DISSENY DE LA PERSISTÈNCIA

Tema 6

Enginyeria del Programari
ETS Enginyeria Informàtica
DSIC – UPV

Curs 2021-2022

Objectius

- Comprendre la necessitat de mantenir la persistència en el desenvolupament de programari
- Conèixer el patró d'accés a dades per a aconseguir una correcta abstracció i separació de capes
- Persistència en BD relacionals vs. BD objectuals.
- *Observacions:* El disseny lògic relacional de la BD a partir d'un model OO (diagrama de classes) s'estudia a l'assignatura BDA.

Continguts

1. Introducció
2. Patró d'accés a dades DAO
3. Patró Repositori + unitat de treball
4. Persistència en BDOR i BDOO
5. Conclusions

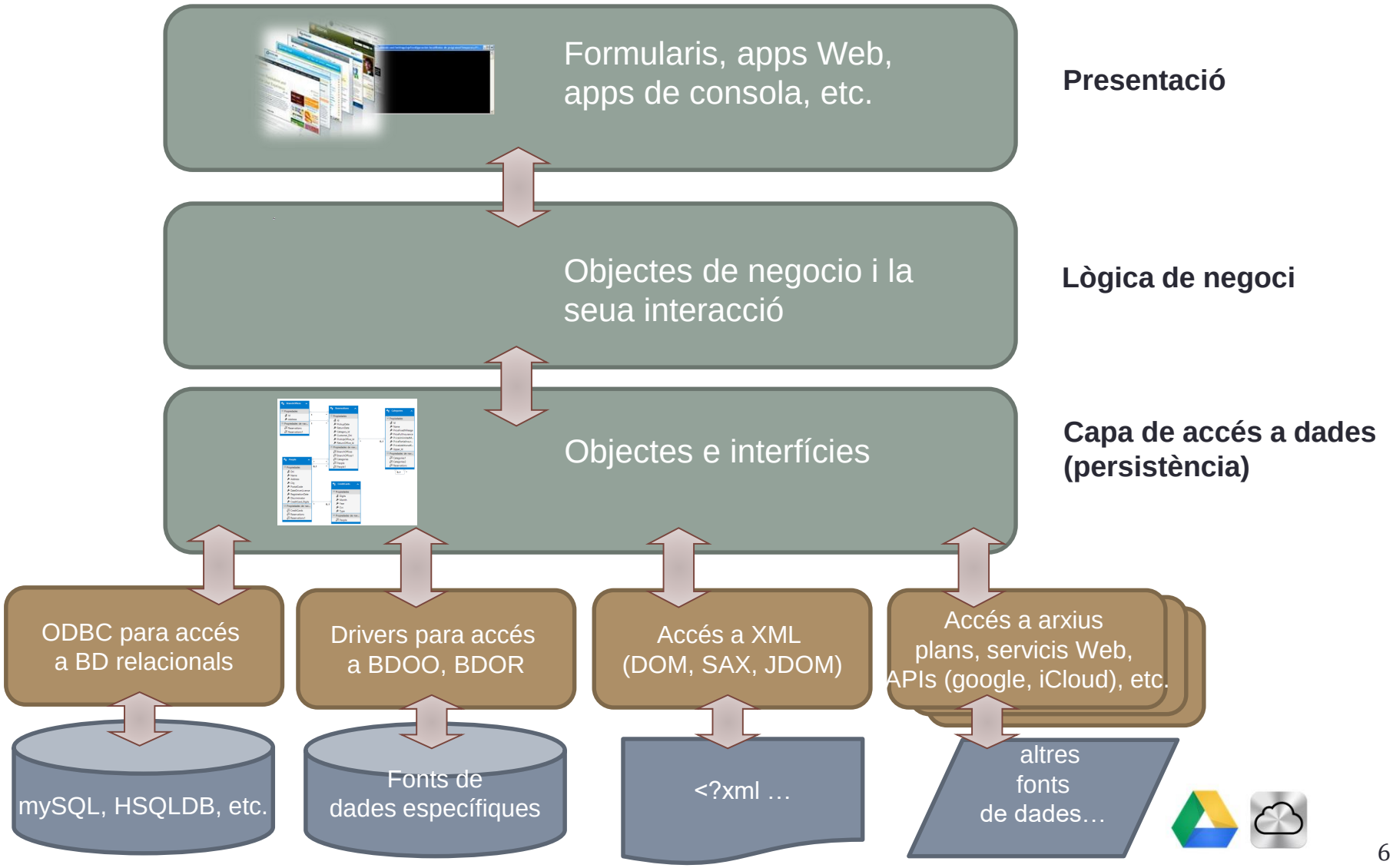
INTRODUCCIÓ

Introducció

- En la majoria d'aplicacions l'emmagatzematge **no volàtil** de la informació és essencial
  - Es pot utilitzar un format específic per a cada aplicació (compatibilitat limitada)
  - Es pot usar un format basat en XML, BD relacionals, OO o mixt (compatibilitat molt major – ex. BD-SQL)

- La utilització de BD es tradueix en l'ús de biblioteques per a gestionar l'accés a les dades (JDBC, ADO, ODBC, etc.)

Introducció: Arquitectura multicapa



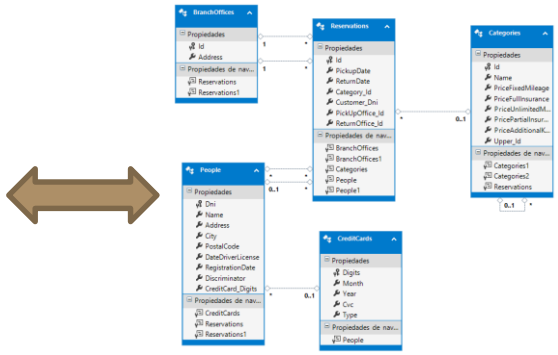
PATRÓ D'ACCÉS A DADES (**DAO**)

- ✓ Estructura
- ✓ Avantatges i inconvenients
- ✓ Implementació

Estructura del patró DAO (*Data Access Objects*)

