Metamodellezés

Simon Balázs

BME IIT, 2011.

Tartalom

- Bevezetés
- Metamodellezés
- EMF & ecore

Bevezetés

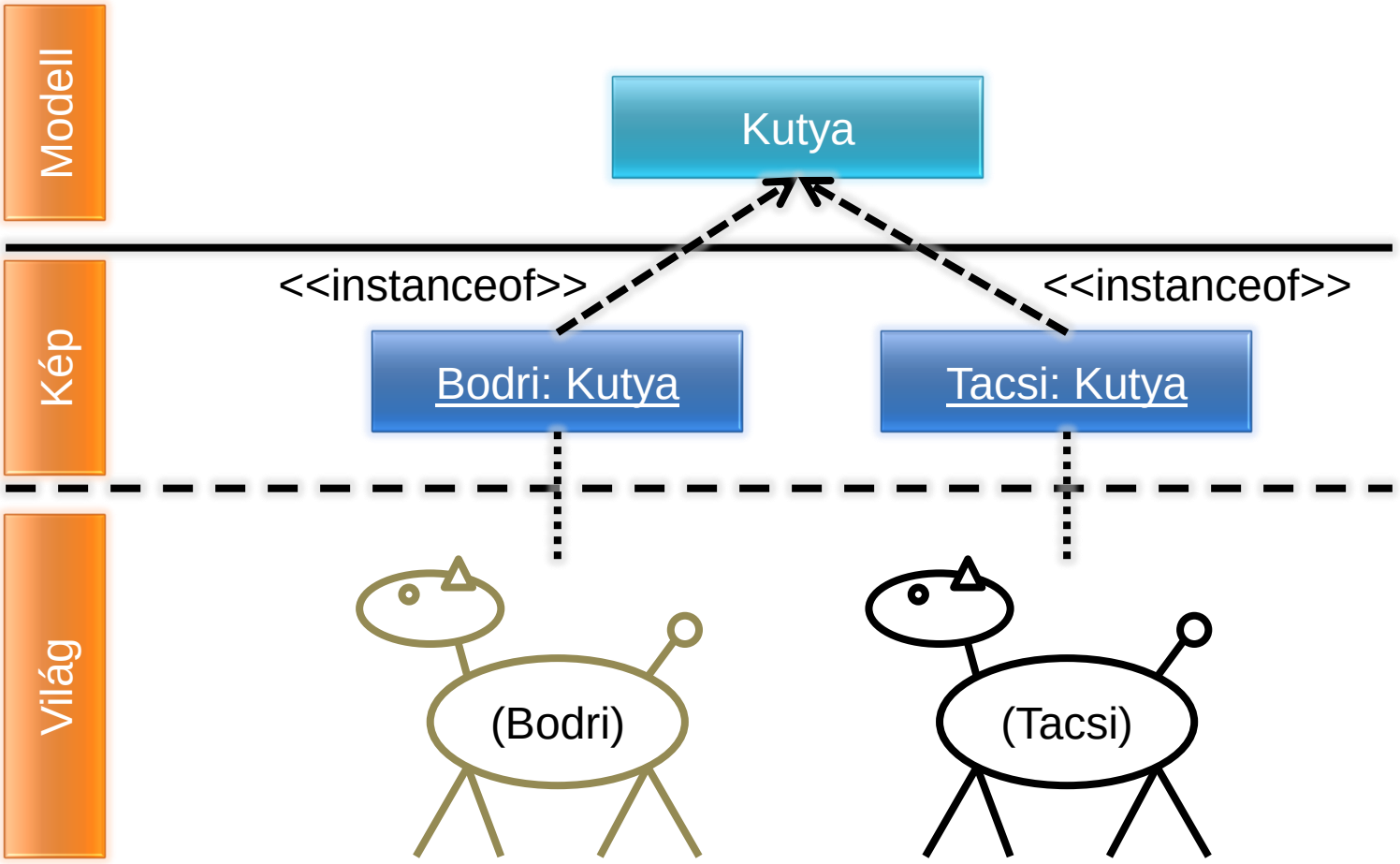
- Hétfő: Simon Balázs
  - hetente felváltva: előadás és gyakorlat
  - metamodellezés
  - kódgenerálás
  - fordítóelmélet
- Csütörtök: Dr. László Zoltán
  - kéthetente

Célok

- Fejlett modellező eszköz használata
- Saját metamodell tervezése
- Kódgenerátorok készítése
- Programkódok feldolgozása
- Fordítóelmélet alapjainak megismerése
- A fenti módszerek használata a szoftverfejlesztés egyes fázisaiban

