7. rész: A specifikációtól az implementációig az EJB rétegben

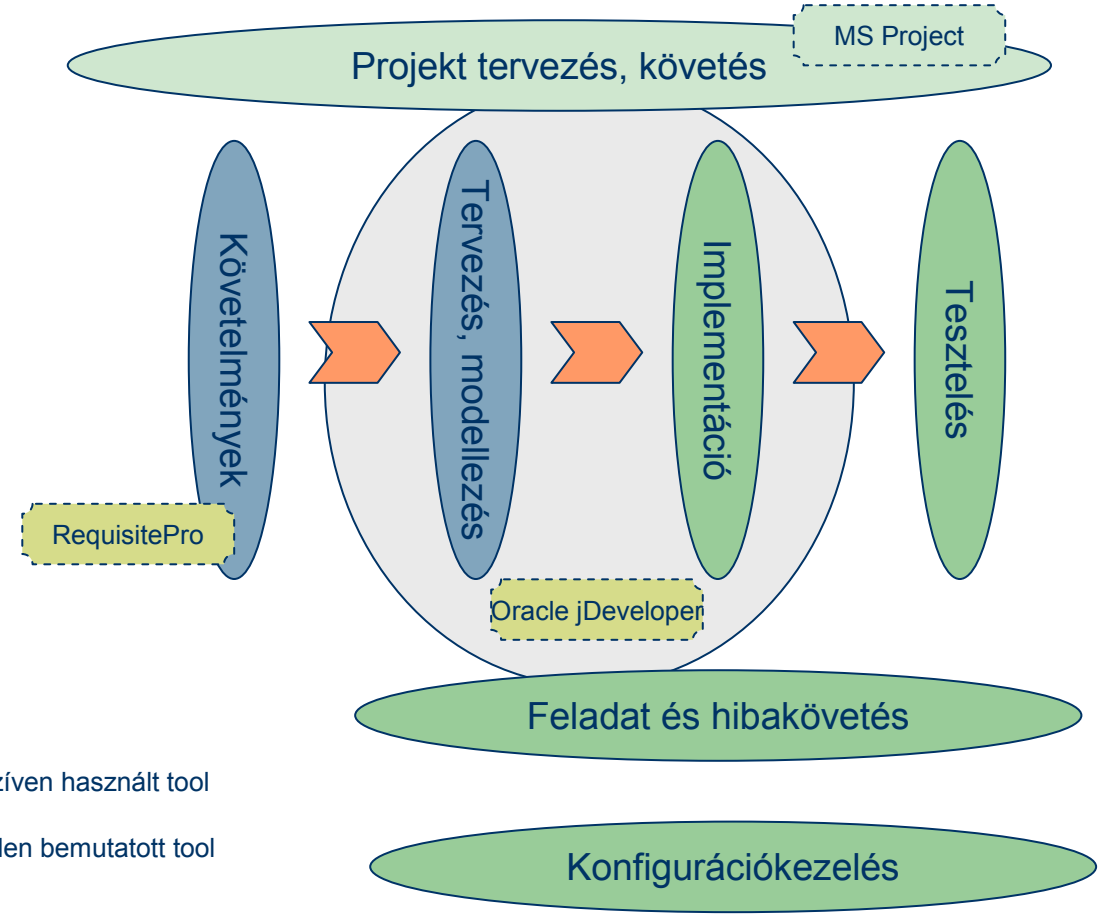
Bakay Árpád
NETvisor kft
(30) 385 1711
arpad.bakay@netvisor.hu