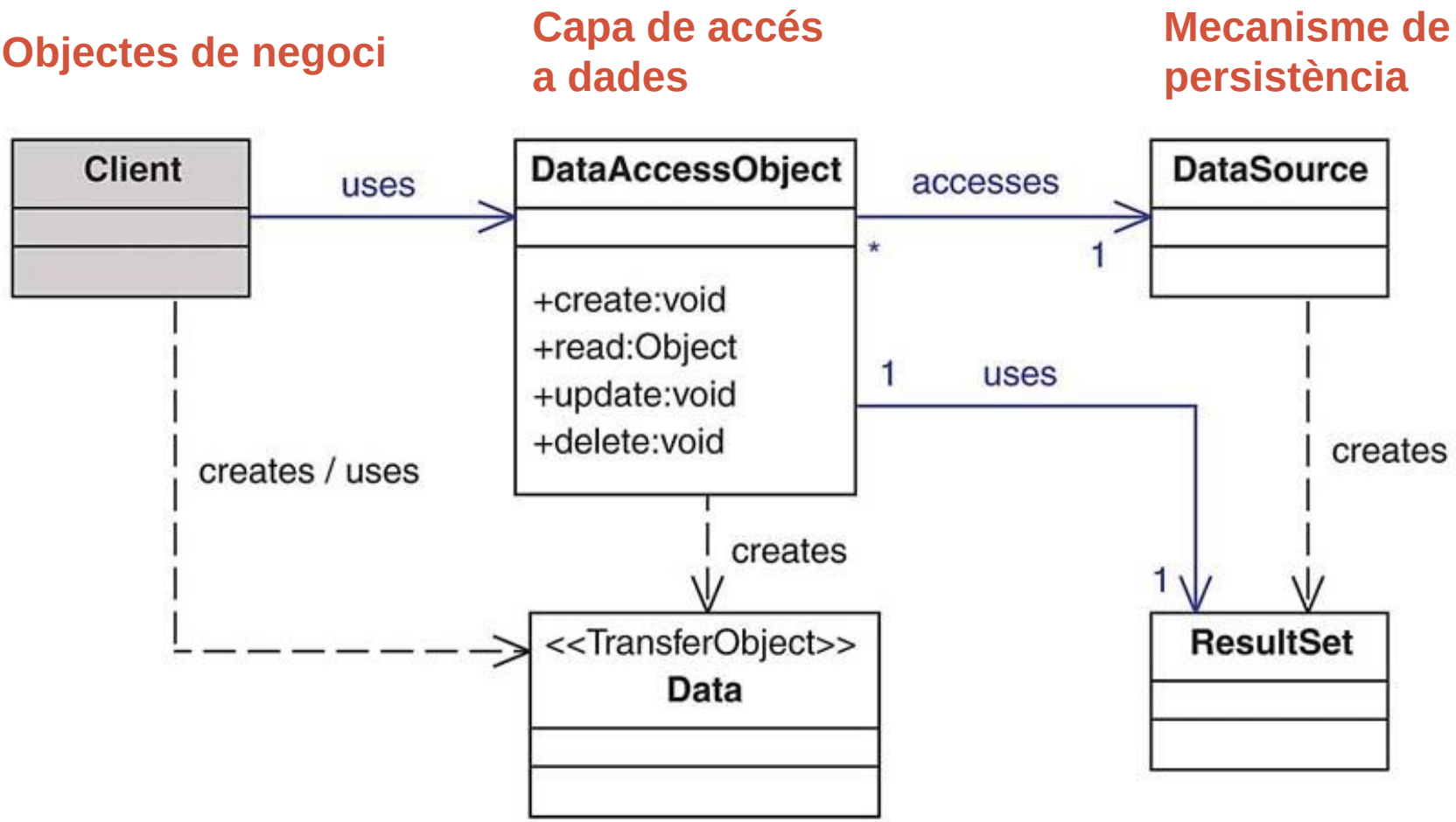
- Ofereix un pont d'implementació entre:
  - Els objectes del domini del problema (lògica de negoci)
  - Dades emmagatzemades de forma persistent (ex. XML, BDR,BDOR, fitxers, etc.)
- Abstrau el mecanisme concret de persistència. Ofereix una interfície d'accés a dades independent de la implementació amb mètodes per a afegir, actualitzar, cercar i esborrar registres

lògica de negoci (objectes)



dades emmagatzemades (ex. SQL)

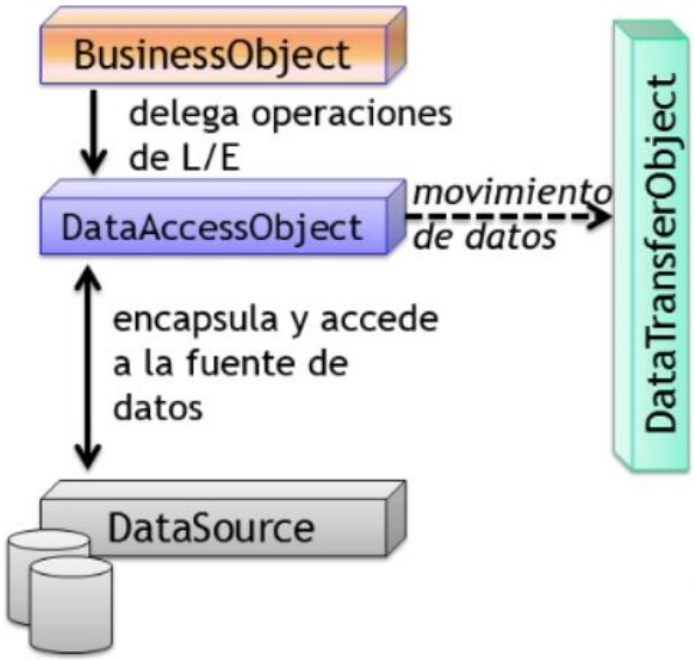
Estructura del patró DAO



Més info en: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Estructura del patró DAO - **Elements**

- ***BusinessObject:*** objecte de la capa de negoci que necessita accedir a la font de dades, per a llegir o emmagatzemar
- ***DataAccessObject(DAO):*** abstrau la implementació subjacent de l'accés a dades a la capa de negocis per a aconseguir un accés transparent a la font de dades. BusinessObject delega en el DAO les operacions de lectura i escriptura de dades
- ***DataTransferObject (DTO):*** representa l'objecte portador de les dades. DAO pot retornar les dades al BusinessObject en un DTO. El DAO pot rebre les dades per a actualitzar la BD en un DTO
- ***DataSource***: implementació de la font de dades (SGBDR, SGBDOO, SGBDOR, repositori XML, fitxers plans, etc.)