Metamodellezés

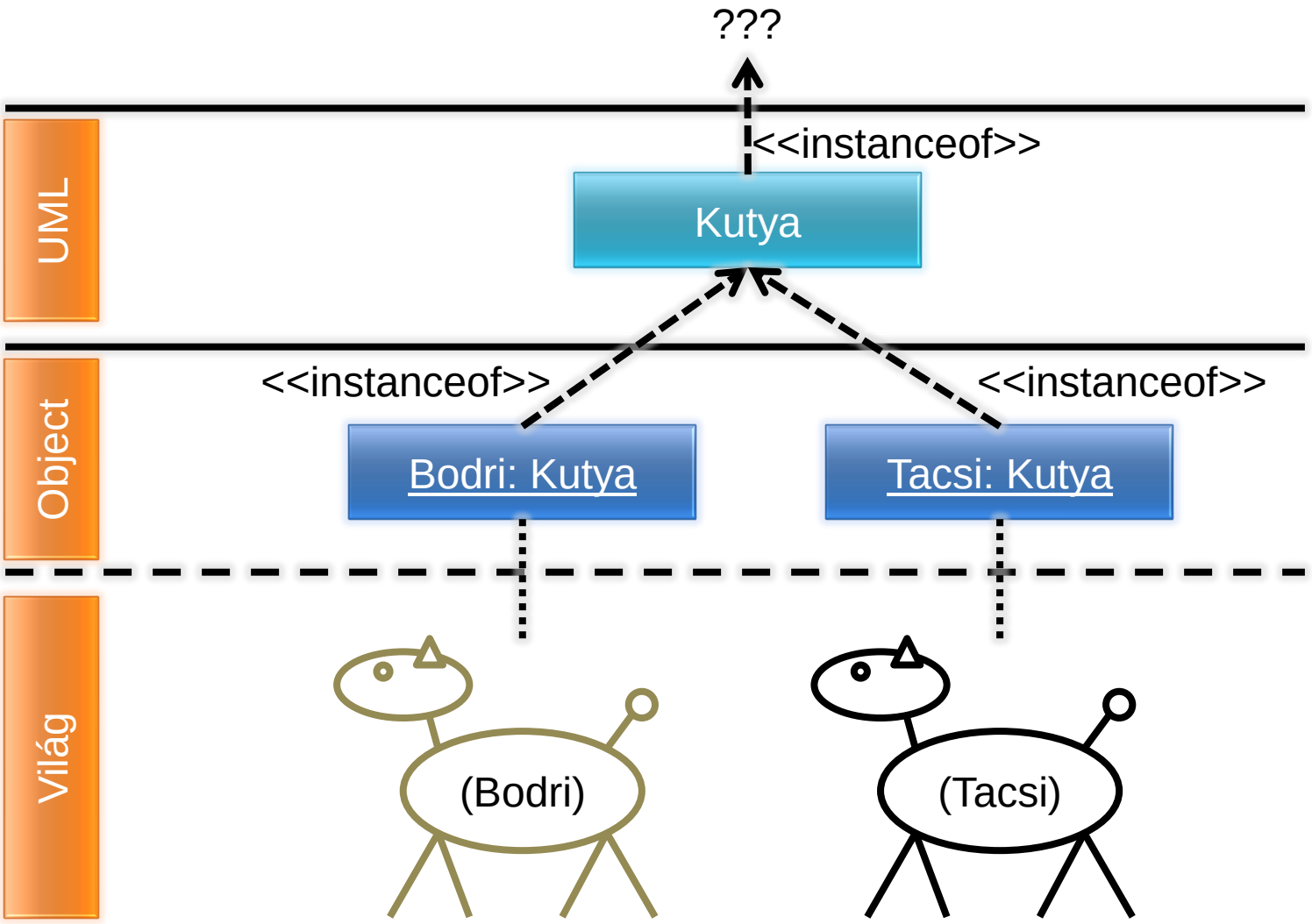
Modell


Modell
Kép
Világ
Kutya
<<instanceof>>
<<instanceof>>
Bodri: Kutya
Tacsi: Kutya
(Bodri)
(Tacsi)

Modellezés

- Model
  - Abstraction
    - az adott környezet számára érdektelen információk figyelmen kívül hagyása
  - Classification
    - a közös tulajdonsággal rendelkező fontos információk csoportosítása
- Metamodel
  - modellezési nyelv modellje
  - megadja a modellek egy családjának építőelemeit, struktúráját, szemantikáját és kényszereit

Metamodellezés



Metamodellezés

- UML – Unified Modeling Language (M2)
  - use-case diagram
  - osztálydiagram
  - szekvencia diagram
  - …
- MOF – MetaObject Facility (M3)
  - ebben van leírva az UML
  - legfelső meta-szint
  - önleíró
  - az UML egy kicsi részhalmaza
- XMI – XML Metadata Interchange
  - modellek cseréje eszközök között XML formátumban

Metaszintek

Metaszint	Leírás	Elemek
M3	MOF meta-metamodell (önmaga metamodellje)	MOF: Class, Property, Association stb.
M2	UML metamodell	UML: Class, Property, Association, State, Activity stb.
M1	UML modell	„Kutya” nevű osztály
M0	objektumok és adatok: az M1 réteg elemeinek példányai	„Bodri” nevű Kutya típusú objektum

Metamodellezés

- UML (M2):
  - nem mindig elegendő
  - pl. gráfok, fák, egyéb speciális alkalmazási területek
- DSL – Domain Specific Language (M2)
  - szakterület-specifikus nyelvek
  - speciális szakterületre kialakított nyelvek
  - az **absztrakt szintaxis** (modell struktúrája, elemei és kényszerei) MOF-ban leírva
  - a **konkrét szintaxis** (felhasználó számára) lehet:
    - szöveges (speciális programnyelv)
    - grafikus (speciális dobozok és vonalak)
  - előny: a felhasználó a saját fogalmaiból építkezik

Metaszintek

Metaszint	Leírás	Elemek
M3	MOF meta-metamodell (önmaga metamodellje)	MOF: Class, Property, Association stb.
M2	a MOF által leírt metamodellek: a MOF elemeinek példányai	UML: Class, Property, Association, State stb.; DSL metamodell
M1	az M2 réteg metamodelljei által leírt modellek: az M2-es metamodellelemek példányai	„Kutya” nevű osztály; DSL modell
M0	objektumok és adatok: az M1 réteg elemeinek példányai	„Bodri” nevű Kutya típusú objektum; DSL objektumok

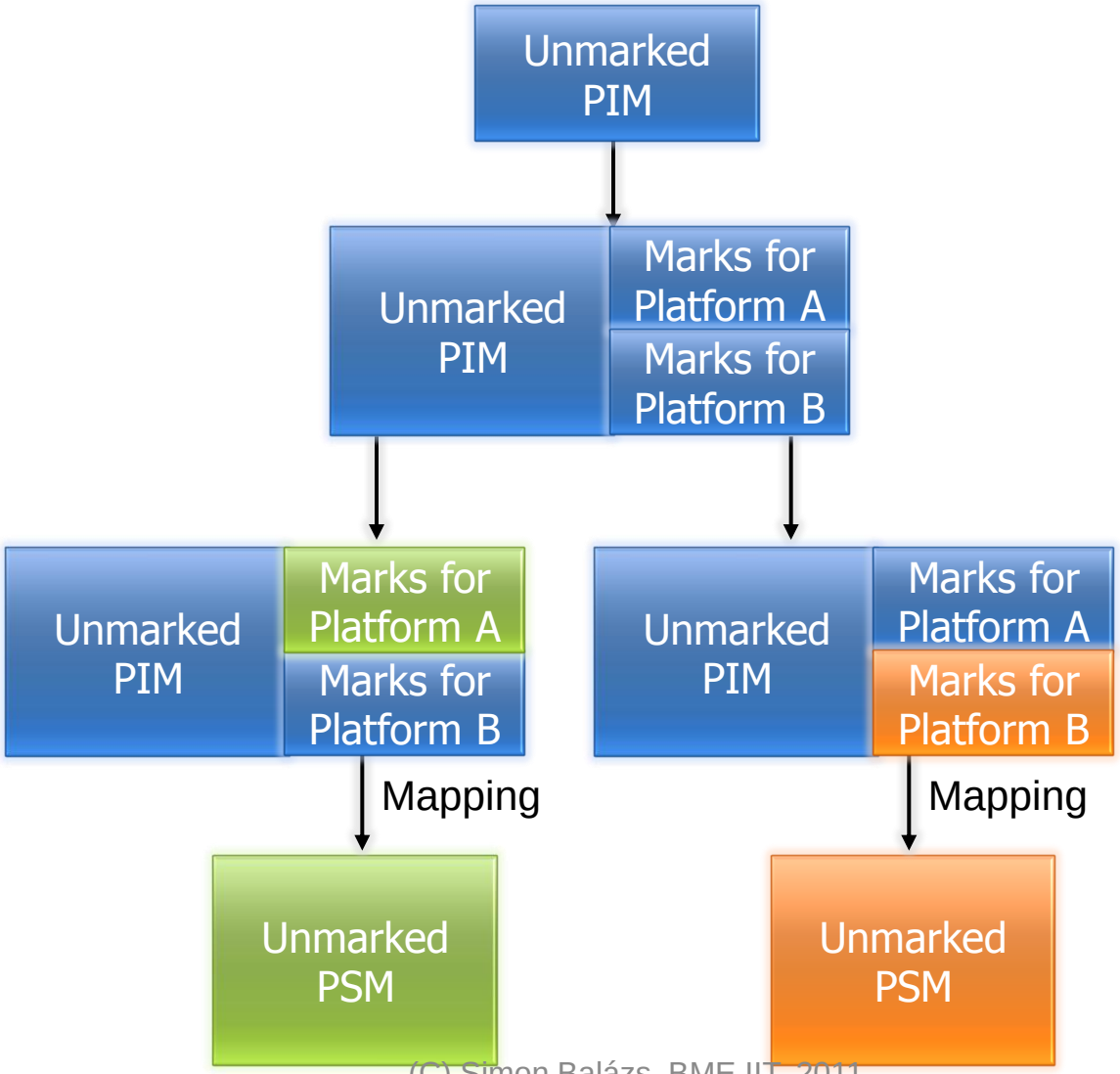
Megjegyzés:
Két egymás feletti metaszint közötti kapcsolat relatív,
így tetszőlegesen sok metaszint létezhet!

