Arhitectura sistemelor soft enterprise. Platforma .NET

Curs 6

Pattern-uri de logica a domeniului

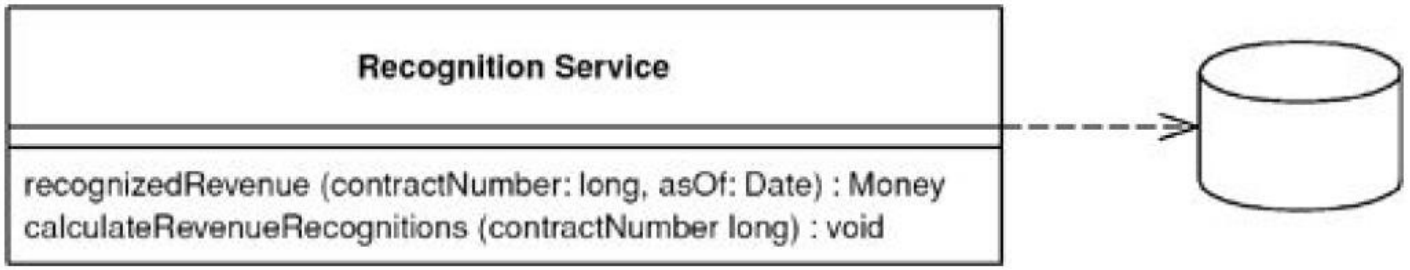
Continut

- Transaction script
- Domain Model
- Table Module
- Service Layer

- Bibliografie: PoEAA, cap 9

Pattern-uri de Domain Logic

- Transaction Script: organizeaza business logic sub forma de functii
  - fiecare din acestea manipuleaza cate o cerinta de la nivelul prezentare
- Aplicatia este vazuta ca o succesiune de ***tranzactii*** care se cer dinspre interfata utilizator

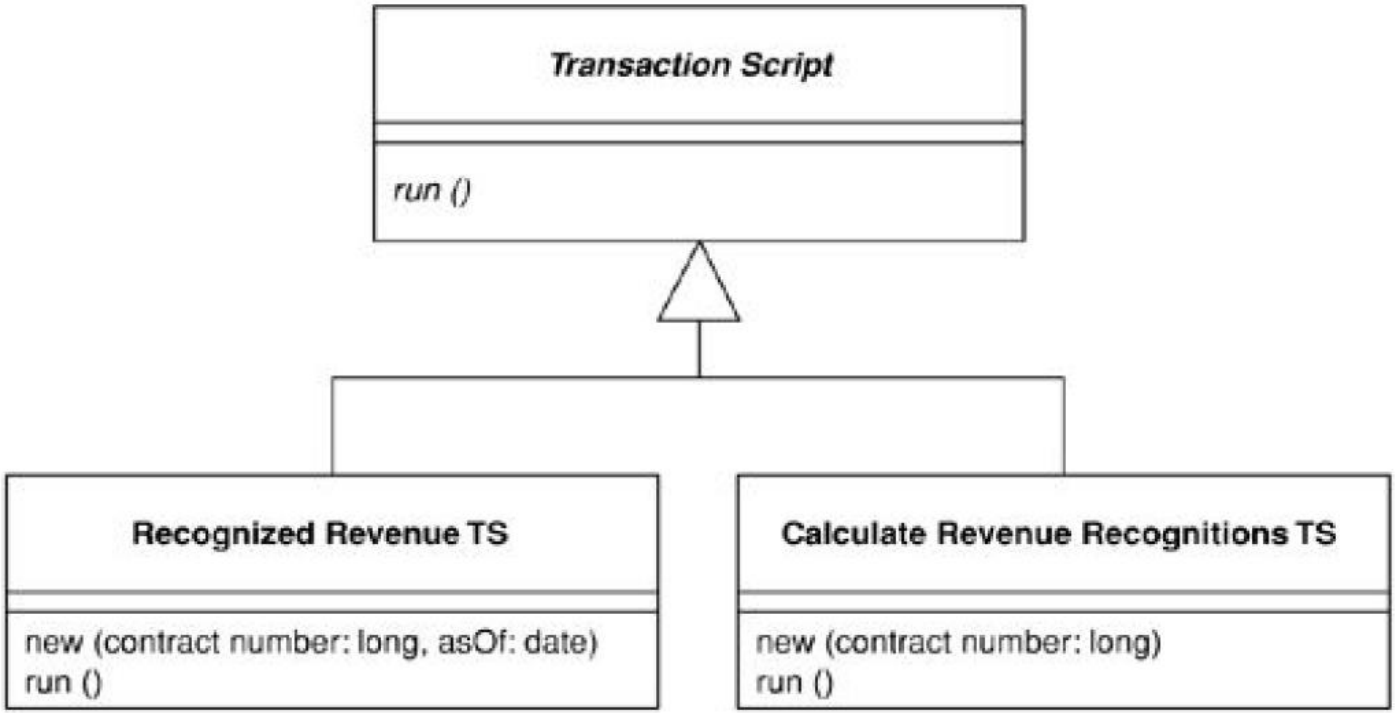


Transaction script

- Exemplu: pentru inchirierea unei camere la hotel, tranzactia verifica daca e libera camera, calculeaza pretul, adauga inregistrari in baza de date despre ocuparea ei
- Apelurile fie se fac direct la serverul de baze de date, fie via un nivel de acces la date
- Metodele (tranzactiile) sunt dictate de diagrama use-case
- E posibila structurare in module, pentru regasire sau reutilizare usoara
- De preferat izolare fata de prezentare si accesul la sursa de date => posibilitate de testare si independenta de bibliotecile/framework-urile folosite pentru GUI