Avantatges i inconvenients

Patró DAO. **Avantatges**:

- **Encapsulació.** Els objectes de la capa de negoci no coneixen detalls específics d'implementació de l'accés a dades, ocults en el DAO.
- **Migració més fàcil**: migrar el sistema a un gestor de dades diferent suposa canviar la capa de DAO per un altra.
- **Menor complexitat** en la capa de negoci en aïllar-se de l'accés a dades.
- Accés a dades **centralitzat** en un nivell.

Avantatges i inconvenients

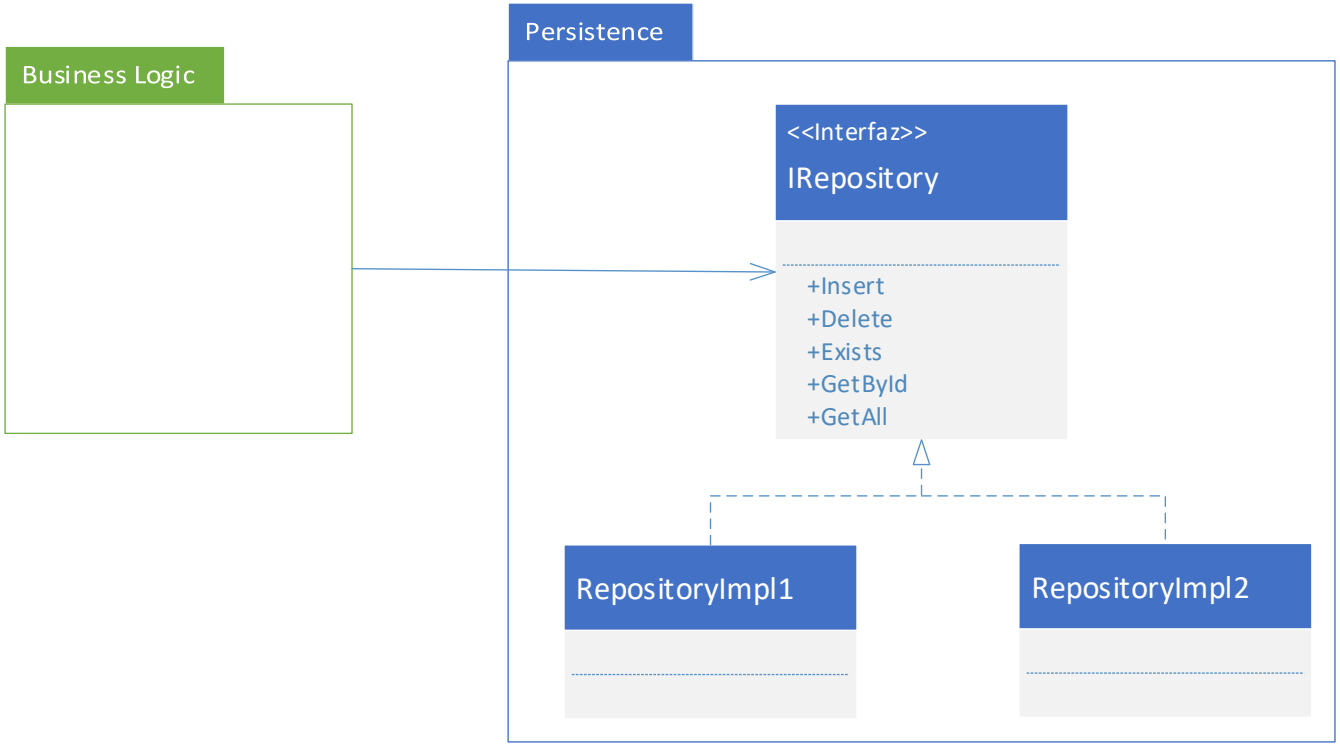
Patró DAO. **Inconvenients**:

