# Frameworks de persistència: un estudi comparatiu

**Manuel Calvo Pons**
Enginyeria en Informàtica

**Consultor : Javier Ferró Garcia**

15 de gener de 2007

## ÍNDEX DE LA MEMÒRIA

# 1. INTRODUCCIÓ

## 1. DESCRIPCIÓ DEL PROJECTE

Aquest projecte s'engloba dins la tecnologia Java 2 Enterprise Edition (J2EE), que és una plataforma oberta i estàndard per desenvolupar i desplegar aplicacions empresarials multicapa amb n-nivells, distribuïdes i basades en components. És una tecnologia molt de moda i utilitzada en aquests moments ja que aprofita els avenços en comunicacions per aconseguir programari que es pugui executar descentralitzat i amb plataformes heterogènies.

Dins aquesta tecnologia, el problema de la persistència és un tema al que s'ha dedicat importants esforços per intentar minimitzar la no correspondència entre els objectes del paradigma de la programació orientada a objectes i els registres d'una taula dins un sistema de gestió de bases de dades. Aquest projecte pretén ser un estudi dels diversos frameworks de persistència que pretenen ajudar al desenvolupador a aconseguir la perdurabilitat de les dades.

## 2. RESUM

El projecte és un estudi dels distints productes que es poden trobar per aconseguir la persistència dels objectes entre diferents sessions. Primerament s'explica l'arquitectura típica que es sol trobar dins les aplicacions distribuïdes (en components) de tres capes. Dins la capa intermèdia trobem la capa d'integració, que és la que s'encarrega de guardar els objectes en un medi perdurable (bases de dades, per exemple). Així centrem l'objectiu de l'estudi: aprofundir en el coneixement dels diversos productes que ofereixen persistència. Seguidament es presenten les propietats que han de tenir un framework de persistència i es fa una petita introducció a una sèrie de productes que es poden considerar com proveïdors de persistència. Es continua amb un anàlisi de les propietats que es poden trobar en un framework de persistència, tant les propietats necessàries com ajudes al desenvolupador de l'aplicació. S'han considerat dues característiques bàsiques per l'elecció de dos productes per realitzar un estudi en profunditat, el tipus de codi i si compleix amb algun estàndard. D'aquí s'han obtingut els dos productes (TopLink

Essentials i JPOX) que s'han considerat més interessats per l'objectiu del projecte. S'ha presentat un petit cas d'estudi (programari de manteniment de piscines) per poder estudiar les propietats (construcció d'esquema, transaccionalitat, transparència, rendiment, mecanismes de consulta i cardinalitat) d'una manera pràctica i presentar així uns resultats quantitatius i qualitatius per ajudar a l'elecció per part dels programadors del framework de persistència més adient a les seves necessitats. Per finalitzar, es presenten les conclusions del projecte: la persistència d'objectes és una àrea prou desenvolupada i presenta eines prou correctes per considerar-la un problema suficientment resolt.

S'ha afegit un apartat sobre un llenguatge de consulta orientat a objectes, JDOQL que és el llenguatge que presenta l'especificació JDO (i podem trobar al producte JPOX). La descoberta d'aquest tipus de llenguatge a les proves realitzades prèviament ens ha fet considerar necessari l'estudi i comparació més profund d'aquests llenguatges, que podem trobar a quasi tots els frameworks de persistència.

## 2. ÍNDEX DE FIGURES

# 3. FRAMEWORKS DE PERSISTÈNCIA: UN ESTUDI COMPARATIU

## 1. INTRODUCCIÓ

La programació orientada a components proporciona mecanismes apropiats per al desenvolupament i la implementació de components de programari distribuïts, que aprofiten les avantatges dels avenços en comunicacions per aconseguir programari que es pugui executar descentralitzat i amb plataformes heterogènies, amb un alt grau de reutilització i amb molta llibertat de moviments en relació a l'aprofitament del programari.

Dins aquest paradigma, la tecnologia Java 2 Enterprise Edition (J2EE) és una plataforma oberta i estàndard per desenvolupar i desplegar aplicacions empresarials multicapa amb n-nivells, distribuïdes i basades en components.

Una aplicació J2EE es divideix en tres capes d'alt nivell, que normalment es subdivideixen en 5 capes lògiques.

- capa client
- capa intermèdia
  - capa de presentació
  - capa de negoci
  - capa d'integració
- capa EIS

La capa d'integració s'encarrega d'accedir i tractar les dades que poden estar emmagatzemades en fonts molt heterogènies. A la plataforma J2EE hi ha diverses aproximacions possibles per representar les entitats (dades):

- POJO, que són classes java que utilitzen alguna tècnica per accedir a les dades
- EJB d'entitat, són components EJB que ens permeten mostrar i treballar com objectes les dades que tinguem a les bases de dades relacionals
- Frameworks de persistència, són sistemes que automatitzen les operacions CRUD (creació, lectura, actualització i esborrat) d'entitats a les bases de dades relacionals

# 1. DESCRIPCIÓ DEL PROJECTE

Aquest projecte final de carrera vesarà sobre l'estudi d'aquesta darrera aproximació, els frameworks de persistència, que solen oferir una eina completa de mapeig objecte / relacional.

La necessitat d'existència d'aquest frameworks la trobem en que els EJB d'entitat (EJB creats precisament per gestionar la persistència) han rebut moltes crítiques al llarg de les seves versions: al principi no es podien implementar relacions entre EJB d'entitat, oferien un rendiment molt pobre, el CMP tenia grans deficiències.

Així, han anat apareixent diferents aproximacions a fer més rendible la persistència dins les aplicacions orientades a components. Les avantatges que podem trobar emprant els frameworks de persistència són facilitat d'ús, fa transparent la persistència, moltes optimitzacions de rendiment, generació automàtica de les classes Java que representen les entitats i les relacions a partir de fitxers de definició. La desavantatge és que no és una tecnologia estàndards. Amb la nova especificació d'EJB 3.0 sembla que es camina cap a l'estandarditzación de les eines de mapeig objecte /relacional.

# 2. OBJECTIUS GENERALS I ESPECÍFICS

Amb aquest projecte es pretén aprofundir en el coneixement dels frameworks de persistència. Donat que hi ha on elegir (Hibernate, Castor, Jakarta OJB, TopLink, Expresso, JPOX, Ibatis,...), després de l'estudi de les característiques més rellevants d'uns quants dels frameworks coneguts, ens centrarem en un estudi comparatiu de dos, tant en la vesant teòrica com en la pràctica.

Així, l'objectiu general és bàsicament que són i com funcionen el frameworks de persistència.

Després, al centrar-nos en dos exemples de frameworks concrets, podrem aprofundir en la part pràctica, realitzant un petit projecte que implementi les tasques bàsiques de persistència (CRUD) d'accés a les dades d'una base de dades, per poder comparar i valorar les avantatges i inconvenients. Dins aquest estudi més exhaustiu es valorarà la mancança d'alguna operació o la seva millora per poder aportar alguna cosa nova dins el tema.

# 3. PLANIFICACIÓ DEL PROJECTE

Al diagrama 3.1 podem veure una temporalització del projecte. Aquesta temporalització pot variar segons es plantegin les necessitats. Així, si es veu que hi ha temps per l'estudi de qualque framework més, s'ampliarà aquest

nombre de frameworks estudiats. També els frameworks estudiats poden variar depenent de la informació a la que es tingui accés.

Id
Nombre de tarea
1
PFC - Frameworks de persistència
2
Estudi frameworks existents
3
Hibernate
4
TopLink
5
Jakarta OJB
6
Altre framework
7
Decisió framework a desenvolupar
8
Desenvolupament programari 1er framework
9
Desenvolupament programari 2on framework
10
Estudi millores framework de persistència
11
Redactament memòria
12
Entrega PFC
01/10
08/10
15/10
22/10
29/10
05/11
12/11
19/11
26/11
03/12
10/12
17/12
24/12
31/12
07/01

Diagrama 3.1

## 4. PRODUCTES OBTINGUTS

El primer que presentem és una petita presentació d'una sèrie d'eines que proporcionen persistència, amb un quadre on es relacionen les propietats que han de tenir un bon framework de persistència. Propietats tan variades com si el producte proporciona arquitectura de cache, permet la construcció de consultes dinàmicament, consultes desades a memòria, generació automàtica de l'esquema de la base de dades, quines cardinalitats suporta, tipus de codi o si suporta qualque estàndard de persistència ens permet elegir dins la gran varietat de productes que trobem un parell d'exemples representatius per realitzar un estudi en més profunditat.

Seguidament es presenta un exemple on es convenient la utilització d'aquestes eines i es realitza un petit programari per realitzar les proves de comparació que ens proporcionaran els resultats de transaccionalitat, construcció d'esquema, transparència, rendiment, mecanismes de consulta complexos i cardinalitat suportada. Aquestes proves que porten implícit l'estudi d'ús dels dos frameworks de persistència escollits són la base per les conclusions a les que s'arriba sobre l'estat actual de l'estudi i solucions existents del problema de la persistència dins l'arquitectura orientada a components de Java Enterprise Edition.

L'estudi del framework de persistència JPOX ens ha introduït en el tema dels llenguatges de consulta orientats a objectes. El descobriment de JDOQL ens ha portat a afegir als objectius primers un estudi d'aquest llenguatge, ja que afecta a les conclusions del projecte. Així podem trobar al joc de proves un exemple d'ús d'aquest llenguatge.

## 5. DESCRIPCIÓ DE LA MEMÒRIA DEL PROJECTE

El que podem trobar en els següents apartats són els resultats del desenvolupament de l'estudi d'una sèrie de frameworks de persistència (no tots, ja que la llista és molt ampla).

El primer que s'obté és una breu descripció dels frameworks considerats, amb els trets més rellevants segons la informació que ens dóna el fabricant del producte o la comunitat de desenvolupadors que ha programat el producte (vull fer notar que hi ha una majoria de productes desenvolupats que són de codi lliure i desenvolupats sense ànim de negoci, açò ens pot donar idea de la importància que es dona a aquest tema pels propis programadors) i els llocs oficials d'on s'ha extret la informació.

Seguidament es passa a analitzar totes les propietats que ha de tenir un framework de persistència. Aquestes propietats s'han extret de diversos estudis realitzats per diferents autors (consulteu bibliografia).

Per acabar el tema d'anàlisi, es fa un resum dels criteris de comparació més importants considerats per realitzar l'eleció de dos frameworks als que es dedicarà el següent apartat. Es presenta també una taula amb un resum de les propietats de cada producte considerat.

Després d'obtenir els dos frameworks considerats com més rellevants per a l'estudi, es presenta la fase d'implementació de la memòria, amb un estudi de sis criteris de comparació (construcció d'esquema, transaccionalitat, transparència, rendiment, mecanismes de consulta i cardinalitat) que ens poden donar resultats quantitatius per valorar els avantatges i inconvenients dels productes estudiats. Aquest estudi presenta l'exemple d'una petita aplicació d'una empresa dedicada al manteniment de piscines en la seva part de persistir els clients (piscines) i les medicions de químics de l'aigua que es realitza periòdicament. Amb aquest exemple s'ha desenvolupat el estudi dels dos frameworks, presentant també els trets més rellevants del codi que hi ha programar, així com dels fitxers que es necessiten per aconseguir la persistència.

Per acabar, es presenten les conclusions a que s'han arribat amb l'estudi dels resultats.

S'ha afegit un apartat (no inclòs als objectius primers) sobre el llenguatge de consulta JDOQL, ja que ens ha semblat un estudi necessari per poder arribar a conclusions més reals sobre les eines considerades.

# 1. PROPIETATS DELS FRAMEWORKS DE PERSISTÈNCIA

Com s'ha explicat a la primera part d'aquest document, la integració de les tecnologies de programació orientada a objectes i bases de dades relacionals ens porta al problema conegut com Object – Relational Impedance Mismatch, ja que no es pot relacionar directament tots els objectes i les taules d'una base de dades (per exemple, identitat, classe, herència i polimorfisme no són suportades de forma primitiva per una base de dades relacional).