Transaction script

- Organizarea pe clase:
  - Varianta cea mai simpla: scripturi de tranzactii grupate pe clase; clasele sunt structurate pe subiectul de lucru
  - Fiecare script e scris in propria lui clasa: design pattern-ul Command (design pattern obiectual)



Transaction script

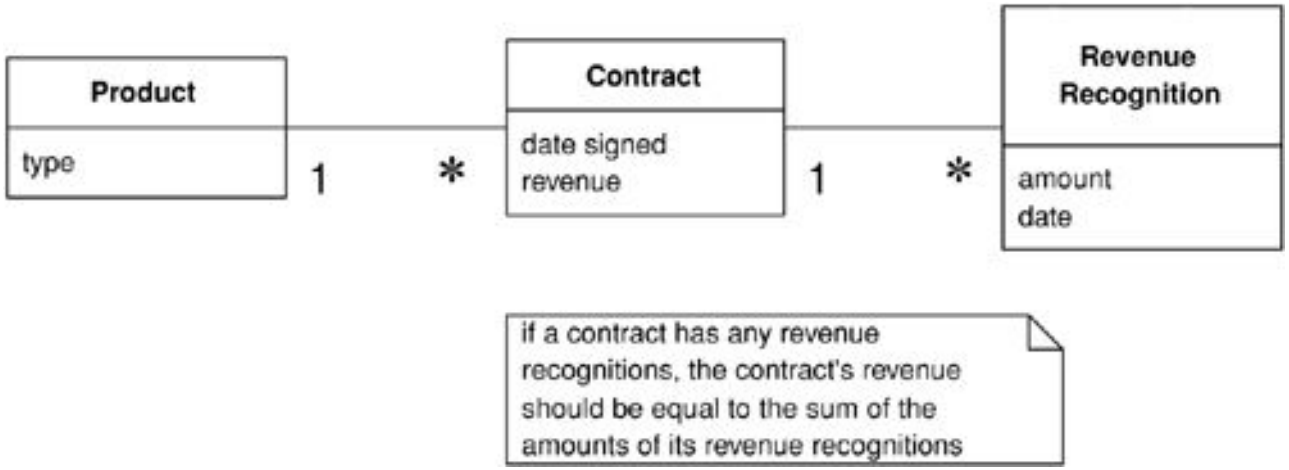
- Motivatia alegerii TS: simplitatea evidenta => usurinta in intelegere si intretinere
- Daca partea de modelare a logicii aplicatiei devine complexa, trebuie refacuta arhitectura (architectural refactoring)
- Dificil de zis cand lucrurile devin prea complexe = cand ar trebui reconsiderata optiunea curenta de arhitectura? Ce este complexitatea ciclomatica?
- Calculul veniturilor aferente contractelor: proces simplu pentru majoritatea tranzactiilor
- Dar in cazul in care suma se plateste in avans pentru un an, cum se calculeaza veniturile pana la o anumita perioada? Suma/12 luni * numarul lunii curente? Sau se considera incasarea doar la sfarsitul perioadei? Se considera inflatia?

Transaction script

- Pentru problema calculului de venit: regulile pot fi multe si volatile
- Unele reguli date de legile in vigoare, altele de standarde interne (specifice business-ului), sau in functie de tipul de client implicat, sau de perioada de efectuare a tranzactiei (e.g. perioade cu TVA diferite, perioade cu reduceri de preturi etc.)
- Problema calculului venitului poate deveni complexa => o singura procedura evolueaza in multiple intructiuni “if”
- Solutie: folosirea claselor polimorfice (sau strategy design pattern)

Transaction script

- Exemplu: vanzarea a trei feluri de produse: procesoare de text, servere de baze de date si programe de calcul tabelar
- Pentru procesor de text, regula este: venitul se inregistreaza in momentul platii (toti banii se dau odata)
- Pentru server BD: o treime azi, o treime in 30 de zile, restul peste 60 de zile
- Pentru aplicatie de calcul tabelar: o treime azi, o treime peste 60 de zile, restul peste 90 de zile
- Schema conceptuala:



