Arhitectura sistemelor soft enterprise. Platforma .NET

Curs 2

Organizarea pe straturi si organizarea logicii de domeniu

Scopul sectiunii

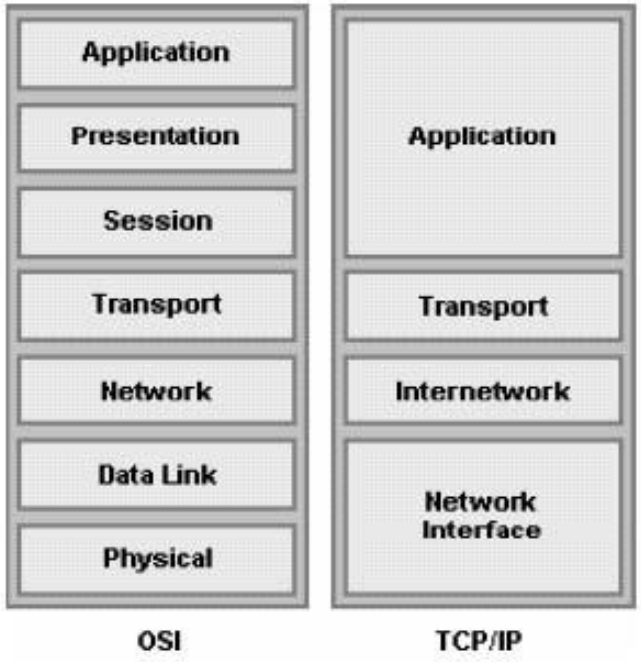
- Enunt de probleme
- Rezolvarea e doar schitata
  - Durata: 4 saptamani
- Detaliile de rezolvare sunt date in partea a doua a semestrului sub forma de pattern-uri, cu schite de implementare

- Bibliografie pentru azi: cap 1, 2 din PoEAA

Structurarea pe straturi (layering)

- Cea mai comuna tehnica pentru a stapani complexitatea unui sistem - software sau de alta natura
- Exemple:
  - Instructiuni in limbaj de programare de nivel inalt -> apeluri de functii ale sistemului de operare -> executie de catre procesor
  - Straturile din modelele OSI si

TCP/IP:



Structurarea pe straturi (2)

- Un strat comunica doar cu cele adiacente
- Un strat foloseste servicii ale stratului de dedesubt si furnizeaza servicii celui de deasupra
- Rareori se face salt peste straturi
- Beneficiile structurarii pe straturi:
  - Fiecare strat poate fi inteles separat, fara a cere intelegerea deplina a celorlalte
  - Se pot substitui diferite implementari de straturi fara a fi nevoie sa se schimbe toata aplicatia; trebuie mentinute doar serviciile enuntate initial

Structurarea pe straturi (3)

- Beneficii (continuare)
  - Se pot minimiza dependintele intre straturi neadiacente
    - intr-o aplicatie enterprise, daca se schimba modul de acces la date, atunci partea de logica a aplicatiei nu e obligatoriu sa fie modificata
  - Straturile pot fi standardizate: TCP si IP precizeaza clar serviciile pe care le ofera si astfel au devenit standarde
  - Un strat poate fi folosit de mai multe altele imediat superioare: TCP e folosit de HTTP, FTP, POP3, SSH etc.

Structurarea pe straturi (4)

- Dezavantaje:
  - Straturile nu pot incapsula intotdeauna bine elementele continute; uneori o modificare poate sa atraga schimbari in cascada: exemplu - adaugarea unui camp in baza de date inseamna modificarea in partea de acces la date, in cea de modelare a domeniului si in interfata utilizator
  - Prea multe straturi pot afecta negativ performanta: comunicarea intre straturi adiacente necesita un tip de date comun convenabil ales, transformarea din reprezentarea interna in tipul comun inseamna cicli procesor consumati; idem, alternative de a folosi impachetarea si despachetarea de date
    - **Dar beneficiile structurarii pe straturi intrec dezavantajele**
- Cel mai dificil aspect: deciderea asupra a ce straturi sunt necesare si care sunt responsabilitatile fiecaruia

Istoricul structurarii pe straturi (1)