Aquest problema ha conduït a l'estudi de millorar la relació entre aquests dos paradigmes, que són de les tecnologies més emprades en aquest moments. D'aquests estudis van sorgir les solucions automatitzades que reben el nom de mapejadors objecte – relacional (frameworks de persistència).

Aquest mecanisme es basa en l'ús d'un component que s'encarrega del mapeig d'objectes de negoci a un model relacional i en sentit contrari. Aquest component sol ser unes metadades que expliquen com equivalen un component amb l'altre dins de cada paradigma.

Avantatges:

- L'ús d'un mecanisme d'aquest tipus permet reduir la quantitat de codi necessari (desenvolupat pel desenvolupador, s'entén)
- facilita el manteniment
- permet que el desenvolupador pensi tot son en objectes.

Inconvenients:

- s'ha d'afegir i mantenir les metadades
- es perd el control de com es realitza la persistència
- es dificulta el canvi de consultes, ja que s'ha de saber com funciona el mapeig
- hi ha consultes complexes en les que no es pot optar per la solució programada
- al no haver un estàndard dominant, l'ús d'un framework en concret et lliga

En els darrers anys han aparegut al manco dos intents d'estandardització que intenten arrengar aquest darrer problema:

- JPA (Java Persistence API) és l'API de persistència desenvolupada per la plataforma JavaEE i inclosa dins l'estàndard EJB3. L'objectiu és no perdre les avantatges de l'orientació a objectes al interactuar amb una base de dades, com passava amb EJB2, i permetre emprar objectes regulars (coneguts com Bojos)
- JDO (Java Data Objectes) és una especificació de persistència d'objectes Java que presenta com característica més rellevant la transparència dels serveis de persistència al model de domini. També empra Bojos i sorgeix de l'estàndard general per a la persistència ODMG concretat al llenguatge Java. La darrera especificació llançada és la JDO 2.0 encara que l'epacta JDO Project ja està desenvolupant la versió 3.0

## 1. PRODUCTES CONSIDERATS

Com ja s'ha comentat, mecanismes per aconseguir la persistència en podem trobar molts i de moltes castes. Seguidament, presentarem una llista amb una part d'aquests productes, amb una petita descripció.

- Hibernate

És una eina ORM Open Coure que ha quedat com un dels mapejadors objecte / relacional més emprats. Tal és cert això que el nou estàndard implementat d'EJB 3.0 s'aproxima molt a el que ofereix Hibernate. Suporta quasi totes les bases de dades relacionals ja que cada dialecte SQL te associada una classe que la implementa, a més es pot crear fàcilment el seu dialecte, extenent un existent directament o extenent la classe abstracta net.sf.hibernate.dialect.Dialect. A la figura 3.2 es presenta una sèrie de mecanismes i APIs que empra per poder realitzar les tasques.

Lloc oficial: http://www.hibernate.org

| | |
|---|---|
| Reflexió (Java) | Permet descobrir informació sobre els atributs, els mètodes i els constructors |
| POJO (Java) | Objectes normals de java |
| JDBC | Connexió amb la base de dades |
| JTA | Gestionar el suport de transaccions |
| JNDI | Consultar el servei de directori dels datasource d'accés a la base de dades |
| XML | Per especificar el mapeig i les propietats de les taules |
| Dom4j | Per realitzar l'anàlisi dels fitxers xml |
| Collections, Lang i Logging | Per ampliar la funcionalitat dels components que ens dóna el JDK |

Diagrama 3.2

- Ibatis

Ibatis és una eina de codi obert que no és un ORM pròpiament i s'empra principalment per les aplicacions on el Model de Dades ja està creat i no està normalitzat.

No és totalment transparent, ja que el programador manipula SQL. Empra un descriptor XML per ajuntar els objectes amb les sentències SQL.

La seva simplicitat així com la gran tolerància a errades de cast són les majors avantatges d'Ibaits.

Es divideix en dos elements: SQL Maps proporciona una manera senzilla de moure les dades entre els objectes Java i la base de dades relacional, obtenint tota la potència del SQL sense escriure una retxa de codi JDBC; i Ibatis Data Access Objectes (DAO) és la implementació del patró DAO, una capa d'abstracció que oculta els detalls de la capa de persistència i proporciona un API únic per a totes les aplicacions.

Ibatis suporta els cursors d'oracle, mapping de resultSets, suport per les propietats privades dels beans, afegeix un log a les operacions de connexió a la BD i pot treballar amb Spring, Log4j i OsCache.

Lloc oficial: http://ibatis.apache.org/

- TopLink Essentials

És un framework de persistència desenvolupat per Oracle però que darrerament s'ha alliberat el codi. Implementa l'especificació EJB3 JPA, sent el referent d'aquesta especificació. Els mapejos es poden definir amb fitxers xml o amb annotations de Java.

TopLink empra el seu propi llenguatge d'expressions per realitzar les consultes. Les expressions són molt potents, suportant conceptes relacionals avançats com GROUP BY, encara que també es pot emprar SQL si es desitja. També suporta CMP, BMP o una barreja de tot dos, donant flexibilitat a l'eina.

TopLink crea un conjunt de metadades, els descriptors o mapes que defineixen com els objectes seran guardats a la base de dades relacional. Aquests descriptors permeten generar en temps d'execució les declaracions d'SQL dinàmicament i permeten canviar l'esquema de la base de dades sense fer canvis dins la lògica de negoci. Aquests descriptors són independents del dialecte SQL i de la base de dades.

Lloc oficial: http://www.oracle.com/technology/products/ias/toplink/jpa/index.html

- Castor JDO

Framework de persistència open source, molt senzill de configurar. El llenguatge de consulta està basat en JDO, encara que no compleix la especificació JDO.

El sistema es configura amb XML, així com la definició dels mapejos entre objectes i taules.

L'accés al sistema per a l'execució de consultes i comandes SQL es fa per mitjà de la classe Database, i aquesta s'accedeix emprant mètodes estàtics de la classe JDOManager.

Es pot emprar tant per aplicacions client com per J2EE. Per aquest darrer, Castor JDO és un ORM escrit 100% en Java. Permet la creació de Java Data Object sobre esquemes definits per l'usuari, així com també dóna als JDO la capa de persistència.

Lloc oficial: http://www.castor.org/jdo.html

- Cayenne

És un framework de persistència de codi obert amb llicència Apache que proporciona mapeig objecte – relacional i serveis remots. Relaciona una o més esquemes de bases de dades directament a objectes Java, manipulant simples commit i rollbacks, generació d'SQL, unions, seqüències,…

Cayenne està pensat per ser senzill d'emprar sense sacrificar flexibilitat i disseny. Suporta enginyeria inversa i generació de bases de dades. Aquestes funcions són controlades directament per mitjà de CayenneModeler, una completa i funcional eina gràfica. A diferència de molts altres framework

estudiats, no hi ha que definir l'esquema de la base de dades amb XML, sinó que directament mapeja aquesta a objectes Java.

Cayenne te altres característiques importants, com caching, una completa sintaxis de consulta, cerca de relacions, objectes baix demanda o falta de relacions, herència d'objectes, auto – detecció de base de dades i objectes persistents genèrics.

Darrerament, Cayenne està en procés de complir l'estàndard JPA.

Lloc oficial: http://cayenne.apache.org/

- CocoBase

És un motor de persistència comercial fàcil d'emprar que proporciona eines per mantenir la relació objecte – registre per mitjà d'un mapeig de la relació. Suporta, a més, Dynamic POJO Data Persistence, Dynamic Transparent Persistence, EJB, així com aplicacions J2SE (standalone). La darrera versió enllestida (5.1) suporta l'especificació JPA (especificació de l'API de Java EE per a la persistència que trobem dins l'estàndard EJB3).

CocoBase agafa la informació de la base de dades necessària per generar l'arxiu de mapeig, a més genera SQL dinàmic per emmagatzemar, consultar, eliminar i modificar les dades dels objectes que trobem a la base de dades.

Ofereix eines que fan més fàcil la creació de mapejos quan el model d'objecte es diferencia molt del model de dades, però si aquest objectes són POJO l'eina perd la seva utilitat.

Lloc oficial: http://www.thoughtinc.com/

- Form

Mapejador objecte – relacional de codi propietari que ofereix una gran flexibilitat i molta facilitat d'ús. Encara que Form no posa cap restricció particular a l'arquitectura de l'aplicació, es va crear pensant en les aplicacions construïdes amb tres capes (presentació, negoci i dades).

Encara que no implementa completament l'especificació JDO, Form tendeix cap a aquesta especificació, ja que l'objecte persistent ha d'implementar una interfície, te un generador de codi i altres capacitats que especifiquen JDO.

Per configurar-lo, n'hi ha proa amb descriure com el domini dels objectes es tradueixen a l'esquema de la base de dades. El mapeig es pot construir tenint les taules i els objectes o l'arxiu de mapeig pot construir les taules.

Form treballa amb una sèrie de classes que realitzen les funciones necessàries per realitzar la persistència (connexió a la base de dades, mapejos, manipulació dels registres, consultes) que fa que no sigui aconsellable per petites aplicacions, ja que la seva arquitectura és invasiva.

Lloc oficial: http://www.chimu.com/projects/form/index.html

- JGrinder

És un framework propietari i una eina de desenvolupament que fa fàcil integrar les aplicacions Java i les dades de negoci.

Permet definir els objectes de mapeig i les seves relacions amb un entorn gràfic. Les principals capacitats de JGrinder són:

- servei transaccional: proporciona serveis de persistència basats en transaccions
- facilitat d'ús: tota la informació emprada pel framework es genera, evitant falta de sincronització amb el model o errades de mapeig
- flexibilitat: permet treballar amb col·leccions com Vectors, Hashtables i sèries natives
- escalabilitat: els serveis de transaccions d'objecte es construeixen per determinar tots els camps que s'han modificat, permitint un alt grau de concurrència i actuació si comparem amb SQL
- senzilla recuperació de l'objecte: la recuperació de les dades necessàries es fa d'una manera transparent al programador
- consultes genèriques: permet la creació de consultes dins un magatzem i aquestes consultes són consistents amb els canvis de diferents fabricants de bases de dades
- integració: pot treballar amb qualsevol SGBD i per quasi tots els drivers JDBC
- compatibilitat: es pot emprar tant per noves bases de dades com per esquemes ja existents
- suport de diferents arquitectures: suporta tant l'arquitectura de tres o dos capes com l'arquitectura client – servidor amb client pesant

JGrinder proporciona una solució viable per RDBMS i pot extendre fàcilment el framework per incloure recolzament a ODBMS.

Darrerament ha migrat a una eina Open Source Code que compleix l'estàndard JDO.

Lloc oficial: http://jgrinder.sourceforge.net/

- Kodo

És una eina de codi privat que ofereix accés a les bases de dades relacionals per mitjà dels estàndards JDO i EJB3, Open JPA (Java Persistance API) és una

part de Kodo i és un API de persistència que es dedica a emmagatzemar objectes de bases de dades relacionals a la RAM, que simplifica el desenvolupament i millora la performance.

http://www.solarmetric.com/solarmetric_jump_page

- TJDO

TriActive JDO és una implementació open source de l'especificació JDO de Sun. La versió actual ha estat provada satisfactòriament amb bases de dades tan variades com Cloudscape, DB2, Firebird, MySQL, Oracle, SAP DB i MS SQL Server.