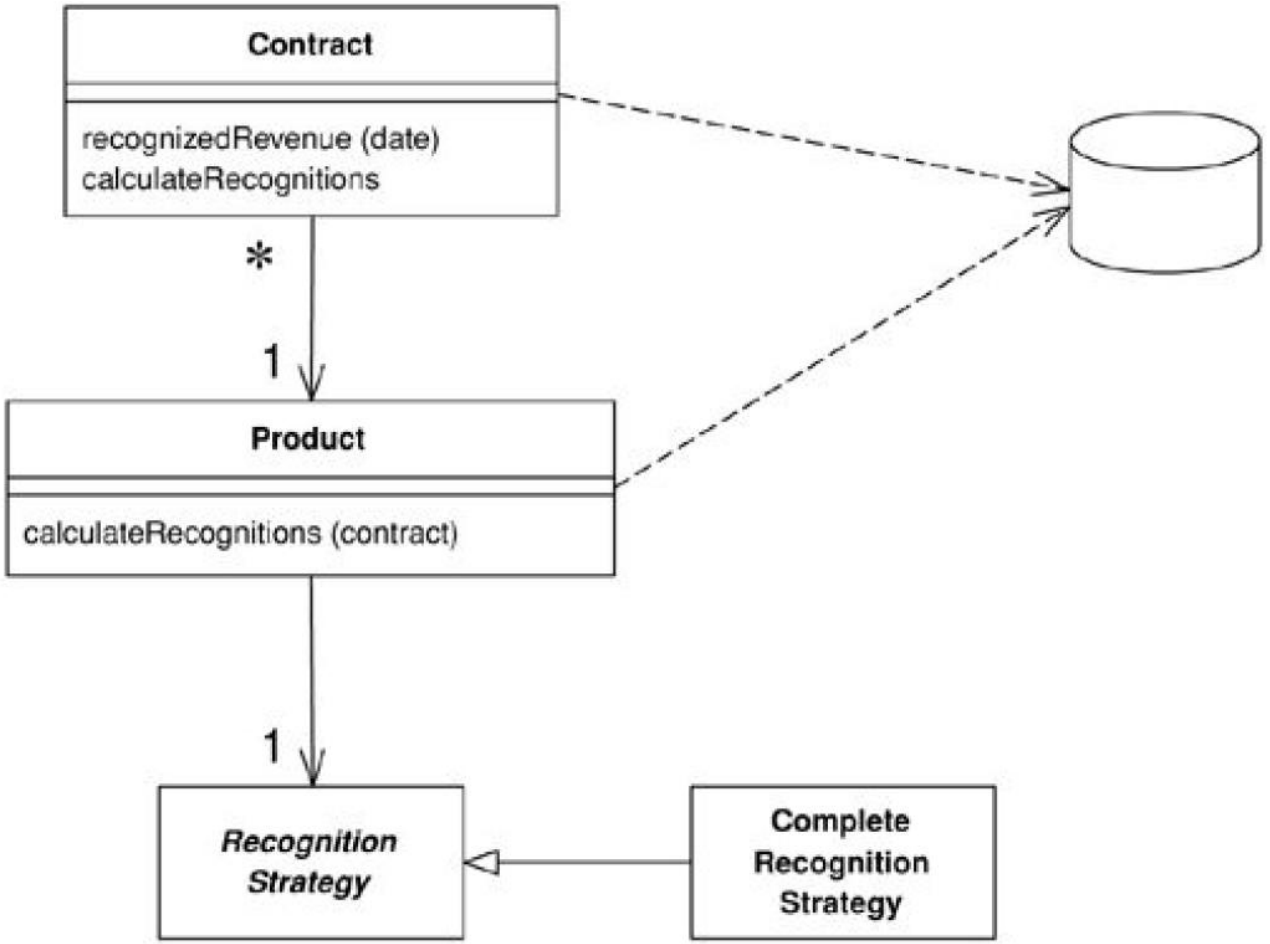
Transaction script

- 2 scripturi de tranzactii: unul pentru a spune care este valoarea incasata pana la o anumita data, pentru un anumit contract, altul pentru a insera in BD ratele ce trebuie platite pentru un anumit contract
- Posibilitate: script SQL (procedura stocata) care returneaza direct valoarea calculata
- …sau creare de cod SQL din script si trimitere catre serverul de baze de date
- Exemplu de cod: PoEAA, pag 92+

Domain Model

- Esenta: un model obiectual care incorporeaza atat **datele** cat si **comportamentul** ce opereaza asupra lor
- Se expune strict partea de interes pentru cel ce acceseaza clasa din exterior = incapsulare; detaliile specifice de implementare vor fi ascunse
- Se remarca separarea conceptelor (vezi pagina urmatoare): Contract, Produs, *Strategie*, Strategie concreta
- Rezultat: retea de obiecte interconectate cu responsabilitati bine definite
- Obiecte "fine", cu o interfata granulara:
  - Metode pentru proprietati
  - Metode **polimorfice** pentru implementare de operatii

Domain Model



Domain Model

- Este posibil ca obiectele sa se asocieze natural cu structura bazei de date
- Dar nu este obligatoriu (si deseori nici recomandat): in BD avem proces de normalizare, la clase - nu
- Un model bogat al domeniului va arata diferit de modelul relational
  - Diferentele pot fi reduse, dar nu neaparat anulate de utilizarea de view-uri in BD
- Rezultat: 2 tipuri de Domain Model: unul **simplu**, apropiat ca structura de BD; altul **complex**, usor de extins, dar mai greu de asociat cu baza de date

Domain Model

- Daca baza de date este orientata pe obiecte: efortul de asociere obiecte – date e zero
- Pentru model relational – asociere naturala sau creata de programator
- Model simplu: Active Record
- Model complex (= modificari de cerinte, comportament variat etc.): Data Mapper
- Cum se alege: varianta vizata trebuie sa fie usor de **testat** si **extins** relativ la cerintele proiectului
- Risc potential: clasele sa devina prea mari => refactoring

