters.

No es recomanable utilitzar alguns tipus de dades en les claus primàries, com per exemple els *float* o els *BigInteger,* perquè pot donar problemes segons el sistema de gestió de bases de dades.

És necessari que l'usuari implementi els *getters* i *setters* de totes les propietats persistents de la seva classe. El nom del *getter* ha d'estar format per "get" més el nom de l'atribut amb la primera lletra en majúscula. Per exemple, si un atribut té el nom "edat", llavors el seu *getter* ha de ser "getEdat()". I el nom del *setter* ha d'estar format per "set" més el nom de l'atribut amb la primera lletra en majúscula.

A més, també es obligatori que l'usuari implementi el constructor sense paràmetres de la seva classe persistent, encara que estigui buit.

Per a finalitzar, només es permet obrir una transacció per sessió. Un cop iniciada una transacció, no es podrà tornar a iniciar cap altra fins que la iniciada es confirmi o es cancel·li.

10.5. Claus foranes

MyFrameUOC ha de permetre l'ús de claus foranes mitjançant l'anotació *@ForeignKey*. Una clau forana és un atribut d'una entitat que es relaciona amb una altra entitat. Un exemple de la seva implementació en la base de dades seria el següent:

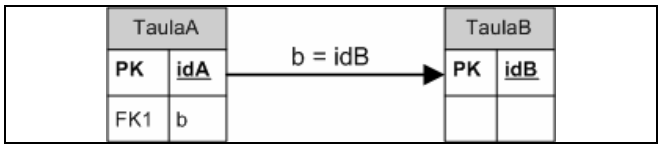

Figura 39. Clau forana

La implementació d'aquest exemple en Java i mitjançant MyFrameUOC seria el següent:

```
@Entity(Table = "TaulaA")
public class A{
    @PrimaryKey(Name = "PK_TaulaA", Field = "idA")
    private String identificadorA;

    @ForeignKey(Name = "FK_AB", Fields = "b-identificadorB")
    private B b;
}

@Entity(Table="TaulaB")
public class B{
    @PrimaryKey(Name = "PK_TaulaB", Field = "idB")
    private String identificadorB;
}
```

Figura 40. Clau forana amb *@ForeignKey*

Observar com en l'anotació *@ForeignKey* s'indica que la clau forana en la *TaulaA* s'implementa mitjançant el camp *b* que fa referència a la clau primària *identificadorB* de l'entitat *B*. És a dir, en la propietat *Fields* s'indiquen els camps de la taula origen (*TaulaA*) que faran referència a la taula destí (*TaulaB*) i mitjançant un guió (-) s'indica cap a quina clau primària de l'entitat destí fan referència. En l'exemple de la figura 40 s'indica *Fields="b-identificadorB"*, on *b* és el camp de la *TaulaA* i *identificadorB* és l'atribut de la clau primària de l'entitat a la qual es fa referència. És responsabilitat de l'usuari indicar la correspondència entre els camps origen i els atributs de les claus primàries de l'entitat destí.

Els camps d'una taula que formen part d'una clau forana no poden ser obligatoris, és a dir, poden tenir valors nuls. Es decideix imposar aquesta restricció perquè en cas contrari s'hauria de desactivar les restriccions de claus foranes per a poder inserir registres en les taules i això podria provocar que altres usuaris modifiquessin la taula i la deixessin inconsistent. Per a simplificar-ho, s'opta per l'ús de claus foranes que poden contenir valors nuls.

10.6. Relacions d'un a un (1 a 1)

MyFrameUOC ha de permetre les relacions d'un a un (1 a 1) entre dues entitats i es considera que aquest tipus de relació es pot implementar de dues formes diferents:

a) **Sense una taula per a implementar la relació**. La relació un a un s'implementa mitjançant una clau forana en cadascuna de les dues taules.
b) **Amb una taula per a implementar la relació**. S'utilitza una taula per a implementar la relació un a un entre dues taules.

Per un costat, un exemple de la implementació d'una relació un a un sense una taula per a la relació seria el següent:

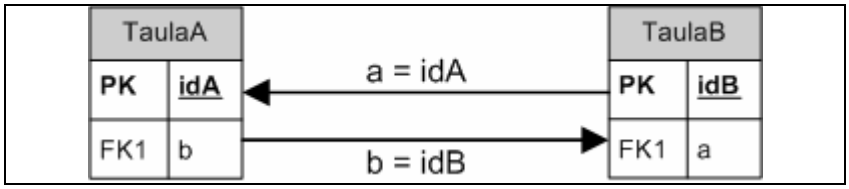

Figura 41. Relació d'un a un sense taula

Les relacions 1 a 1 sense taula s'han de configurar mitjançant una anotació *@ForeignKey* en cadascuna de les dues classes. En la figura 42 es mostra un exemple.

```
@Entity(Table="TaulaA")
public class A{
    @PrimaryKey(Name = "PK_TaulaA", Field = "idA")
    private String identificadorA;

    @ForeignKey(Name = "FK_AB", Fields = "b-identificadorB")
    private B b;
}

@Entity(Table="TaulaB")
public class B{
    @PrimaryKey(Name = "PK_TaulaB", Field = "idB")
    private String identificadorB;

    @ForeignKey(Name = "FK_BA", Fields = "a-identificadorA")
    private A a;
}
```

Figura 42. Relació d'un a un amb *@ForeignKey*

Per un altre costat, no es pot crear una relació 1 a 1 sense taula amb l'anotació *@OneToOne*. Aquesta anotació només serveix per a configurar les relacions 1 a 1 que utilitzen una taula per a implementar la relació.

Un exemple de la implementació de la relació un a un amb taula per a la relació seria el següent:

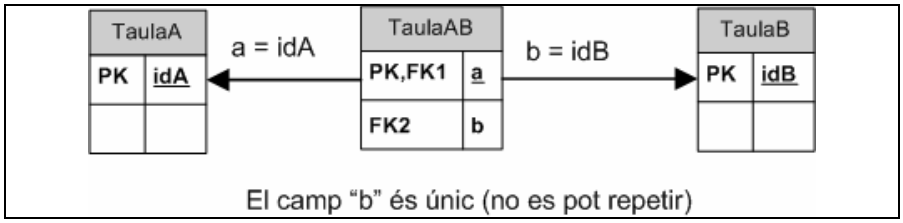

Figura 43. Relació d'un a un amb taula

La configuració en MyFrameUOC es fa mitjançant l'anotació *@OneToOne* tal i com es mostra en la figura següent:

```
public class A{
    @PrimaryKey(Name = "PK_TaulaA", Field = "idA")
    private String identificadorA;

    @OneToOne(Table = "TaulaAB", NameFK_To = "FK_AB", NameFK_From = "FK_BA", Fields_To
= "b-identificadorB", Fields_From = "a-dentificadorA")
    private B objecteB;
}

@Entity(Table="TaulaB")
public class B{
    @PrimaryKey(Name = "PK_TaulaB", Field = "idB")
    private String identificadorB;
}
```

Figura 44. Relació d'un a un amb *@OneToOne*.

La propietat *Fields_To* de l'anotació *@OneToOne* serveix per a configurar els camps que apunten cap a l'entitat destí. Es poden incloure diverses correspondències, separades per una coma(,), on cada correspondència ha

d'estar formada per dos valors, separats per guió (-). El primer valor és el nom del camp que tindrà el camp en la taula que s'utilitza per a implementar la relació. El segon valor és el nom de l'atribut de la clau primària de l'entitat destí amb el qual es relaciona l'atribut de l'anotació de l'entitat origen.

La propietat *Fields_From* serveix per a configurar els camps de l'entitat destí que apunten cap a l'entitat origen.

10.7. Relacions d'un a molts (1 a N)

MyFrameUOC ha de permetre les relacions d'un a molts (1 a N). Aquest tipus de relacions s'implementaran mitjançant una taula

Un exemple d'implementació d'una relació d'un a molts amb una taula per a la relació és el següent:

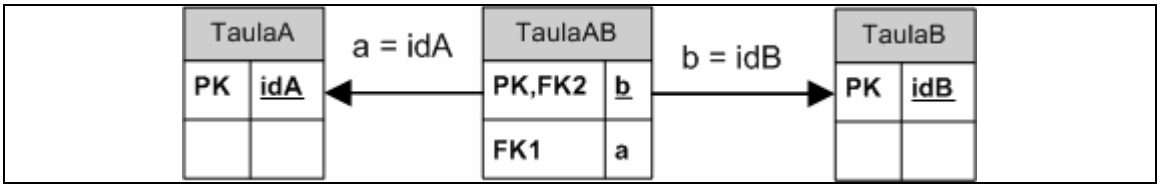

Figura 45. Relació d'un a molts amb taula

La configuració d'aquest exemple en MyFrameUOC seria la que es mostra a continuació:

```
@Entity(Table="TaulaA")
public class A{
    @PrimaryKey(Name = "PK_TaulaA", Field = "idA")
    private String identificadorA;

    @OneToMany(Table="TaulaAB", NameFK_To = "FK_AB", NameFK_From = "FK_BA", Fields_To =
"b-identificadorB", Fields_From = "a-identificadorA")
    private ArrayList<B> b;
}

@Entity(Table="TaulaB")
public class B{
    @PrimaryKey(Name = "PK_TaulaB", Field = "idB")
    private String identificadorB;
}
```

Figura 46. Relació d'un a molts amb *@OneToMany*

10.8. Relacions de molts a molts (M a N)

MyFrameUOC ha de permetre les relacions de molts a molts i es considera que aquest tipus de relacions s'implementen de la manera següent:

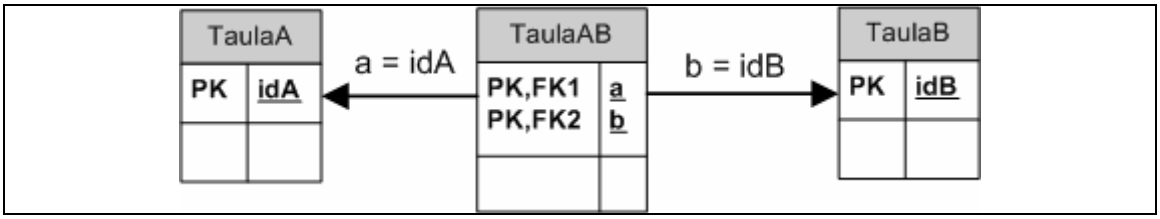

Figura 47. Relació de molts a molts

Aquesta relació es pot configurar amb l'anotació *@ManyToMany* tal i com es mostra en la figura 48.

```
@Entity(Table = "TaulaA")
public class A{
    @PrimaryKey(Name = "PK_TaulaA", Field = "idA")
    private String identificadorA;

    @ManyToMany(Table  =  "TaulaAB",  NameFK_To  =  "FK_AB",  NameFK_From  =  "FK_BA",
Fields_To = "b-identificadroB", Fields_From = "a-identificadorA")
    private ArrayList<B> llistaB;
}

@Entity(Table="TaulaB")
public class B{
    @PrimaryKey(Name = "PK_TaulaB", Field = "idB")
    private String identificadorB;

    @ManyToMany(Table  =  "TaulaAB",  NameFK_To  =  "FK_BA",  NameFK_From  =  "FK_AB",
Fields_to = "a-identificadorA", Fields_From = "b-identificadorB")
    private List<A> llistaA;
}
```

Figura 48. Relació de molts a molts amb *@ManyToMany*

Observar com l'atribut de la relació és de tipus *Collection* on cada element és un objecte de l'entitat destí amb el qual està relacionat un objecte de l'entitat origen.

10.9. Gestió de la base de dades

MyFrameUOC ha de permetre que les taules no estiguin creades en el moment en que l'usuari vulgui emmagatzemar, llegir, actualitzar o esborrar un objecte d'una entitat. Per tant, s'ha d'encarregar de verificar si les taules existeixen i de crear-les en cas de ser necessari. Aquesta verificació forma part del cas d'ús anomenat "Gestió BBDD" que s'ha mostrat en el diagrama de casos d'ús.

També ha de permetre que l'usuari pugui esborrar la taula que està associada a una entitat. Aquesta funcionalitat s'ha d'encarregar d'esborrar

la taula de l'entitat i, a més, totes les taules que estiguin relacionades amb la taula de l'entitat que es vol esborrar.

10.10. Llenguatge UOCQL

MyFrameUOC ha de proporcionar un llenguatge de consultes propi que serveixi per a fer cerques d'objectes d'entitats en la base de dades. L'objectiu és que l'usuari pugui definir les seves pròpies cerques a més de la cerca per clau primària que s'ofereix per defecte mitjançant el mètode *read(Object, ArrayList<Object>)* de la classe *Session*.

UOCQL ha de ser un llenguatge de consultes que es basi en l'SQL però que s'apropi al paradigma d'orientació a objectes. Ha de ser senzill i ha de permetre obtenir un conjunt d'objectes d'una entitat de la base de dades. La seva sintaxis es va especificar en l'anàlisi de requeriments, però és necessari entrar més en detall.

En una consulta UOCQL només hi podrà participar una entitat i el seu resultat serà un conjunt d'objectes d'aquesta entitat (o nul si no es troba cap instància en la base de dades).

En el següent exemple es mostra una consulta UOCQL que obté tots els objectes de l'entitat *Empleat* que tinguin l'edat compresa entre 25 i 50 anys:

```
FROM Empleat WHERE edat > 25 AND edat < 50
```

Figura 49. Exemple d'una consulta UOCQL

Una consulta UOCQL estarà formada per un o diverses restriccions separades pels operadors lògics *AND* o *OR*. Cada restricció estarà formada per un operand dret, un operador de comparació i un operand esquerre.

L'operand esquerre ha de ser un atribut de l'entitat que apareix en la clàusula *FROM*. L'operand dret ha de ser un valor constant. Els operadors relacionals que es poden utilitzar són els següents: >, <, =, <>, >= i <=.

11. Disseny

En aquest capítol es fa una descripció del sistema MyFrameUOC mitjançant el llenguatge UML[12]. En primer lloc, es presenta l'esquema invariant o model conceptual amb la informació que gestiona el sistema. En segon lloc, es mostra el diagrama de classes del framework. I en tercer lloc, s'exposen els diagrames de seqüències de les operacions més importants.