A tananyag készült az
ELTE-IKKK
projekt támogatásával

Tartalom

- Tervezés lépései
- Adattervezés
- Design patternek alkalmazása

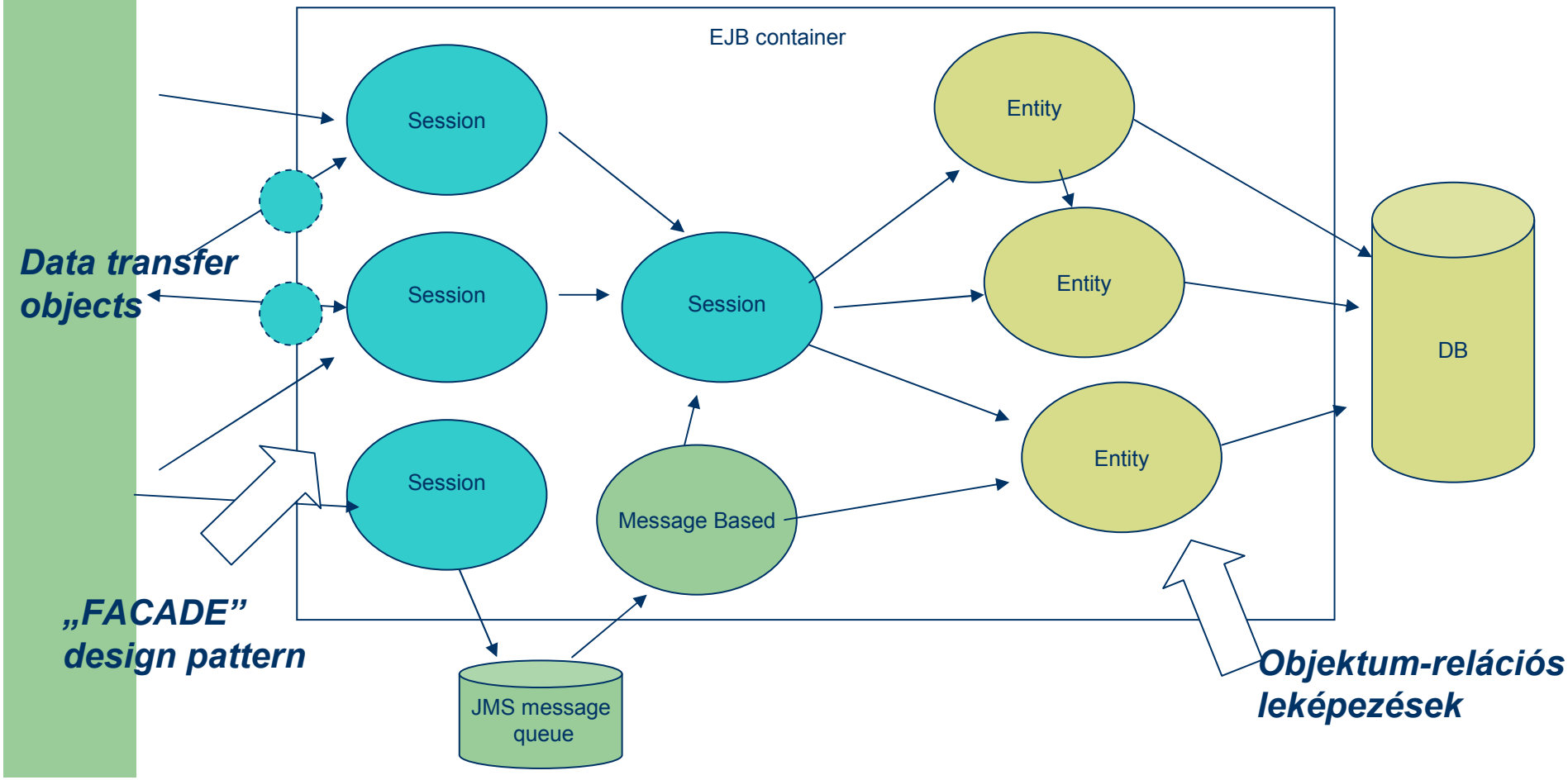
Hol tartunk?