MDA:
Model Driven Architecture

MDA: Model Driven Architecture

- Platform
  - futtatási környezet specifikálása modellek egy halmazára
  - legalább egy implementáció
  - pl. Java, .NET; Windows, Linux; Oracle, MySQL
- PIM: Platform Independent Model
- PSM: Platform Specific Model
- Mapping
  - leképzés, függvény: PIM-ből a PSM előállítása
- Marker
  - ez alapján dől el, milyen platformra készül a PSM
  - A leképzés plusz bemenete, nem része a PIM-nek!

MDA: Model Driven Architecture

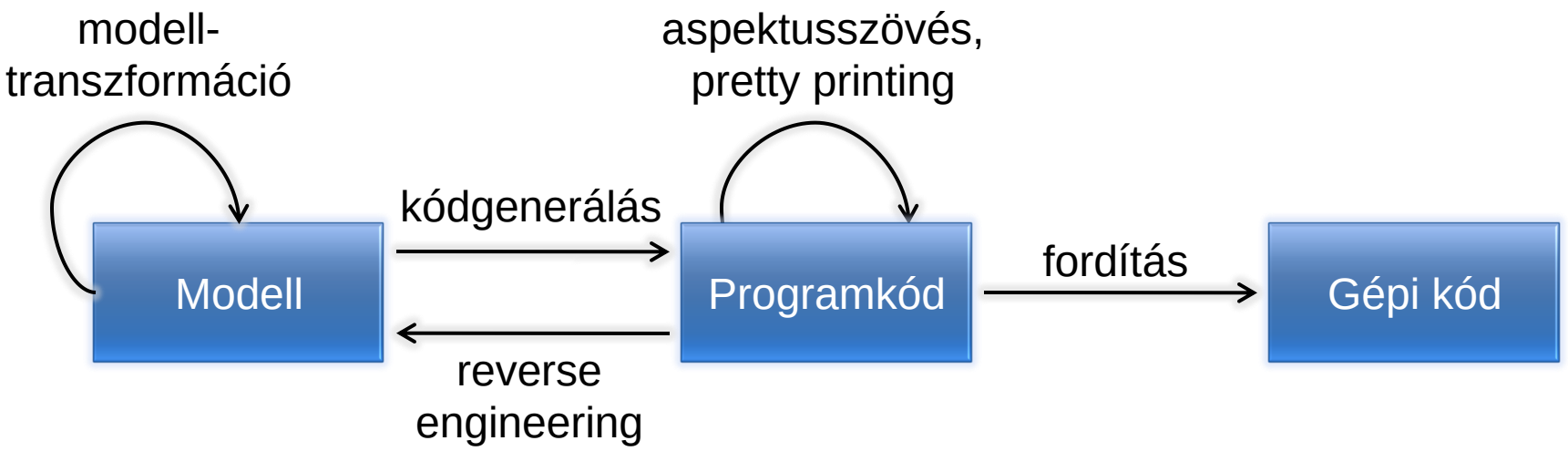


MDA: Model Driven Architecture

- Model elaboration
  - a modell általában nem teljes: finomítani kell
  - kiindulás: magas absztrakciós szintű modell
  - átmenet: köztes modellek
  - kimenet: alacsonyabb absztrakciós szintű modell
  - példák:
    - get-set metódusok automatikus hozzáadása a modellhez
    - Java kód generálása modellből
- Reverse engineering
  - modell visszafejtése
  - az absztrakciós szint emelése
  - példa:
    - Java kód alapján UML osztálydiagram előállítása