[12] UML són les sigles en anglès d'*Unified Modeling Language* (llenguatge unificat de modelatge) i és un llenguatge que serveix per descriure i modelar un sistema.

11.1. Esquema invariant

Segons l'anàlisi de requeriments i les consideracions anteriors, es pot descriure el sistema des del punt de vista de la informació. I per a fer-ho es considera convenient elaborar un esquema invariant o model conceptual que mostri la informació que gestionarà MyFrameUOC. El seu contingut es mostra en la següent figura:

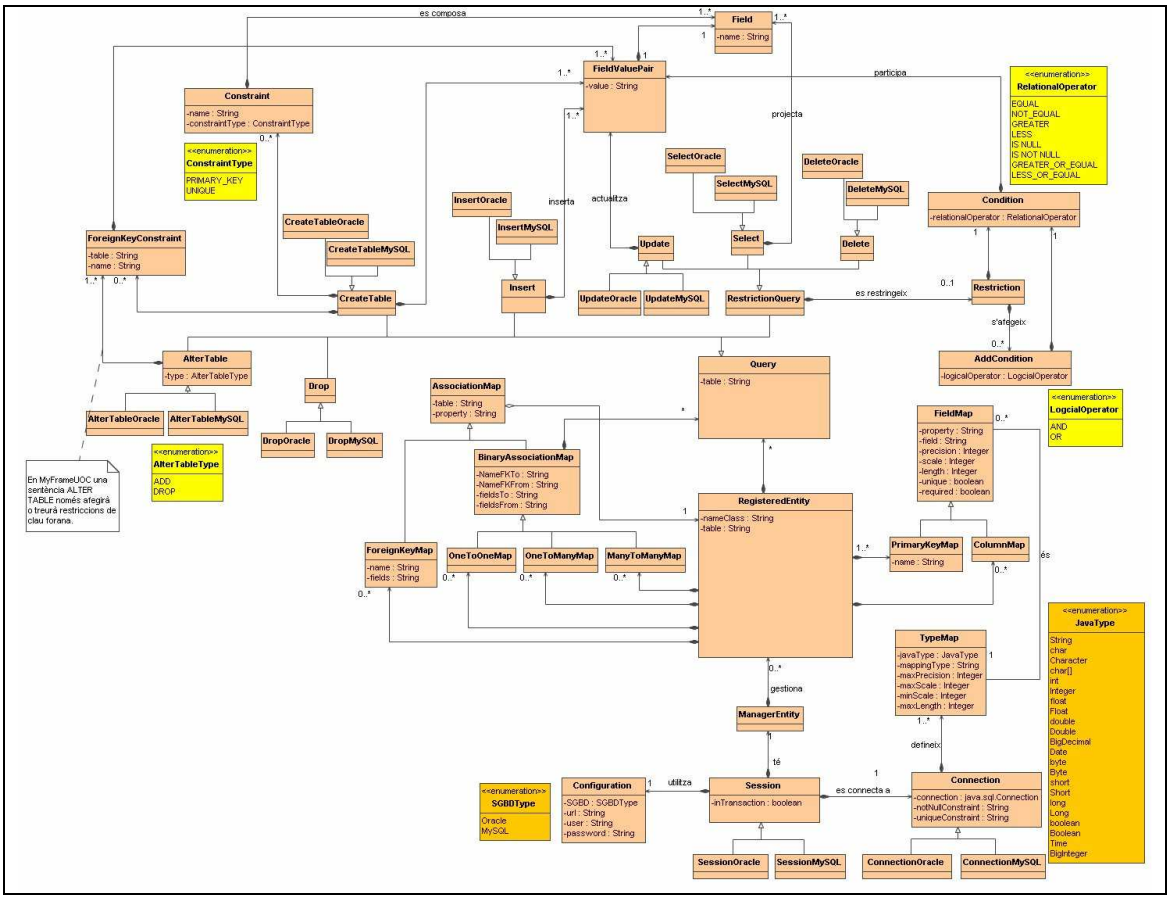

Figura 50. Esquema invariant de MyFrameUOC

11.2. Diagrama de classes

El diagrama de classes que s'obté del sistema MyFrameUOC és el següent:

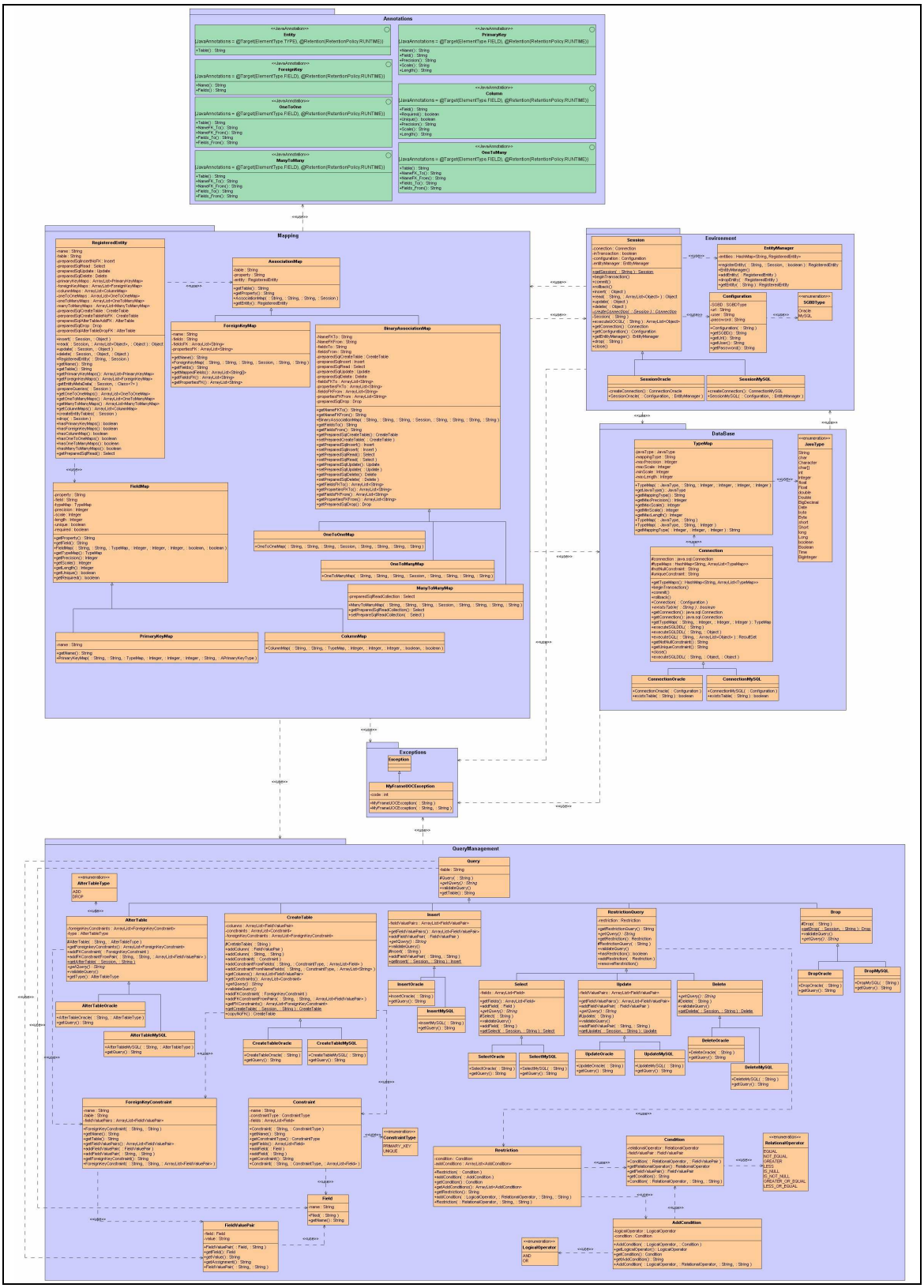

Figura 51. Diagrama de classes de MyFrameUOC

Tal i com es pot veure en l'anterior figura, les classes s'ubiquen en cinc paquets segons les seves funcionalitats principals:

- **Annotations**. S'ubiquen les interfícies que serveixen per a definir les anotacions que es poden utilitzar per a configurar les entitats de MyFrameUOC.
- **Environment**. S'ubiquen les classes que gestionen l'entorn de MyFrameUOC. En aquest paquet s'ubica la classe *Session* que és la que l'usuari haurà d'utilitzar per a interactuar amb el sistema.
- **DataBase**. S'ubiquen les classes que s'encarreguen de la part de la base de dades (connexió, execució de consultes sobre la base de dades, etc.).
- **Mapping**. S'ubiquen les classes que s'encarreguen del mapeig entre les entitats i els objectes relacionals de la base de dades.
- **QueryManagement**. S'ubiquen les classes que representen i gestionen les consultes o sentències SQL que s'utilitzen en el sistema.
- **Exceptions**. S'ubiquen les classes que s'encarreguen de la gestió d'excepcions.

Les decisions de disseny que s'han pres en aquesta activitat són les següents:

- Ús del patró mètode factoria per a implementar la creació de connexions i de sentències SQL segons el SGBD que es configuri en el fitxer de configuració.
- En cadascun dels paquets s'aplica el patró façana de tal manera que l'accés a cada nivell o paquet es fa mitjançant una única classe.

11.2.1. Capa *Annotations*

En aquesta capa s'ubiquen les interfícies que s'utilitzen per a definir les anotacions que es poden utilitzar en MyFrameUOC i que serveixen per a configurar el mapeig entre els objectes de Java i les bases de dades.

El diagrama de classes corresponent a la capa *Annotations* és el següent:

Figura 52. Diagrama de classes de la capa *Annotations*

Les classes o interfícies que s'ubiquen en aquesta capa són les següents:

- **Entity**. Serveix per a definir la configuració d'una entitat.
- **PrimaryKey**. Serveix per a definir la configuració d'una clau primària.
- **Column**. Serveix per a definir la configuració d'una columna.
- **ForeignKey**. Serveix per a definir la configuració d'una clau forana.
- **OneToOne**. Serveix per a definir la configuració d'una relació d'un a un.
- **OneToMany**. Serveix per a definir la configuració d'una relació d'un a molts.
- **ManyToMany**. Serveix per a definir la configuració d'una relació de molts a molts.

11.2.2. Capa *Environment*

En aquesta capa o nivell s'ubiquen les classes que s'encarreguen de gestionar l'entorn de MyFrameUOC. En ella s'ubica la classe *Session* que és a la que l'usuari ha d'accedir per utilitzar el sistema.

El diagrama de classes corresponent a la capa *Environment* és el següent:

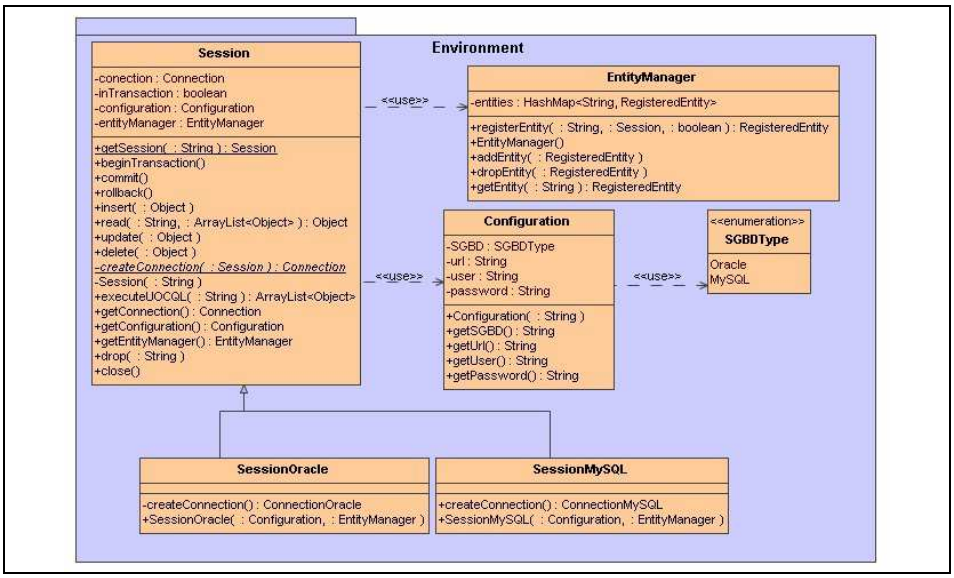

Figura 53. Diagrama de classes de la capa *Environment*

Una descripció de les classes que s'ubiquen en aquesta capa és la següent:

- **Session**. Implementa la gestió d'una sessió i ofereix a l'usuari els mètodes que pot utilitzar per a proporcionar persistència a les seves classes.
- **SessionOracle**. Especialització de la gestió d'una sessió per al cas del sistema de gestió de bases de dades d'Oracle. En aquest tipus de sessió es crea una connexió amb Oracle.

- **SessionMySQL**. Especialització de la gestió d'una sessió per al cas del sistema de gestió de bases de dades de MySQL. En aquest tipus de sessió es crea una connexió amb MySQL.
- **Configuration**. Gestiona la informació corresponent a una configuració com per exemple el tipus de sistema de gestió de bases de dades que s'utilitza, la direcció url per a connectar-se a la base de dades, l'usuari, la contrasenya, etc.
- **SGBDType**. Emmagatzema els diferents tipus de sistemes de gestió de bases de dades que es poden utilitzar en MyFrameUOC. Tal i com ja s'ha comentat, en una primera versió s'implementa el sistema perquè es pugui utilitzar en Oracle i en MySQL.
- **EntityManager**. Gestiona les entitats que estan registrades en memòria i que, per tant, es poden utilitzar per a emmagatzemar, llegir, actualitzar o esborrar objectes de la base de dades. Aquesta classe s'encarregarà de crear les entitats segons es necessitin i proporciona a la classe *Session* les entitats perquè aquesta les pugui utilitzar.

11.2.3. Capa *DataBase*

En aquesta capa o nivell s'ubiquen les classes que s'encarreguen de gestionar les connexions amb els sistemes de gestió de bases de dades i es responsabilitzen d'executar les sentències SQL que la capa *Mapping* necessitarà llençar contra la base de dades.