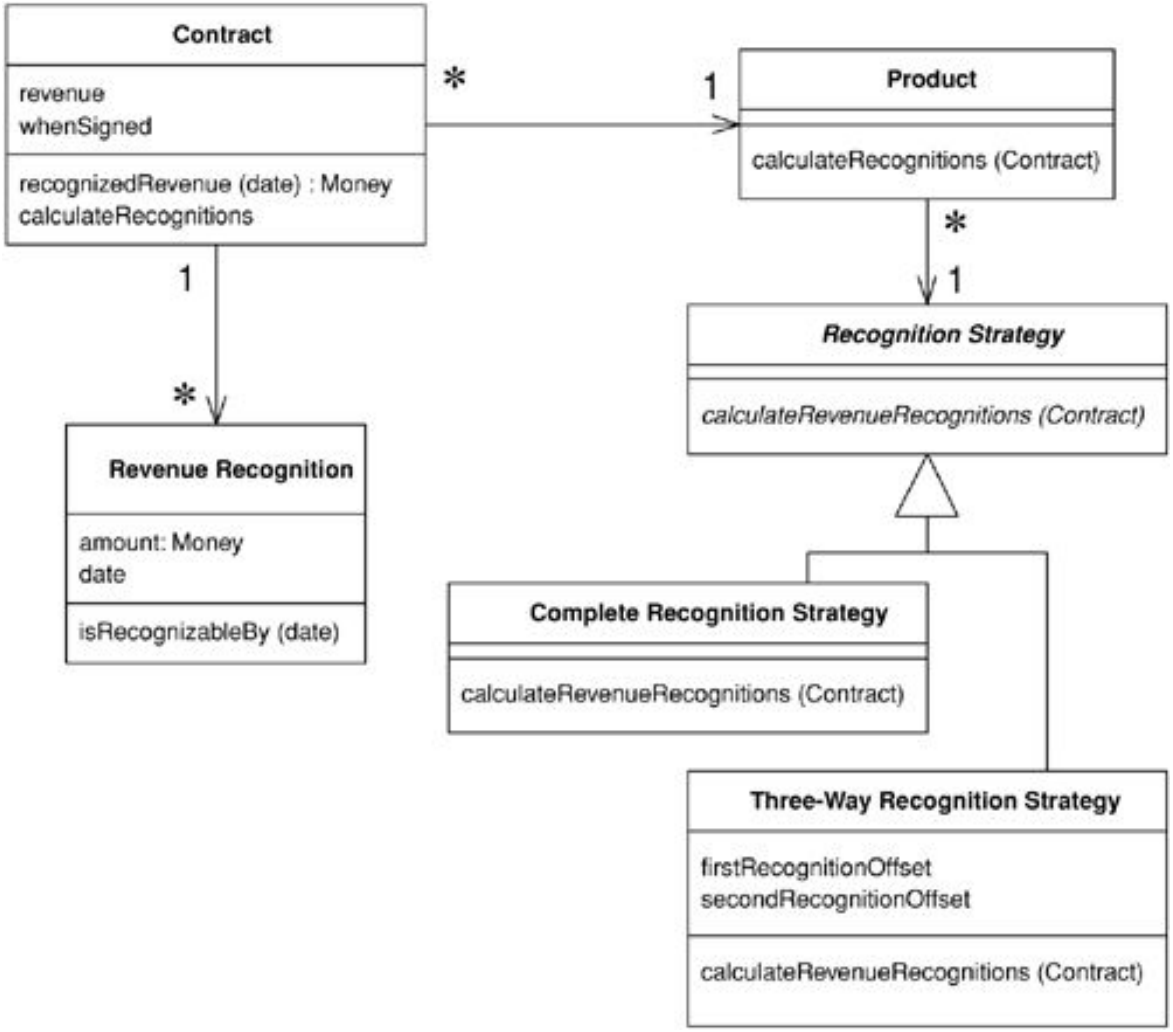
Domain Model

- Daca se cere persistarea starii obiectelor intre doua apeluri din cadrul unei tranzactii de business => pastrarea starii (client session state, server session state, database session state)
- Mod de stapanire a complexitatii: design patterns (e.g. factory method, abstract factory, strategy, template)
- Mare atentie la duplicarea comportamentului => risc de inconsistenta; se evita prin definirea clara a responsabilitatilor claselor
- Indicatie: sa se studieze comportamentele legate de utilizare (specifice unor anumite ecrane, de exemplu); duplicarile pot fi astfel usor observate

Domain Model

- Cand se foloseste?
  - Reguli de business complicate sau care se schimba des
  - Calcule complexe dpdv al implementarii, workflow complex
  - Echipa de dezvoltatori stie/poate invata OOP + object oriented design + Design Patterns
- Atentie la dependentele de mediul de rulare (discutie in PoEAA despre EJB)
- Dificultate: Data Mapper (transformare DM <-> BD)
- Cand NU se foloseste: reguli simple, flux de lucru aproape liniar => transaction script

Domain Model



- Exemplu: PoEAA, pag 97+

Table Module

- O instanta care manipuleaza logica de business pentru mai multe inregistrari dintr-o tabela sau dintr-un view