- Prima varianta: aplicatiile de tip client – server
  - 2 straturi: (interfata utilizator + logica de business + cod acces la sursa de date); (sursa de date = baza de date – cel mai frecvent relationala)
  - Probleme:
    - de regula, partea de implementare a logicii aplicatiei era inclusa in partea de prezentare => dificultate in a o modifica (actualiza)
    - Duplicarea codului (copy/paste) in loc de factorizare a sa
    - Pseudo-alternativa: scrierea logicii de domeniu in baza de date, sub forma de proceduri stocate = lipsa de flexibilitate in implementare, probleme mari la schimbarea dialectelor SQL proprietare, probleme legate de testare, versionare de cod inexistenta

Istoricul structurarii pe straturi (2)

- A doua varianta: 3 straturi
  - Miscare datorata impunerii programarii orientate pe obiecte
  - Modelarea domeniului se face intr-un al treilea strat situat intre interfata utilizator (user interface, UI) si baza de date
  - Factor de presiune: raspandirea aplicatiilor web, in care UI trebuie separat de restul codului
  - Un sistem care are nivelul de modelare a domeniului incapsulat intr-un strat este usor de extins sa poata fi folosit intr-o aplicatie web
  - Rezultat: dispare legatura directa intre partea de UI si accesul la date
- Confuzie frecventa de termeni: strat (layer) si nivel (tier)
  - Nivelurile implica separare fizica; sistemele client server sunt de fapt cu 2 niveluri (two tiers)

Straturile principale (1)

- Cea mai populara varianta: 3 straturi (sau chiar niveluri)
  - prezentare (UI)
  - modelare de domeniu
  - sursa (surse) de date
- **Stratul de prezentare**: cum se manipuleaza interactiunea intre utilizator si aplicatie: linie de comanda, clickuri de mouse, apasari de taste; sinonim: interfata utilizator (UI)
- **Modelare de domeniu** (business logic): legata de cerintele functionale ale sistemului, implica flux de procesari de date
- **Sursa de date**: comunicarea cu surse de date (e.g. SGBD SQL sau NoSQL, fisiere de date, servicii web etc.), utilizare de sisteme de mesaje, distributed event streaming platform, control de tranzactii

Straturile principale (2)

- **Stratul de prezentare** poate fi si un serviciu care se foloseste de catre alte aplicatii, nu exclusiv Graphical UI
- Frecvent (si de dorit): partea de modelare de domeniu este independenta de particularitatile sursei de date
  - Motiv: trecerea la o alta sursa de date (relationala/semistructurata/date obtinute prin acces de servicii)
- Alteori **stratul de prezentare** acceseaza direct sursa de date
  - Posibil, dar rareori incurajat
  - Exemplu: legarea controalelor de surse de date SQL in pagini ASP.NET (data binding); scrierea codului SQL in pagina PHP
- Sursa de date poate sa aiba multiple implementari, pentru a face legatura cu diferite servere de baze de date

Straturile principale (3)

- Regula: **partea de modelare a domeniului** si de **acces la date** ar trebui sa nu depinda de stratul de prezentare
  - Motiv: se permite introducerea de multiple tipuri de prezentare si substituirea sau imbogatirea lor
  - Se poate modifica partea de prezentare fara a afecta (prea mult sau chiar deloc) straturile inferioare
- Proba: daca exista multiple implementari de GUI si modificarea unei reguli de business cere modificare in toate GUI-urile, atunci acea regula de business nu a fost implementata complet in stratul de modelare a domeniului.

Straturile principale (4)

- Exemplu:
  - Afisarea unei liste de produse in care cele care au crestere de vanzare de 10 procente sau mai mult sunt colorate cu rosu
  - Implementarea 1: programatorul face in stratul de prezentare comparatia intre vanzarile actuale si cele din luna precedenta; daca e cazul, le coloreaza in rosu
  - Implementarea 2: partea de domain model sa furnizeze pe langa lista de produse si cate un indicator asociat celor care au cresteri mari = o valoare logica pe fiecare produs
  - Partea de interfata interogheaza aceasta valoare logica si coloreaza corespunzator

Locatia straturilor (1)

- Unde se ruleaza fiecare strat?
  - Separarea pe straturi este utila chiar daca rularea se face pe aceeasi masina
- Cea mai comuna intrebare: unde sa ruleze procesarea datelor?
  - Cazul cel mai simplu: rulam totul pe server de aplicatii
  - Motiv: usor de intretinut si versionat daca codul este intr-un singur loc
  - Efort de instalare: aproape zero
  - Lipsa problemelor legate de particularitatile calculatoarelor client

Locatia straturilor (2)