El diagrama de classes corresponent a la capa *DataBase* és el següent:

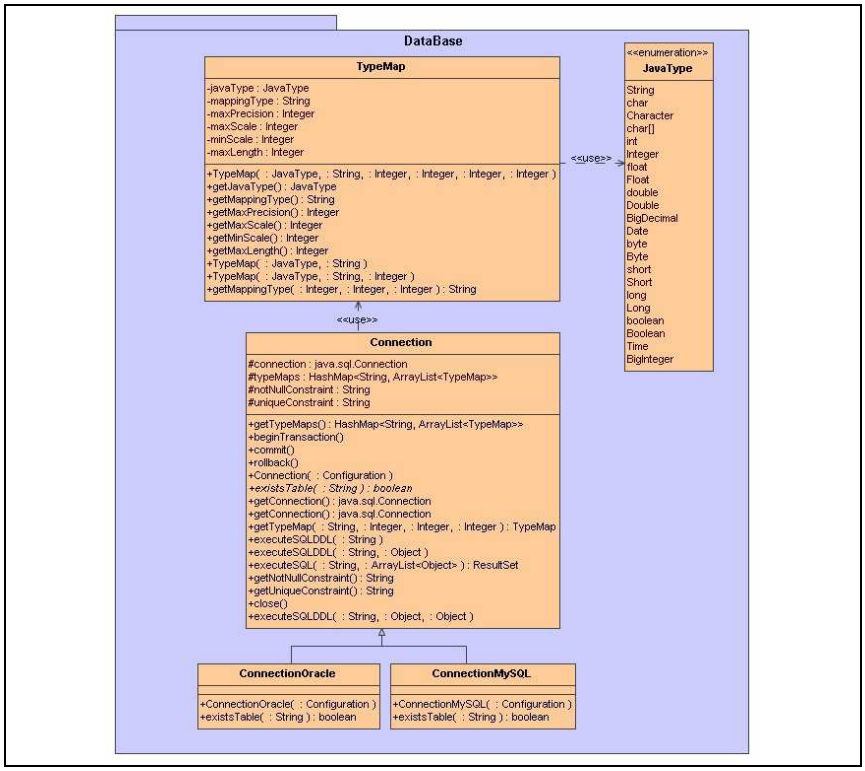

Figura 54. Diagrama de classes de la capa *DataBase*

La descripció de les seves classes és la següent:

- **TypeMap**. Emmagatzema i gestiona el mapeig entre un tipus de dades en Java i el tipus de dades del sistema de gestió de bases de dades.
- **JavaType**. Emmagatzema i representa els tipus de dades en Java que es poden utilitzar en MyFrameUOC.
- **Connection**. Gestiona una connexió amb la base dades. Permet emmagatzemar el mapeig entre els tipus de Java que es poden utilitzar i els tipus de dades utilitzat segons el sistema de gestió de bases de dades. A més, permet executar sentències SQL contra la base de dades.
- **ConnectionOracle**. Especialització d'una connexió amb la base de dades per al cas d'Oracle. S'encarrega de crear una connexió amb el sistema de gestió de bases de dades d'Oracle mitjançant el driver JDBC específic. A més, s'encarrega de crear el mapeig entre els tipus de dades en Java i els que s'utilitzen en Oracle.
- **ConnectionMySQL**. Especialització d'una connexió amb la base de dades per al cas de MySQL. S'encarrega de crear una connexió amb el sistema de gestió de bases de dades de MySQL mitjançant el driver JDBC específic. A més, s'encarrega de crear el mapeig entre els tipus de dades en Java i els que s'utilitzen en MySQL.

11.2.4. Capa *Mapping*

En aquesta capa o nivell s'ubiquen les classes que s'encarreguen del mapeig entre les entitats i els objectes relacionals de la base de dades.

El diagrama de classes corresponent a la capa *Mapping* es mostra en la figura 55.

Una descripció de les seves classes és la següent:

- **RegisteredEntity**. Emmagatzema i gestiona les entitats registrades en memòria. S'encarrega d'obtenir la informació de les anotacions d'una classe, d'emmagatzemar el mapeig dels atributs de l'entitat i de generar les sentències SQL preparades que són necessàries per a dur a terme les operacions CRUD d'una entitat. Una sentència SQL preparada és una sentència SQL en la qual hi ha indicats els paràmetres que cal substituir per a executar la sentència sobre la base de dades.
- **FieldMap**. Representa i gestiona el mapeig d'un atribut d'una entitat que no sigui una relació.
- **PrimaryKeyMap**. Representa i gestiona el mapeig d'una clau primària.
- **ColumnMap**. Representa i gestiona el mapeig d'una columna.
- **AssociationMap**. Representa i gestiona el mapeig de les relacions.
- **ForeingKeyMap**. Representa i gestiona el mapeig de les claus foranes.

- **BinaryAssociationMap**. Representa i gestiona el mapeig de les relacions bidireccionals.
- **OneToOneMap**. Representa i gestiona el mapeig de les relacions d'un a un (1 a 1).
- **OneToManyMap**. Representa i gestiona el mapeig de les relacions d'un a molts (1 a N).
- **ManyToManyMap**. Representa i gestiona el mapeig de les relacions de molts a molts (N a M).

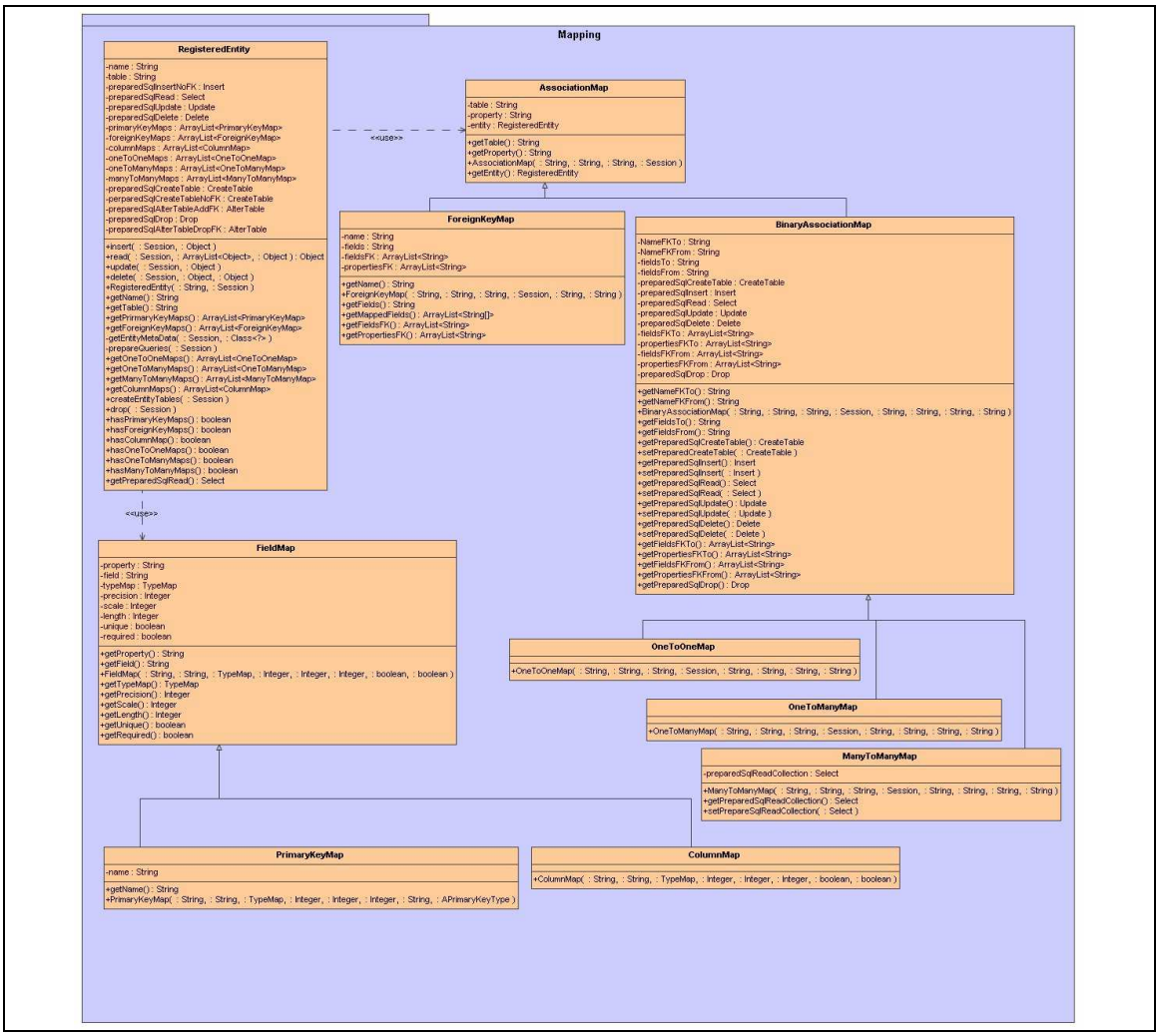

Figura 55. Diagrama de classes de la capa *Mapping*

11.2.5. Capa *QueryManagement*

En aquesta capa o nivell s'ubiquen les classes que representen i gestionen les sentències SQL que s'utilitzen en MyFrameUOC.

El diagrama de classes corresponent a la capa *QueryManagement* és el següent:

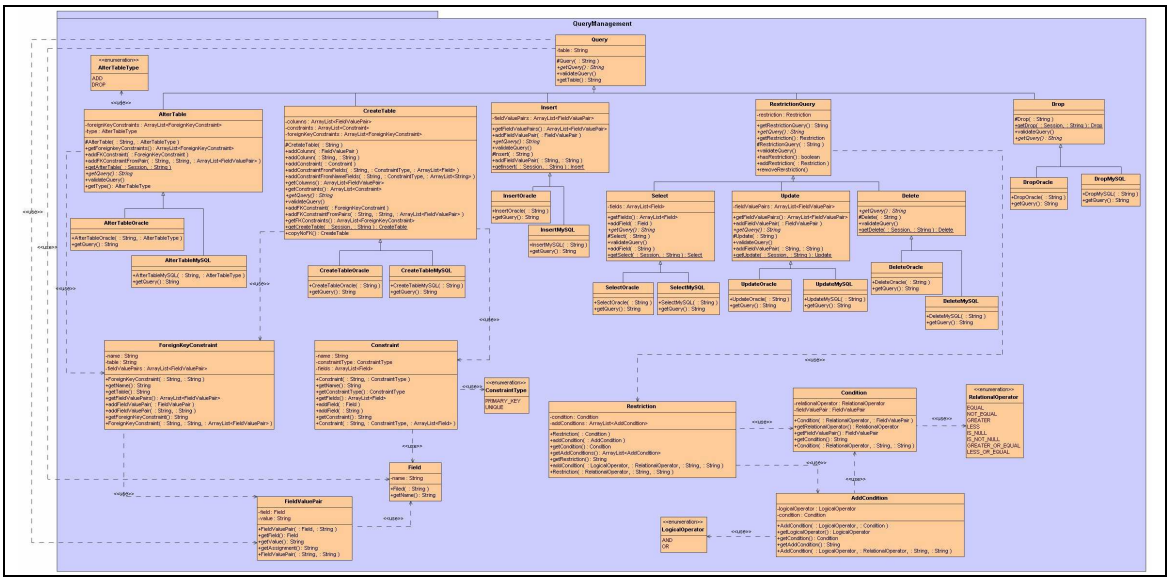

Figura 56. Diagrama de classes de la capa *QueryManagement*

La descripció de les seves classes es mostra a continuació:

- **FieldValuePair**. Representa i gestiona un parell camp-valor o, simplement, un parell de valors. S'utilitza per a representar els camps i els valors d'una sentència SQL.
- **Field**. Representa i gestiona un camp.
- **ForeignKeyConstraint**. Representa i gestiona la definició d'una restricció de clau forana.
- **ConstraintType**. Emmagatzema els tipus de restricció que es poden definir i que no són claus foranes (restriccions de clau primària i úniques).
- **Constraint**. Representa i gestiona la definició d'una restricció que no sigui la d'una clau forana.
- **Query**. Representa i gestiona les sentències SQL.
- **AlterTable**. Representa i gestiona les sentències ALTER TABLE.
- **AlterTableOracle**. Especialització de les sentències ALTER TABLE per al cas d'Oracle.
- **AlterTableMySQL**. Especialització de les sentències ALTER TABLE per al cas de MySQL.
- **CreateTable**. Representa i gestiona les sentències CREATE TABLE.
- **CreateTableOracle**. Especialització de les sentències CREATE TABLE per al cas d'Oracle.
- **CreateTableMySQL**. Especialització de les sentències CREATE TABLE per al cas de MySQL.
- **Insert**. Representa i gestiona les sentències INSERT.
- **InsertOracle**. Especialització de les sentències INSERT per al cas d'Oracle.
- **InsertMySQL**. Especialització de les sentències INSERT per al cas de MySQL.
- **Rrestriction**. Representa i gestiona les restriccions que es poden definir en una sentència SQL.

- **RelationalOperator**. Emmagatzema els tipus d'operadors relacionals que es poden utilitzar en una condició.
- **Condition**. Representa i gestiona una condició.
- **LogicalOperator**. Emmagatzema els tipus d'operadors lògics que es poden utilitzar per afegir condicions en una restricció d'una sentència SQL.
- **AddCondition**. Representa i gestiona les condició que s'afegeixen a les restriccions de les sentències SQL.
- **RestrictionQuery**. Representa i gestiona les sentències SQL que poden tenir una restricció.
- **Select**. Representa i gestiona les sentències SELECT.
- **SelectOracle**. Especialització de les sentències SELECT per al cas d'Oracle.
- **SelectMySQL**. Especialització de les sentències SELECT per al cas de MySQL.
- **Update**. Representa i gestiona les sentències UPDATE.
- **UpdateOracle**. Especialització de les sentències UPDATE per al cas d'Oracle.
- **UpdateMySQL**. Especialització de les sentències UPDATE per al cas de MySQL.
- **Delete**. Representa i gestiona les sentències DELETE.
- **DeleteOracle**. Especialització de les sentències DELETE per al cas d'Oracle.
- **DeleteMySQL**. Especialització de les sentències DELETE per al cas de MySQL.
- **Drop**. Representa i gestiona les sentències DROP TABLE.
- **DropOracle**. Especialització de les sentències DROP TABLE per al cas d'Oracle.
- **DropMySQL**. Especialització de les sentències DROP TABLE per al cas de MySQL.