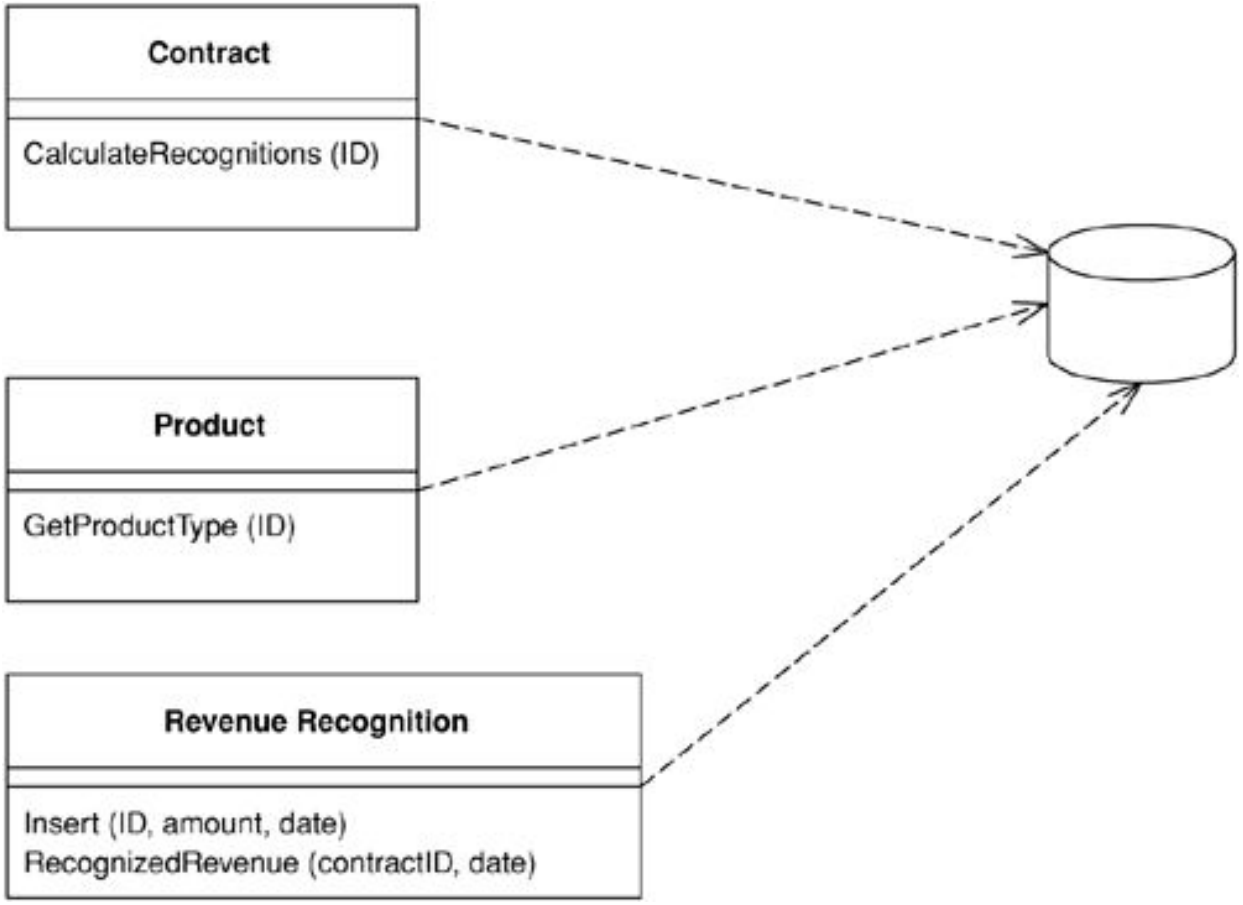


Table Module

- Rezolva simplu una din problemele de Domain Model: complexitatea legarii la date = complexitatea transformarii intre cele doua tipuri de reprezentare
- Rezultat: se grupeaza intr-un singur loc datele si comportamentul asociat (ca la domain model) + aparenta lucrului cu baze de date relationale in RAM
- Clasele de Table Module sunt structurate conform tabelelor/view-urilor din baza de date
- Diferenta majora fata de DM: daca la DM pentru un set de inregistrari ai un set de obiecte care le reprezinta, la TM ai un singur obiect care contine toate inregistrarile de interes

Table Module

- Datele rezultate intr-un obiect de tip Tabel Module se reprezinta printr-un **Record Set**
- Table Module furnizeaza o interfata explicita pentru accesarea datelor din BD
- De regula: un table module pentru fiecare tabel
- In plus: table module pentru view-urile interesante (dar ceva mai mult de lucru la modificare/inserare/stergere, deoarece implica operatii pe mai multe tabele)

Table Module

- Deseori, mai multe Table Modules actioneaza pe acelasi RecordSet (mai jos: theDataSet)

