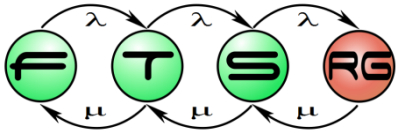
Domain-specifikus modellezés az Eclipse Modeling Framework használatával



Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Domain-specifikus nyelvek: miért?

- A mai szoftverfejlesztés kihívásai
  - Komplexitás
    - Növekvő fejlesztési idő és költség
  - Sokféleség
    - Domain, követelmény, implementációs technika, eszköz
  - Változás
    - Környezet, követelmények, hibajavítások…

Domain-specifikus nyelvek

- Cél: a szakterületi fogalmakkal fejezzük ki magunkat
  - Absztrakciós szint növelése
  - Szakértő (nem fejlesztő) is értse a modellt!
- Domain-specifikus nyelv (Domain Specific Language)
  - Pl. HTML, SQL, reguláris kifejezések
- Lehetne általános célú nyelv (General Purpose Language)
  - Pl. Java, C stb.
  - De túl implementációspecifikus
  - Szakértő nem tudja hatékonyan használni

Domain-specifikus nyelvek: mikor?

- Előnyök
  - Specifikus problémák gyorsabb megoldása
  - Követelmények kifejezése a problématér szintjén
  - Automatikus validáció
- Hátrányok
  - Jártasság szükséges a szakterületben (költség)
  - Nyelvtan megalkotása szubjektív, bonyolult
  - Eszköztámogatás fejlesztése költséges lehet (a behatárolt alkalmazhatósághoz képest)
  - Inkompatibilis DSL-ek elterjedhetnek ugyanarra a területre
    - Szabványosítás

Nyelv-alapú fejlesztés

- Nyelv:
  - Modellek leírása
  - Emberek számára
  - Cél: problématerület áttekintése
- Biztosítandó
  - Futtatás
  - Analízis
  - Tesztelés
  - Implementáció
  - …

Nyelvek együttes kezelése

- Feladat: több nyelv együttes kezelése
  - Transzformáció
  - Aspektus integráció
  - Szinkronizáció
  - Finomítás és ekvivalencia vizsgálata
  - Evolúció
- Hasonlóan az aspektus-orientált programozáshoz
  - Több szempont
  - Azonos rendszer

Az UML lehetőségei és korlátai

- Előnyök:
  - Standard közös nyelv
  - Vizuális
- Hátrányok:
  - Nem domain-specifikus
  - Korlátozott hatókör
    - UML Profilok
  - Nem formális szemantika
    - Nincs egy közös, általánosan használható szemantika
  - Unified (NOT Universal) Modeling Language

Metamodellezés

Modellező nyelvek tervezése

Modellező nyelvek tervezése

- Metamodellezés: Tervezési metodológia modellező nyelvekhez
- Metamodell: egy modellező nyelv modellje
- Részei:
  - Absztrakt szintaxis
  - Konkrét szintaxis
  - Jólformáltsági szabályok
- További lehetőségek:
  - Szemantika
  - Leképezések más nyelvekre

Konkrét szintaxis

- Ahogy a felhasználó “látja”
- Nyelvenként lehet többféle is
- Fajtái
  - Szöveges szintaxis
  - Grafikus szintaxis