11.2.6. Capa *Exceptions*

En aquesta capa o nivell s'ubiquen les classes que s'utilitzen per a emmagatzemar i gestionar les excepcions que es produeixen en el framework MyFrameUOC.

El diagrama de classes d'aquesta capa és el següent:

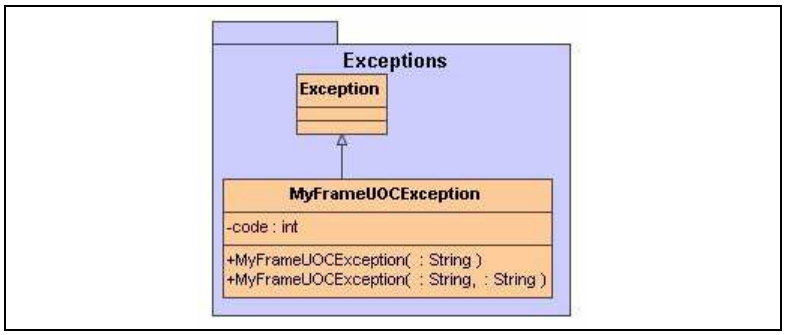

Figura 57. Diagrama de classes de la capa *Exceptions*

Dins d'aquesta capa es posaran totes les classes que es necessitin per a emmagatzemar els diferents tipus de missatges dels errors que es poden produir en el sistema de MyFrameUOC.

11.3. Diagrames de seqüències

En els següents apartats es mostren els diagrames de seqüències de les operacions més importants que s'han d'implementar en el sistema MyFrameUOC.

11.3.1. Obtenció d'una sessió

Tal i com ja s'ha esmentat, l'usuari interacciona amb el sistema mitjançant una sessió que l'ha d'obtenir amb un mètode que ofereix la classe *Session*. El diagrama de seqüències de l'execució d'aquest mètode és el que es mostra en la figura 58.

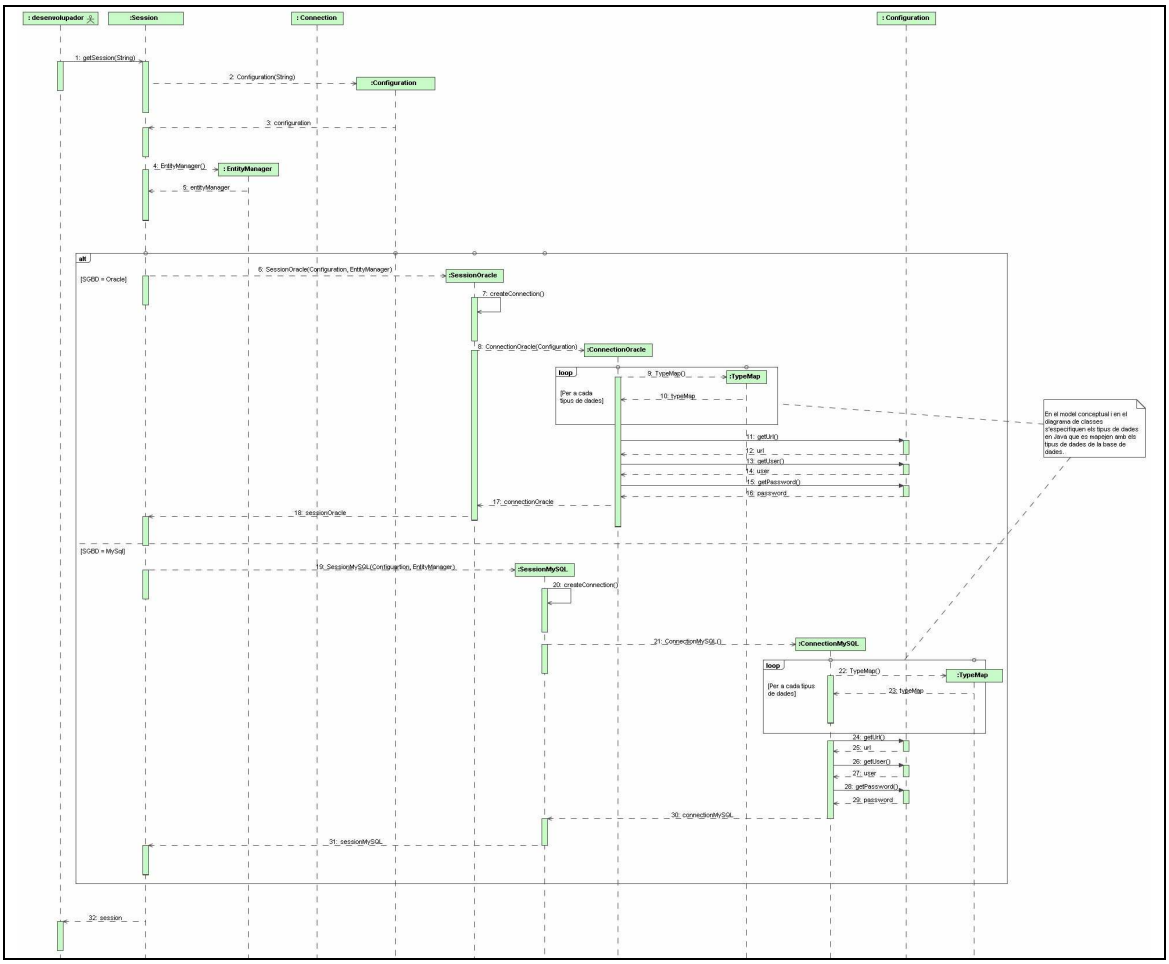

Figura 58. Diagrama de seqüències de l'obtenció d'una sessió

11.3.2. Registre d'una entitat

La classe *EntityManager* implementa el mètode *registerEntity(String, Session, boolean)* que s'encarrega de registrar i d'obtenir una entitat. Aquest procés es repeteix en diferents processos i, per tant, es farà referència a ell quan sigui necessari però sense tornar a fer la seva descripció en detall. El seu diagrama de seqüències es mostra en la figura 59.

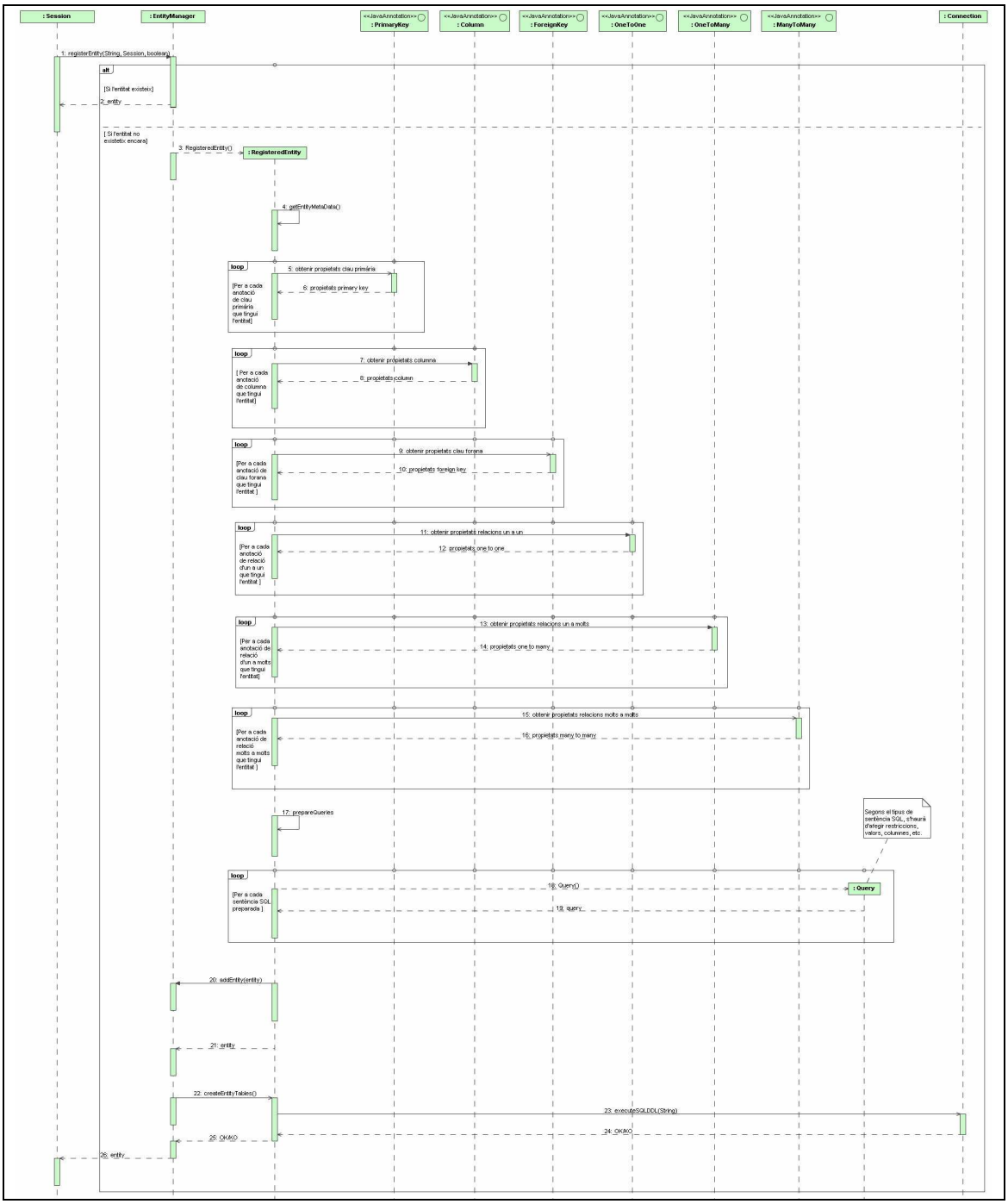

Figura 59. Diagrama de seqüències del registre d'una entitat

11.3.3. Emmagatzemament d'un objecte

L'usuari pot afegir un objecte d'una entitat en la base de dades mitjançant el mètode *insert(Object)*. El seu diagrama de seqüències es mostra en la figura 60.

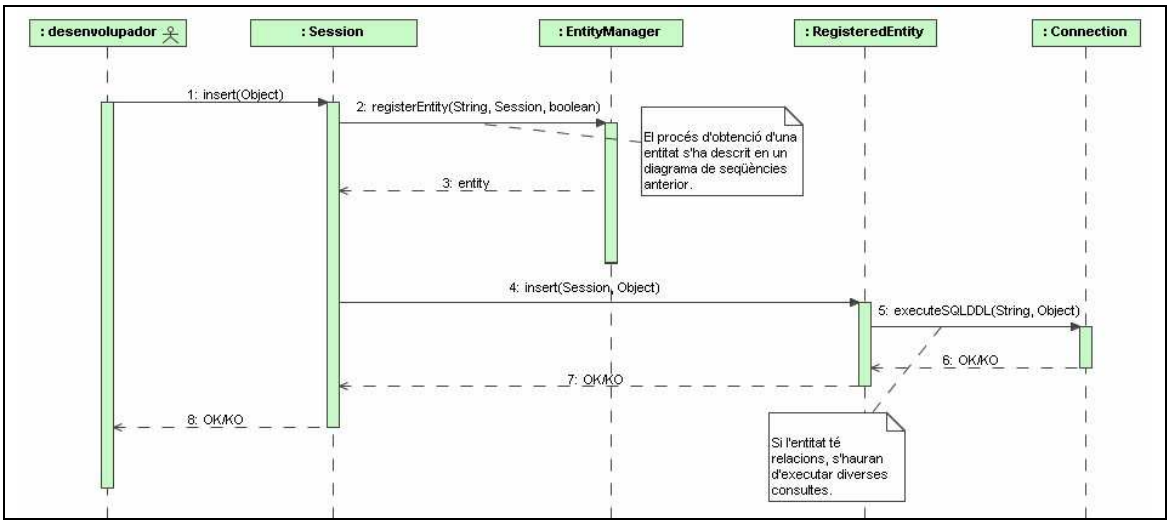

Figura 60. Diagrama de seqüències de l'emmagatzemament

11.3.4. Recuperació d'un objecte

Per a obtenir un objecte que estigui emmagatzemat en la base de dades s'ha d'utilitzar el mètode *read(String, ArrayList<Object>)*. Aquest rep com a paràmetres el nom de la classe de la qual es vol recuperar un objecte i un *ArrayList<Object>* que conté els valors de la clau primària per les quals s'ha de buscar l'objecte. Aquesta llista ha de contenir els valors de les claus primàries en el mateix ordre en el qual es defineixen en l'entitat. El seu diagrama de seqüències es mostra en la figura 61.

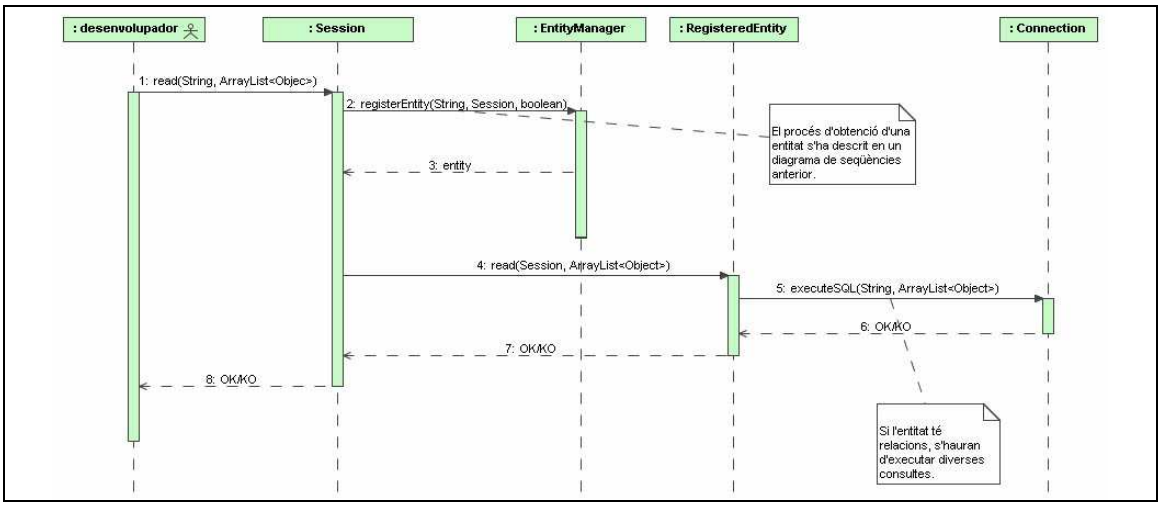

Figura 61. Diagrama de seqüències de la recuperació

11.3.5. Actualització d'un objecte

El procés d'actualització d'un objecte s'encarrega de sincronitzar els valors que té l'objecte amb els que hi ha en la base de dades. El seu diagrama de seqüències es mostra en la figura 62.