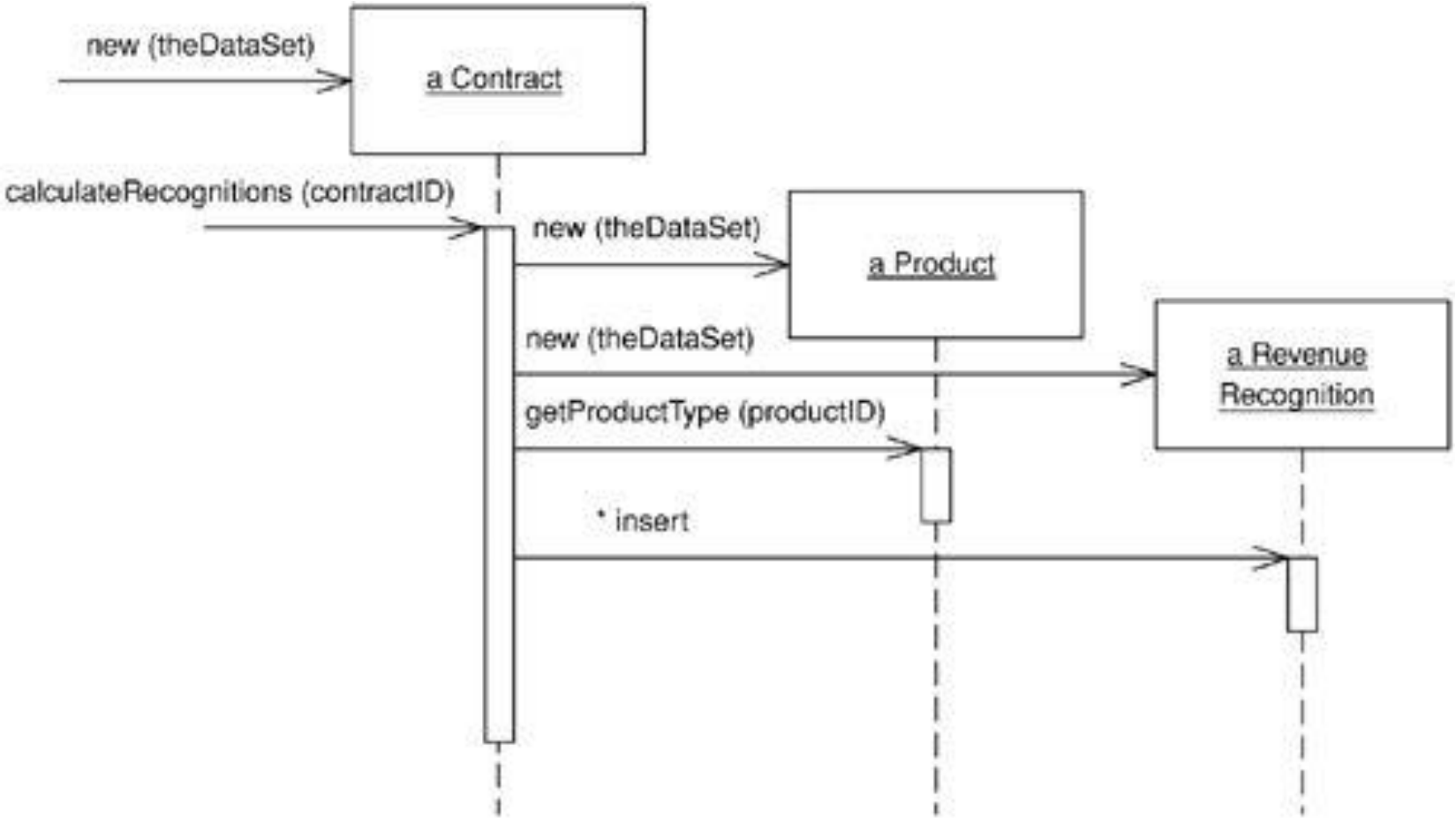


Table Module

- Table module: gandit ca si clasa statica sau clasa instantiabila
- La a doua varianta: se poate face initializarea unui obiect table module pe baza unui RecordSet provenit dintr-o alta interogare
- Tot pentru aceasta varianta: se poate utiliza mostenirea intre clase (reutilizare de cod)
- Un Table Module poate sa includa diferitele interogari care vor popula obiectul
- Alternativa: folosirea unui Table Data Gateway intermediar intre baza de date si Table Module, in stratul de date
- Avantaj: un TDG pentru fiecare sursa de date; dezavantaj: trebuie scrisa o clasa pentru fiecare tabel in parte

Table Module

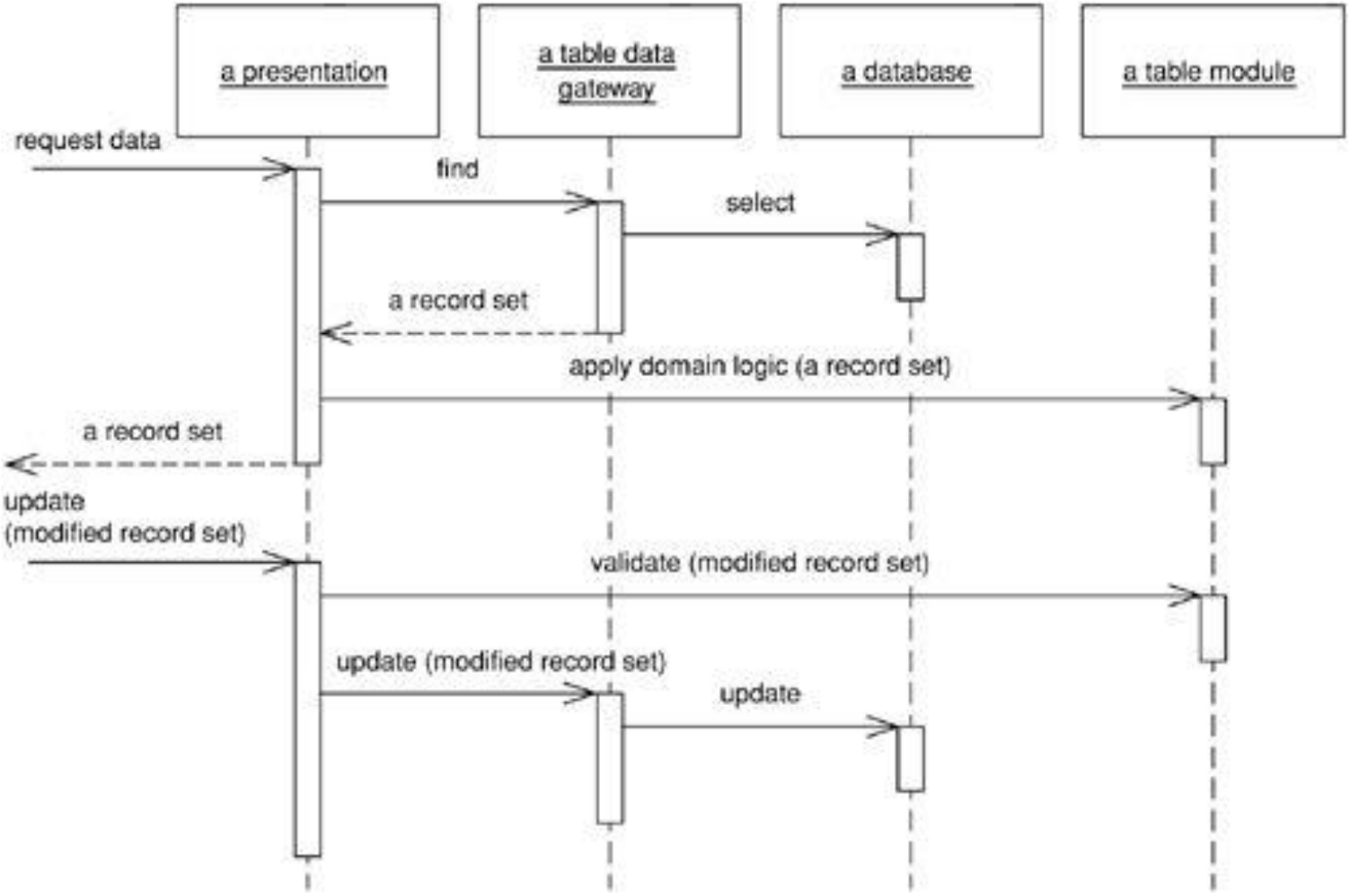
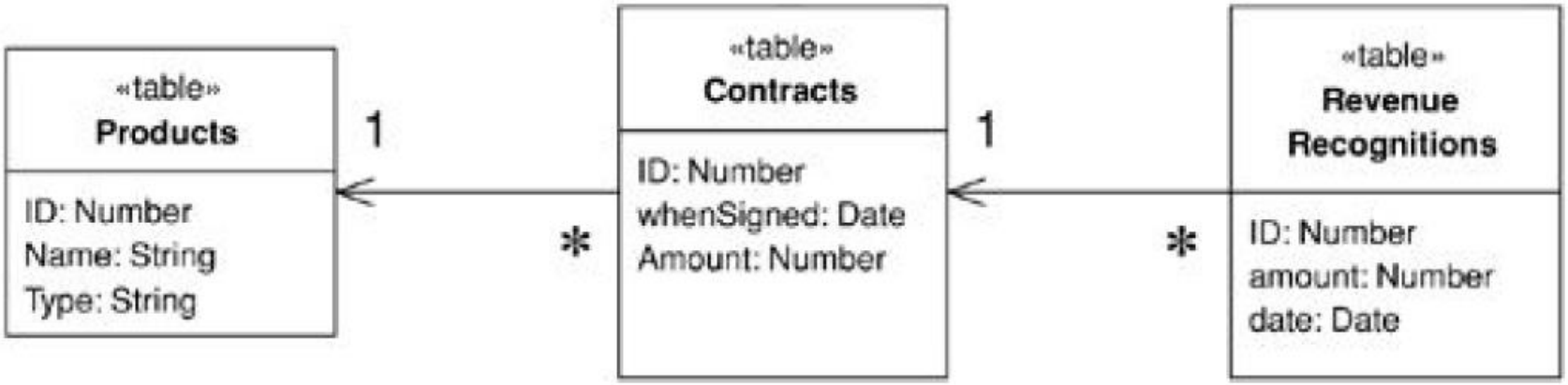