Implementa el llenguatge de consulta de l'especificació JDOQL, i com detalls més significatius pot crear els esquemes necessaris de les bases de dades amb les classes definides, auto - valida els canvis d'estructura en aquests esquemes en temps d'execució i pot mapejar les classes de java a SQL.

http://tjdo.sourceforge.net/

- Torque

Torque és un framework de l'organització Apache que permet accedir i manipular les dades d'una base de dades relacional. Al contrari dels altres frameworks ORM no empra reflexió per accedir a les classes, es descriu l'estructura de la classe per mitjà d'un arxiu XML. Aquest esquema també es pot emprar per genera i executar SQL per crear les taules de la base de dades.

Es pot dividir en quatre parts:

- Runtime: conté tot el necessari per a que l'aplicació pugui accedir a les dades
- Generador: és el framework de construcció ant
- Maven – plugin: crea el mapeig ORM i classes per crear objectes i per accedir a la base de dades, encara que també pot crear mapeig XML i l'SQL per crear les taules
- Templates: generadors d'SQL que es poden modificar i canviar per aconseguir major rendiment o en casos especials

Torque és un framework molt flexible i potent que requereix un coneixement d'altres frameworks d'apache per ser una bona solució.

http://db.apache.org/torque/

- JPOX

Producte de codi obert i referència en la implementació de l'especificació JDO 2 de Sun.

És una implementació completa de l'estàndard amb les següents característiques: suporta la majoria de bases de dades disponibles en el mercat; s'integra amb JEE; funciona amb una cache per augmentar el rendiments; i permet la persistència de col·leccions (Map, List,...)

http://www.jpox.org/

## 2. ANÀLISI DE LES PROPIETATS QUE HA DE TENIR UN FRAMEWORK DE PERSISTÈNCIA

Seguidament es presenten una sèrie de propietats que trobem als diferents frameworks de persistència per poder comparar les prestacions de cadascú. Aquest criteris no són una resposta única al problema de la comparació, però si que és representatiu per l'experiència de proba dels mecanismes. També s'ha de remarcar que l'avaluació d'aquest criteris no dóna una puntuació o llista ordenada de mecanismes de persistència sinó que aproxima una solució a l'elecció de dos frameworks de persistència que donin uns resultats interessants dins la segona part de la comparació, on sí que obtindrem una resposta quantitativa per mitjà del desenvolupament d'un programari de proba amb el que es pot medir les respostes.

- Arquitectura de cache: si el mapejador té la propietat de caching d'objectes s'indica la arquitectura que empra.
- Bulk Data Manipulation: si la aplicació empra el patró Bundled Write, que permet al mapejador enviar conjunts de sentències SQL dins una mateixa operació
- Claus primàries compostes: s'estudia si s'accepta claus primàries compostes per més d'un atribut
- Construcció GROUP BY: si el llenguatge de consulta suporta la agrupació esmentada
- Construcció manual d'SQL: es valora si el desenvolupador ha d'escriure codi SQL
- Constructor per defecte: es refereix a la necessitat o no de que l'entitat tingui constructor
- Consultes dinàmiques: si es pot generar en temps d'execució de consultes
- Consultes desades: si és possible que consultes que es repeteixen es guardin en el mapejador per evitar la creació repetida de la mateixa consulta
- Consultes polimòrfiques: si suporta polimorfisme a nivell de les consultes
- Creació d'esquema: si el mapejador pot creat les sentències SQL per a la creació de les taules i demés
- DBMS suportats: conjunt de bases de dades relacionals que suporta
- Fetching: si l'eina permet especificar com s'obtenen els atributs i les associacions d'un determinat objecte. Generalment, es valora si es recupera el valor quan sigui necessari o quan es realitza la consulta

- Herència: conjunt de mecanisme que te el mapejador per tractar l'herència
- Funcions avançades SQL: es valora si el mapejador pot tractar consultes amb funcions agregades (count, max,...)
- Generació automàtica d'identificador: suport per la generació automàtica de claus primàries tipus identificador
- Llenguatge de consulta: es valora si el mapejador està provist d'un llenguatge de consulta propi orientat a objectes
- Mapeig d'una classe a N taules: si suporta el mapeig a més d'una taula
- Mapeig de N classes a una taula: si es pot realitzar el mapeig d'un conjunt de classes a una taula
- Objectes associats amb Outer Joins: si els objectes relacionats es poden obtenir per mitjà de l'ús de outer joins
- Ortogonalitat: possibilitat de persistir objectes de classes arbitraries sense necessitat de superclasse o interfície específica
- Persistència d'EJBs: si suporta BMP i CMP, o les dues
- Cardinalitat: quin tipus de relacions suporta (1 – 1, 1 – n,...)
- Annotations: si pot aconseguir les metadades per mitjà de les annotations de java
- Cascading: s'avalua si el mapejador permet especificar la propagació d'operacions de persistència
- Detached Objectes: possibilitat d'emprar entitats utilitzades en altres sessions
- Optimistic Locking: si es pot programar el llançament d'una excepció quan es detecti una possible corrupció de dades en els accessos concurrents
- Paginació: si es possible recuperar la totalitat d'entitats en diferents rangs
- Taules extra de la base de dades: si fan falta taules afegides per mantenir els locks, metadades,...
- Tipus personalitzats: si es poden definir nous tipus de manera que puguin ser mapejats a la seva columna de la taula
- Unique: si les consultes suporta la funció SQL unique (o distinct)
- Persistència de camps privats: s'avalua si es permet la persistència de propietats de camps declarats privats
- POJO: si permet la utilització de plain objects java per realitzar la persistència
- Generació de codi: refereix si el mapejador necessita a qualque moment la generació o el processament de codi font
- Llicència: s'estudia el tipus de llicència que té el producte
- Plataformes: llista de plataformes amb les que es pot emprar el producte
- Documentació: estudi sobre la quantitat i qualitat de documentació a la que es te accés
- Aprenentatge: dificultat d'aprenentatge en la utilització del producte
- Eines de recolzament: eines que ajuden a l'ús dels producte considerats
- Estàndard: avaluació de si tenen la certificació de complir amb qualque estàndard o especificació publicada

Aquesta llista de criteris són més un estudi teòric del que ens pot oferir els productes estudiats, ja que no requereixen d'un cas pràctic per la seva comparació, basant-se en les explicacions que proporciona el fabricant.

No tots aquests criteris s'empraran per la comparació de productes, sinó que s'han seleccionat els considerats més importants i adients per la selecció.

# 3. RESULTATS OBTINGUTS

De l'estudi de les propietats que poden tenir els productes considerats es pot extreure la llista mínima de propietats que ha de tenir un framework per aconseguir el seu objectiu de persistir les instàncies dels objectes. Aquestes característiques són:

- Caching: els frameworks de persistència haurien d'oferir opcions de cache per emprar-se freqüentment, per així aconseguir una productivitat major
- Quering: és convenient que el programador pugui dissenyar les consultes, amb un llenguatge de consultes orientat a objectes o, en el seu defecte, amb SQL directament
- Locking: s'ha de permetre la concurrència, però garantitzant la integritat de les dades persistents
- Sequencing: el producte ha de suportar diferents estratègies per manipular la indexació
- Suport a transaccions: s'ha de garantir la integritat de les dades. Una modificació d'un objecte persistent ha de quedar reflexat a la base de dades

Quasi tots els productes considerats tenen aquestes funcions mínimes per ser considerats com un framework de persistència. Un estudi més ampli, fora de l'abast d'aquest projecte, hauria d'aprofundir en l'estudi de cada propietat per aconseguir un quadre comparatiu del que ofereix cada producte.

Les poques diferències rellevants que podem trobar en els productes considerats ens han fet centrar-nos en dos consideracions bàsiques alhora d'escollir la utilització d'un framework de persistència sobre els altres. Aquests dos criteris són:

- tipus de codi, propietat que considerem molt important ja que la possibilitat d'escollir un codi que podem manipular facilita les modificacions per aconseguir un producte a mida dels gusts personals del programador. Així, s'escollirà per l'estudi dos frameworks de codi obert o al manco un codi alliberat no propietari, per poder accedir i manipular aquest sense limitacions.
- Estàndard, quan es comença a emprar un framework de persistència costa bastant canviar-lo. Per aquesta raó s'ha de cercar un producte que estigui recolzat per qualque grup de desenvolupadors o empreses. Així,

és molt convenient l'elecció d'un producte que compleixi amb una de les especificacions que hi ha definides, per evitar quedar-se amb un producte sense actualitzacions (necessàries quan apareixen noves tecnologies) o sense suport.

Tenint en compte el que s'ha exposat a l'apartat anterior, s'han escollit dos productes per realitzar una implementació de l'estudi comparatiu. El primer producte escollit és TopLink Essentials ja que és una de les referències de l´'especificació JPA (part de l'especificació EJB 3.0) i tot i haver estat creat per una empresa comercial (Oracle) s'ha alliberat el codi per poder manipular-lo. El segon producte escollit és JPOX, framework de persistència que implementa l'especificació de Sun JDO (JPOX compleix tant la versió 1 o com la 2) i és un producte desenvolupat per una comunitat de programadors en la que qualsevol pot aportar les seves idees.

## 2. ESTUDI COMPARATIU

Com s'ha explicat a l'apartat anterior, la selecció dels dos productes a estudiar en profunditat s'ha degut per damunt dels demés criteris a ser aplicació i referència dels dos estàndards que podem trobar actualment en el món de la persistència.

- TopLink Essentials compleix JPA, part de l'especificació d'EJB 3.0 que podem trobar a la versió 5 de la plataforma JEE. És una especificació que simplifica la persistència d'objectes Java senzills (Bojos), posant èmfasi en la definició mínima de metadades per aconseguir el mapeig d'objectes a la base de dades relacional. Aquesta especificació és el resultat de la feina d'un grup d'experts que han pres en consideració idees dels frameworks líders del mercat (Hibernate, Oracle TopLink, ...) O sigui, se ha construït des de la pràctica que ja es portava en el desenvolupament de certs productes.

  La intenció que trobem darrera aquesta especificació és que sigui implementat per diferents proveïdors aconseguint així que productes distints ofereixin la mateixa interfície.

  Característiques a tenir en compte d'aquesta especificació són l'ús d'annotations per la definició de les metadades que permet la no definició dels mapejos fins temps d'execució o configuració per omissió (encara que l'especificació permet emprar també XML).

- JPOX és la única implementació de l'estàndard JDO 2.0, estàndard desenvolupat per Sun per aconseguir la persistència en Java (és l'estàndard ODMG per Java). Els objectes persistents JDO són classes normals en Java que no requereixen d'implementar certes interfícies o extensions de classes especials.

Les característiques més rellevants són que la definició dels objectes s'han de posar en metafitxers externs XML i els productes que compleixen aquesta donen recolzament a la modificació de classes compilades per poder fer transparent la persistència per mitjà de enhancers (enhancer no és part de l'estàndard JDO, però és el mecanisme per implementar-ho).

# 1. MECANISMES DE COMPARACIÓ

Després de la presentació i estudi dels criteris de comparació teòrics que ens han portat a l'elecció dels dos frameworks de persistència a estudiar, seguidament es presentaran els criteris pràctics per obtenir els resultats cercats.

- Construcció d'esquema: s'avalua la capacitat de les eines estudiades de construir les taules de la base de dades, segons el mapeig presentat.
- Transaccionalitat: possibilitat d'avortament d'una acció sinó es pot dur a terme d'una manera completa.
- Transparència: s'avaluarà si l'eina necessita preocupar-se de les consultes SQL.
- Rendiment: amb un cas d'estudi concret, presentat al següent apartat, es valorarà quantitativament els temps de resposta a distintes operacions de persistència.
- Mecanismes de consulta: s'avaluarà una sèrie de funcions avançades d'SQL, si són possibles de fer amb les eines estudiades i el seu rendiment (avg, sum, max, min, count, clàusula group by i unique).
- Cardinalitat: seguint el cas d'estudi, s'estudiarà com tracten els dos productes considerats les relacions entre classes i quines relacions suporten.