- Arquitectura SW lleugerament **més complexa**
- Cal implementar **més codi** per a oferir aquest nivell de indirecció addicional
- Des del punt de vista de l'eficiència es pot **ralentir** el procés

PATRÓ REPOSITORI + UNITAT DE TREBALL

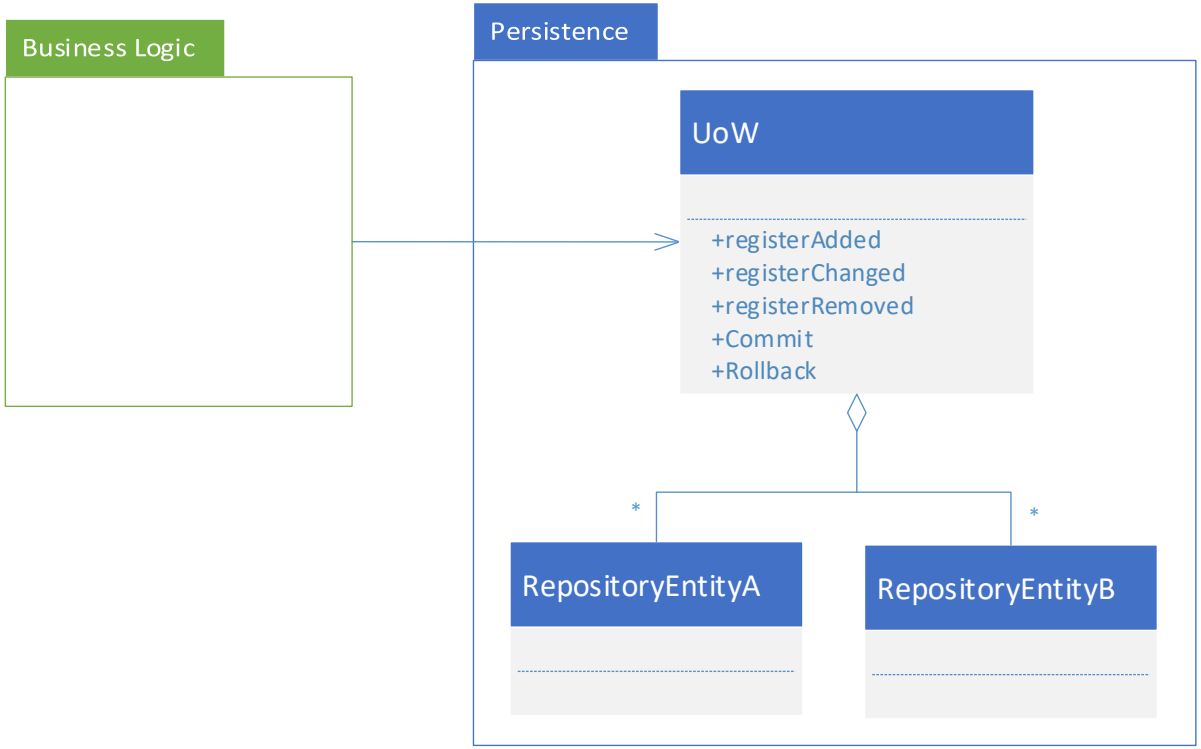
Patró Repositori

- **Repositori** (Repository): pont entre la capa de domini i la de persistència (típicament una BDOR). Ofereix una interfície tipus **col·lecció** per a accedir als objectes de domini directament (sense DTOs).



Patrons Repositori + Unitat de Treball

- **Unitat de Treball** (Unit of Work, or UoW):
  - manté un registre dels objectes afectats durant una transacció (operació atòmica sobre objectes de negoci) i
  - s'encarrega de persistir els canvis (o desfer-los en cas d'avortar la transacció) i resoldre qualsevol problema de concurrència que poguera sorgir.