Table Module

- Cand se foloseste:
  - accesarea datelor se face in mod tabular la afisare sau la prelucrare
  - plus: nu se cere de la beneficiar o logica de domeniu complexa, precum la Domain Model
- Exemplu de RecordSet: ADO.NET DataSet = in principal colectie de obiecte DataTable

Table Module

- Exemplu in .NET:



Schema bazei de date pentru problema declararii veniturilor

Exemplu: clasa de baza

```
class TableModule{
        protected DataTable table;
        protected TableModule(DataSet ds, String tableName)
        {
                table = ds.Tables[tableName];
        }...}
```

Table Module

- Clasa TableModule este mostenita ulterior:

```
class Contract : TableModule{
    public Contract (DataSet ds) : base (ds, "Contracts") {}
    …
}
```

- Creare de obiect de tip Contract:

```
contract = new Contract(dataset);
```

- Utilizare de indexatori:

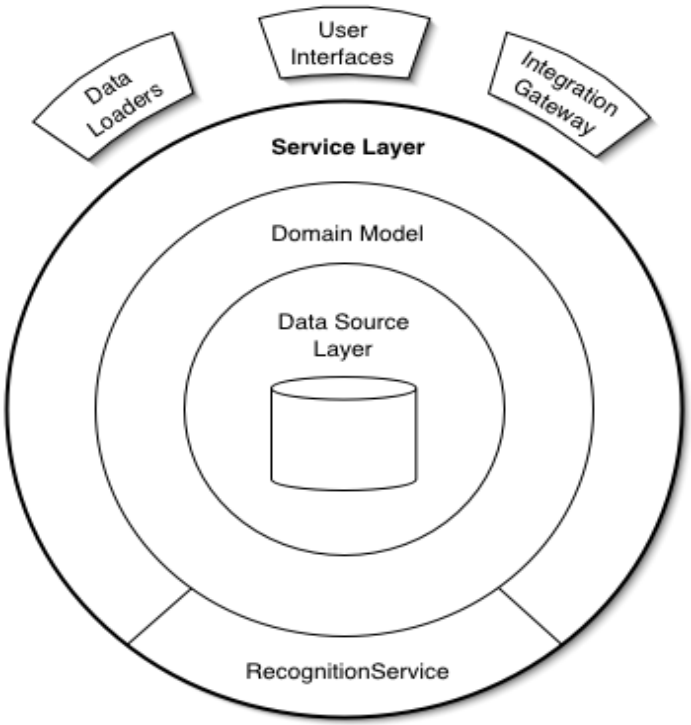
```
public DataRow this [long key] {
    get {
        String filter = String.Format("ID = {0}", key);
        return table.Select(filter)[0];
    }
}
```