Projekt tervezés, követés
MS Project
Követelmények
RequisitePro
Tervezés, modellezés
Implementáció
Oracle jDeveloper
Tesztelés
Feladat és hibakövetés
Konfigurációkezelés
Intenzíven használt tool
Röviden bemutatott tool

1. Áttekintés: Alkalmazástervezés a specifikáció alapján

1. UML Modellezés
   - Pl. use case, sequence, class, deployment diagram
2. Adattervezés
   - Objektum vagy relációs szemlélettel
3. Üzleti réteg külső interfészek megtervezése, modellezése
4. Entity és session EJB-k generálása, implementálása
   - Modul szintű teszteléssel, teszt környezetben, spec eszközökkel
5. Az üzleti logika szolgáltatásainak tökéletesítése
   - Leggyakoribb cél: a sebesség növelése
   - Biztonság fokozása
   - Kényelmi funkciók támogatása (pl. gyorskeresés, undo/redo)

Design patternek alkalmazása az EJB réteg kialakításánál



1.1 UML Modellezés

- Az UML jól használható bonyolult rendszerek szemléletes leírására, de…
- A legtöbb diagram közvetlenül és automatikusan nem használható implementációra
  - Inkább „szemléltető” jelleg, mint precíz spec.
  - Nem adnak teljes leírást, pl:
    - UC diagram csak áttekintést ad, az UC-k lényege rejtve marad
    - Sequence, collaboration stb. diagramokat csak a legfontosabb, legérdekesebb műveletekre készítenek
- Kivétel a class diagram
  - Ezt viszont sok fejlesztőeszköz átvette, és integráltan megvalósítja (pl. a jDeveloper is).

Az UML alapú tervezés mai helyzete

- A modellező és a fejlesztő eszközök összeolvadnak:
  - Az egybe csomagolt integrált rendszerek megbízhatóbban működnek, mint külső integráció
  - *Ráadásul sok esetben ingyenesek!!!*
- Következmények:
  - A tervezés/modellezés és az implementáció már nem választható el élesen egymástól
  - *A modellező eszközök piaca beszűkült*
    - *Leginkább bonyolult, real-time rendszerekhez használják*

1.2 Adattervezés

- Objektum-alapon, v. Relációs DB alapon?
- Lényeges különbségek a két szemléletben:
  - Öröklődés
    - RDBMS: nincs öröklődés és nincsenek absztrakt osztályok
    - Három módszerről ld.
      http://www.objectmatter.com/vbsf/docs/maptool/ormapping.html
  - Relációk
    - Pl: List<Kezelés> ←→ FK_KEZELES_ESETID
    - OO: OneToMany, OneToOne. etc
    - OO uni- és bidirectional
    - Rel: minden reláció one to many és lényegében bidirectional
    - Rel: a many-to-many kapcsolórekordokkal (join tables)

Adattervezés: tanulságok

- UML class diagram interpretálása a részletek szintjén nem egyértelmű
  - Relációk property neve, átjárhatósági iránya, relációk collection típusa, számossági (0-1-*) constraintek
  - Attribútumként használt objektumok (pl. collectionok)
- A RDBMS alapú tervezés alacsonyabb szintű, de általában szabatosabb
- Aut. generálás mindkét irányban lehetséges
  - Kisebb-nagyobb bökkenőkkel
    - Pl. öröklődési hierarchia leképezése (O->R esetben), ill. detektálása (R->O esetben)

1.3 Üzleti funkciók tervezése

- Jó ösztönökkel kell megkeresni az optimális határt a prezentáció és az üzleti réteg között
  - Felhasználhatók a Use case-ek, GUI modellek
- Funkciókat logikai vagy actor/jogosultság szempontok szerint csoportosítani
  - Ezek lesznek a Session bean-ek
- Definiálni a teljes igényelt funkcionalitást (de nem többet!) megvalósító metódus-készletet.
- Az üzleti funkciók szolgálják a prezentációt
  - Ne legyenek „öncélúak”, „öntörvényűek