# 2. CAS D'ESTUDI

Amb l'objectiu de poder valorar quantitativament els dos productes estudiats, es proposa una petita part de l'arquitectura d'un sistema de gestió de manteniment de piscines. La part presentada és la part de persistència, amb un senzill model de dades i el seu equivalent en el món relacional.

L'anàlisi conceptual ens mostra la relació entre l'objecte piscina i les seves medicions de l'estat de l'aigua en el decurs del temps.

L'objecte piscina té un identificador únic (id) que es genera automàticament, i uns atributs adients per poder fer estudis de àrees (direcció, població) o grandària (volum).

L'altra objecte a considerar és la lectura diària dels químics de l'aigua, i que es relacionarà amb l'objecte piscina amb una clan forana (id_piscina). Hem d'afegir

a l'objecte químics una clan primària (id), ja que els frameworks de persistència necessiten d'aquesta clau per poder funcionar. Sinó, bastaria amb la combinació de id_piscina i data per aconseguir l'identificador únic.

En el diagrama 3.3 es presenta el diagrama de classes del model.

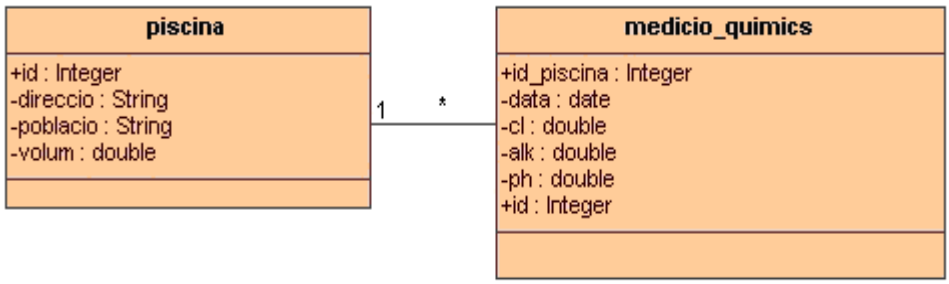


Diagrama 3.3

Aquest model ens permetrà fer consultes de distinta complexitat per l'objecte d'estudi, comparació dels diferents frameworks presentats, ja que permet fer consultes senzilles (dades de lectura de químics a la piscina x el dia y) com consultes més complexes (mitja de la quantitat de cloro que conté l'aigua de la piscina z).

També permet estudiar les distintes relacions entre objectes: cardinalitat 1 - 1, 1 - n i n – m (per exemple, seria interessant per l'empresa saber l'estat de les piscines d'una població determinada).

La capa de persistència ens permet definir i implementar la correspondència entre el les dades que hem de guardar de forma persistent i el codi Java que ha d'interactuar amb les mateixes.

El primer que s'ha de decidir és quina informació de la nostra aplicació s'ha de fer persistent i després estudiar com es guardarà aquesta informació a la base de dades relacional.

En el cas d'estudi concret aquestes preguntes són senzilles de respondre. Tota la informació presentada (objectes piscina i medicions) han de quedar guardats i aquesta persistència correspon a dues taules amb les columnes que coincideixen amb els atributs corresponents. Al diagrama d'entitat relació 3.4 podem veure aquesta correspondència

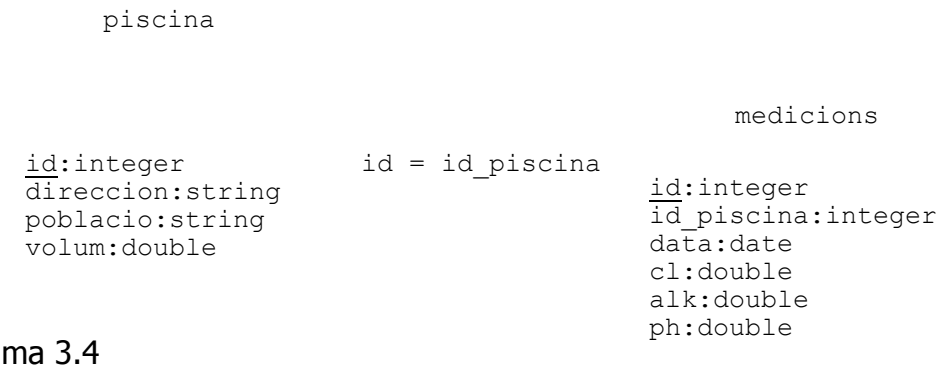


Diagrama 3.4

La configuració de hardware i software on es faran les probes es presenta en el diagrama 3.5.

| | |
|---|---|
| Processador | Intel Pentium a 2.4 GHz |
| Memòria RAM | 1GB |
| Sistema Operatiu | Windows XP Profesional |
| Java | 1.6.0_02 |
| Gestor de base de dades | MySQL 5.0 |
| Driver JDBC | Mysql-connector-java-5.0.6 |

Diagrama 3.5

## 3. COMPARACIÓ DE PRODUCTES: PARÀMETRES A TENIR EN COMPTE

Dels dos productes estudiats, seguidament s'explicarà les necessitats i paràmetres necessaris per la seva utilització.

- **TopLink Essentials (JPA)**

| | |
|---|---|
| Versió | 2.41 |
| Codi | Codi obert |
| Lloc oficial | http://www.oracle.om/technology/products/ias/toplink/jpa.index.html |
| Estàndard | JPA |

Diagrama 3.6

TopLink Essentials és la referència de l'especificació JPA i part del projecte de servidor d'aplicacions Glassfish d'Oracle.

Una consideració importants de l'especificació JPA és que els objectes que s'han de persistir són POJOs, que implica que no s'ha de tenir cap consideració especial sobre els mateixos i no fa falta afegir codi si es tracta una aplicació ja existent.

Per aconseguir el mapeig no es fa necessari (encara que és possible) definir aquest amb un fitxer especial. L'especificació empra els annotations de Java per

aconseguir la relació entre els camps de la basa de dades i els objectes en Java. Al diagrama 3.7 podem veure un exemple, amb l'entitat presentada Piscina.

```
....
@Entity
public class Piscina {
@Id
@GeneratedValue
private Long id;
....
```

Diagrama 3.7

En temps d'execució, el programa llegeix les annotations i crea el mapeig objecte – relacional per aconseguir la persistència.

Les especificacions de la connexió s'han d'introduir en el fitxer persistence.xml. Aquí s'especifica a quina base de dades ens connectarem, amb quina connexió i amb quin usuari. Un exemple amb les propietats bàsiques el podem veure al diagrama 3.8. Hi ha altres possibilitats de configuració que no es tracten en aquest projecte.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
        <persistence-unit name="PiscinaServices" transaction-type="RESOURCE_LOCAL">
                <class>mcp.Piscina</class>
                        <properties>
                                <property                                name="toplink.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
                                <property                                   name="toplink.jdbc.url"
value="jdbc:mysql://localhost:3306/mcp_aqua"/>
                                <property name="toplink.jdbc.user" value="*****"/>
                                <property name="toplink.jdbc.password" value="*****"/>
                        </properties>
        </persistence-unit>
</persistence>
```

Diagrama 3.8

L'especificació JPA empra el paquet javax.persistence per aconseguir la persistència. Hi ha que crear un EntityManager i cridar als mètodes necessaris per realitzar la unió entre objecte i base de dades (persist, remove, find,...).

- **JPOX (JDO)**

| | |
|---|---|
| Versió | 1.1.9 |
| Codi | Codi obert |
| Lloc oficial | http://www.jpox.org |
| Estàndard | JDO |

Diagrama 3.9

JPOX és el framework de referència de les especificacions JDO1 i JDO2, desenvolupades per Sun, que són les especificacions concretes per Java de l'ODMG (especificació general per a la persistència).

JDO vol ser l'estàndard per interactuar amb els sistemes de gestió de dades donant una visió d'aquestes consistent i uniforme constituïda per Bojos (objectes Java). JDO resumeix amb unes interfícies de programació els serveis necessaris per aconseguir la persistència d'objectes Java amb qualsevol sistema de gestió de bases de dades sense cap necessitat de conèixer els mecanismes de persistència emprats o sigui, d'una manera transparent.

Les característiques del mapeig del POJO Piscina.java a la taula piscina s'han de detallar al fitxer package.jdo. Aquí s'ha d'especificar com a mínim com es referència l'id de la classe (identity-type) i quins atributs de l'objecte es corresponen amb els camps de la taula i de quin tipus són. Al diagrama 3.8 es pot veure un exemple d'aquest fitxer.

Dins el fitxer jdo.properties, s'ha de posar la configuració d'accés a la base de dades especificant quin sistema de base de dades s'empra, quin driver de connexió i l'usuari i el password. Un exemple de configuració es pot veure al diagrama 3.11.

Al diagrama 3.10 es poden veure els trets bàsic per aconseguir la persistència. El fitxer no presenta diferències amb l'estudiat amb el TopLink o a altres frameworks.

L'especificació JDO fa necessari també la creació de la classe DAO.java, que implementa les transaccions per mitjà de l'API jdo.PersistenceManager, així com la classe PersistenceManagerHelper.java pel tractament d'excepcions. El patró DAO s'aplica per tenir un únic accés als distints sistemes de gestió de bases de dades.

A l'aplicació (que tant pot ser JSE com JEE) s'aconsegueix la persistència cridant als mètodes de la classe DAO creats. Aquest crea una instància de PersistenceManager que realitza la transacció. Al diagrama 3.12 es pot veure un exemple amb les instruccions més rellevants.

```
<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC
   "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 2.0//EN"
   "http://java.sun.com/dtd/jdo_2_0.dtd">
<jdo>
   <package name="mcp">
      <class name="Piscina" identity-type="application">
         <inheritance strategy="new-table"/>
                        <field name="id" primary-key="true" value-strategy= "autoassign"/>
         <field name="direccion" persistence-modifier="persistent">
            <column length="100" jdbc-type="VARCHAR"/>
         </field>
         <field name="poblacio" persistence-modifier="persistent">
            <column length="255" jdbc-type="VARCHAR"/>
         </field>
         <field name="volum" persistence-modifier="persistent"/>
      </class>
   </package>
</jdo>
```
Diagrama 3.10

```
javax.jdo.PersistenceManagerFactoryClass=org.jpox.PersistenceManagerFactoryImpl
javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
javax.jdo.option.ConnectionURL=jdbc:mysql://localhost:3306/mcp_aqua
javax.jdo.option.ConnectionUserName=*****
javax.jdo.option.ConnectionPassword=*****
```
Diagrama 3.11

També és molt important remarcar ja que afectarà al temps d'execució, que JPOX crea al manco dos fitxers de logs. Aquest fitxers són molt útils pel seguiment de les transaccions realitzades, però encareixen el cost de temps.