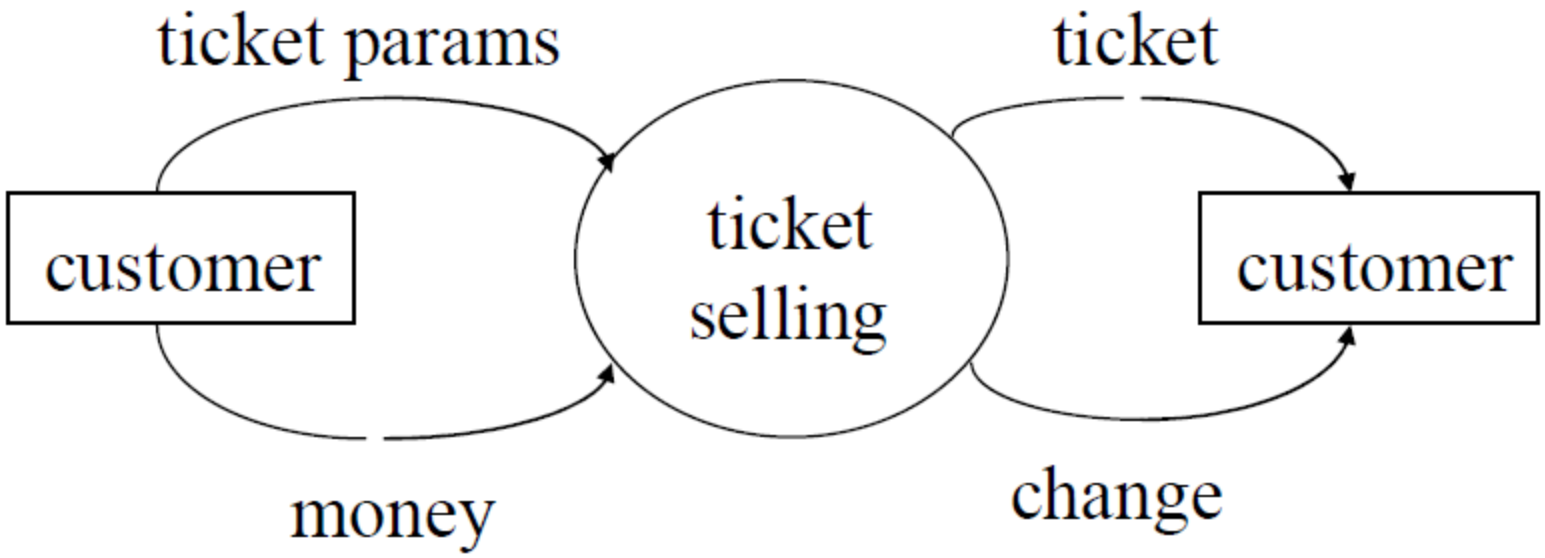
Transzformációk

- Modell feldolgozását igényli:
  - modelltranszformáció, kódgenerálás
- Programkód megértését igényli:
  - aspektusszövés, pretty printing, reverse engineering, fordítás