Façade Design Pattern

- Az üzleti réteg egyféle firewall az adatok előtt
  - Akár helyes, akár érvénytelen hívássorozatra alapszinten megvédi az adatok biztonságát, konzisztenciáját
    - Tranzakciókezelés
    - Logikai ellenőrzés: pl. van-e elég egyenleg
    - Hozzáférések naplózása
  - Valódi biztonsági firewallként (pl. jogosultság-ellenőrzésre) is használható
    - Az EJB-khez beállítható jogosultság-feltételekkel

Adatok átadása a Façade-on: Data Transfer Object design pattern

Egy, általában vmelyik entitáshoz kapcsolódó „value” objektum, amely a távoli hívásokhoz teljesen serializálható, és amelyen keresztül a gyakran ismétlődő attribútum-lekérdezési szekvenciák hossza csökkenthető.

- Sok esetben egy 3.0-s Entity osztály önmagában megfelel
  - az Entity-k legtöbbször serializálhatók, hiszen egyszerű adatokat tartalmaznak
- Biztonsági, v. egyéb megfontolásból lehet egy másik objektum is, pl:
  - az entitás egyes adatainak elrejtése a kliensek elől
  - az entitásban nem szereplő, pl. kapcsolódó entitások alapján aggregált mezők (pl. számla összeg a számlatételek összegéből)
- Alternatíva DTO helyett: attribútumok egyenkénti elérése a facade-on: pl.: `getTipus()/setTipus()`, `getLoero()/setLoero()`, stb.
  - Objektum azonosítása
    - Hívási paraméterként átadott PK `getTipus(String rendszam);`
    - Stateful bean: `setCurrentObject(String rendszam); … getTipus();`

Mi legyen façade mögött?

- Bonyolultabb rendszereknél jellemzően találhatók további, lokális Session EJB-k
- … esetleg üzenetsorok és Message Driven EJB-k
- … de nálunk a Façade várhatóan közvetlenül kezeli majd az Entitásokat

1.4 Generálás, implementáció, tesztelés

- … a következő órán

1.5 A rendszer tökéletesítése

- Cél legtöbbször a sebesség
- Általánosan alkalmazható eszközök
  - Iteráció helyett entity *findAllxxx* named query-k bevezetése
    - Pl: findAllUsersByLastName, findAllBooksByPriceRange
  - Data Transfer Object (DTO) használata
  - Message Driven processing
  - DB stored procedures, server-side Java

2. Gyakorlati lépések

- Vannak már rendszerezett követelményeink
- Ismerjük az implementálandó szoftver architekturát

- …lépjük át a szemantikai szakadékot

2.1 Tervezés első lépése: a „kockás papír”

- … vagy text editor
- Összegyűjteni és vázlatosan ábrázolni
  - az entitás, típusokat, relációkat
    - Felhasználandó: DATA követelmények a ReqPro-ban
  - A prezentáció által igényelt funkciókat
    - Csoportosítani kategóriánként -> facade session EJB-k
    - Felhasználandó: UC-k a ReqPro-ban
  - A lehetséges DTO-kat

A kockás papír lényege

- A spec alapján azonosítani a kulcs-entitásokat és a fontosabb funkciókat.
- Nem használunk szoftver eszközt, ami elterelhetné a tervező figyelmét.
- Vázlatos leírás, tehát azonnal fel kell használni, nem napok, hetek múlva.

A „Kockás papír fázis” statisztikája az Ursulában

- *„Szoftverfejlesztési eszköz”*: MS Windows Notepad

- 7 entitás / tábla
- 5 facade session bean
  - UserManager, EllatoNavigator, EllatoManager, BetegManager, BetegSelfCare
- 44 facade metódus
- 4-5 DTO

- Munkaigény: 90 perc
- Később több kisebb változás a tervben
  - nevváltozás, egy-egy plusz paraméter, stb.

2.2 Implementáció a fejlesztőeszközben

- A következő órán:
  - Projekt előkészítése
  - Adattervezés séma-genrálás
  - Entitások elkészítése
  - Üzleti funkciók elkészítése