```
....
static void save(Piscina piscina) {
   PersistenceManager pm = PersistenceManagerHelper.getPersistenceManager();
   try {
      Transaction tx = pm.currentTransaction();
      tx.begin();
      pm.makePersistent(piscina);
      tx.commit();
.....
```
Diagrama 3.12

## 4. COMPARACIÓ DE PRODUCTES: CONSTRUCCIÓ D'ESQUEMA

És una propietat que trobem present en els dos frameworks estudiats.

TopLink permet la possibilitat de creació de les taules amb els objectes a persistir si posem a l'arxiu de configuració persistence.xml la propierty toplink.ddl-generation al valor create-tables. Si s'intenta guardar un objecte dins una taula que no ha estat creada o es demana llistar els objectes de la mateixa taula no existent, llança l'ordre SQL corresponent per crear-la (si no s'ha posat l'annotation @GeneratedValue a l'objecte POJO a persistir, la taula es crea igualment, però alhora d'inserir el nou objecte persistent es llança una excepció ja que no es troba l'identificador únic). Per un POJO com l'exemple del diagrama 3.7 es crea la taula presentada al diagrama 3.13, amb els camps que s'han creat segons els tipus dels mètodes set i get de l'objecte Java. Malauradament, un punt en contra és que si tenim la propietat toplink.ddl-generation al valor create-tables i la taula ja existeix es llançarà una excepció. Per tant, és molt més recomanable crear les taules amb els objectes a persistir directament a la base de dades.

| **FIELD** | **TYPE** | **NULL** | **KEY** |
|---|---|---|---|
| ID | Bigint(20) | NO | PRI |
| DIRECCION | Varchar(255) | YES | |
| POBLACIO | Varchar(255) | YES | |
| VOLUM | double | YES | |

Diagrama 3.13

JPOX permet també la possibilitat de creació de l'esquema de les taules. Per permitir-ho, basta amb posar a true el valor de l'etiqueta org.jpox.autoCreateSchema del fitxer de configuració jpox.properties. Així, si intentem llistar els registres de una taula Piscina que no existeix o inserir un nou objecte persistent, es crea aquesta taula i la taula de control jpox_tables. La taula creada conté els camps que s'han definit al fitxer de mapeig package.jdo, amb els tipus corresponents. A l'exemple que es pot veure del diagrama 3.10, crea la taula piscina amb els camps següents

| **FIELD** | **TYPE** | **NULL** | **KEY** |
|---|---|---|---|
| ID | Int(11) | NO | PRI |
| DIRECCION | Varchar(100) | YES | |
| POBLACIO | Varchar(255) | YES | |
| VOLUM | double | YES | |

Diagrama 3.14

## 5. COMPARACIÓ DE PRODUCTES: TRANSACCIONALITAT

La transaccionalitat (recordem que és la propietat per la qual una interacció amb una base de dades és realitza completament o no es realitza res) és una altra propietat que ofereixen les dues eines estudiades d'una manera explícita.

A TopLink una transacció es realitza entre els mètodes d'un EntiyManager getTransaction().begin() i getTransaction().commit(). Si aquesta darrera

sentència no s'executa, la persistència no queda reflexada (per exemple un insert o un delete).

Amb JPOX la transaccionalitat s'aconsegueix, com es pot veure al diagrama 3.12, també per mitjà de l'ordre commit, ja que si aquesta ordre no s'executa no s'executa la inserció o esborrament del registre. Aquest mètode pertany a la classe Transaction del paquet javax.jdo.

## 6. COMPARACIÓ DE PRODUCTES: TRANSPÀRENCIA

Quan es ralla de transparència ens referim a una mida quantitativa inversament proporcional a l'esforç de realitzar una tasca. O sigui, quan més temps dediquem a aconseguir la persistència, menys transparència hi haurà.

Per mesurar la transparència en els dos productes estudiats ens considerarà la quantitat de línees de codi mínimes (considerant tant el codi del llenguatge de programació com l'SQL com l'XML) a escriure per aconseguir les tasques de persistència.

Aquesta solució es pot considerar una solució mínima. S'hauria de fer un estudi més profund sobre la necessitat del programador de conèixer les diferents tecnologies que s'empren per aconseguir la persistència. A més, quan més transparència trobem, més rigidesa del model hem de tenir. Per exemple, si en un producte s'ha d'emprar SQL (menys transparència) es guanya en potencionalitat ja que es troba més llibertat alhora de realitzar els distints accessos a la base de dades.

| **TopLink** | |
|---|---|
| **TIPUS** | **QUANTITAT** |
| POJO | 20 |
| Codi Mètode Insert | 11 |
| Codi Mètode Cerca per Id | 6 |
| Mapeig (XML) | 0 |
| **Total** | **37** |

Diagrama 3.15

| **JPOX** | |
|---|---|
| **TIPUS** | **QUANTITAT** |
| POJO | 16 |
| Codi Mètode Insert | 20 |
| Codi Mètode Cerca per Id | 18 |
| Mapeig (XML) | 17 |
| **Total** | **71** |

Diagrama 3.16

Mentre TopLink realitza el mapeig per mitjà de les annotations (dins de l'objecte Java), amb JPOX s'ha d'especificar en el fitxer xml corresponent. També és necessari remarcar que les crides a la persistència de JPOX s'han realitzat (és necessari segons l'estàndard JDO) amb tractament d'excepcions dins un bloc try catch, mentre que amb TopLink no es fa necessari aquesta condició.

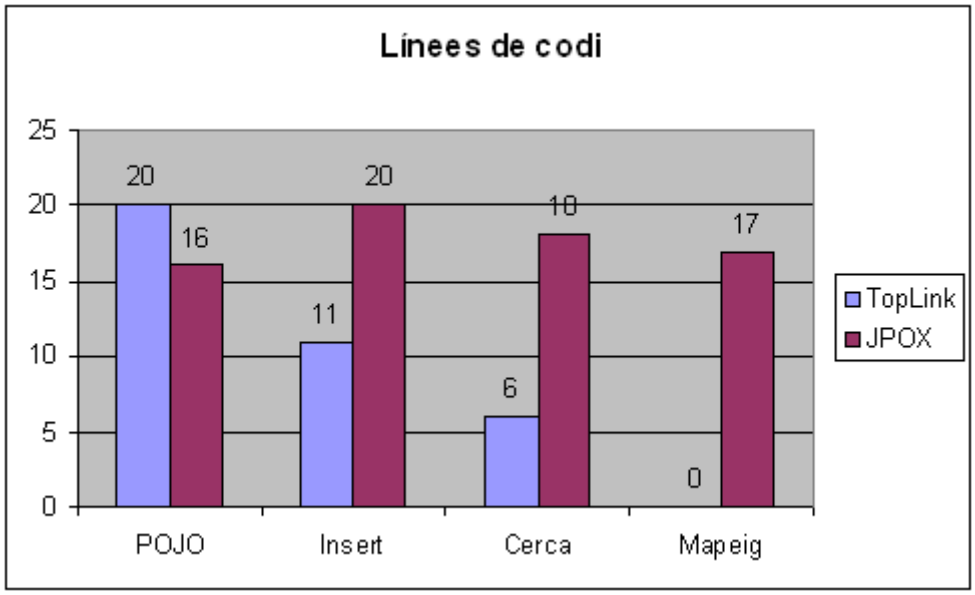


Diagrama 3.17

Més significatiu que les línees de codi emprades és la quantitat de tecnologies a conèixer. Amb TopLink s'empra xml pel fitxer de configuració (opcionalment també es podrien definir els mapejos) i dins Java s'ha de saber definir les annotations i les consultes amb el llenguatge de consulta orientat a objectes JPQL. Amb JPOX no fa falta emprar aquest dos darrers recursos Java per salvar o obtenir un objecte de la base de dades, tot i que en JPOX trobem també el llenguatge de consulta orientat a objectes JDOQL per definir les consultes més complexes.

## 7. COMPARACIÓ DE PRODUCTES: RENDIMENT CRUD (ALTES, BAIXES, MODIFICACIONS)

Els resultats de les probes realitzades les presentem seguidament. Aquestes s'han realitzat dins l'entorn explicat a l'apartat anterior i s'han executat des d'una aplicació JSE directament. El temps exposat s'expressa en milisegons. Les probes s'han realitzat en dos ambients, amb una taula buida (100 registres) i amb una taula carregada (3000 registres).

- Insert

A la figura 3.18 es poden observar els resultats del test d'inserció de 100 noves piscines.

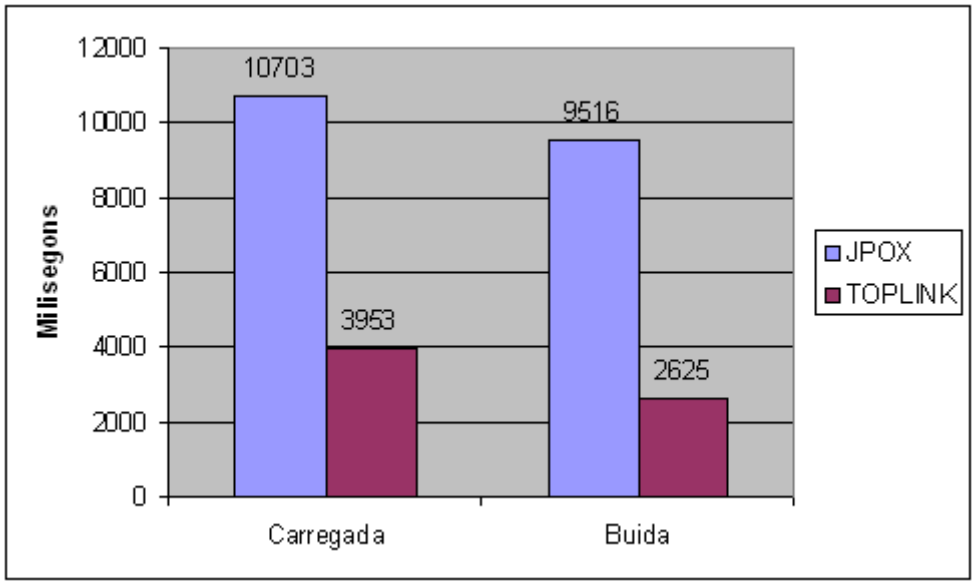


Diagrama 3.18

- List de tota la taula

Els resultats d'obtenir tota la taula es poden observar al diagrama 3.19.

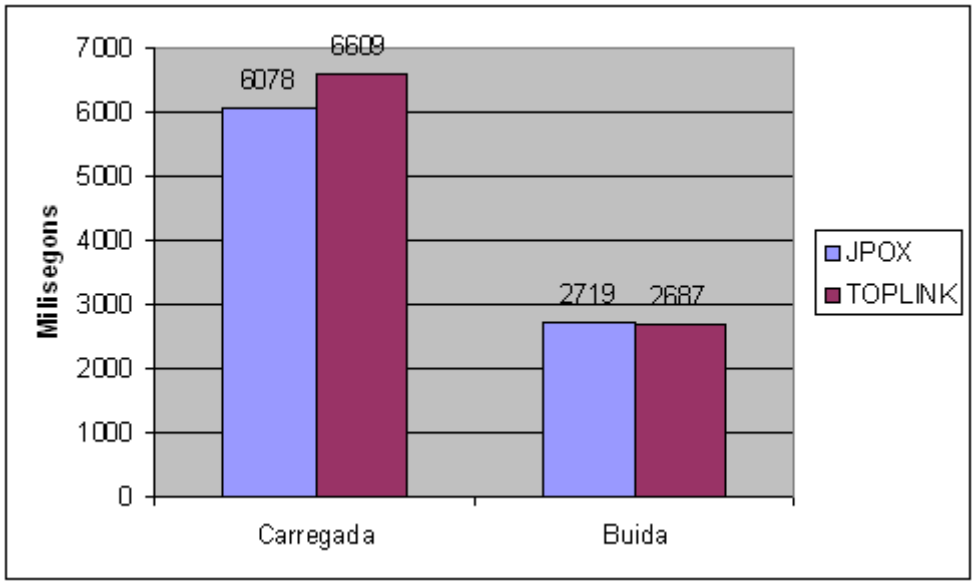


Diagrama 3.19

- List per identificador

El framework realitza una cerca per l'identificador d'una piscina.

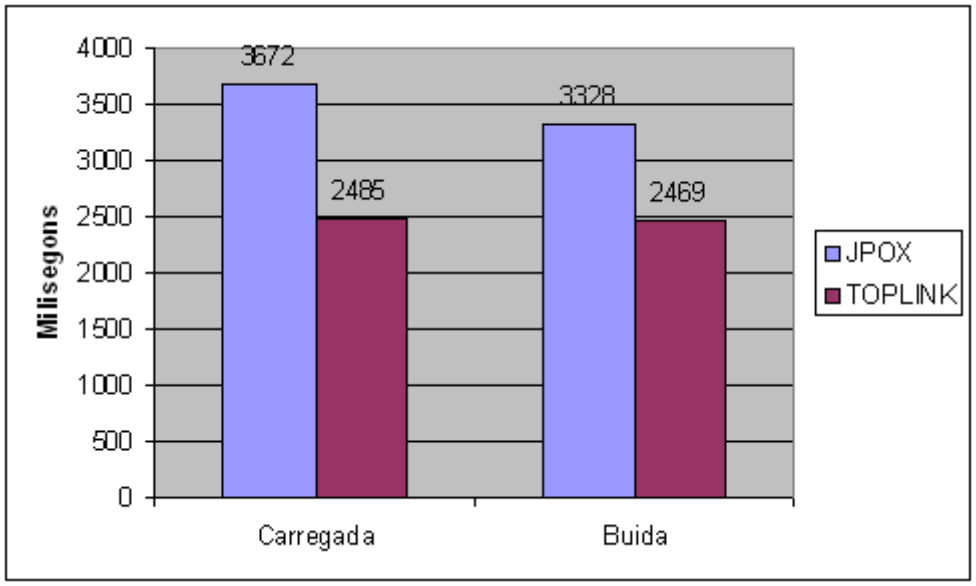


Diagrama 3.20

- List els registres que compleixen un requisit

S'avaluarà la resposta del framework a una cerca per un criteri (volum de la piscina menor de 10).

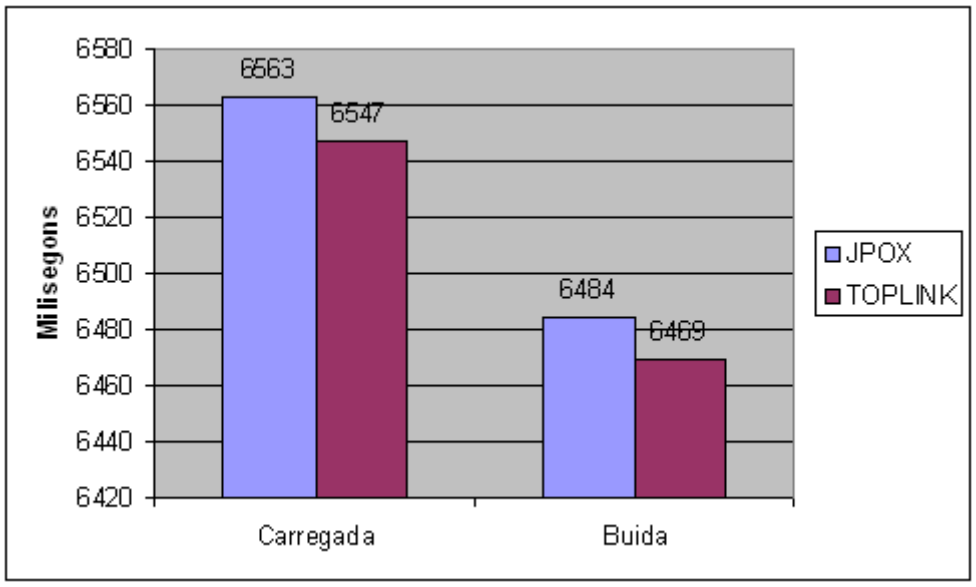


Diagrama 3.21

- Delete un objecte

Eliminar un registre de la base de dades segons l'identificador únic.

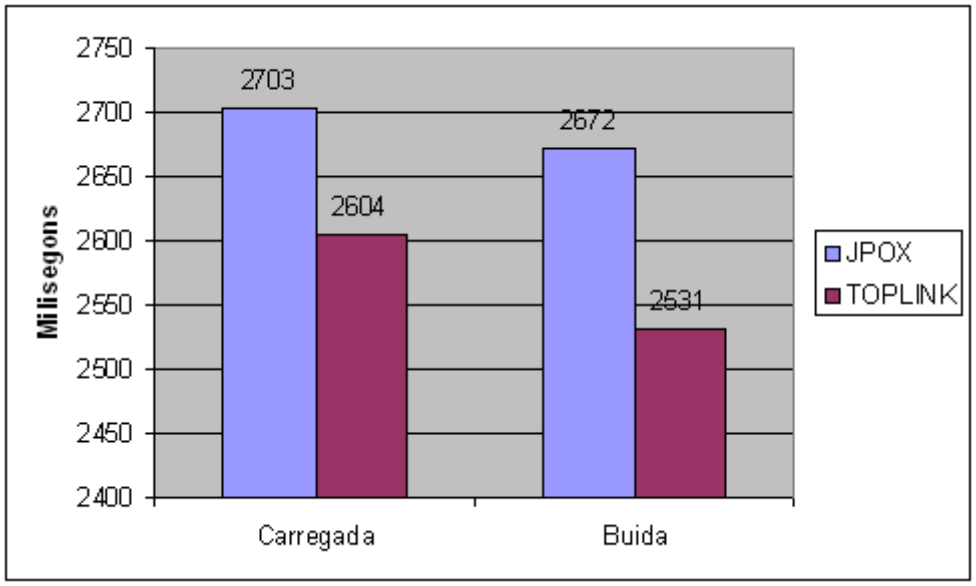


Diagrama 3.22

- Update una piscina

Per realitzar la modificació d'un objecte tan sols fa falta recuperar l'objecte (list per identificador) i assignar-li a aquest el nou valor de l'atribut a canviar. Quan es confirma la transacció (commit) el framework directament guarda l'estat de l'objecte

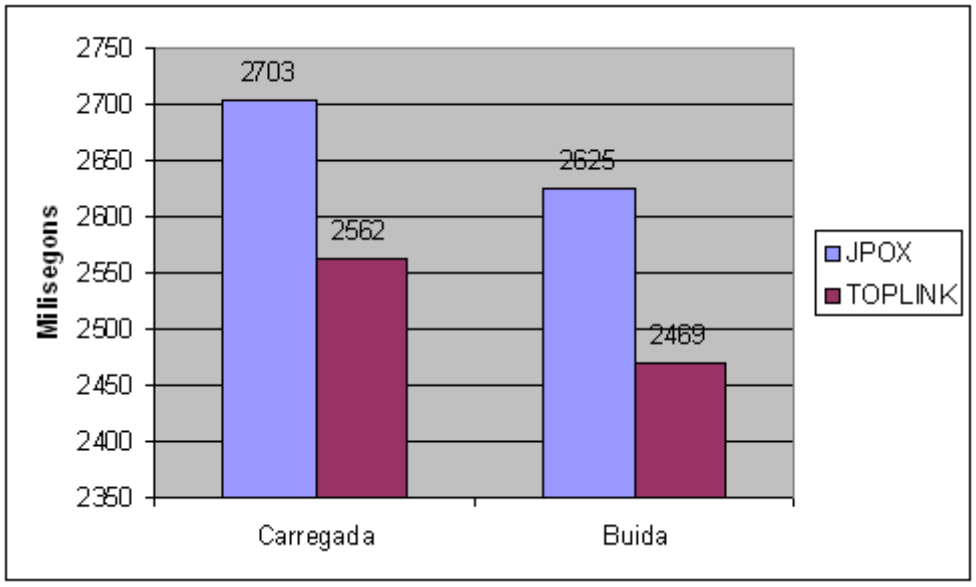


Diagrama 3.23

## 8. COMPARACIÓ DE PRODUCTES: MECANISMES DE CONSULTA

Els dos productes estudiats possibiliten les consultes amb funcions agregades (sum, max, min, avg i count) d'SQL. Al diagrama 3.24 es poden veure resultats dels temps de resposta a diverses consultes.

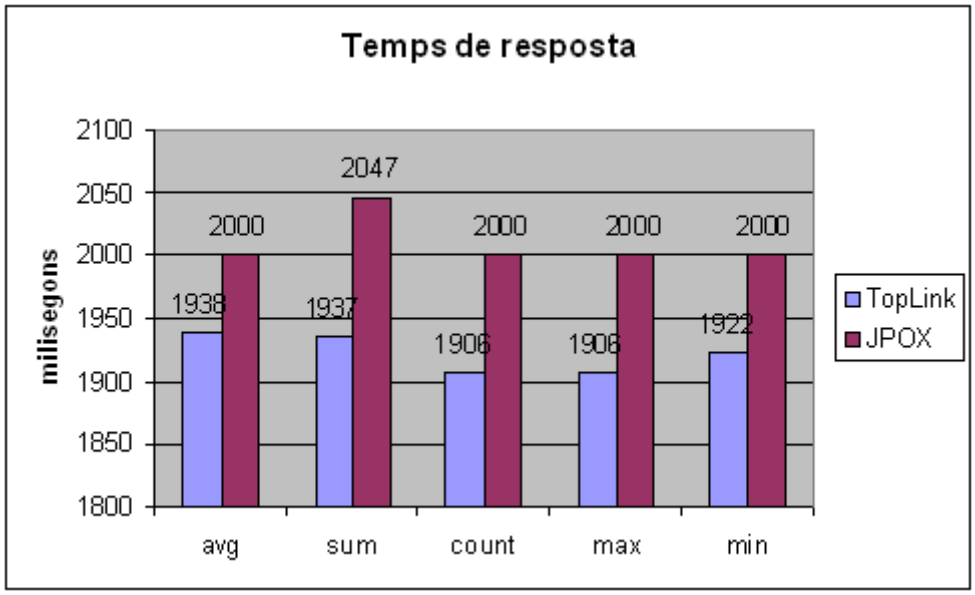


Diagrama 3.24

També els dos productes ofereixen suport per realitzar consultes més complicades amb agrupaments (clàusula group by, clàusula having), ordenades (order by) i sense repeticions (distinct). La funcionalitat oferta és pràcticament la mateixa, encara que hi podem trobar consultes que el gestor de bases de dades mySql i JPOX accepten i TopLink no les suporta. Aquestes consultes són consultes amb la clàusula having: per exemple

