Modele de proiectare a aplicațiilor de întreprindere

Cursul 10 - 14

Sumar

- Alte modele
  - Blackboard
  - Dependency Injection
  - Value Object
- Arhitecturi pentru aplicații de întreprindere
- Modele specifice nivelului de persistență a datelor (1)

Blackboard

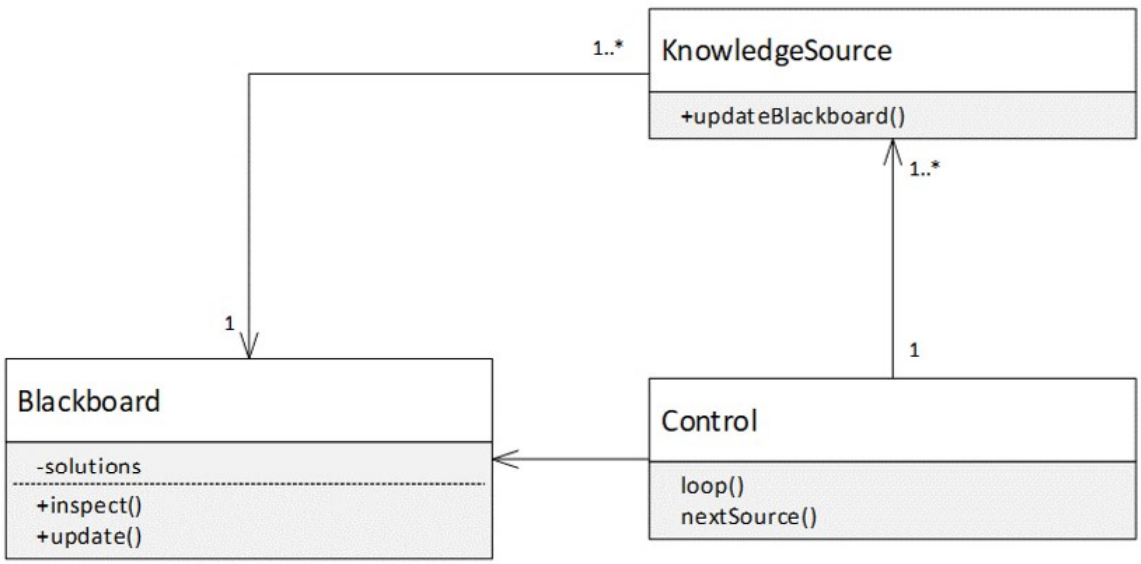
Problema

- Existența unui domeniu în care nici o abordare la o soluție nu este cunoscută sau fezabilă
- Se identifică o serie de arii de expertiză
- Soluțiile la problemele parțiale necesită reprezentări și paradigme diferite
- Fiecare secvență de transformare poate genera soluții alternative
- Exemple
  - Recunoașterea vocală – transformările necesită expertiză acustică, fonetică și statistică
  - Identificarea autovehiculelor

Scop

- Mai multe subsisteme specializate combină cunoștințele pentru a construi o soluție eventual parțială sau aproximativă

Diagrama de clase



Componente

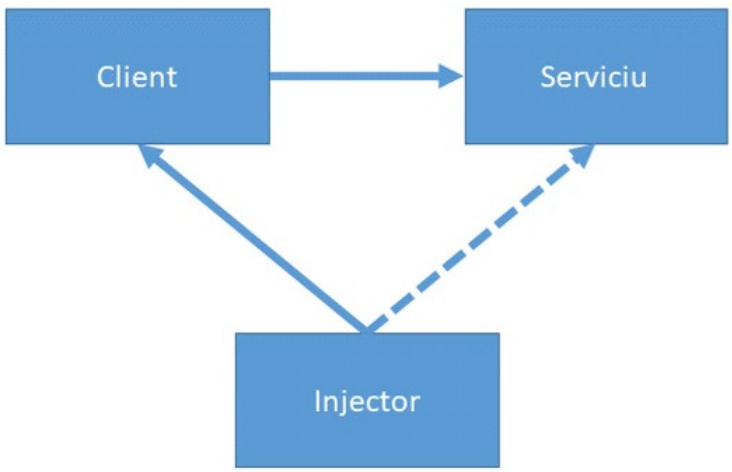
- **Blackboard**
  - Include obiectele din spațiul soluțiilor
- **KnowledgeSource**
  - Module specializate cu reprezentări specifice
- **Control**
  - Responsabil cu selectarea, configurarea și execuția modulelor

Dependency Injection

Scop

- Asigurarea unei dependențe reduse între obiecte
- Obiectele nu trebuie să fie responsabile pentru crearea propriilor dependențe
- Mecanism pentru crearea și transmiterea obiectelor (dependențelor) către un alt obiect
- Se bazează pe principiile
  - Dependecy Inversion (SOLI**D**)
    - Clasele trebuie să depindă de abstractizări și nu de implementări concrete
  - Inversarea controlului (IoC)
    - Inversarea controlului asupra obiectelor în scopul obținerii unei cuplări scăzute

Alte abordări



Componente

- **Client**
  - Clasa dependentă de **Serviciu**
- **Serviciu**
  - Furnizează servicii **Clientului**
- **Injector**
  - Creează un **Serviciu** și îl transmise clasei **Client**

Implementare

- Modalități diferite de transmitere a obiectelor
- Opțiuni
  - Constructor
  - Proprietăți (set)
  - Interfețe
    - Metode de tip set