- Alta varianta: totul pe client
  - Pro: aplicatia poate avea timpul de raspuns mic + se foloseste puterea de calcul a sistemelor client
  - Contra: clientul trebuie sa ruleze o anumita platforma, sa aiba instalate bibliotecile necesare procesarilor locale
  - Contra:versionare ingreunata
    - Exceptii: Java Web Start =Java Network Launching Protocol (JNLP); Microsoft ClickOnce
    -

Locatia straturilor (3)

- Cum decizi?
- De regula, stratul de date ruleaza pe o masina aparte
  - Exceptie: smart client, in care o parte de date pot fi preluate pe client; necesita rezolvarea problemei de sincronizare
- Stratul de prezentare: depinde cum se materializeaza
  - XHTML pe thin client -> trebuie sa existe server web
    - Constientizam partea de roundtrip, responsiveness (viteza de reactie a interfetei)
    - Varianta: GUI facut in Web Assembly, e.g. https://www.qt.io/qt-examples-for-webassembly
  - Rich (fat) client -> pe masina clientului
  - Varianta: o parte din domain model pe client, alta pe server

Locatia straturilor (4)

- Ce e de evitat:
  - Duplicarea codului pe client si pe server
  - Implementarea straturilor in *procese* diferite pe aceeasi masina = performanta scazuta si complexitate mai mare (trebuie remote façade si data transfer object);
    - comunicarea intre procese este posibila si e mediata de sistemul de operare, dar mai lenta decat comunicarea in cadrul aceluiasi proces, intre fire de executie
- Uneori, cerinte stringente de performanta, securitate sau existenta de aplicatii mostenite dicteaza locatia straturilor

Organizarea logicii de domeniu

- Trei modalitati de rezolvare (pattern-uri):
  - Transaction script
  - Domain model
  - Table module

Transaction script (1)

- Cel mai simplu si usor de inteles
- Un **transaction script** este o procedura sau functie care:
  - Preia intrarea de la stratul de prezentare
  - O proceseaza, deci contine validari si calcule sau transformari
  - Stocheaza/interogheaza datele (d)in baza de date
  - Optional: formeaza raspunsul si il returneaza apelantului
- Organizare: cate o procedura pentru fiecare actiune sau tranzactie de business
- Se poate sparge in mai multe subrutine ce pot fi partajate de mai multe transaction scripts

Transaction script (2)

- Avantaje:
  - Model procedural simplu pe care il intelege orice programator
  - Se implementeaza simplu in conjunctie cu un strat de sursa de date de tip Row Data Gateway sau Table Data Gateway
  - Arata clar unde incep si se termina tranzactiile: procedura incepe cu deschiderea unei tranzactii si se termina cu inchiderea ei
    - inchidere de tranzactie = commit sau rollback
  - Rezultat: set de rutine bine separate

Transaction script (3)

- Dezavantaje:
  - Cand creste complexitatea logicii de domeniu, scripturile de tranzactie devin depasite de situatie
  - Fenomen frecvent: duplicarea codului
    - Evitabila prin factorizarea codului, dar deseori omisa din motive de graba
  - Rezultat: pentru o problema de dimensiuni mari va rezulta un graf de functii cu legaturi multiple si fara o structurare clara
    - Cum gasesti functie care implementeaza o anumita operatie?

Domain Model (1)

- Modelarea logicii complexe este sprijinita de programarea orientata pe obiecte
- Se construieste un model al domeniului sub forma de clase relationate
  - Cum alegi clasele: in prima faza pleci de la substantivele cele mai des intalnite in enuntul cerintelor
  - Sistem de leasing: act de inchiriere, serviciu/obiect care se inchiriaza, client, formula etc.
- Logica de efectuare a validarilor si a calculelor se gaseste tot aici
  - Pentru un obiect ce urmeaza a fi livrat, logica sa poata sa includa calculul cheltuielilor de impachetare si livrare – sarcina altui obiect

Domain Model (2)

- Vor exista clase care implementeaza diferite responsabilitati
- Obiectele instanta comunica intre ele
- Modul de creare a claselor poate fi dictat de reprezentarea responsabilitatilor (o clasa/responsabilitate)
- Este esential un curs de design orientat pe obiecte