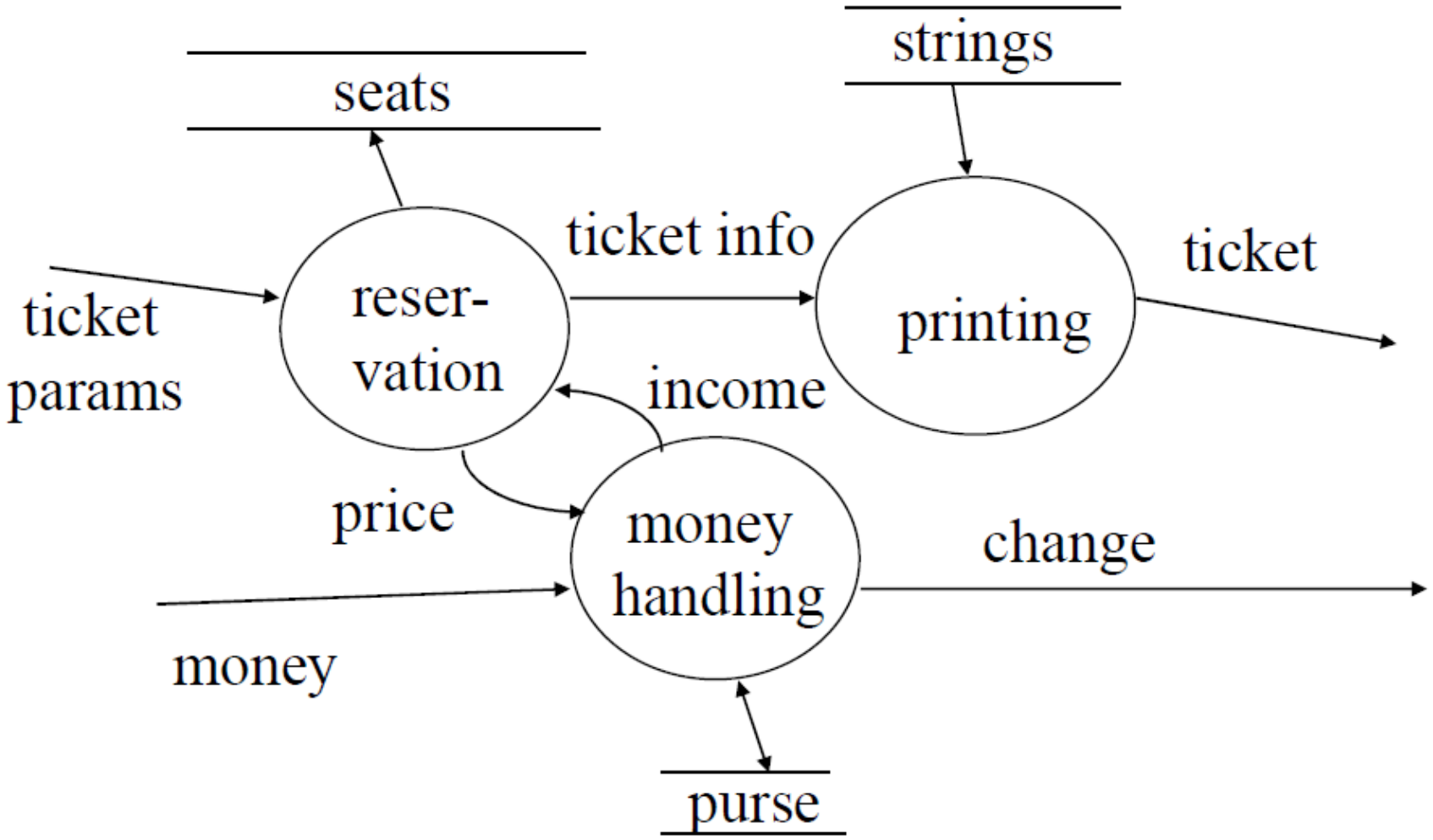


Gyakorlatok

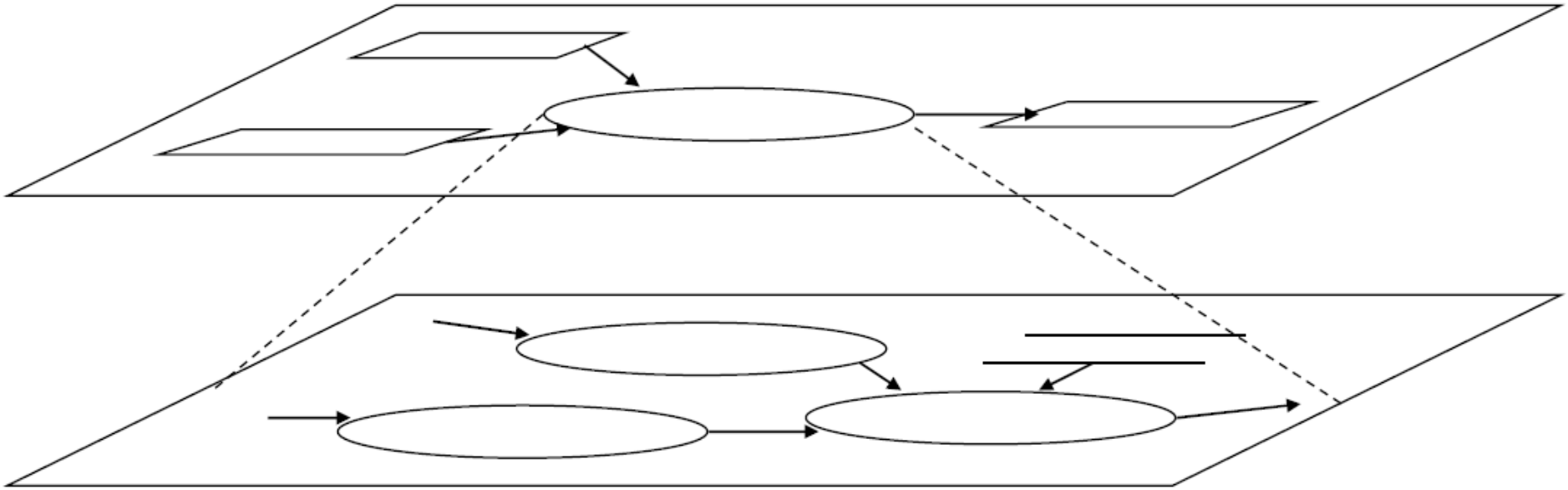
Mi ez?


ticket params
ticket
customer
ticket selling
customer
money
change

Mi ez?

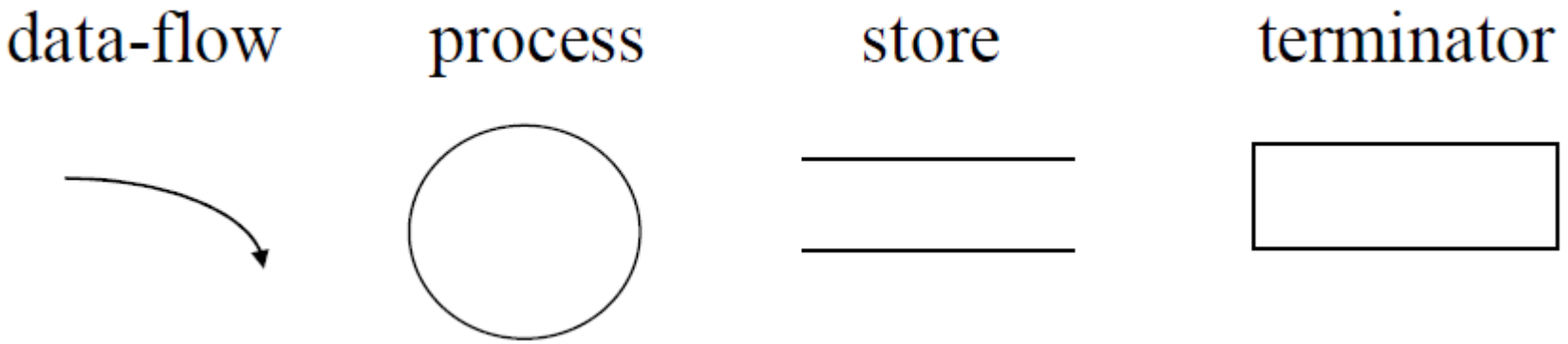

seats
strings
ticket info
ticket
ticket
params
reser-
vation
printing
income
price
money
handling
change
money
purse

Context Diagram & Data Flow Diagram



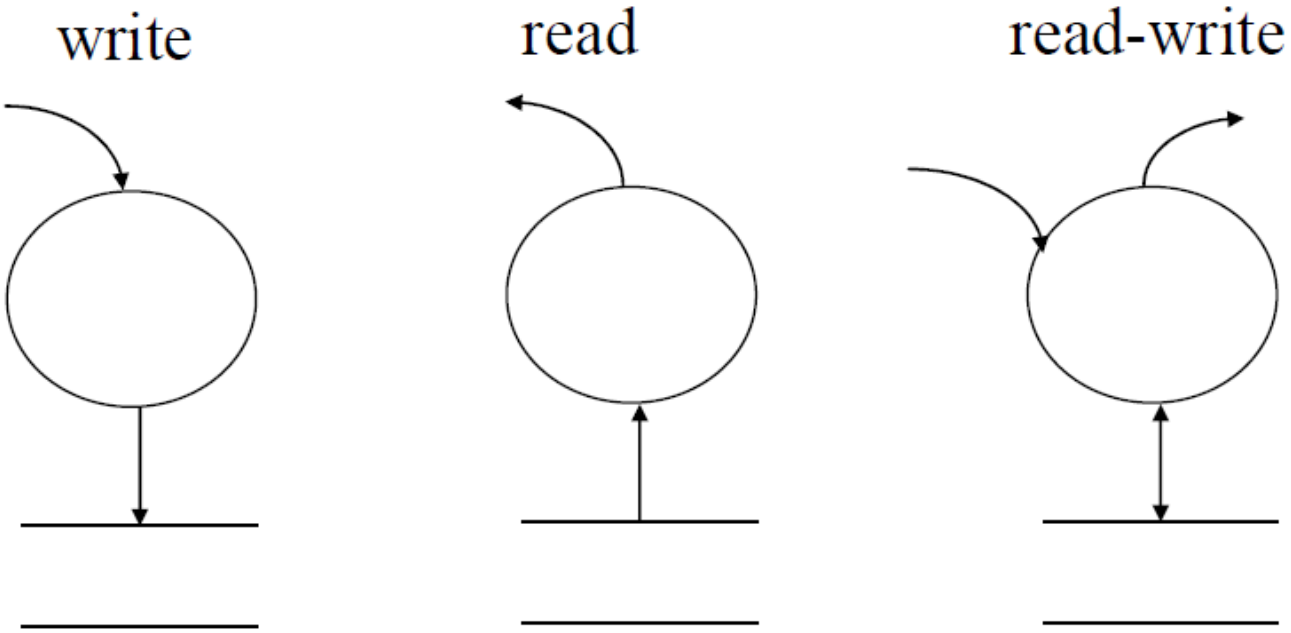
Context & Data Flow Diagram

- Data-flow: adat
- Process: folyamat
- Store: tár
- Terminator: külső forrás/nyelő