Implementare

```
class Serviciu {
...
}

class Client {
   Serviciu serviciu;

   Client() {
      serviciu = new Serviciu();
   }
...
}
```

```
class Client {
   Serviciu serviciu;

   Client(Serviciu serviciu) {
      this.serviciu = serviciu;
   }
...
}

class Injector {
   void inj() {
      Serviciu s = new Serviciu();
      Client c = new Client(s);
      ...
   }
}
```

Avantaje

- Suport pentru testare
- Cuplare scăzută a componentelor
- Extinderea cu ușurință a aplicațiilor
- Scăderea complexității codului

Dezavantaje

- Identificarea dinamică a tipurilor (la execuție)
- Complexitatea învățării
- Apariția excepțiilor (la execuție)
  - de la compilare

Implementări existente

- Autofac
- Dagger
- Guice
- LightInject
- Ninject
- Spring
- Unity

Value Object

Value Object

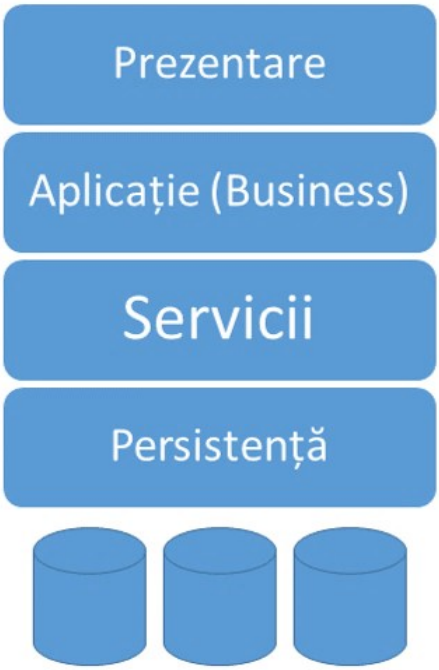
- Obiecte simple, a căror egalitate nu se bazează pe identitate
- Similar tipurilor primitive simple
- Valori monetare, date, constante enumerative etc.
- Uzual, se transmit prin valoare și nu prin referință
- Uzual, aceste obiecte sînt imutabile

Modele arhitecturale (cont.)

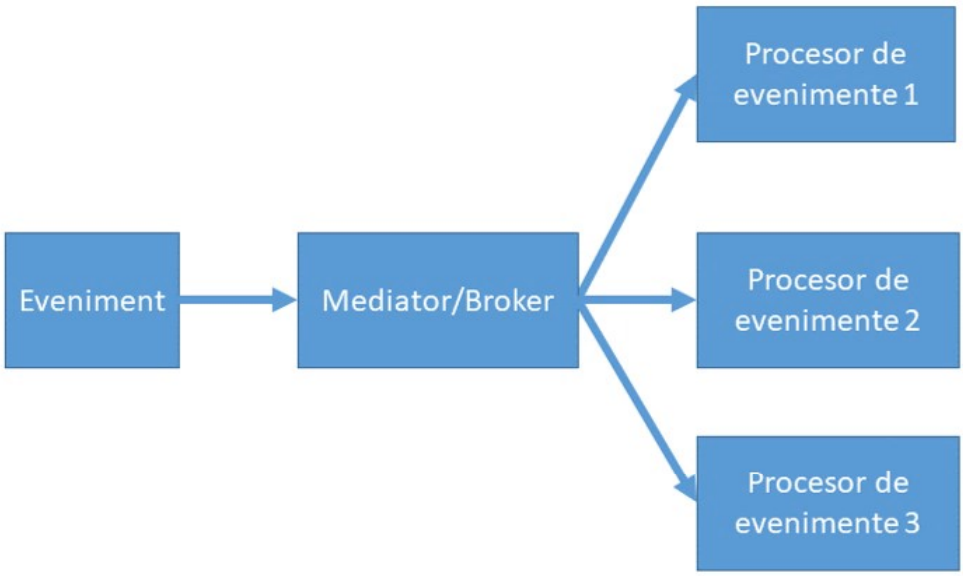
Modele arhitecturale (cont.)

- Arhitectură stratificată
- Arhitectură condusă de evenimente
- Arhitectură de tip micro-kernel
- Arhitectură bazată de microservicii
- Arhitectură bazată pe cloud (space-based)

Arhitectură stratificată



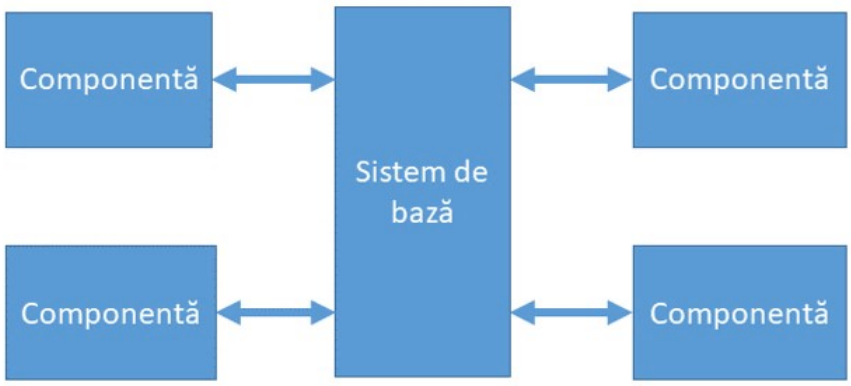
Arhitectură condusă de evenimente



Componente

- **Eveniment**
  - Inițial
  - De procesare
- **Mediator**
  - Evenimentele sînt gestionate de o singură entitate
  - Util în situațiile în care este necesară gestionarea secvenței de prelucrare a evenimentului
- **Broker**
  - Mesajele sînt distribuite într-o manieră înlănțuită
  - Procesări simple, care nu necesită un control global
- **Canale**
  - Permit transmiterea mesajelor asincrone către procesor
  - Uzual, cozi de mesaje
- **Procesor de evenimente**
  - Include logica aplicației (business)
  - Componente decuplate
  - Prelucrează evenimentele