El patró Repositori

- Ha augmentat la seua popularitat des que va ser introduït com a part de l'enfocament denominat **Domain Driven Design** (Evans, 2004)

- Ofereix una **abstracció** per a accedir a les dades com si foren **col·leccions en memòria.**
  - Interfície amb mètodes per a afegir, esborrar, actualitzar i cercar objectes del domini de forma directa, sense necessitat de crear objectes de transferència

  - Ajuda a reduir l'acoblament entre la lògica de negoci i la capa de persistència: objectes de negoci ignorants respecte al mecanisme de persistència

Un repositori por objecte del domini

- L'enfocament més senzill, especialment quan s'introdueix en un sistema antic, és crear **un repositori per cada objecte del domini a persistir**

- Solament es necessita implementar en cada repositori els mètodes que es vagen utilitzar i res més

- El major benefici d'aquest enfocament és que no es perd temps implementant mètodes que mai es van a utilitzar (YAGNI, You Aren't Gonna Need It)

Repositori genèric

- Un altre enfocament consisteix a crear un repositori genèric vàlid per a qualsevol objecte de domini. Es necessita una sola interfície i una única implementació per cada mecanisme de persistència que vulguem utilitzar (i no una implementació per cada classe a persistir)
- Un exemple d'interfície per a un repositori genèric en C# seria:

```
public interface IRepository
{
  void Add<T>(T entity) where T : class;
  void Delete<T>(T entity) where T : class;
  T GetById<T>(IComparable id) where T : class;
  bool Exists<T>(IComparable id) where T : class;
  IEnumerable<T> GetAll<T>() where T : class;
}
```

Repositori vs DAO

- El **concepte de DAO** està més proper al mecanisme de persistència subjacent; és un enfocament **centrat en les dades**. Per açò normalment es crea un DAO per cada taula o vista d'una base de dades.

- El **concepte de repositori** es troba més proper a la capa de negoci, doncs treballa directament amb els objectes de negoci.
  - Internament, un repositori podria usar BDOR, però també mecanismes de baix nivell com DAO/DTO

- En *dominis anèmics* (sense verdadera lògica de negoci, solament get/set), repositori i DAO són intercanviables

Patró Unitat de Treball

- Actualitzar la informació en la base de dades cada vegada que es modifica un objecte del domini:
  - És poc eficient, incorreria en molts canvis xicotets, i si alguna cosa ix malament durant una transacció cal desfer els canvis
  - Condueix a problemes de consistència en l'accés concurrent (si altres clients han obtingut les dades amb canvis que hi ha hagut que desfer)

- Una Unitat de Treball manté un registre de canvis en els objectes de negoci durant una transacció completa. No realitza els canvis en el mecanisme de persistència fins que no es completa la transacció, i si cal desfer alguna cosa se soluciona en memòria
  - És més eficient i evita problemes de consistència en l'accés concurrent

Patró Unitat de Treball

- **No necessàriament cal implementar-ho ad-hoc per a cada projecte** ja que ho trobem habitualment en frameworks de persistència, per exemple
  - La interfície ITransaction en Nhibernate
  - La classe **DbContext** en **Entity Framework**

- Per a una explicació més detallada sobre el patró Repository + UoW en C# (EF5 & MVC4, amb exemples) es recomana aquest artícle. També hi ha una versió per a EF6 & MVC5.

PERSISTÈNCIA EN BDOR I BDOO

- ✓ Objectius
- ✓ Avantatges
- ✓ Referències addicionals

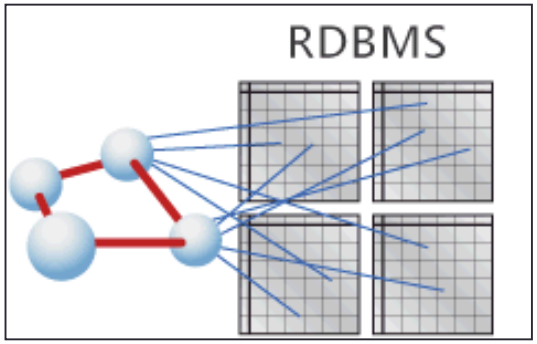
Objectius

En lloc d'implementar una BD relacional es proporciona una BDOO (no relacional)