Konkrét szintaxis

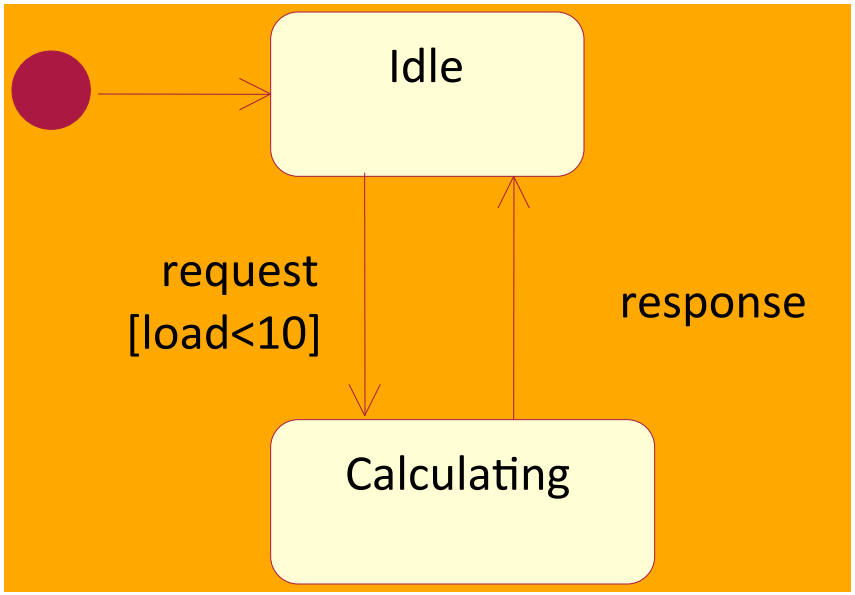
- Grafikus leírás
  - + Könnyen olvasható: Intuitív, könnyen érthető jelölésrendszer
  - + Biztonságosan írható: Csak szintaktikailag helyes modellek készíthetőek
  - – Nehéz írni: A grafikus szerkesztés lassabb…
  - – Nem jól skálázódik
    - – Nagy modellek kezelése problémás
- Eclipse technológiák: jövő héttől

Konkrét szintaxis

- Szöveges leírás
  - + Könnyen írható
    - Komplex kifejezések megadása gyorsan
  - + Jól skálázódik
    - 10k soros fájl jól kezelhető
  - – Nehéz olvasni
    - Áttekinthetőség, kapcsolatok megjelenítésének hiánya
  - – Nehéz karbantartani
    - Névvel történő hivatkozások
- Eclipse technológiák: októberben

Példa: konkrét szintaxis

- Grafikus jelölés



- Szöveges jelölés

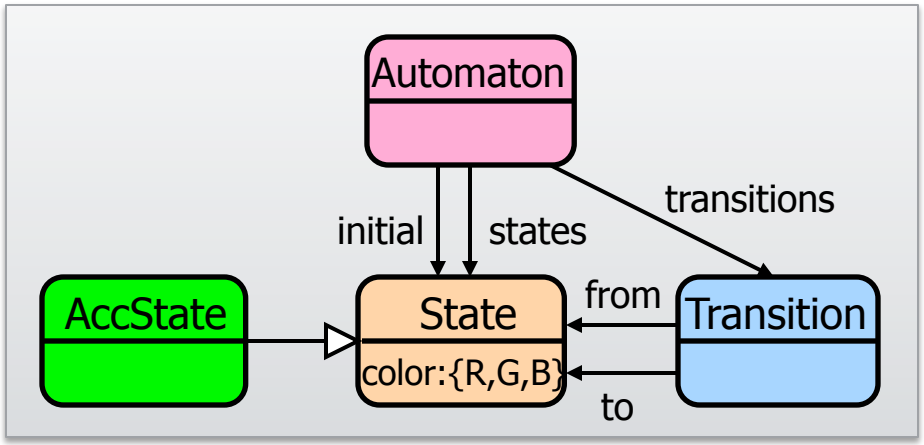
```
void request() {
 if(state == IDLE &&
     this.load < 10)
        state =
  CALCULATING;
}

void response() {
 if (state ==
  CALCULATING)
        state = IDLE;
}
```

Absztrakt szintaxis (metamodell)

- Metamodell: a modellező nyelv modellje
- Cél: definiálni...
  - a nyelv fogalomkészletét
  - a fogalmak közötti kapcsolatokat
    - Asszociációk/referenciák
    - Attribútumok
    - Absztrakció/finomítás (ld. taxonómiák, ontológiák)
  - Jólformáltsági szabályok
    - Számosság
    - Egyéb kényszerek (pl. egyedi név)