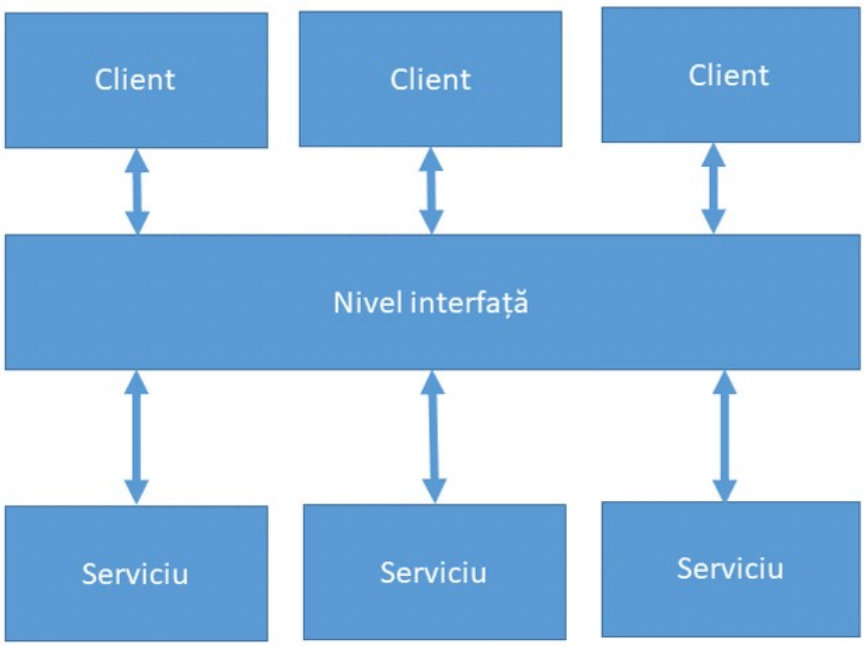
Arhitectură de tip micro-kernel



Componente

- Sistem de bază
  - Include logica de bază a aplicației
  - Pun la dispoziție un mecanism de conectare și identificare a componentelor adiționale
- Componente
  - Module de sine-stătătoare, independente
  - Includ prelucrări specializate
  - Extind funcționalitățile sistemului de bază

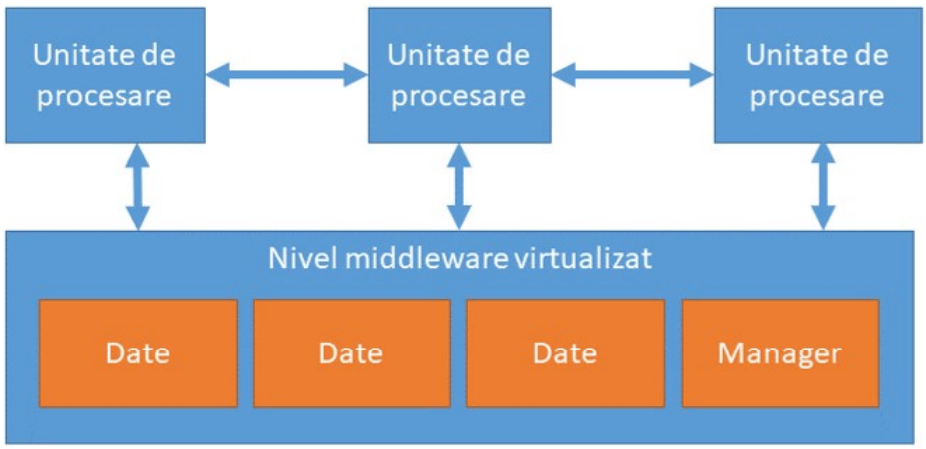
Arhitectură bazată de microservicii



Componente

- **Client**
  - Inițiază cereri la nivelul aplicației
- **Nivel interfață**
  - Asigură comunicarea dintre clienți și aplicație
  - Interfață de programare a aplicațiilor (API)
  - Aplicație (interfață utilizator)
  - Broker de mesaje
- **Servicii**
  - Componente dedicate, independente
  - Nivel de granularitate diferit
    - o singură acțiune
    - implementări parțiale ale logicii aplicației

Arhitectură bazată pe cloud (space-based)



Componente

- **Unități de procesare**
  - Componente ale aplicației
  - Granularitate diferită
    - Aplicații de complexitate redusă
    - Componente care implementează diferite funcționalități ale aplicației
  - Includ
    - modulele aplicației
    - date în memorie asociate
    - suport pentru persistență/replicarea datelor
- **Nivel middleware virtualizat**
  - Asigură comunicarea între componente
  - Suport pentru sincronizare și prelucrarea cererilor
  - Includ componente pentru
    - Mesagerie (gestiunea cererilor și a sesiunilor)
    - Date (asigură replicarea datelor la nivelul aplicației)
    - Procesare (prelucrarea și coordonarea cererilor)
    - Managementul unitățilro de proiectarea (inițializare, terminare, monitorizare etc.)