Comparatie Transaction script vs Domain model

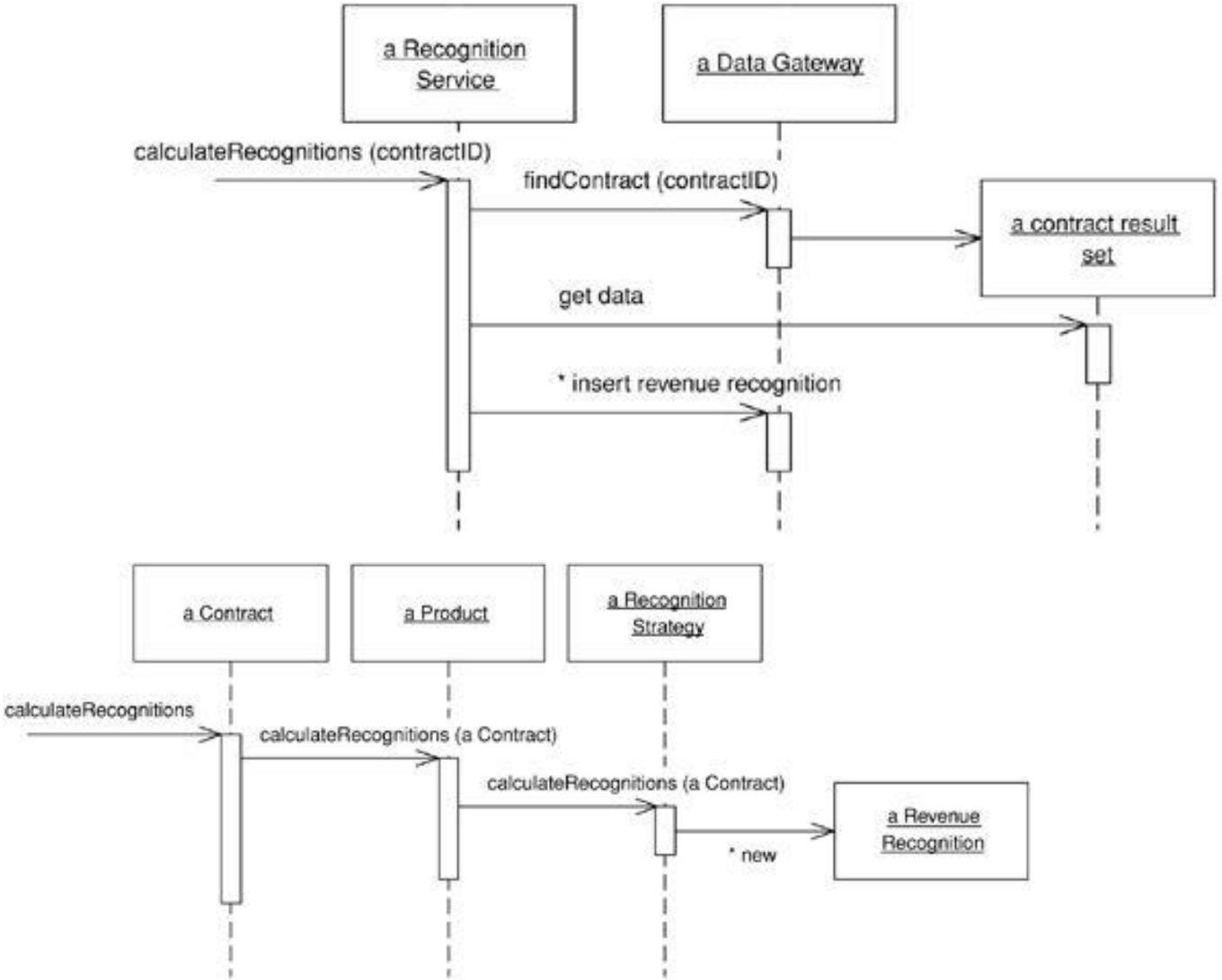


Table module (1)

- Domain model: clase pentru contracte, produse, venituri
- In timp ce la Domain model am o instanta pentru fiecare contract, in Table module (TM) exista **o singura instanta ce reprezinta un intreg set de date** (e.g. contracte)
- TM lucreaza cu design patternul enterprise RecordSet