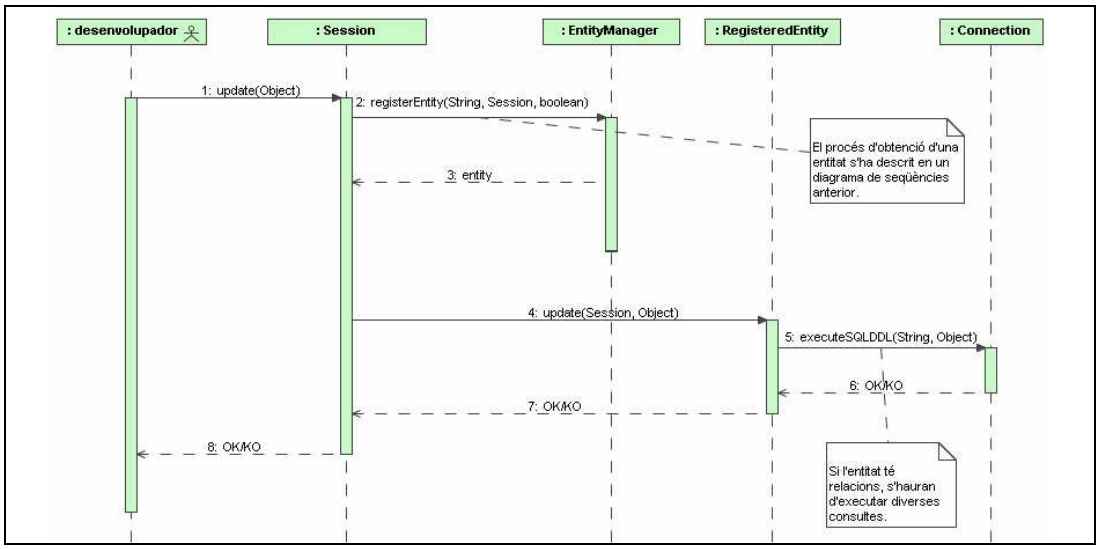

Figura 62. Diagrama de seqüències de l'actualització

No es podran actualitzar les claus primàries d'un objecte. Per tant, només es podran actualitzar els valors dels atributs que siguin columnes o relacions.

11.3.6. Eliminació d'un objecte

L'eliminació d'un objecte de la base de dades es porta a terme mitjançant el mètode *delete(Object)* que implementa la classe *Session*. El seu diagrama de seqüències es mostra en la figura 63.

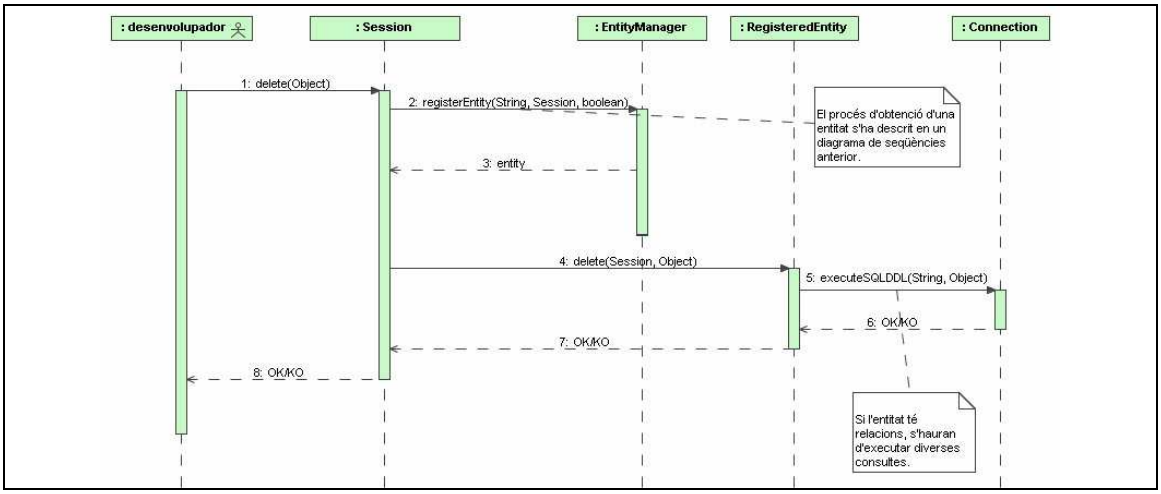

Figura 63. Diagrama de seqüències de l'eliminació

12. Implementació

En els següents apartats es tracten alguns aspectes rellevants sobre la implementació del sistema MyFrameUOC. En primer lloc, es mostra el software i el hardware que s'ha utilitzat per a desenvolupar el framework. En segon lloc, es presenta i s'explica l'estructura del directori del projecte Java utilitzat. I en tercer lloc, es descriuen i s'argumenten algunes decisions que s'han pres durant la construcció del codi font.

12.1. Eines i maquinari utilitzat

Les eines que s'han utilitzat per a desenvolupar el framework de persistència MyFrameUOC són les següents:

- Sistema operatiu *Windows XP Professional*, versió 2002 i amb el *Service Pack* 3
- Eclipse Helios versió 3.6.2
- Java 1.7
- MySQL versió 5.5.28.3
- Oracle 11g versió 2

I el maquinari que s'ha utilitzat per al desenvolupament i per a les proves ha estat el següent:

- Intel Pentium Mobile 1,70 GHz i 2 GB de RAM.
- Intel Pentium 4 CPU 2,80 GHz i 1 GB de RAM.

12.2. Estructura del directori del projecte MyFrameUOC

Per dur a terme la implementació del sistema s'ha utilitzat un projecte Java que s'ha anomenat MyFrameUOC. La seva estructura i contingut es mostra a continuació:

- **src**. S'ubica el codi font del framework de persistència.
- **test**. S'ubiquen les classes de JUnit per a dur a terme les proves.
- **lib**. S'ubiquen les llibreries Java utilitzades en el projecte. Aquestes llibreries són fitxers .jar que són necessaris tenir per a poder utilitzar l'eina MyFrameUOC.
- **doc**. S'ubica la documentació de l'API de MyFrameUOC.

La divisió dels diferents paquets del framework s'ha fet segons els nivells o paquets virtuals que s'han obtingut en el disseny. A més, s'ha creat algun paquet més perquè s'ha considerat convenient. La descripció de tots els paquets implementats és la següent:

- **com.uoc.myframeuoc.annotations**. S'ubiquen les classes o interfícies relacionades amb la definició de les anotacions que s'han d'utilitzar per a configurar les entitats mitjançant MyFrameUOC.

- **com.uoc.myframeuoc.database**. S'ubiquen les classes relacionades amb la part de les bases de dades (connexions, traducció de tipus de dades, etc.).
- **com.uoc.myframeuoc.environment**. S'ubiquen les classes que gestionen l'entorn de MyFrameUOC. En primer lloc, la classe *Session* que dóna accés a les funcionalitats del sistema. En segon lloc, la classe *EntityManager* que s'encarrega de gestionar les entitats que estan registrades en el sistema. I en tercer lloc, la classe *Configuration* que es responsabilitza de gestionar la configuració de MyFrameUOC.
- **com.uoc.myframeuoc.exceptions**. Conté les classes per a gestionar les excepcions de MyFrameUOC.
- **com.uoc.myframeuoc.mapping**. Conté les classes que s'encarreguen del mapeig entre una entitat i la base de dades.
- **com.uoc.myframeuoc.utils**. Conté classes d'utilitat per al framework. En aquest paquet també s'inclouen aquelles classes que s'utilitzen per a emmagatzemar valors constants globals.
- **com.uoc.myframeuoc.querymanagement**. S'ubiquen les classes que s'encarreguen de representar i gestionar les consultes SQL amb les quals tracta el sistema. Aquestes classes faciliten la creació i tractament de consultes SQL que es necessiten utilitzar per a crear, llegir, actualitzat i esborrar objectes de la base de dades.

12.3. Decisions preses en la implementació

En la fase d'implementació s'han pres algunes decisions que es consideren necessàries esmentar per a poder entendre correctament el funcionalment del codi font.

En primer lloc, el mapeig dels tipus de dades s'ha fet segons la millor optimització possible i, per tant, s'ha intentat que les dades emmagatzemades en la base de dades ocupin el mínim d'espai possible. Malgrat tot, es deixa que l'usuari pugui configurar alguns paràmetres (com la precisió, l'escala o la longitud d'un cadena) perquè pugui decidir com s'han d'emmagatzemar les dades en la base de dades. Si l'usuari decideix utilitzar els valors per defecte d'aquests paràmetres, és a dir, no els defineix, llavors l'eina sempre intentarà utilitzar el millor cas possible per tal d'aconseguir una millor optimització.

En segon lloc, un *BigInteger* s'emmagatzemen en la base de dades com una cadena de caràcters. Per tant, el seu emmagatzematge depèn dels caràcters que es puguin emmagatzemar en la base de dades. Per aquest motiu, es recomana indicar el paràmetre *Precision* en les anotacions de MyFrameUOC per a indicar quants dígits tindrà el *BigInteger*. Si no s'indica cap, llavors el sistema agafarà cadenes de caràcters de longitud per defecte (normalment, 255 caràcters).

En tercer lloc, les relacions es gestionen en cascada. Això vol dir que una operació amb un objecte que té relacions afectarà als objectes de la relació.

per exemple, si s'elimina un objecte que té una clau forana, llavors també s'eliminarà l'objecte al qual apunta la clau forana.

I per acabar, el mètode *getEntityMetaData()* de la classe *RegisteredEntity* s'ha subdividit en diversos mètodes, un per a cada tipus d'anotació. És a dir, per obtenir la informació de l'anotació *Entity* s'utilitza un mètode i per a obtenir la informació de l'anotació *PrimaryKey* s'utilitza un altre mètode. Això s'ha fet així per modular la implementació de la funcionalitat. No es mostra en el diagrama de classes perquè es considera que és una decisió que es pren en la implementació i no en el disseny, tot i que es podria haver pres en el disseny. Succeeix el mateix amb el mètode *prepareQueries()* també de la classe *RegisteredEntity*.

13. Aplicació d'exemple *FootballTeam*

En aquest apartat es farà el disseny i la implementació d'una petita aplicació J2EE que mostra el funcionament del framework de persistència MyFrameUOC. Es tracta d'una mini aplicació web anomenada *FootballTeam* i que serveix per a gestionar equips de futbol.

En primer lloc, es presenta el seu anàlisi de requeriments. En segon lloc, es mostra el seu disseny. I en tercer lloc, es mostren alguns aspectes sobre la seva implementació.

13.1. Anàlisi de requeriments

L'objectiu del desenvolupament de l'aplicació *FootballTeam* és mostrar l'ús i el funcionament de l'eina MyFrameUOC. Per tant, en cap cas es pretén desenvolupar una aplicació J2EE completa.

FootballTeam és una mini aplicació web que permet gestionar equips de futbol, els seus jugadors i els entrenaments de cadascun d'ells. L'aplicació ha d'oferir les següents funcionalitats:

- Donar d'alta, consultar, actualitzar i esborrar un equip de futbol.
- Incloure i treure jugadors d'un equip.
- Donar d'alta, consultar, actualitzar i esborrar jugadors.
- Donar d'alta, consultar, actualitzar i esborrar un pla d'entrenament.
- Incloure i treure jugadors d'un pla d'entrenament. A partir d'un pla d'entrenament s'ha de poder conèixer els jugadors que el segueixen.
- Iniciar o finalitzar plans d'entrenament d'un jugador. A partir d'un jugador s'ha de poder conèixer quins plans d'entrenament està seguint actualment i s'ha de poder iniciar o finalitzar un pla d'entrenament per a ell.

Cal tenir en compte els següents requeriments:

- L'aplicació ha de ser J2EE i l'usuari ha de poder accedir-hi mitjançant un explorador web. En un principi, qualsevol usuari que tingui accés a Internet podrà accedir a l'aplicació.
- Un equip de futbol està format per jugadors i contindrà la següent informació:
    - **Identificador**. Ha de permetre identificar de forma inequívoca un equip.
    - **Nom**. Nom complet que té l'equip o el club al qual pertany.
    - **Estadi**. Nom del camp de futbol en el qual juga l'equip.
    - **Pressupost**. Quantitat d'euros de què disposa l'equip per a incorporar nous jugadors i cobrir despeses (manteniment, sous, etc.).
- Per a un jugador cal emmagatzemar la següent informació:
    - **DNI**. Número del DNI del jugador. Serveix per a identificar cada jugador.
    - **Nom** i **cognoms**. El nom, el primer i el segon cognom del jugador.
    - **País de naixement**. País en el que va néixer el jugador.
    - **Data de naixement**.
    - **Edat**. Número d'anys que té actualment el jugador.
    - **Demarcació**. Posició que ocupa el jugador dins del terreny de joc (si és porter, defensa, lateral, davanter, etc.).
    - **Sou**. Quantitat d'euros a l'any que té el jugador de sou.
- Un pla d'entrenament serveix per a entrenar als jugadors. Un jugador pot seguir cap o diversos plans d'entrenament en un moment donat i un pla d'entrenament pot estar seguit per cap o diversos jugadors. D'un pla d'entrenament cal emmagatzemar la següent informació:
    - **Identificador**. Codi que identificar el pla d'entrenament.
    - **Nom**. Nom de l'entrenament.
    - **Descripció**. Text descriptiu del pla d'entrenament.
    - **Instruccions**. Passos o instruccions que s'han de seguir per a completar l'entrenament.
    - **Sessions**. Número de sessions de què consta l'entrenament.
    - **Durada de cada sessió**. Número d'hores i de minuts que es triga en completar una sessió de l'entrenament.