Criterii de comparație

- Testabilitatea
- Scalabilitatea
- Implementarea
- Performanțe
- Agilitatea
- Instalarea

Modele pentru aplicații de întreprindere

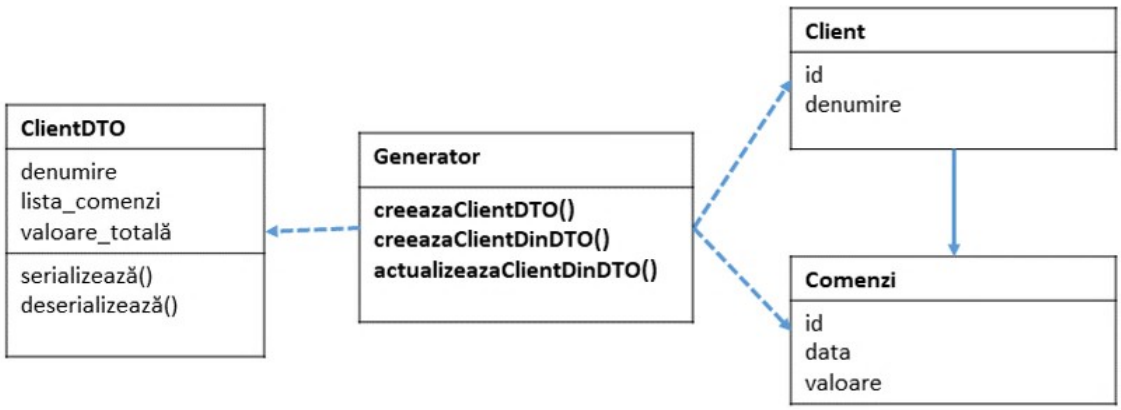
- Transferul obiectelor
  - Data Transfer Object
- Specifice sursei de date
  - Active Record
  - Data Mapper
  - Table Data Gateway
  - Row Data Gateway
  - Data Access Object
- Asociere metadate Obiect/Sursă de date
  - Query Object
  - Repository
  - Object Relational Mapping
- Obiect/Sursă de date
  - Unit of Work
  - Identity Map
  - Lazy Load

Data Transfer Object

Data Transfer Object

- Transferul datelor între diferite subsisteme/niveluri ale aplicației
- Reducerea numărului de parametri ai metodelor
- Returnarea mai multor valori dintr-o metodă
- Reducerea numărului de apeluri/cereri
- Datele sînt încapsulate într-un obiect
- Uzual, obiectele includ doar date, fără logică
- Obiecte serializabile

Diagrama de clase



Componente

- ClientDTO
  - Obiectul de tip DTO obținut pe baza obiectelor din model Client și Comenda
- Generator
  - Generează obiect de tip DTO pe baza modelului
  - Creează obiecte din model pe baza DTO
  - Bazat pe modelul Mapper
- Client, Comanda
  - Obiecte ale modelului

Discuții

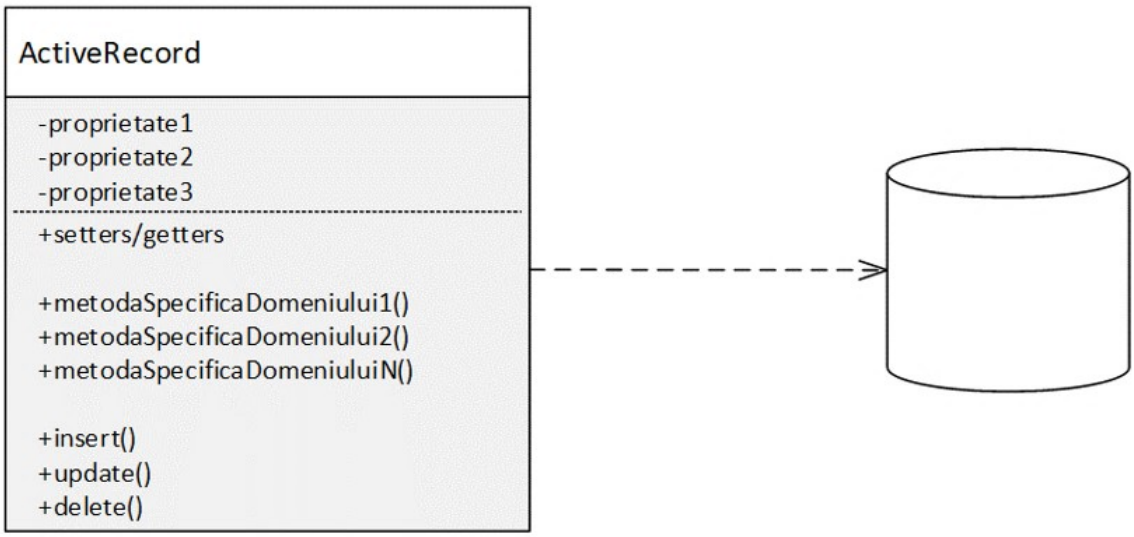
- Formă simplificată a obiectelor domeniului
- Tipuri simple de date
- Date agregate din mai multe surse
- Un caz particular este Result Set

Active Record

Active Record

- Obiect asociat unei înregistrări dintr-o tabelă/view a unei baze de date
- Încapsulează accesul la baza de date
- Include și logica domeniului pentru datele reprezentate