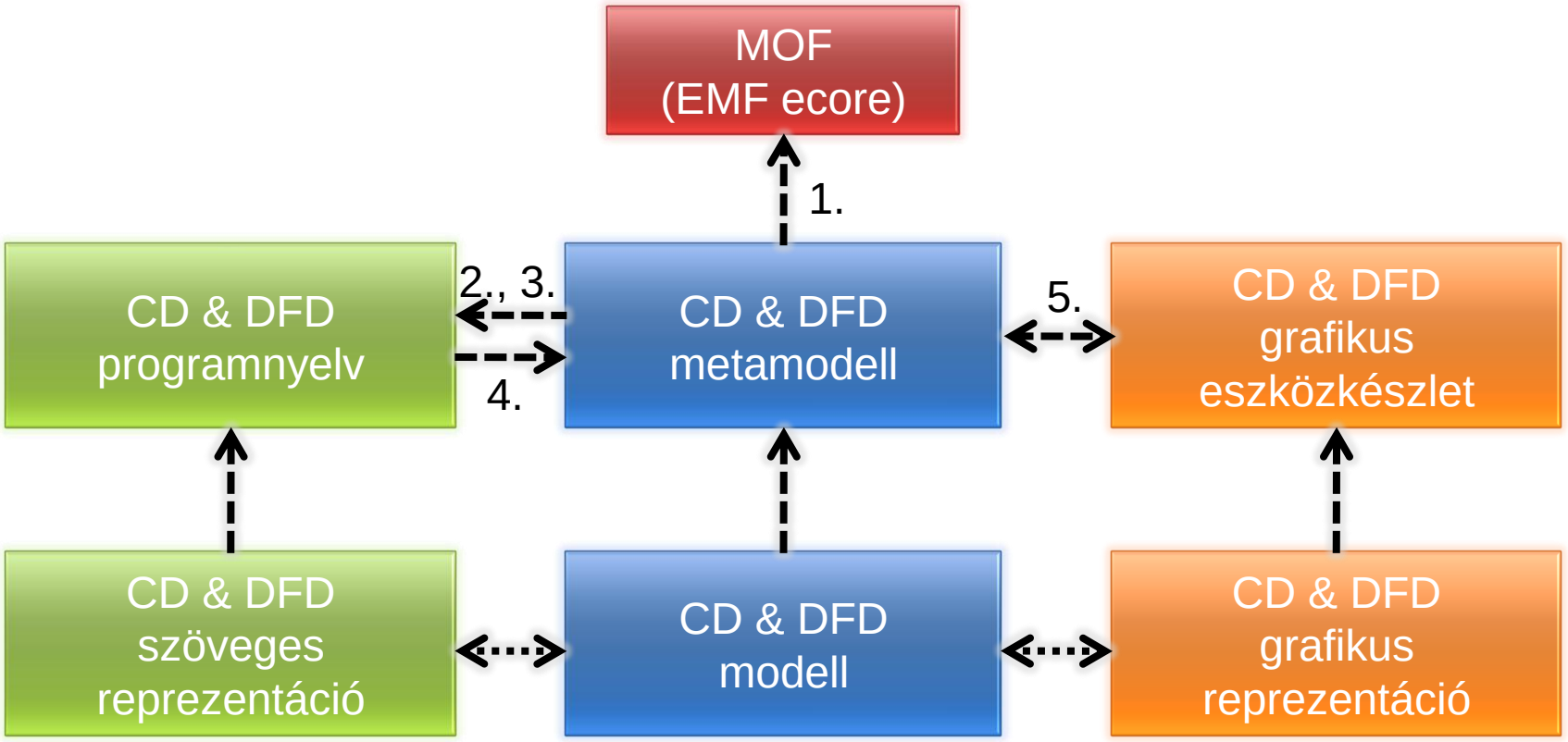


Feltételek

- Store csak process-hez kapcsolható
- Context diagramon nincs store
- Egy store több szinten is előfordulhat



Gyakorlatok



Eclipse Modeling Framework (EMF)

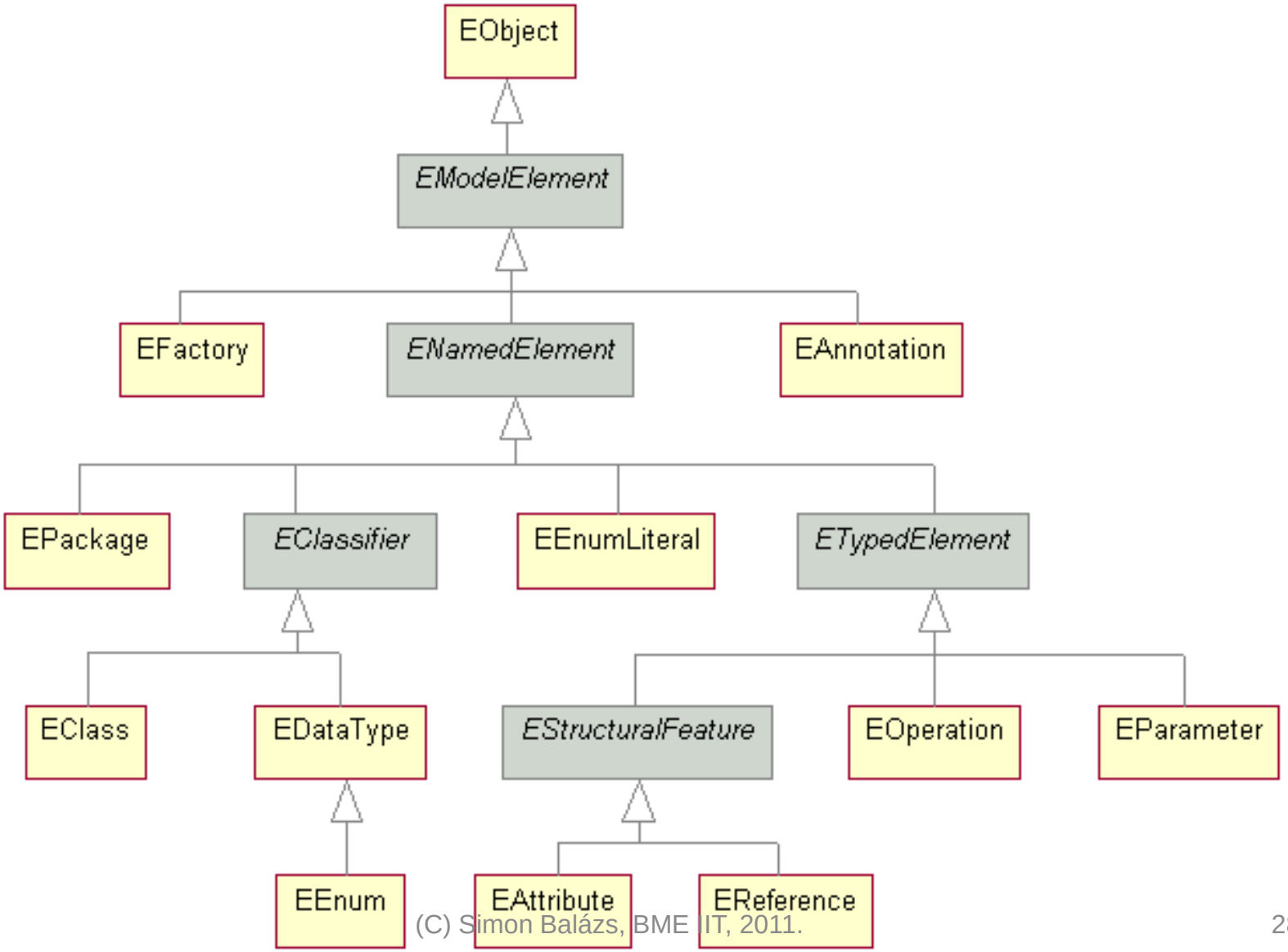
Eclipse Modeling Framework (EMF)