13.2. Disseny

Segons els requeriments que s'han descrit en l'apartat anterior, es considera que el model conceptual o esquema invariant de l'aplicació *FootballTeam* és el següent:

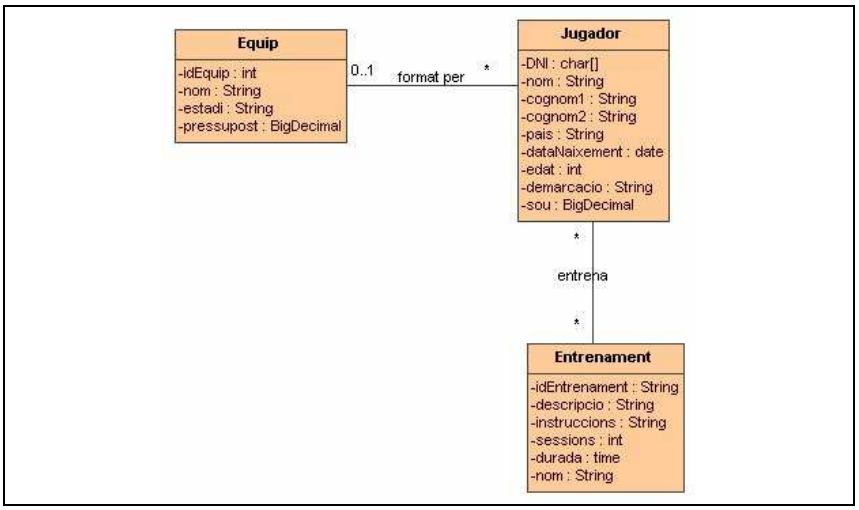

Figura 64. Model conceptual de l'aplicació *FootballTeam*

Una descripció en molt alt nivell del sistema en termes de components arquitectònics i interfícies és el següent:

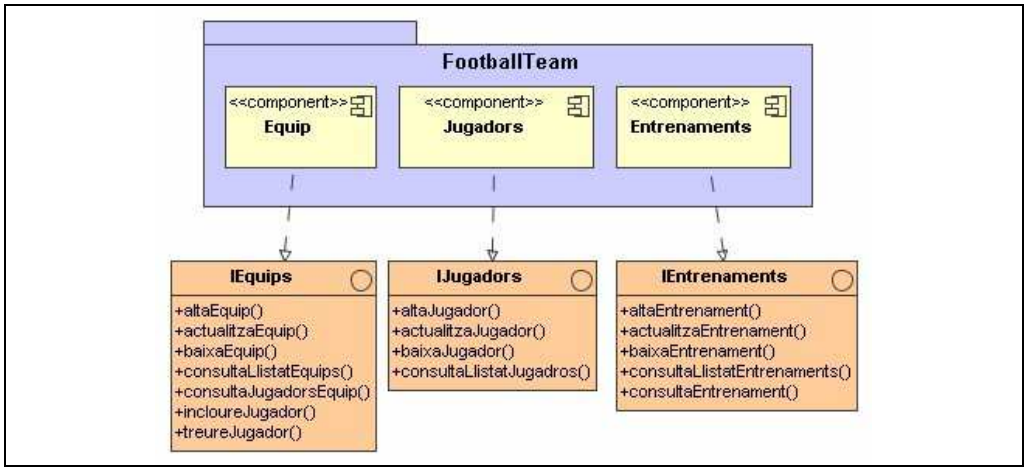

Figura 65. Descripció en molt alt nivell del sistema *FootballTeam*

En aquesta descripció no s'han tingut en compte alguns aspectes com l'estil arquitectònic que s'utilitzarà. En aquest cas, es desenvoluparà una aplicació web i s'utilitzarà un estil arquitectònic heterogeni, model client-servidor i de tres capes. Una capa s'encarregarà de la presentació, una altra de la lògica de negoci i una altra de la integració de les dades.

Per tant, s'ha de dividir el sistema *FootballTeam* en tres capes i per a fer-ho, s'ha de descompondre els components arquitectònics que s'han obtingut anteriorment en components de gra més fi, i cadascun d'aquests s'ubicarà en una capa diferent.

Així, l'arquitectura de programari del sistema *FootballTeam* queda de la següent manera:

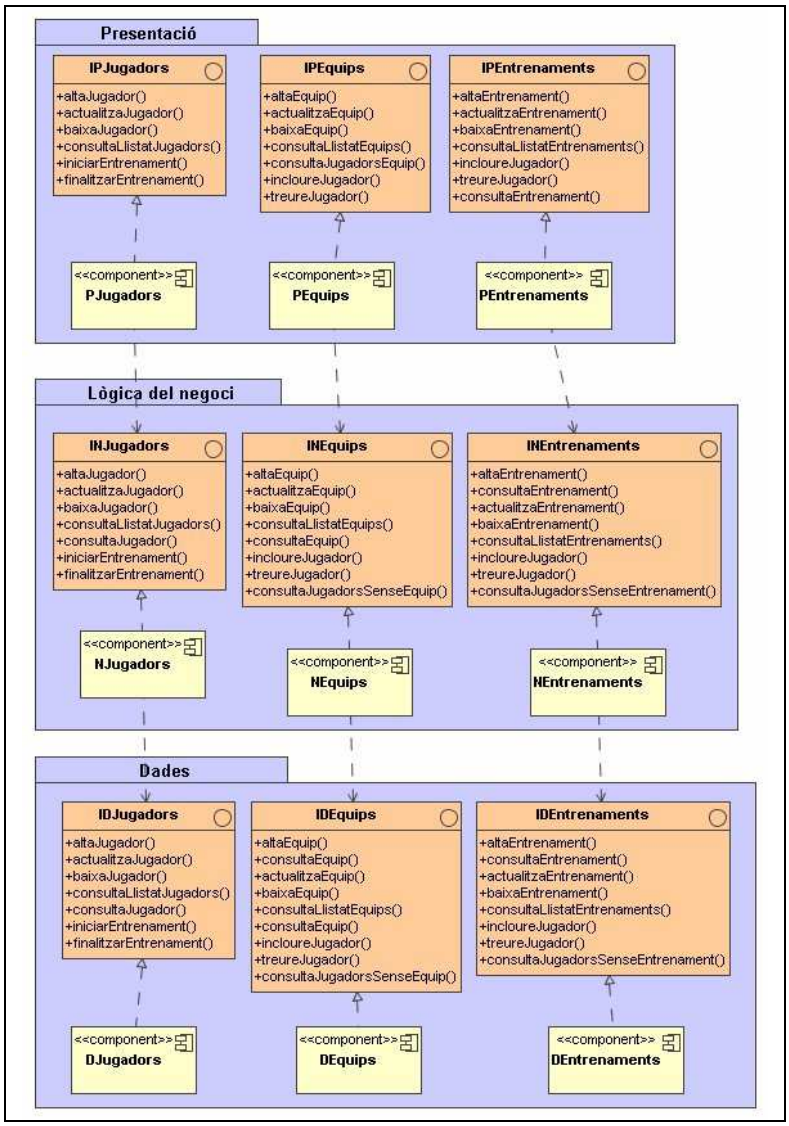

Figura 66. Arquitectura programari del sistema *FootballTeam*

Els components de la capa de presentació encara es poden refinar més. Per a fer-ho, es té en compte que s'aplica el patró MVC (Model-Vista-Controlador) on els components que fan de model estaran implementats en la capa de la lògica de negoci. Per tant, dins dels components arquitectònics de la capa de presentació s'han d'ubicar els components que fan de vista i els que fan de controlador.

Així, l'arquitectura de programari del sistema queda de la següent manera:

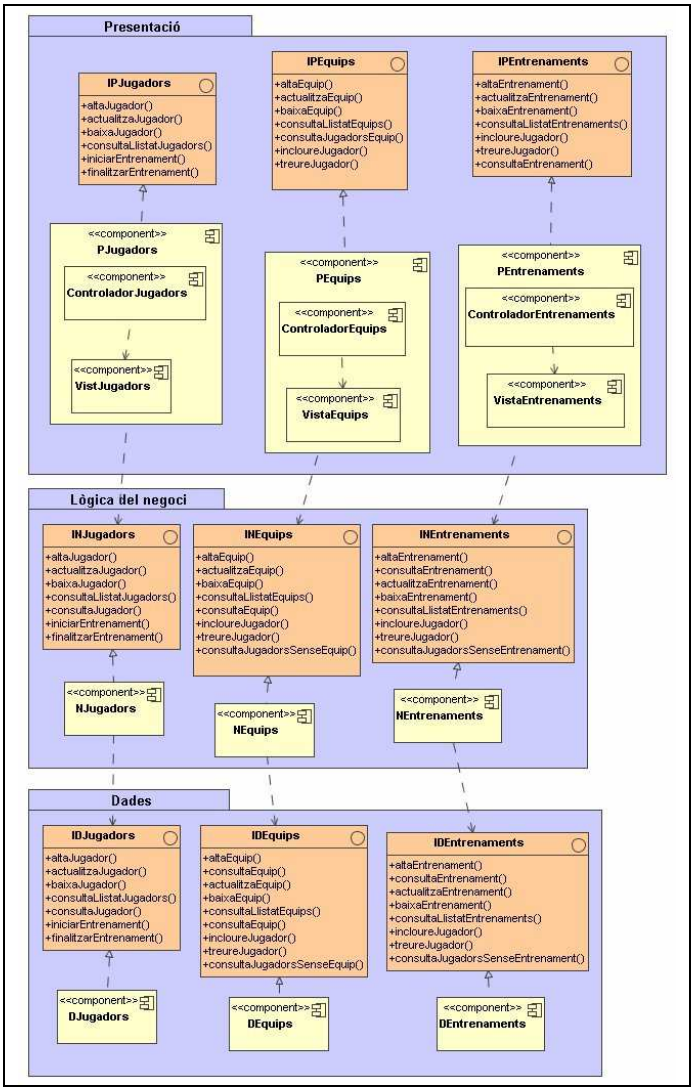

Figura 67. Arquitectura programari refinat del sistema *FootballTeam*

El diagrama de desplegament del sistema es mostra a continuació:

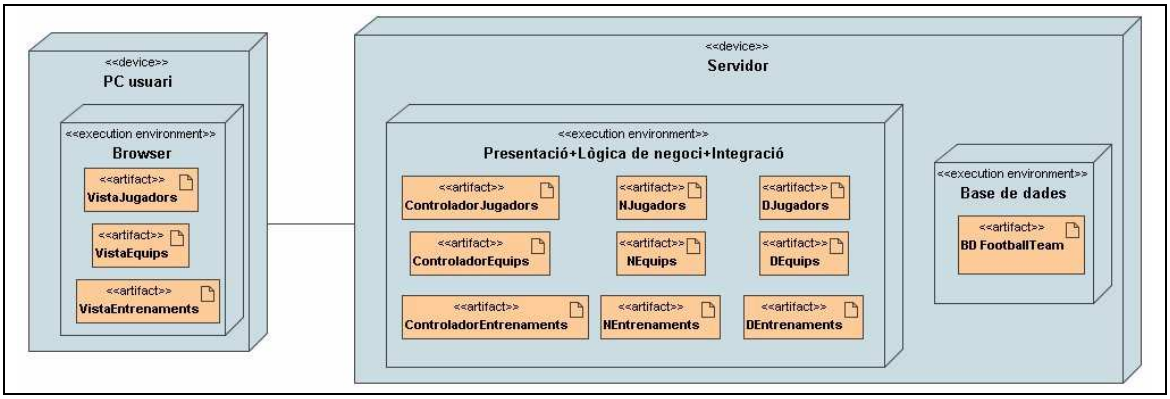

Figura 68. Diagrama de desplegament del sistema *FootballTeam*

Com que s'utilitzarà la tecnologia J2EE per a implementar el sistema, s'ha de fer una fase prèvia de disseny per tal d'adaptar la descripció que ja es té del sistema a la implementació concreta en J2EE. Aquesta adaptació només afecta als components arquitectònics de la capa de presentació (vistes i controladors). Cal tenir en compte que es decideix no utilitzar la tecnologia *Enterprise Java Beans (EJB)* perquè es tracta d'una aplicació web de prova i es considera que és innecessari per aconseguir l'objectiu final (que és mostrar el funcionament del l'eina MyFrameUOC).

Per a tots els components de la capa de presentació es prenen les següents decisions:

- El component que fa de controlador s'implementarà mitjançant un *Servlet*. Aquest s'encarregarà de recollir totes les peticions que es facin cap al component de presentació i cridarà a la lògica de negoci que sigui necessària per a cada cas.
- El *Servlet* també s'encarregarà de seleccionar quina és la següent vista que s'ha de mostrar a l'usuari segons el resultat que s'obtingui de la lògica de negoci.
- Les vistes s'implementaran mitjançant JSP.
- Cada operació tindrà:
    - Una vista d'entrada que servirà per a recollir les dades.
    - Una vista de sortida que informarà del resultat de l'operació.