Diagrama de clase



Discuții

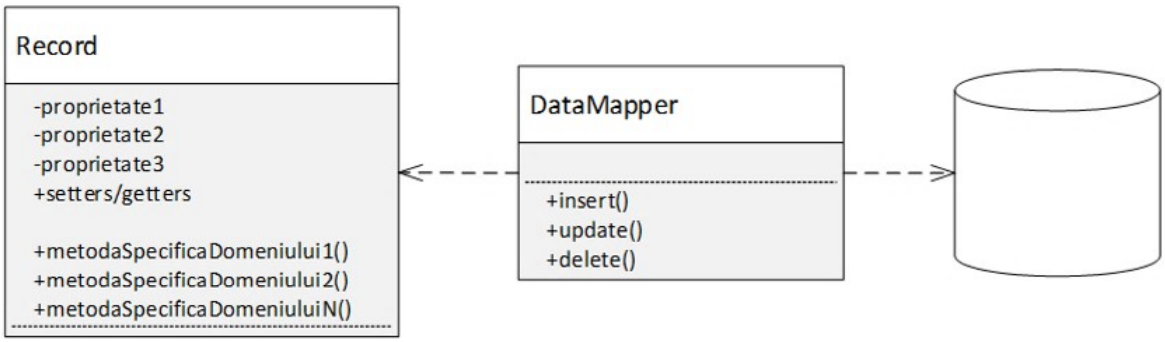
- Suport pentru aplicații de complexitate mică/medie
- Util pentru aplicații de tip CRUD
- Încalcă principiul SRP (Single Resposibility Principle)
- Nivelul asociat logicii domeniului este strîns legat de nivelul de persistență

Data Mapper

Data Mapper

- *Nivel intermediar* care separă obiectele în memorie de baza de date
- Izolează cele două niveluri (aplicație și persistență) între ele și asigură transferul de date între acestea
- Cele două niveluri (aplicație și persistență) sînt independente între acestea, dar și față de obiectul de tip Data Mapper
- Decuplează clasele modelului de nivelul de persistență
- Permite transferul obiectelor între aplicație și baza de date

Diagrama de clase



Discuții

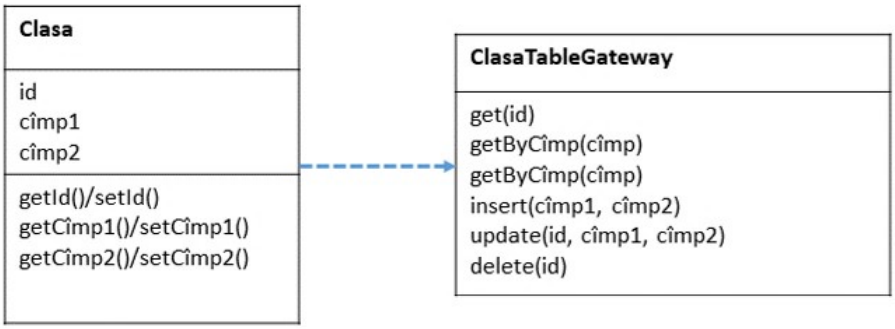
- Este respectat Principiul SRP (Single Resposibility Principle)
- Nivelul asociat logicii domeniului nu este legat de nivelul de persistență
- În forma de bază, include referințe la biblioteci specifice nivelului de persistență

Table Data Gateway

Table Data Gateway

- Obiect care gestionează accesul la o tabelă dintr-o bază de date
- Pune la dispoziție metode pentru:
  - Adăugare
  - Regăsire
  - Modificare
  - Ștergere
- Metodele operează cu parametrii asociați membrilor clasei
- O singură instanță gestionează toate înregistrările dintr-o tabelă

Diagrama de clase



Discuții

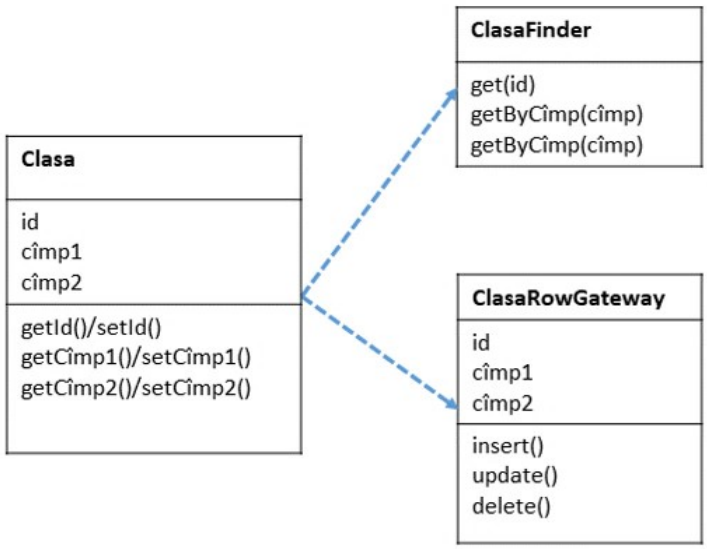
- La nivelul TDG nu există legătură cu entitățile din domeniu
- Include comenzile SQL necesare operațiilor la nivelul tabelei

Row Data Gateway

Row Data Gateway

- Obiect care gestionează accesul la o înregistrare dintr-o tabelă
- O singură instanță pe înregistrare
- Include membri asociați cîmpurilor clasei
- Pune la dispoziție metode pentru
  - Adăugare
  - Modificare
  - Ștergere