```
SELECT p.poblacio,sum(p.volum)FROM Piscina p GROUP BY p.poblacio having sum(p.volum)<100
```

Diagrama 3.25

que amb TopLink no es pot executar, tornant l'excepció: la clàusula having ha d'especificar condicions de cerca sobre els registres del group by o sobre els de les funciones agregades que s'apliquen a la agrupació. Caldria esperar que les funcionalitats ofertes pel gestor de bases de dades estiguessin presents en el framework.

TopLink ofereix el llenguatge de consulta orientat a objectes JPQL que és molt semblant a l'SQL (per tant, no requereix massa aprenentatge si es domina aquest). S'ha de definir la consulta amb una cadena (classe Query) per obtenir després els resultats amb el mètodes de la classe Query getResultList(), getSingleList(),... depenent dels resultats esperats.

JPOX ofereix dues maneres de programar les consultes: amb el llenguatge JDOQL, d'una manera molt semblant al que ofereix TopLink, també s'ha de crear la consulta amb una cadena a la classe Query i després executar-la amb el mètode execute(); i d'una manera declarativa. Per exemple, per realitzar la consulta

```
select max(volum) from piscina where poblacio=”Alaior”;
```

Diagrama 3.26

s'ha de cridar a una sèrie de mètodes de la classe Query

```
static void funcDeclaratiu() {

PersistenceManager pm = PersistenceManagerHelper.getPersistenceManager();
        try {
                Transaction tx = pm.currentTransaction();
                tx.begin();
                Query query = pm.newQuery(mcp.Piscina.class);
                query.setFilter("poblacio == \"Alaior\"");
                query.setResult("max(this.volum)");
                Double results = (Double) query.execute();
                System.out.println(results);
                query.closeAll();
                tx.commit();
        } catch (Exception ex) {
                System.err.println(ex);
        } finally {
                if (pm.currentTransaction().isActive()) {
                pm.currentTransaction().rollback();
                        }
                pm.close();
        }
}
```

Diagrama 3.27

Aquesta manera de tractar la consulta presenta menys transparència i més aprenentatge, però proporciona moltes més facilitats alhora de realitzar consultes més complicades.

El llenguatge JDOQL posseeix moltes més funcionalitats addicionals. Per exemple, es pot extendre les consultes a les subclasses d'una classe donada amb la classe Extent, multitud de mètodes de classe que ajuden a programar les consultes,...

JPOX també ofereix una explicació de les excepcions molt més comprensible i recordem que qualsevol connexió i consulta amb la base de dades queda reflexada en un fitxer de log. Aquestes propietats ajuden molt a entendre els errors quan es programen els mètodes.

## 9. COMPARACIÓ DE PRODUCTES: CARDINALITAT

Tant TopLink Essentials com JPOX suporten les diferents cardinalitats que podem trobar en les associacions entre objectes (1 – 1, 1 – n, n - 1, n – m). Seguidament estudiarem que hi ha que fer per aconseguir la relació que trobem en el cas pràctic d'exemple que hem emprat. Es porta l'historial de medició de químics d'una sèrie de piscines per saber-ne l'estat d'aquestes. O sigui que un objecte Piscina es relaciona amb n objectes Medicions i al contrari, Medicions té una relació n a 1 amb Piscina.

- **TopLink Essentials**

El primer que s'ha de fer és definir les relacions en el fitxer de la classe (POJO). Igual que per definir els mapejos, TopLink pot emprar les annotations java per definir-les. Així, a la classe que ja teníem Piscina hi ha que afegir la relació OneToMany amb Medicions.