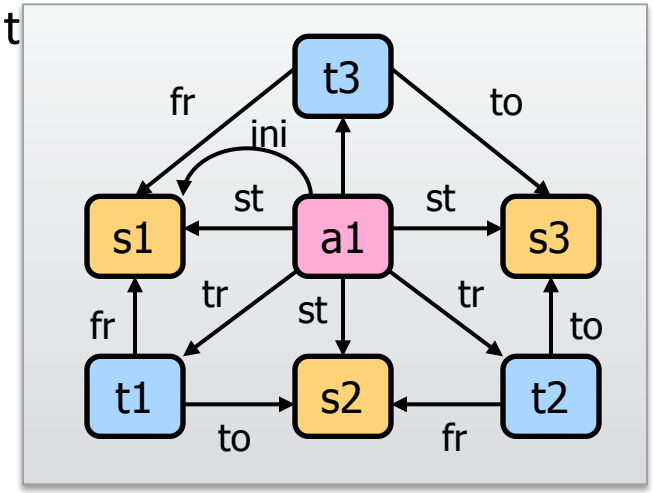
Metamodellek és példányok



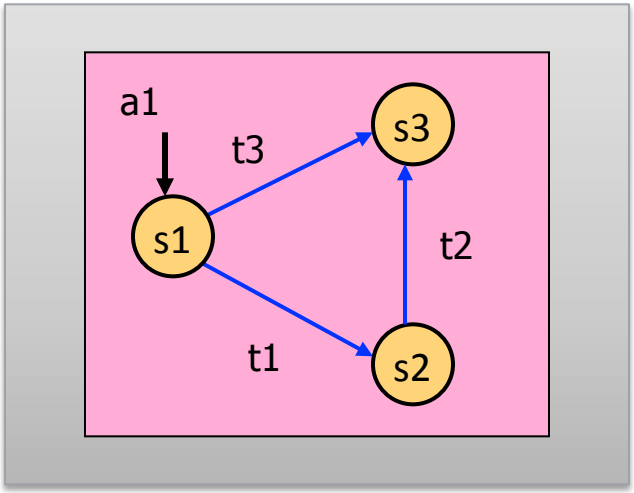
Metamodell

Metamodell szint

Modell szint

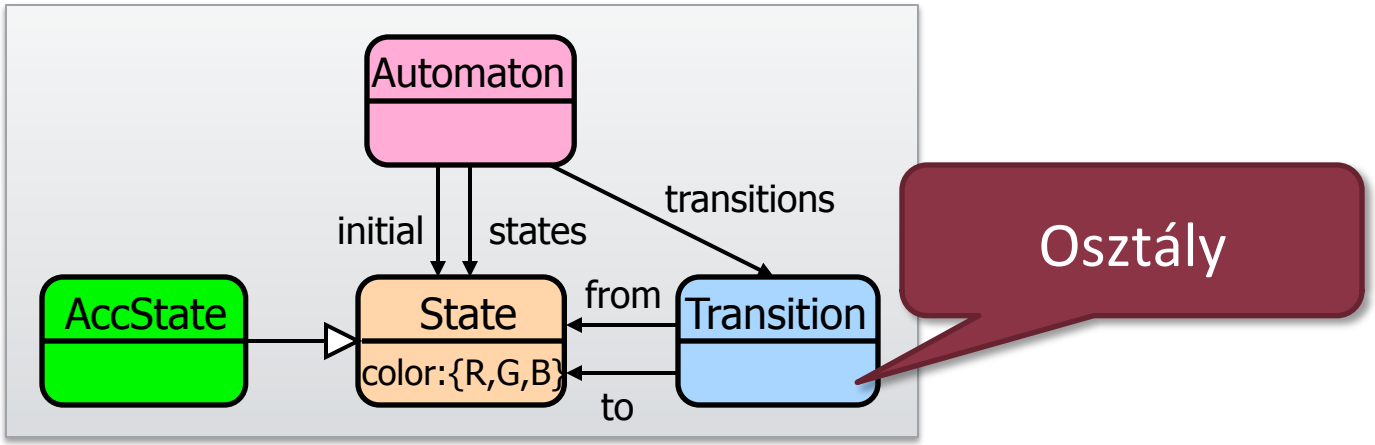


Absztrakt szintaxis



Konkrét szintaxis