Diagrama de clase

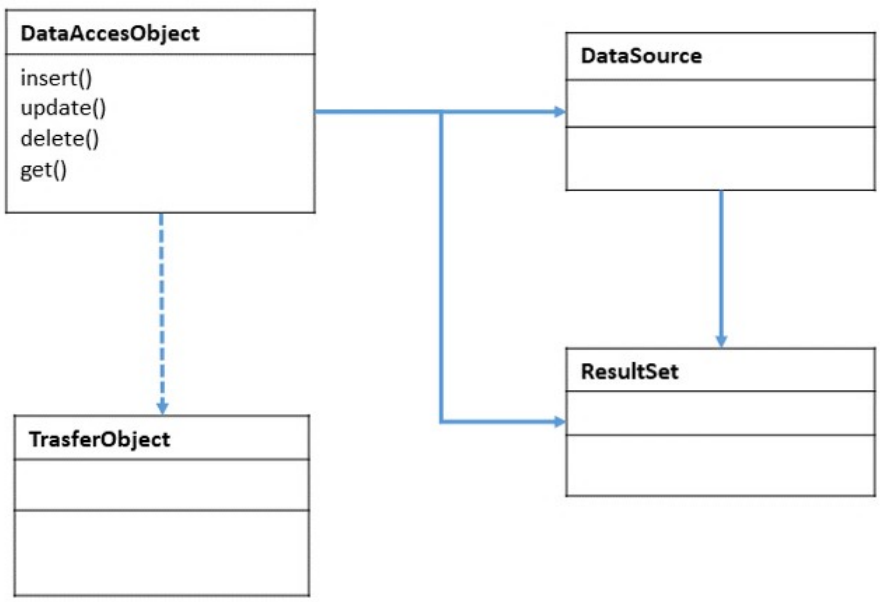


Data Access Object

Data Access Object

- Similar modelelor anterioare de acces la date
- Uzual, gestionează obiect de tipul DTO (Data Transfer Object)
- Acționează ca un adaptor între sursa de date și componente

Diagrama de clase



Componente

- **DataAccessObject**
  - Este creat de către **Client**
- **TransferObject**
  - Obiectul utilizat în tranzacții
  - Utilizat de către **Client**
- **DataSource**
  - Sursa de date
    - Bază de date
    - Fișiere
    - Servicii etc.
  - Este accesată de către **DataAccesObject**, utilizînd-se un **TransferObject**
- **ResultSet**
  - Setul de date din memorie
  - Asociat rezultatului unei interogări a sursei de date
  - Pe baza acesteia se creează obiectul de tip **TransferObject**

Discuții

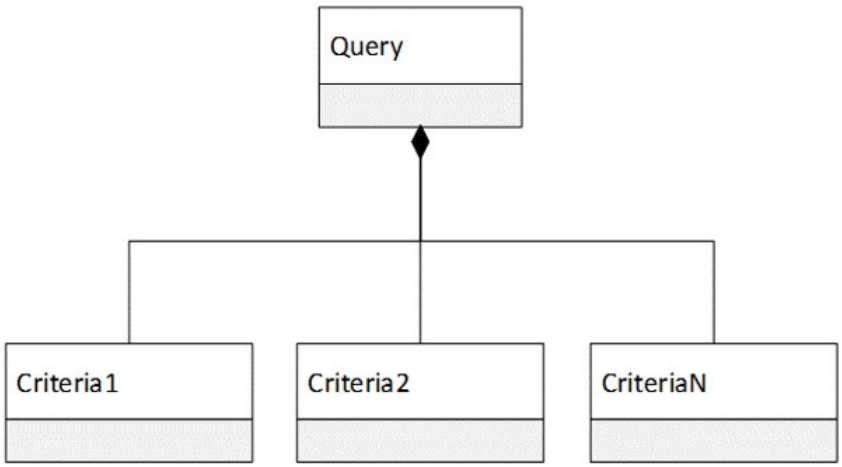
- La nivelul TDG nu există legătură cu entitățile din domeniu
- Include comenzile SQL necesare operațiilor la nivelul tabelei

Query Object

Query Object

- Obiecte asociate interogărilor dintr-o sursă de date
- Implementate în limbajul domeniului
- Interfața independentă de limbajul de interogare al bazei de date
- Uzual, se utilizează împreună cu modelul *Repository*
- Modelele de proiectare *Interpreter*, *Specification*
- Translatare din limbajul domeniului în limbajul de interogare al bazei de date

Diagrama



Componente

- Query
  - Clasa asociată obiectului de tip cerere
  - Generează comenzi specifice sursei de date pe baza criteriile furnizate
  - Utilizează obiecte din dimeniu
- Criteria1, Criteria2, CriteriaN
  - Criteriile specifice interogării
  - Includ cîmpuri prin intermediul cărora se pot genera criterii de selecție
    - operator, cîmp, valoare

Sumar

- Modele specifice nivelului de persistență a datelor (2)