```
@Entity
public class Piscina {

...
private Collection<Medicions> medicions;

/** Nova instància de Piscina */
public Piscina() {
      medicions = new ArrayList<Medicions>();
}

...
@OneToMany(cascade = CascadeType.ALL, mappedBy = "piscina")
public Collection<Medicions> getMedicions() {
          return medicions;
          }

public void setMedicions(Collection<Medicions> medicions) {
          this.medicions = medicions;
          }
...
}
```

Diagrama 3.28

Amb TopLink tenim quatre tipus d'accions en relació al tractament de les associacions:

- Persist: es persisteixen els objectes Medicions quan es persisteix la piscina que els posseeix
- Merge: quan es recupera una entitat piscina, es recuperen totes les medicions que pertanyen a aquesta
- Remove: quan s'esborra una entitat piscina, s'esborren les medicions corresponents
- All: tots els tipus anteriors

A l'altra banda, haurem de crear la classe Medicions amb la relació contrària ManyToOne. L'atribut mappedBy ens mostra que la propietat piscina d'una instància Medicions mapeja a una instància Piscina. Amb açò notifiquem que un identificador d'un objecte piscina existirà com a clau forana en una columna de la taula Medicions.

```
@Entity
public class Medicions {
@Id
private int id;
private double cl;
private double alk;
private double ph;
private Piscina piscina;

/** nova instància */
public Medicions() {
}
....

@ManyToOne
public Piscina getPiscina() {
    return piscina;
}
....
```

Diagrama 3.29

Ara tan sols ens falta afegir al fitxer persistence.xml la nova classe Medicions per poder mapejar les relacions entre classes.

- **JPOX**

Amb JPOX les taules les podem relacionar de dues maneres. De la manera normal les taules es relacionen per mitjà d'una taula join. Es crea una taula que defineix la associació. L'altra opció és la que es defineix com inversa i és relacionar les taules per mitjà d'una clau forana. Les relacions entre classes les hem de definir al fitxer de mapeig package.jdo.

En relació als tractaments de les relacions, la recuperació dels objectes Medicions dins de l'objecte recuperat Piscina és més difícil d'implementar i, per manca de temps, es deixa com tasca pendent. A diferència de TopLink, l'esborrament d'un objecte Piscina que té Medicions persistides (clau forana piscina_id a la taula medicions) no esborra els registres, sinó que canvia el camp de la clau forana a null. Així s'ha d'anar alerta amb no declarar el camp piscina_id com not null, ja que alhora d'intentar l'esborrament d'una piscina amb medicions es produiria una excepció.

```
<package name="mcp ">
<class name="Piscina"
....
<field name="medicions" persistence-modifier="persistent" mapped-by="piscina">
   <collection element-type="mcp.Medicions"/>
   <order>
      <extension vendor-name="jpox" key="list-ordering" value="piscina_id ASC"/>
    </order>
         <foreign-key>
                  <column name="piscina_id"/>
         </foreign-key>
</field>
</class>
<class name="Medicions"
....
<field name="piscina">
   <column name="piscina_id"/>
   <index name="piscina_id"/>
</field>
</class>
</package>
```

Diagrama 3.30

Al següent diagrama es poden estudiar els temps de resposta per les dues operacions amb cascade (recuperar i esborrar).

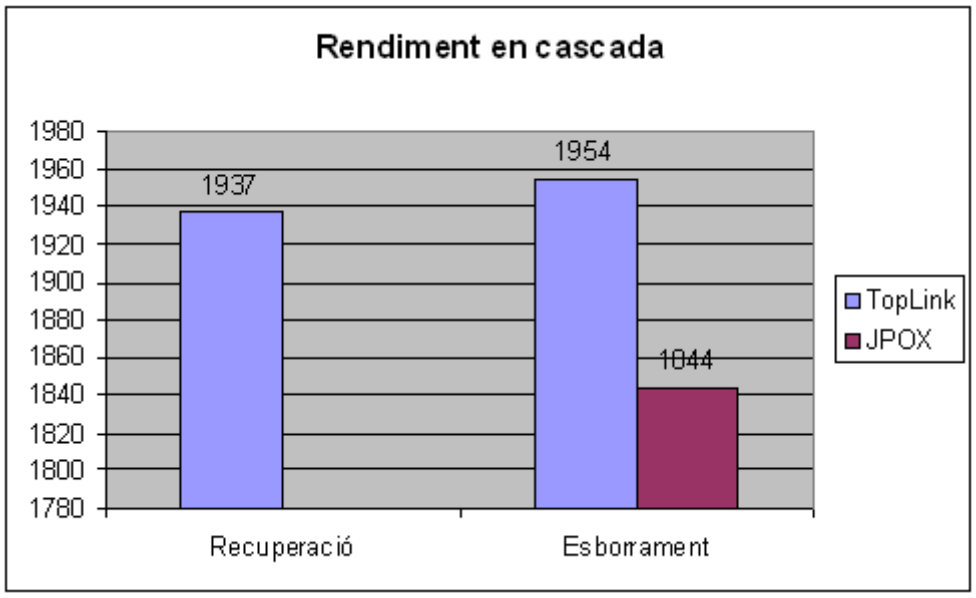


Diagrama 3.31

## 10. ESTUDI RESULTATS OBTINGUTS

Els resultats obtinguts de l'anàlisi de la comparació dels mecanismes no aporten grans diferències. La prova d'una manera pràctica de tots els frameworks presentats (tant a la part d'anàlisi com a la disseny) ens dóna una visió que per una banda és una mica desalentadora del problema: no hi ha un producte que sobresurti per damunt dels altres així que es podria haver estalviat esforç de desenvolupament i unificar criteris per no tenir que optar entre productes semblants. Cada producte pot tenir propietats millores que un altre, però no hi trobem un producte que es pugui dir referent en relació a la persistència. L'altra cara de la moneda és que el problema de la no correspondència entre llenguatges orientats a objectes (Java) i bases de dades relacionals el trobem bastant desenvolupat. Els frameworks de persistència estudiats ens ofereixen una solució suficientment segura, transparent, rendible i completa que ens evita preocupar-nos de molts temes en el desenvolupament d'una aplicació, com pot ser la connexió a la base de dades (JDBC), les consultes (SQL), la integritat de les dades,...

Al no trobar un producte perfecte devers els altres s'ha elegit per fer un estudi de dos productes que implementin especificacions que tenen darrera grups bastant amplis de seguiment. Així no ens fermem a un producte concret que podria desaparèixer (producte d'una empresa que deixa de comercialitzar-lo, per exemple) i deixar-nos sense actualitzacions que implementin les noves tecnologies que sempre van sorgint. Aquestes especificacions difícilment desapareixeran i si encara açò ocorregués, significaria l'arribada d'un estàndard únic pel tractament de la persistència, acceptat per la majoria de la comunitat.

Centrant-nos en l'estudi realitzat dels dos productes, trobem propietats comparades que no tenen pràcticament diferències en els resultats. Així, la integritat (transaccionalitat) està garantida en els dos productes d`una manera semblant, es possible la construcció de l'esquema a la base de dades segons els mapejos definits, tot dos productes ofereixen les mateixes possibilitats en relació al tractament de les relacions entre objectes. Sobre el tema de la transparència, les línees de codi són sensiblement menors a TopLink, però implica la necessitat de dominar altres característiques (annotations) que no fan falta amb JPOX. En relació a les consultes complexes (funcions d'agregació, clàusules having,...) la funcionalitat es pràcticament la mateixa, encara que s'han trobat problemes amb TopLink per emprar la clàusula having. El temps d'execució en relació a aquestes funcions o en relació a les funcionalitat bàsiques (persistir un objecte, recuperar un objecte, modificar un objecte, esborrar un objecte) sempre és menor en el producte TopLink, però açò es deu a que JPOX crea un fitxer de log per depurar errors que no trobem a TopLink. O sigui, que tal vegada per emprar una aplicació ja acabada seria millor emprar TopLink (més rapidesa de resposta, encara que no és tanta aquesta diferència) però per desenvolupar-la i realitzar el seguiment seria millor JPOX, encara que també es pot afegir aquesta funcionalitat de logs a la primera.

En general, podem concloure que per aplicacions que necessiten molts i ràpids accessos a la base de dades (aplicacions en temps real, per exemple) l'eina més adient de les estudiades és TopLink Essentials ja que te un temps de resposta

molt millor. Si l'aplicació el que necessita són consultes i utilitats més complexes, amb un difícil desenvolupament i amb una gran seguretat en trobar fallades ens decantem per JPOX (podeu veure la comparació entre llenguatges de consulta en el següent capítol).

Aquestes no són raons prou de pes per ajudar a decidir-se entre un dels dos productes. Actualment, on la programació de components i Java Enterprise Edition està molt de moda, TopLink el podem trobar integrat dins el projecte d'Oracle d'un servidor d'aplicacions de qualitat empresarial (GlassFish). Un altre servidor molt present és el Jboss. Aquest projecte s'ha decantat darrerament per integrar el framework Hibernate, encara que es poden trobar plugins per emprar altres productes. Per tant, potser una bona solució alhora d'elecció d'un producte concret estudiar quin d'aquests s'adapta millor a l'entorn de treball que estem acostumats a emprar.

# 3. CONCLUSIONS I TREBALL FUTUR DEL PROJECTE

El problema de la persistència dins el paradigma de la programació orientada a objectes (i de la programació en sistemes distribuïts en particular) el podem considerar com resolt satisfactòriament. Les eines estudiades ofereixen els requisits mínims per ajudar al programador a aconseguir objectes perdurables entre sessions. Aquestes eines han d'oferir serveis addicionals per atraure als programador a emprar-les. Serveis com ajudes a paginació, consultes dinàmiques, manteniment de fitxers de logs,.. són importants ja que si ens basem tan sols amb rendiment no trobem diferències prou remarcables per decantar-se per una eina en concret.

Un altre aspecte que s'ha de tenir en compte és que darrerament aquestes eines han estat adjuntades a projectes que engloben camps més amplis. Per exemple Hibernate s'ha integrat dins Eclipse, TopLink forma part del projecte Glassfish. Així la nostra elecció pot estar condicionada per les eines de desenvolupador que estem acostumats a emprar, ja que encara que la integració d'aquests frameworks no vol dir que ens obliguin a emprar-los (sempre s'està en llibertat d'emprar altres diferents) el suport i la integració pot ser una gran ajuda en el procés d'aprenentatge i desenvolupament.

Aquest projecte pretén ser una ajuda a l'elecció d'una eina concreta de persistència, però per açò faria falta un estudi més ampli de tots els productes considerats així com d'un estudi dels serveis addicionals que ofereixen. Aquest projecte hauria de ser el punt de partida per la realització d'un treball on es considerin les virtuts i inconvenients de cada framework de persistència, així com detalls i ajudes a la configuració per la seva utilització.

# 3. APROXIMACIÓ A JDOQL AMB JPOX

Com s'ha comentat a l'apartat de rendiment el llenguatge de consultes JDOQL es pot emprar d'una manera declarativa, d'una manera sintàcticament semblant al llenguatge Java i que facilita tasques de consulta. És per açò que ens ha semblat interessant aprofundir una mica en aquest llenguatge.

## 1. INTRODUCCIÓ

JDOQL és el llenguatge de consultes definit per l'estàndard JDO per actuar amb els mecanismes de persistència. Les condicions de consulta es poden expressar amb una sintaxis Java, sintaxi prou coneguda pel desenvolupador i per tant de fàcil comprensió. El disseny de JDOQL i la interfície que manipula les consultes cerquen els següents objectius:

- Independència del llenguatge devers els mecanismes on persisteixen les dades
- Disposar de la capacitat d'optimització de les consultes
- Disposar de dues maneres de processament de consultes, en memòria o delegat als sistemes de consulta dels gestors de dades
- Possibilitat d'accedir a conjunt de resultats grans, iterant i processant un nombre elevat d'instàncies amb un limitat ús dels recursos
- Possibilitat d'ús de consultes compilades, per no repetir el procés de preparació de les consultes
- Possibilitat d'anidament de consultes, el resultat d'una consulta pot ser emprat per una altra

## 2. ÚS DEL LLENGUATGE JDOQL

Les consultes JDOQL són manipulades per mitjà de la interfície Query de JDO. Tres són els elements bàsics que es requereixen per portar a terme una consulta amb aquesta interfície: la classe d'instàncies que són consultades, el tipus de la col·lecció de resultat (java.util.Collection o Extent) i el filtre de la consulta.

L'ús d'aquest llenguatge facilita diverses tasques devers l'ús de consulta d'un llenguatge més proper a l'SQL. La interfície Query permet incloure a la condició

de cerca paràmetres i variables que formen part del filtre de la consulta, aconseguint així la construcció de consultes dinàmiques. La interfície Extent representa la col·lecció completa de totes les instàncies d'una classe desades en la base de dades, permetent la manipulació de grans conjunts de dades i la gestió dels recursos afectats ja que les consultes són processades en el sistema de gestió de bases de dades.

Per exemple, si volem fer una consulta per aconseguir tots els objectes piscina que tenen un volum menor de 10