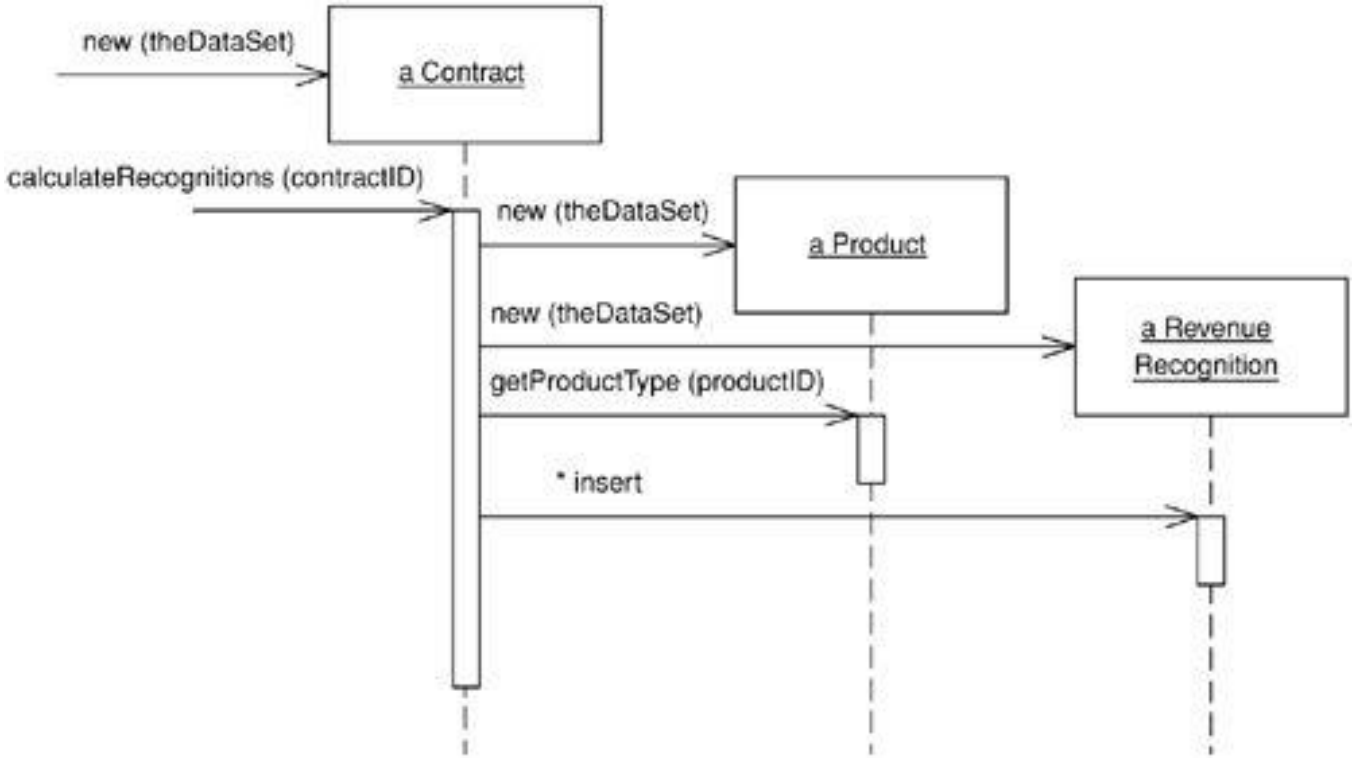
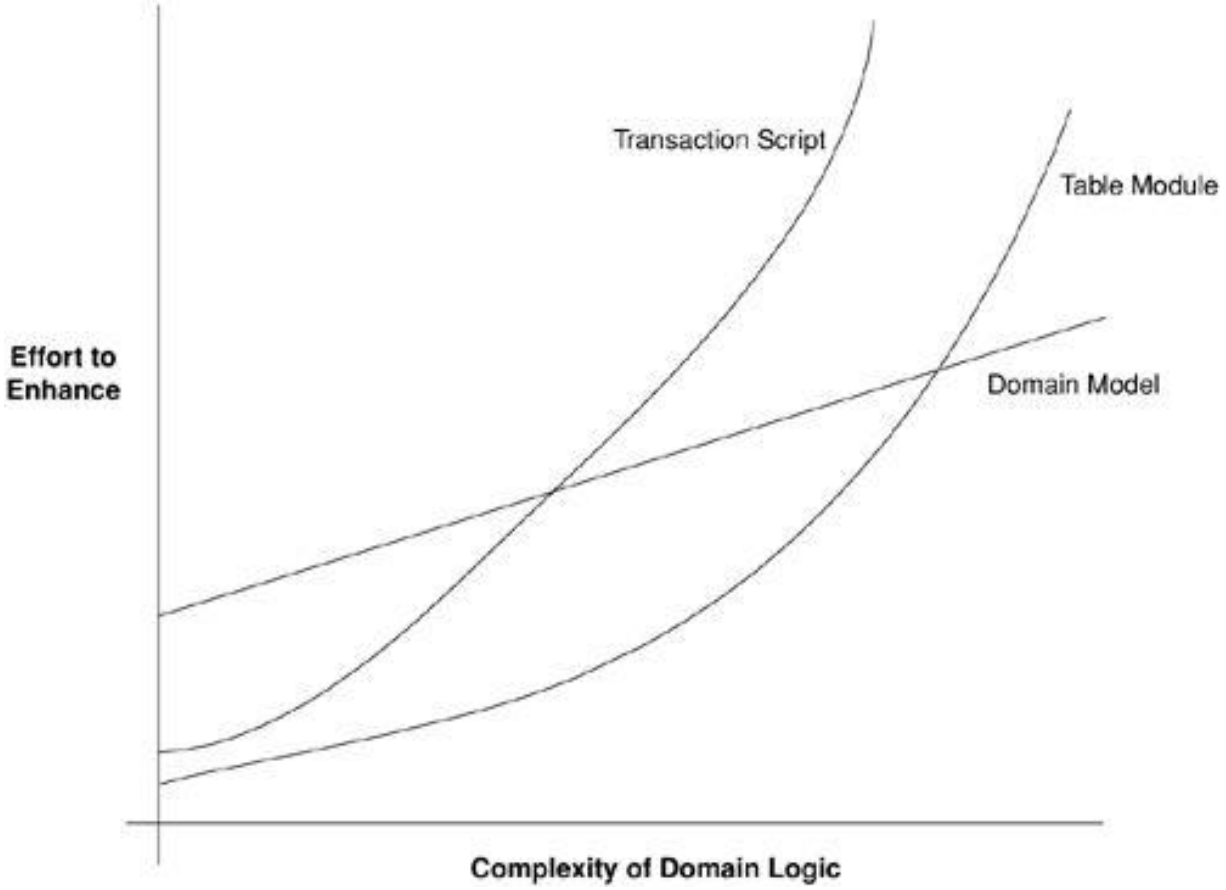