Repository

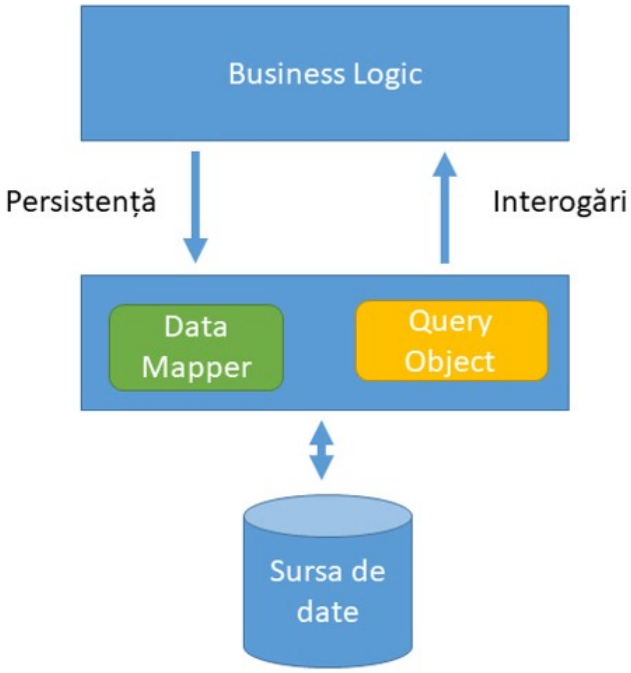
Repository

- Colecție de obiecte, în memorie, de tipul domeniului
- Nivel intermediar între domeniu și nivelul de asociere a datelor (data mapping)
- Suport pentru
  - Adăugare
  - Modificare
  - Ștergere
  - Selecție
- Suport pentru Interogări complexe
- Suport pentru diferite surse de date

Utilizare

- Număr ridicat de obiecte ale domeniului
- Surse multiple de date
- Controlul codului pentru selecția datelor (interogări)
- Optimizarea regăsirii datelor

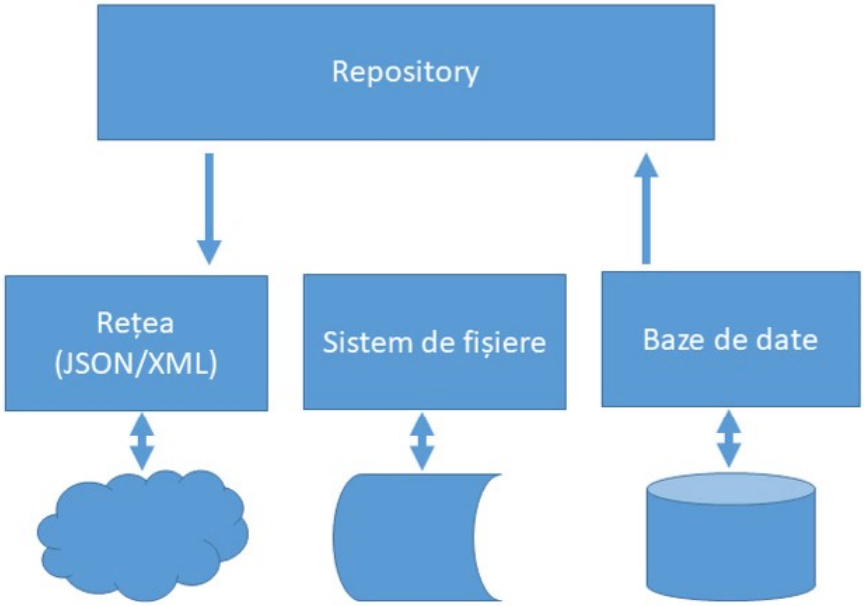
Diagrama



Implemetare

- Repository per entitate
  - O clasă specializată de tip Repository per entitate
- Implementare generică
  - O singură clasă de tip Repository
  - Poate fi utilizată pentru orice tip de obiect din domeniu

Alte abordări

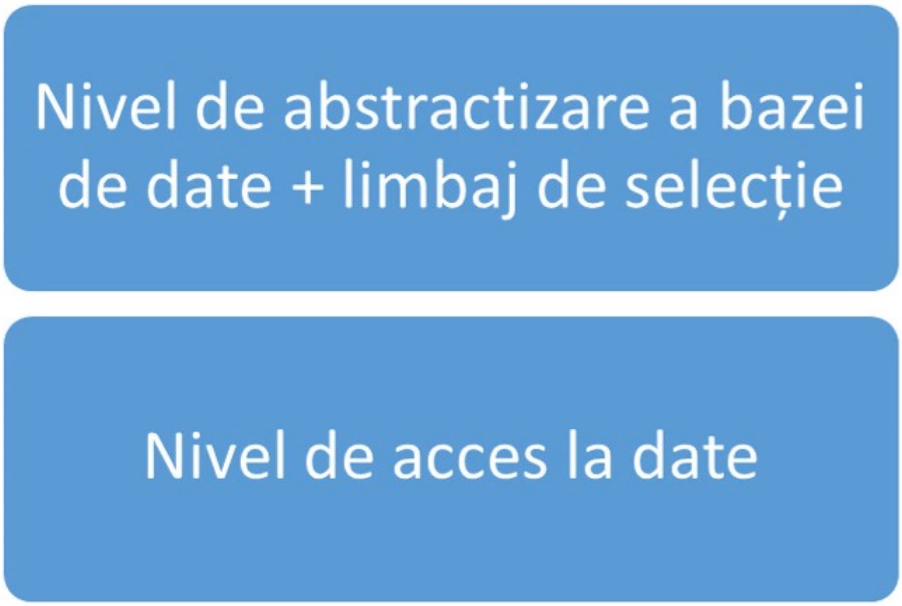


Object Relational Mapping

ORM (Object Relational Mapping)

- Nivel intermediar dintre *baza de date relațională* și aplicație
- Include și un nivel de abstractizare dedicat interogării datelor
- Accesul la baza de date este implementat prin diferite mecanisme

Diagrama



Implementări existente

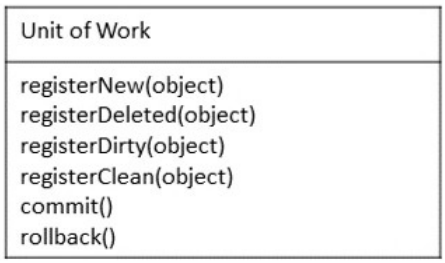
- Django ORM
- OpenJPA
- Hibernate
- Doctrine
- Yii