```
Extent extent = pm.getExtent(Piscina.class, true);
Query q = pm.newQuery(extent, "volum < 10.0");
q.setOrdering("poblacio ascending");
Collection results = (Collection) q.execute();
```

Diagrama 4.1

La interfície Extent retorna totes les instàncies de la classe i de les seves subclasses associades, ja que el segon paràmetre del mètode getExtent està a true. Amb aquest extent creem la consulta amb un filtre que és una cadena amb la condició, permetent així crear consultes dinàmiques ja que aquesta cadena pot ser passada com paràmetre entre classes. JDOQL també permet la creació de paràmetres dins el filtre, amb el mètode declareParameters. A més, es pot definir l'ordre de resposta (com hem vist al diagrama 4.1) o demanar tan sols el primer objecte que compleix la condició (set Unique, com podem veure al següent diagrama)

```
...
Query q = pm.newQuery(extent, "volum == volMin");
q.declareParameters("Double volMin");
q.setUnique(true);
Piscina p = (Piscina)q.execute(volMin); //p.ex. = 10.00
...
```

Diagrama 4.2

A manera d'exemple, presentem la consulta complexa que s'ha definit al cas d'estudi (mitja de volums de les piscines per població), que amb JDOQL es pot definir de la següent manera

```
...
Transaction tx = pm.currentTransaction();
                 tx.begin();
                 Query q = pm.newQuery(pm.getExtent(Piscina.class,false));
                 q.setResult("poblacio, AVG(volum)");
                 q.setGrouping("poblacio");
                 q.setOrdering("poblacio descending");
                 Collection results = (Collection)q.execute();
...
```

Diagrama 4.3

## 3. RESULTATS

Al diagrama 4.4 es pot veure la gràfica en relació a rendiment d'execució de la consulta complexa presentada a l'apartat anterior. Amb JPOX no hi trobem diferències entre emprar JDOQL amb cadena SQL a emprar-lo de la manera declarativa.

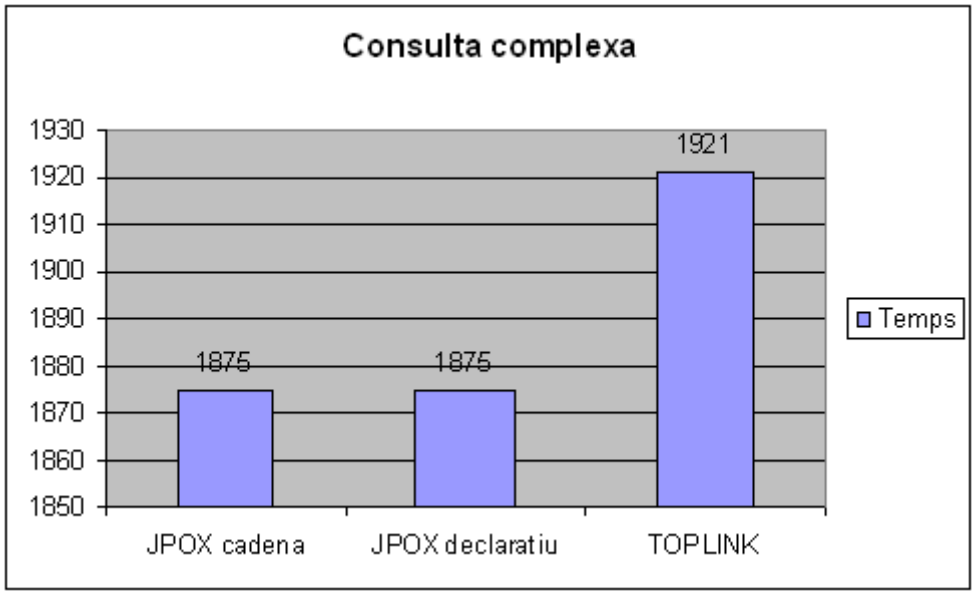


Diagrama 4.4

El que sí que es pot comprovar directament és que el rendiment de TopLink és una mica més baix quan es processen consultes amb ordenacions i agrupacions.

## 4. CONCLUSIONS

Sorprenentment amb una consulta complexa on trobem agrupacions i ordenacions el rendiment de JPOX és millor que el rendiment de TopLink (recordem que a les proves de rendiment de l'apartat 3 sempre s'ha constatat millors temps de resposta amb TopLink).

En relació a emprar JDOQL d'una manera cadena SQL a d'una manera declarativa, per la nostra experiència en les proves realitzades arribem a la conclusió que val la pena l'esforç d'aprendre a emprar JDOQL de forma declarativa ja que, encara que en relació a rendiment no hi trobem diferències, es disposen d'altres mètodes (no estudiats en aquesta memòria, però que es pot afegir al treball futur del projecte) dins l'especificació JDO o extensions amb JPOX que ajuden a l'elaboració de les consultes i els seus resultats. Exemples d'aquest mètodes en trobem al diagrama 4.5 (es pot trobar una llista completa a la pàgina de JPOX).

| Tipus | Mètode | Descripció |
|---|---|---|
| String | startsWith(String) | Retorna si la cadena comença pel paràmetre |
| String | toLowerCase(String) | Retorna la cadena en minúscules |
| Collection | size() | Retorna el nombre d'elements de la col·lecció |

Diagrama 4.5

A més, és molt important considerar que cada programador pot realitzar noves extensions i mètodes per adequar les respostes a les seves necessitats.

# 4. GLOSSARI

- Annotation: tipus de metadada a nivell del codi font que es troba disponible en temps d'execució.
- API: conjunt de funcions i procediments que ofereix una llibreria determinada.
- BMP: persistència amb EJB gestionada pel bean.
- CMP: persistència amb EJB gestionada pel contenidor.
- Consultes dinàmiques: consultes creades en temps d'execució.
- DAO: patró de disseny que permet abstraure i encapsular tots els accessos a la font de dades.
- EJB: API que forma part de l'estàndard JEE, que especifica els objectes que ofereix el servidor.
- Fetching: com i quan es processa una consulta SQL per obtenir unes dades.
- HQL: llenguatge de consulta orientat a objectes que porta l'eina Hibernate.
- Java EE: plataforma de programació per desenvolupar i executar aplicacions Java amb arquitectura distribuïda basada en components.
- Java SE: plataforma per desenvolupar i executar aplicacions Java d'ús general.
- JDBC: API que permet les connexions a les bases de dades.
- JDO: especificació de Sun per la persistència d'objectes.
- JDOQL: llenguatge de consulta orientat a objectes de l'especificació JDO.
- JPA: part de l'especificació EJB 3.0 dedicada a la persistència d'objectes.
- Metadada: dades de les dades.
- ODMG: estàndard per a la persistència general a tots els llenguatges de programació.
- OODBMS: sistemes de gestió de bases de dades orientats a objectes.
- ORM: eines per mapejar objectes i taules relacionals.
- POJO: terme emprat per referir-se als objectes Java que no necessiten cap particularitat per poder ser persistits.
- RDBMS: sistemes de gestió de bases de dades relacionals.
- SQL: llenguatge de consulta a base de dades relacionals.
- Transaccionalitat: interacció completa amb una estructura de dades, on es garantitza la finalització de la mateixa.
- UML: llenguatge de modelat unificat.
- XML: metallenguatge extensible d'etiquetes.

# 5. BIBLIOGRAFIA


- L. JOYANES, M.FERNÁNDEZ: Java 2. Manual de programación. *McGraw-Hill*.
- A. SILBERSCHATZ, H.KORTH, S. SUDARSHAN: Fundamentos de bases de datos. *McGraw-Hill*.
- A. SINGHAL: A comparative study of data access mechanisms for distributed computing in J2EE Platform. *http://www.cs.rit.edu/~axs9690/ArtiSinghal-MSProjectReport.pdf*
- S. YALDIZ: Evaluation of web application development frameworks and object-relational mappers. A case study. *http://www.sts.tu-harburg.de/pw-and-m-theses/2004/yald04a.pdf*
- J. MÁRMOL: Persistencia de objetos. JDO, solución Java. *http://dis.um.es/~jmolina/Persistencia%20de%20Objeto%20JDO.pdf*
- C. GALEANO, E. GOETTE: Persistencia de los objectos Java utilizando base de datos relacionales. *http://emanuelpeg.hostinggratisargentina.com/*
- J. CABOT, J.M. CAMPS, J.CEBALLOS, F.J. DURÁN, N. MORENO, J.R. ROMERO, A. VALLECILLO: Enginyeria del programari de components i sistemes distribuïts. *UOC*
- J. M. CAMPS: Pràctica tutoritzada. Enginyeria del programari de components i sistemes distribuïts. *UOC*

Documentació del llocs web

- Hibernate: http://www.hibernate.org
- Ibatis: http://ibatis.apache.org
- TopLink: http://www.oracle.com/technology/products/ias/toplink/jpa/index.html
- Castor JDO: http://www.castor.org/jdo.html
- Cayenne: http://cayenne.apache.org
- CocoBase: http://www.thoughtinc.com
- Form: http://www.chimu.com/projects/form/index.html
- JGrinder: http://jgrinder.sourceforge.net
- Kodo: http://www.solarmetric.com/solarmetric_junp_page
- TJDO: http://tjdo.sourceforge.net
- Torque: http://db.apache.org/torque
- JPOX: http://www.jpox.org