Table Module

- Exemplu detaliat: PoEAA, pag 104+

Service Layer

- Defineste granitele unei aplicatii printr-un strat de servicii care stabilesc multimea de servicii disponibile si coordoneaza raspunsul aplicatiei la fiecare operatie
- Business logic = Domain logic + Service layer



Service Layer

- Alte responsabilitati:
  - incapsuleaza logica de business a unei aplicatii (workflow logic)
  - face verificari de validitate
  - controleaza tranzactiile
  - coordoneaza raspunsurile
- I: De ce avem nevoie de Service Layer atunci cand putem accesa Domain Model?
- R: Integrarea de logica specifica aplicatiei (logica de API) reduce gradul de reutilizare a claselor; SL izoleaza functionalitatea ceruta de aplicatie de implementarea procesarii; el contine logica apelurilor

Service Layer

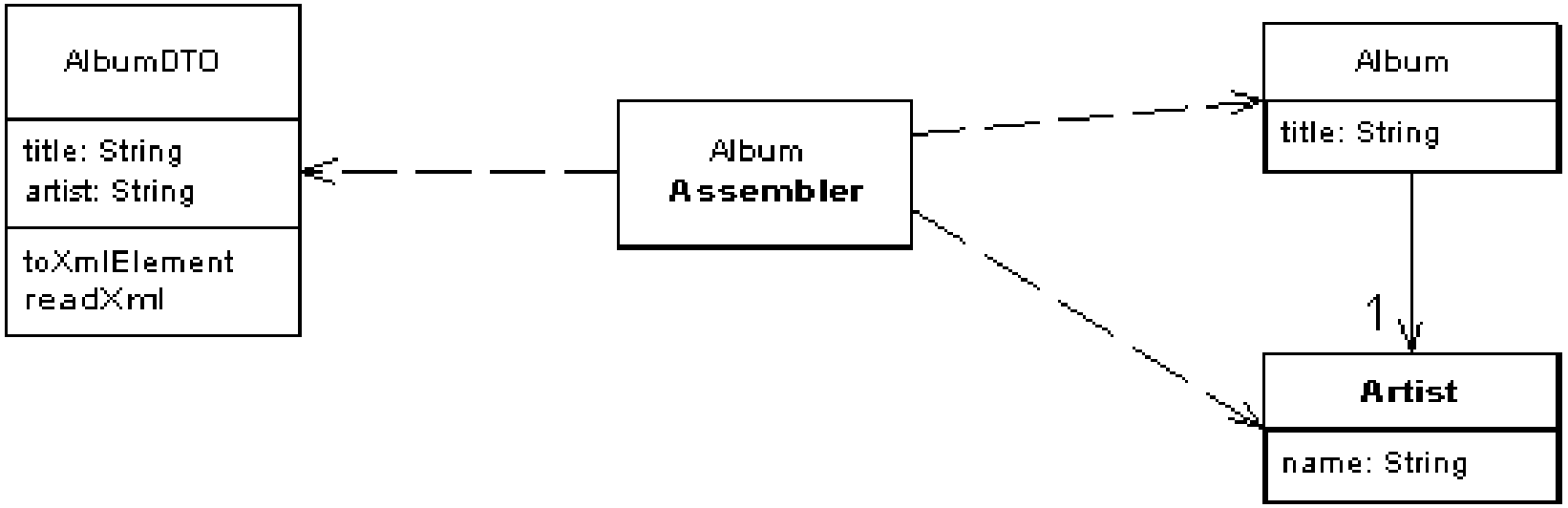
- Variatii de implementare:
  - Domain Façade
  - Operation Script
- Domain Façade:
  - Fatade “subtiri” de apel peste un Domain Model
  - Nu contin business logic
  - Stabilesc (prin publicare) setul de operatii pe care le poate utiliza un client, dar restul este facut de catre Domain Model

Service Layer

- Operation Script:
  - Clase de dimensiuni mari care implementeaza logica aplicatiei dar deleaga apelurile catre obiectele din Domain Logic.
  - Principala realizare: orchestrarea de apeluri catre Domain Logic
  - Poate fi implementat sub forma de script-uri
  - Interfata este “coarse grained”
- Expunerea SL pentru servicii la distanta: o idee buna, dar numai daca asa ceva este cerut explicit

Service Layer

- Apeluri la distanta: problema este cum anume se da raspunsul catre apelant
- Solutie: Data Transfer Object



Service Layer

- Identificarea serviciilor si operatiilor
  - Simple, dictate de necesarul de operatii cerute de client
  - Sunt vizibile din diagrama Use Case a proiectului
  - De multe ori se incepe cu cele 4 operatii de baza: CRUD (Create, Read, Update, Delete)
  - Suplimentar: validare de date
  - Managementul tranzactiilor de business (ACID, rollback), al acceselor concurente
  - Rezultat: cod de tip Transaction script + lucru cu obiecte de Domain Model

Service Layer

- Posibilitate de grupare a functionalitatilor suportate de aplicatie
- Pentru aplicatii mici: un singur grup
- Pentru aplicatii de dimensiuni mari sau pentru care potentialul de dezvoltare ulterioara este mare: multiple subsisteme (e.g. clienti, comenzi, rapoarte)
- *Subsistemele pot fi vandute separat.*

Service Layer

- Cand se foloseste:
  - Avem mai mult de un client al aplicatiei; sau
  - Raspunsuri complexe, formatari in functie de client; sau
  - Orchestrare netriviala de apeluri catre obiectele din DM; sau
  - Cerinta de asigurare a validitatii tranzactiilor de business
- Exemplu: PoEAA, pag 111+