- L'emmagatzematge intern representa als objectes com a tal (no disgregats en taules)
- Ja no és necessari un *middleware* objecte-relacional
  - No obstant açò, les BDOR són un model mixt (extensió d'una BDR per a manejar el paradigma OO) i també suporten SQL
- La majoria d'operacions es realitzen de forma més eficient, doncs no és necessari manejar dades en diferents taules
  - Ex., quan s'accedeix a una relació no es disposa de la clau aliena per a recuperar el registre d'una altra taula sinó del **propi** objecte relacionat

BDOR i BDOO

- En sistemes complexos resulta tediós haver de realitzar la conversió entre dades d'un model OO i els de un model relacional
  - Pas de les característiques del llenguatge de programació a SQL i viceversa



- Existeixen diverses eines que fan aquesta correspondència automàticament per a diversos llenguatges (Java, C#, VB...)
  - Ex. Entity Framework, Hibernate, etc.

Algunos ejemplos

```java
public void store(Car car){
    ObjectContainer db = Db4o.openFile("car.yap");
    car.engine(new TurboEngine());
    db.set(car);
    db.commit();
    db.close();
}
```

```
// OPEN THE DATABASE
d_Database db;
db.open( "business" );

d_Transaction tx;
tx.begin();

d_Ref obj;
obj = new( &db, "Customer" ) Customer();

obj->name = "Luke";
obj->surname = "Skywalker";

// INSERT THE OBJECT AS "MYFRIEND"
db.set_object_name( obj, "MyFriend" );

tx.commit();
```

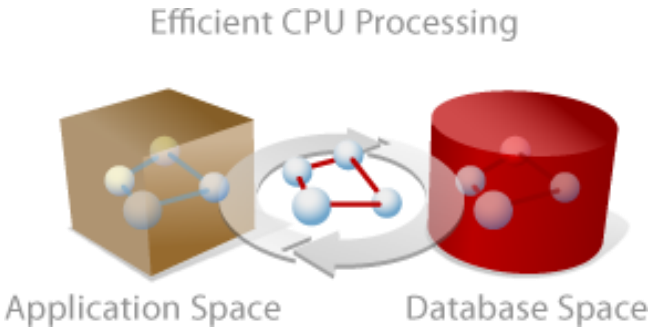
```
// OPEN THE DATABASE
d_Database db;
db.open( "business" );

d_Transaction tx;
tx.begin();

d_Ref obj;

// RETRIEVE THE ENTRY CALLED "MYFRIEND"
obj = db.lookup_object( "MyFriend" );

// DISPLAY THE CUSTOMER NAME
cout << "MyFriend is: " << obj->name;

tx.commit();
```

Avantatges

Tot açò aporta avantatges interessants:

- s'uneix la potència dels objectes amb la flexibilitat de SQL (en alguns casos)

- se simplifica molt el desenvolupament d'aplicacions en capes
  - no és necessari escriure consultes SQL
  - solament requereix un model del domini (i no 2)
  - la comunicació entre totes les capes és en forma d'objectes, sense conversió/mapping



CONCLUSIONS

Resumint…

- Els patrons d'accés a dades (DAO/Repository) permeten abstraure l'accés a la capa de persistència de la seua implementació
- És possible aplicar una senzilla correspondència per a derivar un model relacional a partir d'un model OO
- Les BD objectuals simplifiquen el desenvolupament d'aplicacions perquè:
  - el model de dades es pot projectar (cap a o des de l'aplicació) sense necessitat d'establir correspondències
  - les operacions són simples operacions sobre objectes
  - en general, també són més simples d'usar i més eficients

Bibliografia bàsica

- Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2004

- Feddema H.B., DAO object model: the definitive reference. O'Reilly, 2000.

- Fowler, M., Patterns of Enterprise Application Architecture. Addison-Wesley, 2002