El component *PJugadors* es pot refinar de la manera següent:

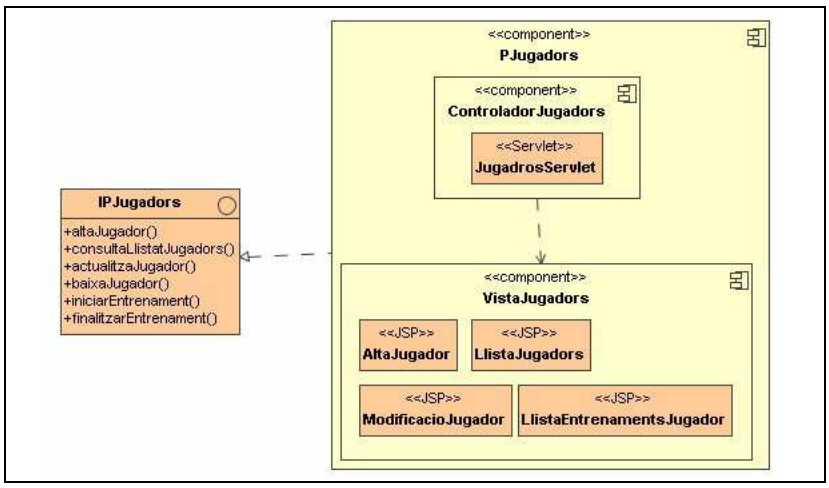

Figura 69. Refinament del component *PJugadors*

El component *PEquips* es pot refinar de la manera següent:

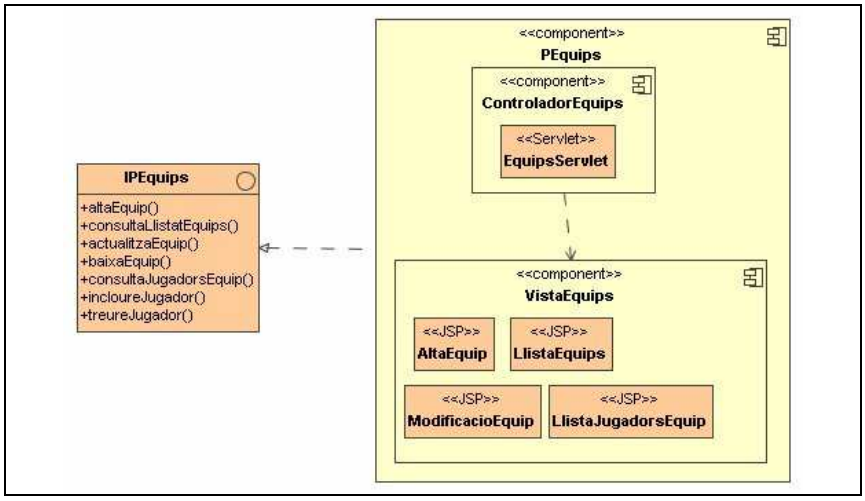

Figura 70. Refinament del component *PEquips*

I el component *PEntrenaments* es pot refinar tal i com es mostra en la figura següent:

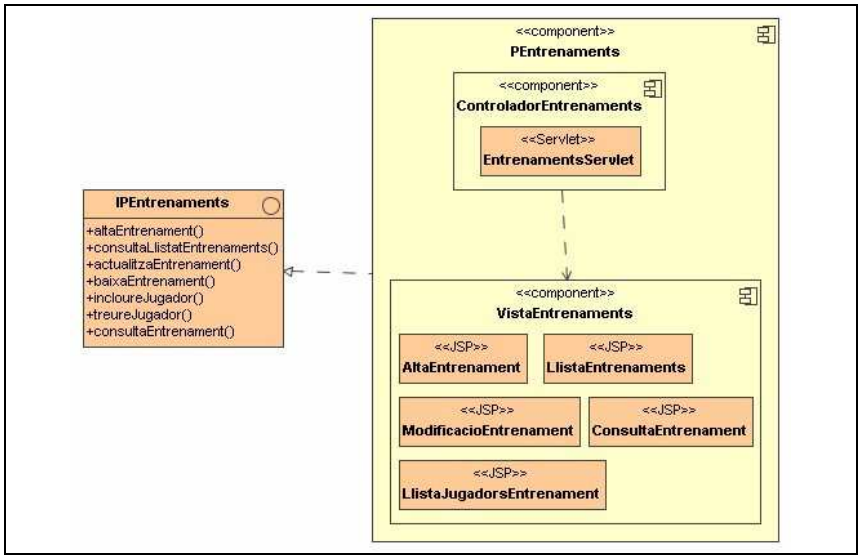

Figura 71. Refinament del component *PEntrenaments*

13.3. Implementació

En els següents apartats s'aborden aspectes relatius a la implementació del sistema *FootballTeam*. En primer lloc, es presenten les eines que s'han utilitzat i les seves configuracions. En segon lloc, es fa una descripció de l'estructura del projecte Java que s'ha utilitzat. I en tercer lloc, es mostren les decisions que s'han pres per a dur a terme la implementació.

13.3.1. Eines i configuracions utilitzades

Per a dur a terme la implementació de *FootballTeam* s'han utilitzat les següents eines:

- JBoss versió 4.2.2.GA
- Eclipse Europe versió 3.2.2. S'utilitza aquesta versió perquè permet treballar amb projectes J2EE.
- Puglins JBoss per a integrar l'eclipse amb JBoss.

Per a poder fer la implementació i utilitzar l'eina MyFrameUOC, el projecte Java *FootballTeam* s'ha de configurar de la següent manera:

- Cal incloure la llibreria JBoss 4.2 Runtime.
- Cal incloure la llibreria (.jar) del framework MyFrameUOC. Aquesta es pot trobar dins de la carpeta *lib*. En qualsevol cas, es pot obtenir a partir del projecte de MyFrameUOC (fent un export i construint un .jar).
- En la construcció del fitxer .war de l'aplicació web cal incloure les llibreries que s'utilitzen en el projecte (totes aquestes llibreries estan incloses en la carpeta *lib* del propi projecte *FootballTeam*). És a dir, cal incloure les següents llibreries .jar:
    - La del framework MyFrameUOC: MyFrameUOCv1.0.jar
    - La del JDBC de MySQL: mysql-connector-java-5.1.20-bin.jar
    - La del JDBC d'Oracle: ojdbc6.jar
    - La del JDom: jdom-2.0.4.jar
- Per a construir i desplegar l'aplicació web (fitxer .war) s'utilitza la vista *Project Archive* de perspectiva *JBoss AS* de l'eclipse. El fitxer .war que hi ha a l'arrel del projecte és el que s'ha utilitzat per a fer les proves i, per tant, està configurat per a utilitzar un SGBD que està en una màquina concreta.

És indispensable tenir instal·lat el SGBD que s'utilitzi (MySQL o Oracle). En cas contrari, l'aplicació *FootballTeam* no podrà accedir a la base de dades i mostrarà un missatge d'error. Per a fer les proves de l'aplicació web s'ha utilitzat les mateixes versions de MySQL i d'Oracle que les que es van utilitzar per a desenvolupar el framework MyFrameUOC.

A més, també cal modificar el fitxer *configuracioMySQL.xml* i el fitxer *configuracioOracle.xml* que es troben en el directori *webapps/WEB-INF* del projecte per a indicar la ruta, l'usuari i la contrasenya d'accés a la base de dades.

Per a finalitzar, l'aplicació està configurada perquè utilitzi el fitxer *configuracióOracle.xml* per defecte, però es pot modificar canviant la variable global anomenada *MYFRAMEUOC_CONF* que es troba en la classe *com.uoc.footballteam.util.FootballTeam*.

13.3.2. Estructura del directori del projecte *FootballTeam*

L'estructura i el contingut del projecte *FootballTeam* que s'ha utilitzat per a fer la implementació és el següent:

- **src**. S'ubiquen les classes Java que implementen l'aplicació.
- **lib**. Conté les llibreries .jar que s'utilitzen en el projecte.
- **webapps**. Conté els fitxers .html i .jsp.
- **webapps/WEB-INF**. Conté els següents fitxers .xml:
    - *configuracioMySQL.xml*. Aquest fitxer conté la configuració de MyFrameUOC que s'ha utilitzat per a fer les proves en MySQL. Cal modificar la url, l'usuari i la contrasenya per al SGBD de MySQL que es tingui instal·lat.
    - *configuracioOracle.xml*. Igual que l'anterior però pel cas d'Oracle.
    - *jboss-web.xml*. Fitxer de configuració per a JBoss.
    - web.xml. Descriptor de desplegament per a indicar els Servlets que s'utilitzen en l'aplicació.
- **webapps/css**. Conté els fitxers .css que s'utilitzen per a donar format a les pàgines .html i .jsp.

13.3.3. Implementació

Tal i com s'indica en el disseny, la implementació de la capa de presentació es fa mitjançant servlets i pàgines .jsp. Per tal de tenir un únic punt de control s'aplica el patró *Facade* mitjançant el servlet *FrontController* que és qui rep totes les peticions que es fan des del client (en aquest cas, des de l'explorador web de l'usuari).

A més, per a crear i gestionar els servlets s'utilitza el patró *Factory* i el patró *Command*, i s'obté així un *Factory* de *Commands*.

En la realització de les proves de l'aplicació FootballTeam es detecta que el driver JDBC d'Oracle té algun problema amb la mida de les cadenes quan s'utilitza alguns caràcters com la 'ñ' o accents o altres signes. Si s'omple un camp de forma completa i s'utilitzen aquests caràcters, el driver afegeix algun caràcter en blanc i provoca que no es pugui emmagatzemar en la base de dades. Tot i així, l'aplicació mostra un missatge per a informar que s'ha produït un error.

Per acabar, en l'annex A que hi ha al final de la memòria es pot consultar les imatges sobre algunes pantalles de l'aplicació FootballTeam que intenten il·lustrar el seu funcionament bàsic.

14. Conclusions

Tot seguit es fa una breu exposició de les conclusions del projecte i dels objectius que s'han aconseguit.

Per una banda, el desenvolupament del projecte ha permès conèixer amb detall la problemàtica del mapeig d'entitat-relació que es va plantejar a l'inici i s'ha estudiat a fons algunes de les tècniques que s'utilitzen per a implementar un framework de persistència.

Tota aquesta investigació ha donat pas al naixement d'un framework de persistència propi anomenat MyFrameUOC. Aquesta eina s'ha desenvolupat de tal manera que facilita i agilitza el procés d'implementació de la capa de persistència de les aplicacions J2EE. Per tant, s'ha cobert l'objectiu principal d'aquest projecte.

Per una altra banda, s'han implementat totes les funcionalitats bàsiques i avançades que es van proposar com a objectius:

- **Operacions CRUD**. L'usuari pot emmagatzemar, llegir, actualitzar i esborrar un objecte de la base de dades.
- **Claus foranes**. L'usuari pot associar dues entitats mitjançant l'ús de claus foranes.
- **Relacions d'un a un**. L'eina permet utilitzar una classe que tingui una relació d'un a un amb una altra classe.
- **Relacions d'un a molts**. Es pot treballar amb classes que tinguin una relació d'un a molts amb una altra classe.
- **Relacions de molts a molts**. Es permeten les relacions de molts a molts.
- **Llenguatge UOCQL**. L'eina proporciona un llenguatge de consultes senzill que permet fer consultes més complexes per a obtenir objectes que estan emmagatzemats en la base de dades.

A més, l'eina facilita la feina al desenvolupador mitjançant l'ús d'anotacions que li permeten configurar les seves classes perquè siguin persistents.

Per acabar, es considera que s'han aconseguit tots els objectius que es van plantejar al principi. Malgrat tot, la falta de disponibilitat de temps ha fet impossible implementar algunes millores que es deixen com a possibles ampliacions.

15. Millores i ampliacions

A continuació es detallen alguns aspectes que es plantegen com a possibles millores i ampliacions d'aquest projecte:

- **Llenguatge UOCQL**. Les consultes UOCQL les ha d'escriure l'usuari manualment. Es podria proporcionar classes de Java per a facilitar-ne la seva definició. En la següent figura es mostra un exemple:

```
UOCQL query                      = new UOCQL();
ArrayList<Condition> conditions = new ArrayList<Condition>();
ResultSet rs                     = null;
Session session                  = null;
String file                      = "configurationOracle.xml";

//FROM Empleat WHERE edat > 25 AND edat < 50
conditions.add(new Condition("edat", Condition.OPER_GREATER, new Integer(25)));
conditions.add(new Condition(Condition.OPER_AND, "edat", Condition.OPER_LESS,
new Integer(50)));

query.setEntity("Empleat");
query.addConditions(conditions);
session = Session.getSession(file);
rs = session.executeUOCQL(query);
```

Figura 72. Millora del llenguatge UOCQL

- **Operacions en cascada**. L'eina treballa amb operacions en cascada de forma automàtica. Això vol dir que si s'elimina un objecte també s'eliminaran els objectes amb els quals està relacionat (segons les relacions que tingui definides en les anotacions). Llavors, es podria fer una ampliació i permetre configurar aquest comportament de tal manera que fos l'usuari qui pugés escollir entre treballar en cascada o no.
- **Obligatorietat de les claus foranes**. En aquesta primera versió de MyFrameUOC, les claus foranes de les taules físiques no han de ser obligatòries (poden contenir valors nuls). Una possible ampliació seria permetre que les claus foranes poguessin ser obligatòries.
- **Anotacions**. En les relacions, la configuració de les correspondències entre els camps d'una taula i els atributs d'una classe es fa mitjançant una simple cadena amb comes i guions. Es podria ampliar la definició de les anotacions per a facilitar aquesta configuració:

```
@Entity(Name = "TaulaA")
public class A(){
    @PrimaryKey(Name="PK_TaulaA")
    private int idA;

    @ForeignKey(Name = "FK_TaulaA", joinColumns={@JoinColumns(name = "b")},
joinProperty={@JoinProperty(name = "idB")})
    private B b;
}

@Entity(Name = "TaulaB")
publlic class B(){
    @PrimaryKey(Name = "PK_TaulaB")
    private int idB;
}
```

Figura 73. Millora de les anotacions de MyFrameUOC

- **Claus primàries**. En la creació d'un objecte l'usuari ha d'informar el valor de la clau primària. Una alternativa seria que es pogués configurar aquest comportament i que, per exemple, l'eina donés valors a les claus primàries de forma automàtica.
- **Tipus de dades**. MyFrameUOC permet utilitzar la majoria de tipus de dades en Java, però hi alguns que no estan permesos com per exemple els arrays (String[], int[], etc.). Tampoc permet treballar amb tipus de dades molt grans com els Blob o els Clob. Es podria ampliar la funcionalitat perquè es suportessin tots aquests tipus de dades.
- **Fitxer de configuració**. En el fitxer de configuració de MyFrameUOC només es permet definir una connexió a una única base de dades. Es podria ampliar perquè es pogués definir diverses connexions en un fitxer de configuració i utilitzar en cada moment la que més convingui.
- **Internacionalització**. Els missatges d'error de l'eina estan tots escrits en català. Es podria ampliar la gestió d'excepcions i permetre utilitzar diferents idiomes.
- **Herència i polimorfisme**. L'eina no està preparada per a suportar l'herència i el polimorfisme. Per tant, es podria estudiar la possibilitat de fer aquesta implementació i permetre emmagatzemar objectes de classes que hereten d'altres classes.