- Java alapú keretrendszer strukturált modellező nyelvek létrehozására
- EMF ecore:
  - az EMF magja
  - a MOF-nak felel meg, de kicsit eltér
- Saját modellező nyelv létrehozása: ecore-ra építve
- Létezik EMF-re épülő UML metamodell is
- Modell mentése, visszatöltése: XMI

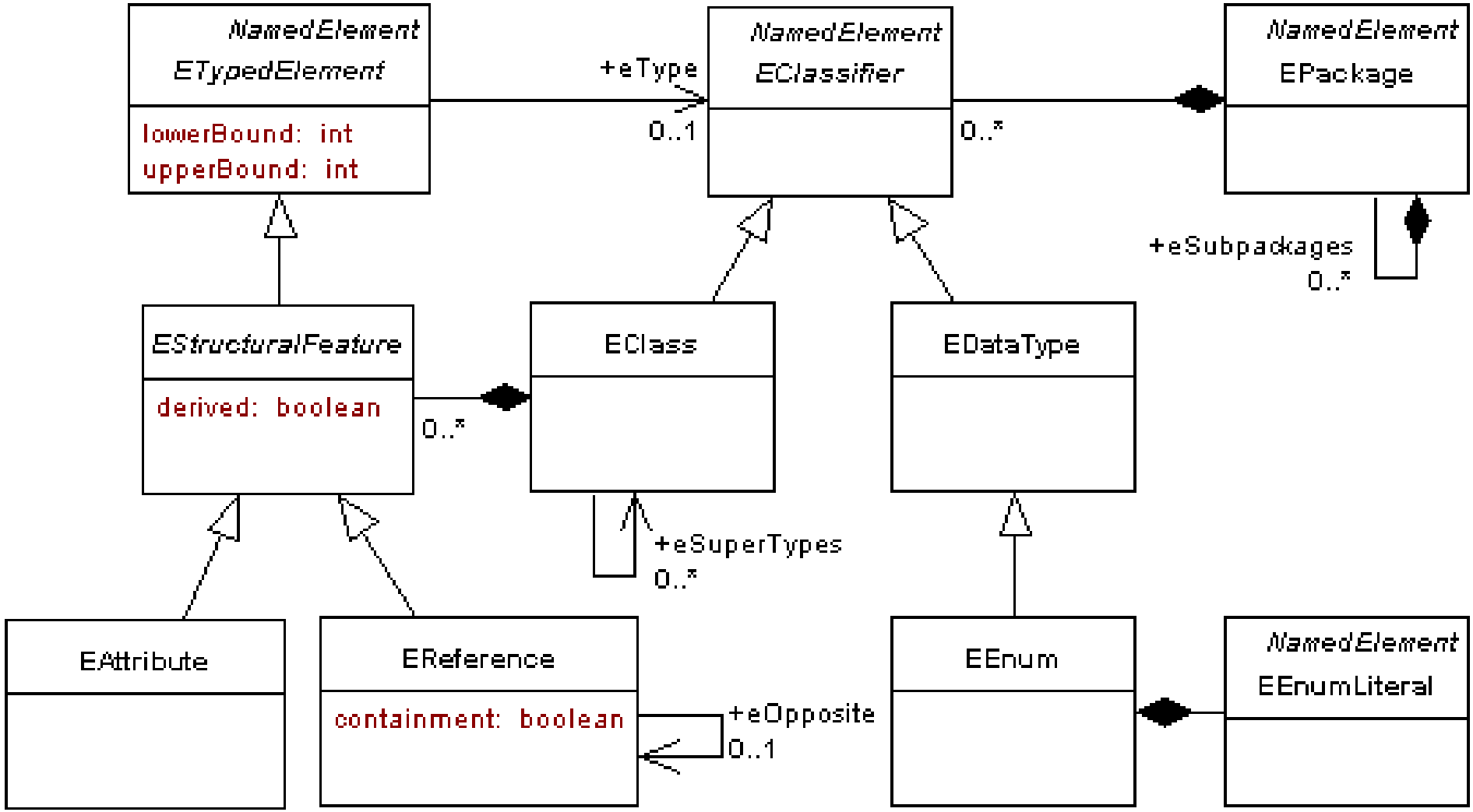
Követelmények

- Tervezéshez:
  - Eclipse Helios (3.6) + “Modeling/EMF – Eclipse Modeling Framework SDK” plugin telepítése
- Egyszerű felhasználáshoz Java kódból:
  - Saját metamodellből generált .jar fájl
  - org.eclipse.emf.ecore.jar
  - org.eclipse.emf.common.jar
- Összetettebb felhasználáshoz (grafikus editorral):
  - Eclipse fejlesztőkörnyezet

ecore


EObject
EModelElement
EFactory
ENamedElement
EAnnotation
EPackage
EClassifier
EEnumLiteral
ETypedElement
EClass
EDataType
EStructuralFeature
EOperation
EParameter
EEnum
EAttribute
EReference