Table module (2)

- Vazut ca o varianta intermediara intre Transaction script si Domain Model
  - Organizarea codului in jurul unui set de date ajuta la structurare si gasirea usoara a codului duplicat
  - Dar nu se pot folosi usor mecanisme specifice OOP: mostenire si suprascriere de comportament, folosirea de strategii alese la momentul rularii sau multe altele din design pattern-urile clasice
- Avantaj: de multe ori este favorizat de modul de realizare a interfetelor utilizator: multe GUI-uri suporta legarea direct la un RecordSet (vezi data binding in WPF sau ASP.NET), deci se poate folosi un Table module pentru validari si furnizare de date

Cum alegi? (1)

- Depinde de complexitatea logicii de domeniu



Cum alegi? (2)

- Experienta echipei care implementeaza este factor important
- Domain model: daca echipa il cunoaste, atunci costul initial este redus; alteori merita sa investesti in training-ul echipei
- Table module: daca mediul de programare suporta RecordSet atunci proiectul poate fi implementat cu viteza crescuta; daca nu exista suport intrinsec din limbaj sau biblioteci, mai bine se evita
- Refactoring-ul este posibil
  - Ajuta mult o refacere de tip Transaction Script->Domain Model
  - Invers: nu exista castiguri substantiale
- Alegerile nu se exclud reciproc: transaction script pentru o parte din model, TM sau DM pentru restul

Service layer (1)

- Structurarea logicii de domeniu in doua parti: un Service Layer scris peste un Table Module sau Domain Model
- Daca se foloseste Transaction Script pentru modelarea logicii de domeniu, atunci acesta este totodata si Service Layer
- Legatura intre nivelul de prezentare si cel de modelare a aplicatiei se face prin intermediul Service Layer
- Beneficiile service layer:
  - Controlul tranzactiilor
  - Securitate
  - Orchestrare de apeluri

Service layer (2)

- Cat comportament pui in SL?
  - Se concepe de obicei ca o fatada, reprezentand interfata de comunicare = API
  - Dupa verificari de securitate, stratul trimite mai departe cererile catre Domain Model / Table Module
  - Service Layer ar fi structurat dupa functionalitatile aplicatiei (use-cases)

Service layer (3)

- Daca asocierea obiectelor este 1:1 cu tabelele din baza de date (exemplu: aplicatie simpla) atunci SL poate fi Transaction Scripts iar obiectele sunt implementate ca Active Record
- Atentie la codul duplicat
- Discutie mai ampla, cu variante si comentarii: PoEAA, capitolul 2

Tema 2

- Care sunt uneltele care pot raporta duplicarea codului? mailto:lucian.sasu@yahoo.com?subject=Duplicare%20cod