Unit of Work

Unit of Work

- Gestionează o listă de obiecte cu modificările intervenite în utilizarea acestora
- Toate modificările aplicate obiectelor se vor reflecta, la un moment dat, în baza de date, printr-o singură tranzacție
- În caz de nereușită la scrierea în baza de date, se revine la starea inițială
  - Este necesară urmărirea operațiilor efectuate în baza de date
- Uzual, este utilizat împreună cu Repository

Diagrama de clase

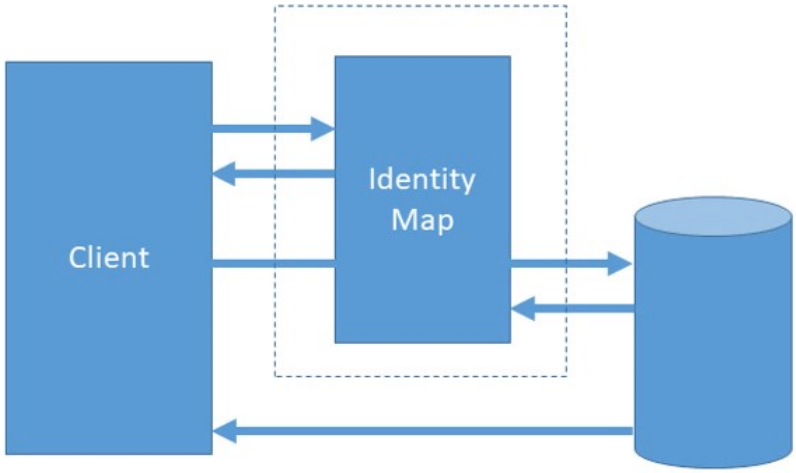


Identity Map

Identity Map

- Operațiile de selecție din baza de date sînt costisitoare din punct de vedere al performanțelor
- În acest sens, se va reține fiecare obiect încărcat într-o structură ce permite regăsirea rapidă (Map)
- La solicitarea unui obiect, mai întîi se verifică în structura de date dacă a fost încărcat în prealabil
- Dacă este identificat obiectul în structură, acesta este returnat
- Dacă nu este identificat obiect, acesta se preia din baza de date și se salvează în structura de date corespunzătoare

Diagrama dinamică



Implementări

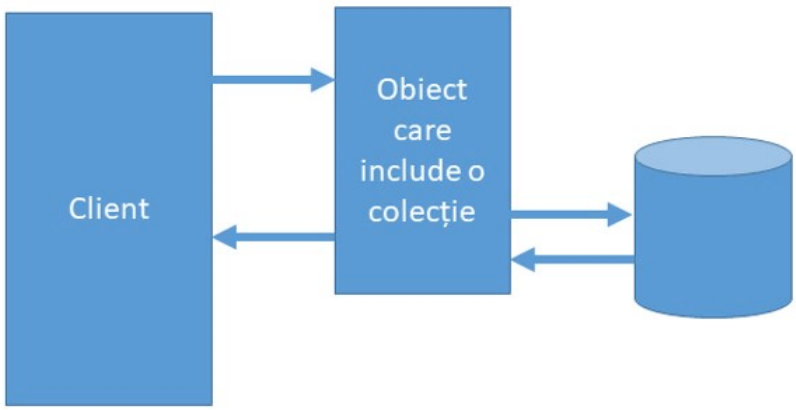
- Selectarea cheii
- Asociate unei tabele sau mai multor tabele
- Pot fi generice sau concrete
- Asociate componentelor de acces la date

Lazy Load

Lazy Load

- Operațiile de selecție din baza de date sînt costisitoare din punct de vedere al performanțelor
- Obiectele sînt încărcate în memorie în momentul în care aceste sînt accesate
- Se bazează pe modelul Lazy Initialization

Diagrama dinamică



Implementare

- Un obiect care conține o colecție de obiecte asociate
- Inițial, lista nu este încărcată și este nulă
- La solicitarea listei, aceasta este inițializată prin preluarea înregistrărilor corespunzătoare din baza de date și returnată clientului
- Soluții
  - Lazy Initialization
  - Virtual Proxy

Referințe

- .NET Design Patterns, http://www.dofactory.com/net/design-patterns
- Data & Object Factory, *Gang of Four Software Design Patterns*, Companion document to Design Pattern Framework™ 4.5, 2017
- Data & Object Factory, *Patterns in Action 4.5*, A pattern reference application, Companion document to Design Pattern Framework™ 4.5, 2017
- Design Patterns | Object Oriented Design, http://www.oodesign.com/
- Design patterns implemented in Java, http://java-design-patterns.com/patterns/
- R. Fadatare, *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall, 2013
- M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford, *Patterns of Enterprise Application Architecture*, Addison Wesley, 2002
- E. Freeman ș.a, *Head First Design Patterns*, O'Reilly, 2004
- J.D. Meier et al, Application Patterns, http://apparch.codeplex.com/wikipage?title_=Application%20Patterns, 2009
- M. Richards, Software Architecture Patterns, O'Reilly, 2015
- D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, Volume 2, John Wiley & Sons, 2000
- A. Shivets, *Design Patterns Made Simple*, http://sourcemaking.com
- Stack Overflow, https://stackoverflow.com/
- D. Trowbridge et. al, *Enterprise Solution Patterns Using Microsoft .NET*, Version 2.0, Microsoft, 2003