Glossari

anotació *f* Mecanisme que permet afegir metadades a una classe de Java mitjançant l'ús d'etiquetes que s'inclouen en el propi codi.

applet *m* Aplicació informàtica simple en Java amb una funció específica, que s'executa mitjançant un explorador web.

aplicació Java stand-alone *f* Aplicació informàtica en Java que s'executa en la màquina virtual de Java que està instal·lada en la màquina del client.

arquitectura client-servidor *f* Estil arquitectònic organitzat com un conjunt de components servidors, que ofereixen uns serveis, i un conjunt de clients que utilitzen aquests serveis.

arquitectura en capes o nivells *f* Estil arquitectònic que representa una organització jeràrquica dels elements d'un sistema, de manera que cada capa proporciona serveis als elements de la capa immediatament superior (o anterior), i se serveix dels serveis que li brinden els elements de la capa immediatament inferior (o següent).

BMP (*Bean* Managed Persistence) *f* La persistència gestionada pel *bean* és una manera d'implementar la gestió de la persistència dels components EJB d'entitat en la que el desenvolupador ha d'implementar els mètodes de *callback* (veure **mètodes de *callback***).

CMP (Container Managed Persistence) *f* La persistència gestionada pel contenidor és una manera d'implementar la gestió de la persistència del components EJB d'entitat en la que el contenidor implementa els mètodes de *callback* (veure **mètodes de *callback***) mitjançant un descriptor de desplegat que el desenvolupador ha de configurar.

contenidor *m* Peça de programari on es despleguen els components i ofereix accés als serveis que proporciona la plataforma.

EJB (Enterprise JavaBean) *m* Components de negoci distribuïts, desenvolupats amb Java que compleixen unes especificacions i, per tant, es poden desplegar a qualsevol contenidor d'EJB d'un servidor d'aplicacions compatible amb la plataforma J2EE.

EJB d'entitat *m* Components que representen les dades persistents del model de negoci de l'aplicació. Es poden veure com una representació Java, en memòria i en forma d'objecte de la informació emmagatzemada en algun mitjà persistent.

esquema invariant *m* Especificació dels predicats que s'han de complir en qualsevol dels estats pels quals travessi un sistema.

estil arquitectònic *m* Conjunt de patrons estructurals que caracteritzen una família de sistemes, d'acord amb una sèrie de decisions essencials sobre l'estructura d'alt nivell i la composició dels sistemes, i establint restriccions importants sobre els seus elements i les relacions possibles entre aquests.

framework de persistència *m* Conjunt d'eines de software o classes utilitzades per a millorar el rendiment del procés d'implementació de la capa de persistència mitjançant la reutilització de codi.

J2EE *f* Plataforma de desenvolupament empresarial que defineix un estàndard pel desenvolupament d'aplicacions empresarials multicapa.

JDBC *f* Són les sigles en anglès de *Java data base connectivity* i són unes classes en Java que permeten accedir a bases de dades i a diferents tipus de fonts de dades com fulls de càlcul o fitxers de text.

JDO (Java Data Objects) *m* Estàndard que defineix les interfícies que un proveïdor ha d'implementar i que el desenvolupador utilitza per a proporcionar persistència a les seves classes.

legacy-system *m* Sistema extern a una l'aplicació, que ja està desenvolupat i a la que l'aplicació necessita accedir.

metadada *f* Dada que dóna informació sobre una altra dada.

mètodes de callback *m* Són un conjunt de mètodes (ejbActivate(), ejbLoad(), ejbPassivate, ejbRemove(), ejbStore(), etc.) que el contenidor d'EJB crida per a dur a terme la gestió de la persistència dels components EJB d'entitat i que els pot implementar el desenvolupador (veure BMP) o el contenidor (veure CMP).

OMR (Object-Relational Mapping) *m* Tècnica de programació que converteix els objectes de la programació orientada a objectes en dades d'una base de dades relacional.

POJO (*Plain Old Java Object*) *m* Classe Java normal.

servidor d'aplicacions *m* Peça de programari que implementa els serveis que ofereixen els contenidors als components.

SGBD (Sistema de gestió de bases de dades) *m* Programari que gestiona i controla bases de dades. Les seves principals funcions són les de facilitar-ne la utilització simultània a molts usuaris de tipus diferents, independitzar l'usuari del món físic i mantenir la integritat de les dades.

UML *m* Llenguatge unificat de modelatge que serveix per a descriure, modelar, documentar i representar gràficament un sistema.

Bibliografia

Bauer, C.; King, G. (2007). *Java Persistence with Hibernate.* New York: Manning.

Begin, C.; Goodin, B.; Meadors, L. (2007). *iBATIS in Action.* New York: Manning.

Caicedo Rendón, O. M. *Hibernate*. [en línia] ftp://jano.unicauca.edu.co/Maestria/cursos/AplicacionesInternet/transp/Hibernate.pdf [data de consulta: 25/10/2012].

Camps i Riba, J. M. (2006). *J2EE. Una plataforma de components distribuïda*. Barcelona: Universitat Oberta de Catalunya, Estudis d'Informàtica i Multimèdia.

Camps Paré, R. (2004). *Introducció als sistemes de gestió de bases de dades*. Barcelona: Universitat Oberta de Catalunya, Estudis d'Informàtica i Multimèdia.

Di Gracia, Nelson O. *Migración hacia un modelo de persistencia orientado a objetos.* [en línia] http://postgrado.info.unlp.edu.ar/Carreras/Magisters/Ingenieria_de_Software/Tesis/Di_Grazia_Nelson_Omar.pdf [data de consulta: 14/10/2012].

Fíguls i Massó, D. (2004). *Programació mitjançant l'SQL.* Barcelona: Universitat Oberta de Catalunya, Estudis d'Informàtica i Multimèdia.

Gutiérrez, Javier. *¿Qué es un framework web?* [en línia]. http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf [data de consulta: 09/10/2012].

Inoto Soto, R. (2006). *Frameworks de persistencia en Java*. [en línia] http://www.jtech.ua.es/jornadas/06/charlas/Persistencia.pdf [data de consulta: 23 /09/2012].

Keith, M.; Schnicariol, M. (2009). *Pro JPA 2: Mastering the Java™ Persistence API.* Estats Units: Appress.

Linwood, J.; Minter, D. (2010). *Beginning Hibernate*. New York: Apress.

Mármol Castillo, J. *Persistencia de objetos. JDO, solución Java.* [en línia] http://dis.um.es/~jmolina/Persistencia%20de%20Objeto%20JDO.pdf [data de consulta: 13/10/2012].

Moreno Vergara, N.; Vallecillo Moreno, A.; Romero Salguero, J.R.; Durán Muñoz, F. (2006). *Arquitectura del programari*. Barcelona: Universitat Oberta de Catalunya, Estudis d'Informàtica i Multimèdia.

Payá Martín, A. *Análisis y uso de frameworks de persistencia en Java*. [en línia] http://www.iit.upcomillas.es/pfc/resumenes/450955e7368ca.pdf [data de consulta: 14/10/2012].

Redondo Garcia, J.L. *Introducción a HIBERNATE*. [en línia] http://iberjava.files.wordpress.com/2010/06/hibernate2.ppt [data de consulta: 25/10/2012].

Richardson, C. (2006). *POJOs in Action. Developping enterprise applications with lightweight frameworks.* Greenwich: Manning.

Roods, Robin M. (2003). *Java Data Objects*. Gran Bretaña: Addison-Wesley.

Rosés Albiol, F. (2003). *Introducción a Hibernate.* [en línia] http://www.froses.com/Assets/Files/Articles/Hibernate_Introduccion_es.pdf [data de consulta: 20/10/2012].

R. Johnson, B. Foote (1988). *Designing Reusable Classes*. Journal of Object-Oriented Programming.

Vallecillo Moreno, A.; Romero Salguero, J. R.; Moreno Vergara, N.; Durán Muñoz, F. (2006). *Disseny d'aplicacions distribuïdes*. Barcelona: Universitat Oberta de Catalunya, Estudis d'Informàtica i Multimèdia.

Annex A: aplicació FootballTeam

La pàgina d'inici consisteix en un menú principal que permet accedir a les funcionalitats que ofereix l'aplicació FootballTeam:

Figura 74. Menú principal

El botó *Equips* permet gestionar els equips de futbol:

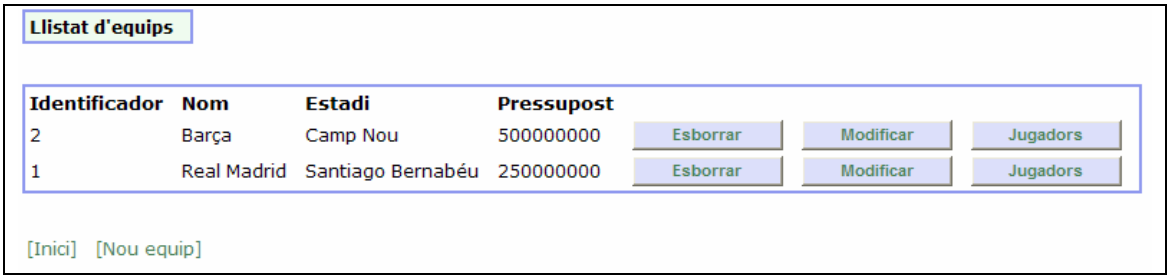

Figura 75. Gestió dels equips de futbol

Des del llistat d'equips es pot accedir als jugadors que pertanyen a un equip mitjançant el botó *Jugadors*:

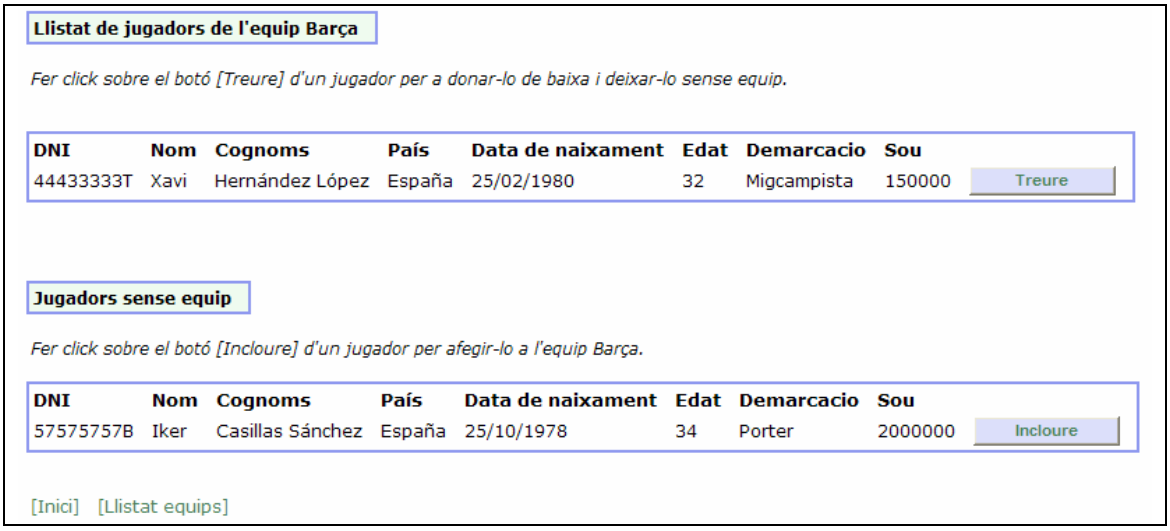

Figura 76. Jugadors d'un equip

Des d'aquesta pantalla es poden incloure o treure els jugadors d'un equip mitjançant els botons *Incloure* o *Treure*.

El botó *Jugadors* del menú principal permet gestionar els jugadors:

Llistat de jugadors

DNI	Nom	Cognoms	País	Data de naixament	Edat	Demarcació	Sou (euros/any)			
57575757B	Iker	Casillas Sánchez	España	25/10/1978	34	Porter	2000000	Esborrar	Modificar	Entrenaments
44433333T	Xavi	Hernández López	España	25/02/1980	32	Migcampista	150000	Esborrar	Modificar	Entrenaments

[Inici] [Nou jugador]

Figura 77. Gestió dels jugadors

Des d'aquest llistat es pot accedir als entrenaments que està seguint un jugador mitjançant el botó *Entrenaments*:

Llistat d'entrenaments del jugador Xavi Hernández López

Fer click sobre el botó [Finalitzar] per a finalitzar l'entrenament del jugador.

Identificador	Nom	Descripció	Sessions	Durada	
1	Xuts a porta	Fer xuts a porta buida però amb l'objectiu de ficar la pilota dins de la porteria.	3	00 h i 15 m	Finalitzar
3	Control pilota	Fer controls amb la pilota.	5	01 h i 30 m	Finalitzar

Entrenaments que no està seguint el jugador Xavi Hernández López

Fer click sobre el botó [Iniciar] perquè el jugador inicii l'entrenament.

Identificador	Nom	Descripció	Sessions	Durada	
2	Carrera contínua	Fer carreres contínues	1	01 h i 30 m	Iniciar

[Inici] [Llistat jugadors]

Figura 78. Entrenaments d'un jugador

Des d'aquesta pantalla es pot iniciar o finalitzar un entrenament per a un jugador en concret. Per a fer-ho s'utilitzen els botons *Iniciar* o *Finalitzar*.

I el botó *Entrenaments* del menú principal permet gestionar els entrenaments:

Llistat d'entrenaments

Identificador	Nom	Número sessions	Duració				
2	Carrera contínua	1	01 h i 30 m	Detall	Esborrar	Modificar	Jugadors
3	Control pilota	5	01 h i 30 m	Detall	Esborrar	Modificar	Jugadors
1	Xuts a porta	3	00 h i 15 m	Detall	Esborrar	Modificar	Jugadors

[Inici] [Nou entrenament]

Figura 79. Gestió dels entrenaments

Des d'aquest llistat també es poden incloure jugadors a l'entrenament mitjançant el botó *Jugadors*. El botó *Detall* permet visualitzar tota la informació d'un entrenament.