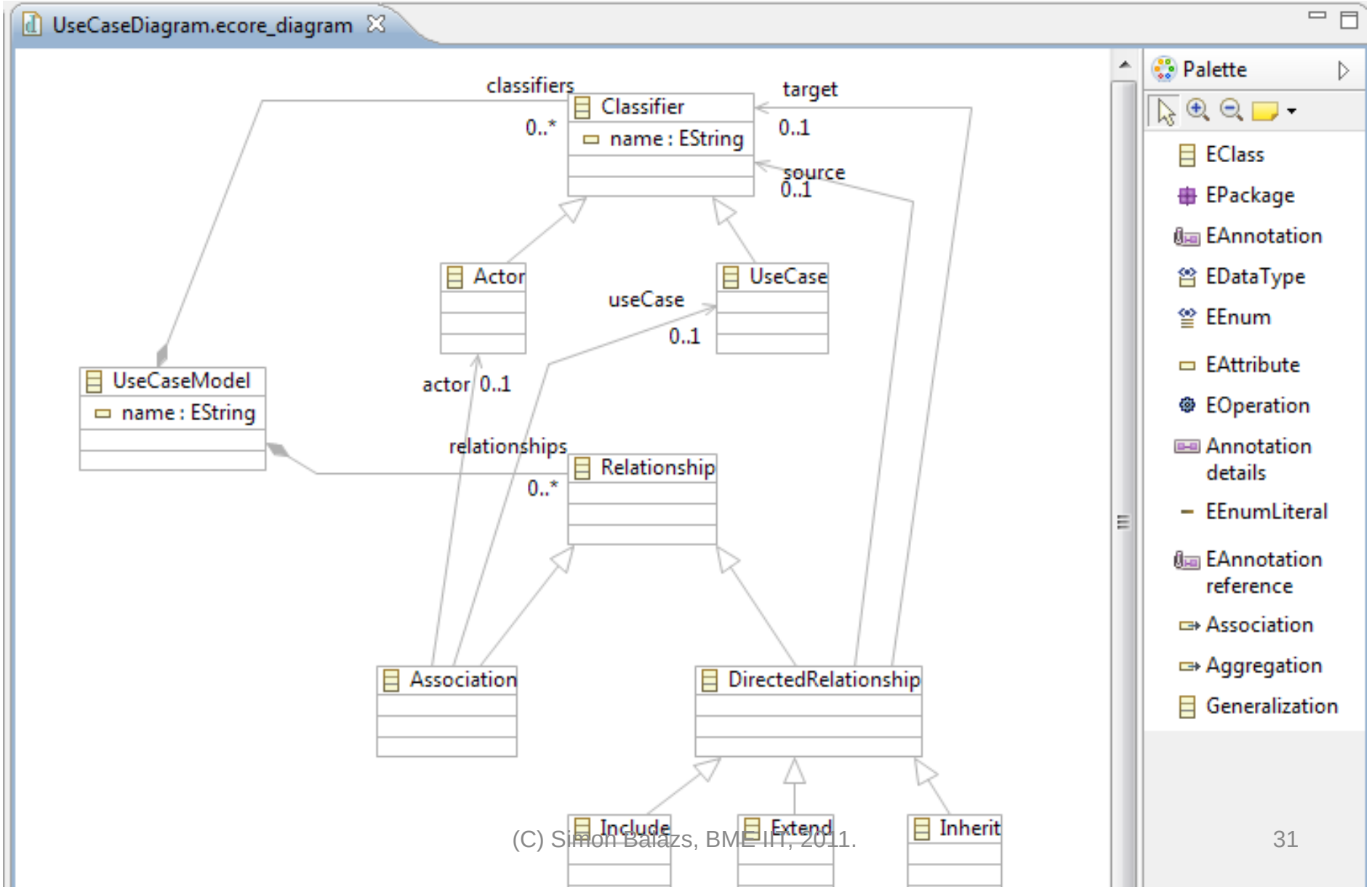
ecore


NamedElement
ETypedElement
lowerBound: int
upperBound: int
+eType
0..1
NamedElement
EClassifier
0..*
NamedElement
EPackage
+eSubpackages
0..*
EStructuralFeature
derived: boolean
0..*
EClass
+eSuperTypes
0..*
EDataType
EAttribute
EReference
containment: boolean
+eOpposite
0..1
EEnum
NamedElement
EEnumLiteral

Saját metamodell létrehozása

- Négy lehetséges módszer:
  - Java interfészek kommentbe írt annotációkkal
  - UML osztálydiagramszerű grafikus editor
  - XMI
  - XML Schema
- Végül mindegyik a Java osztályokra képződik le
  - kicsit megtévesztő: nem Java 1.5-ös annotációk, hanem kommentbe írt speciális karakterláncok

Grafikus editor



Java annotations

- 1. Java csomag létrehozása
- 2. Csomagon belül interfészek létrehozása:
  - minden metamodell-beli elemnek egy-egy
  - interfészek elé komment:
    - (implementációja osztály lesz)

```
/**
 * @model
 */
```

  - használható még az abstract módosító:
    - (implementációja absztrakt osztály lesz)

```
/**
 * @model abstract="true"
 */
```

  - többszörös öröklődés megengedett (interfész szinten)

Java annotations

- 3. Interfészeken belül:
  - getter függvények fejléce, ezekből lesznek az implementáló osztály attribútumai
  - getter-ek előtt komment:

```
/**
 * @model
 */
```

  - lehetséges módosítók:
    - default="…" (alapértelmezett érték megadása)
    - changeable="false" (csak olvasható attribútum)
    - containment="true" (lista esetén: kompozíció)

Példa

```
package tree.model;

import java.util.List;

/**
 * @model
 */
public interface Composite extends Node {
        /**
         * @model containment="true"
         */
        public List<Node> getChildren();
}
```

```
package tree.model;

/**
 * @model
 */
public interface Node {
        /**
         * @model
         */
        public Composite getParent();
}
```

Java annotations

- Az interfészek alapján az Eclipse generálni tudja az implementációt
- Csak a @**model**-lel annotált elemeket veszi figyelembe, minden mást változatlanul hagy
- A gyökérinterfészeket az **EObject**-ből származtatja, implementációjukat pedig **EObjectImpl**-ből
- A listákat lecseréli **EList**-re
- Ha változtatható az attribútum, setter-t is generál (kivéve listáknál)

Generált implementáció

- A saját csomagon belül:
  - **ModelFactory** interfész
  - **ModelPackage** interfész
  - két alcsomag:
    - **[saját csomag].impl**
    - **[saját csomag].util**
- Az **impl** alcsomag tartalma:
  - minden eredeti interfészhez egy **[interfésznév]Impl** nevű implementáló osztály
    - sok generált függvény @**generated** annotációval ellátva
    - ezeket újrageneráláskor felülírja
    - az annotációt eltávolítva saját kód is írható
  - **ModelFactoryImpl** osztály
  - **ModelPackageImpl** osztály

ModelFactory

- A modell példányosításáért felel
- Singleton, elérése:
  - **ModelFactory.eINSTANCE**
- Minden definiált interfészhez egy **.create[interfésznév]()** függvény
- Példa:

```
ModelFactory factory = ModelFactory.eINSTANCE;
Node node = factory.createNode();
```

Összefoglalás

- Metamodellezés
- Transzformációk:
  - metamodell
  - kódgenerálás
  - refaktorálás
  - grafikus modellezés
- Saját metamodell készítése: EMF