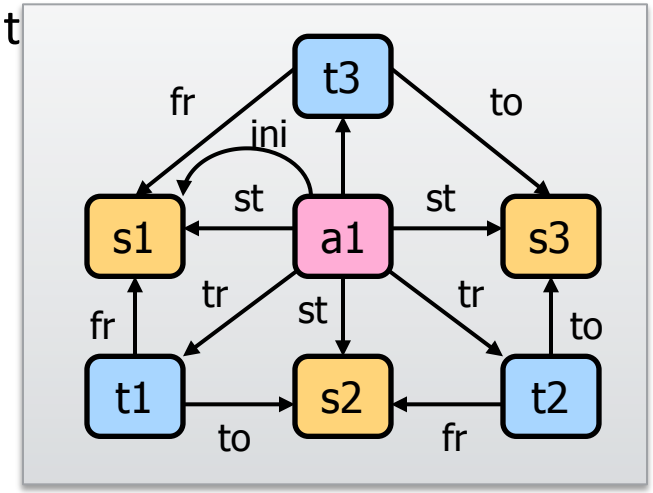
Metamodellek és példányok



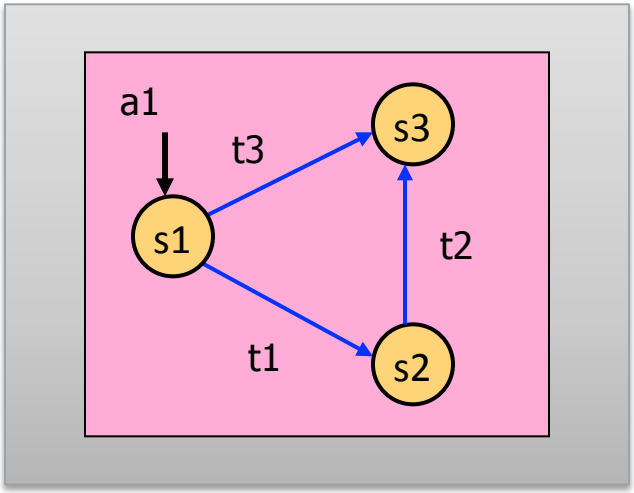
Metamodell

Metamodell szint

Modell szint

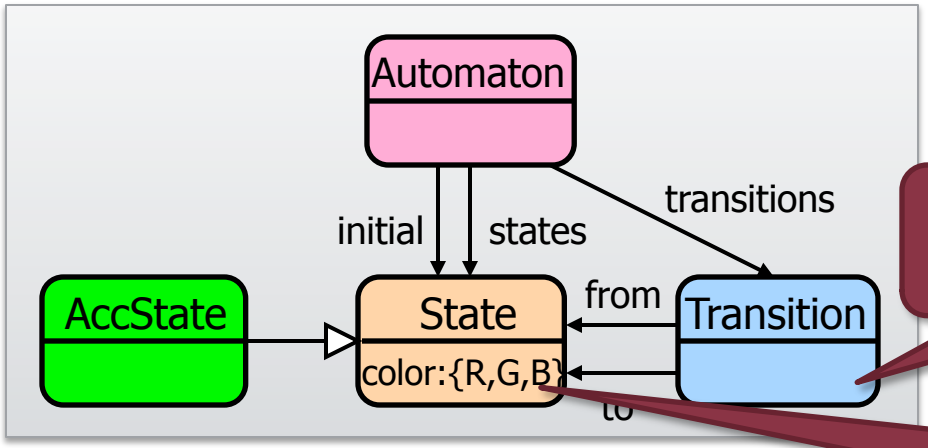


Absztrakt szintaxis



Konkrét szintaxis

Metamodellek és példányok



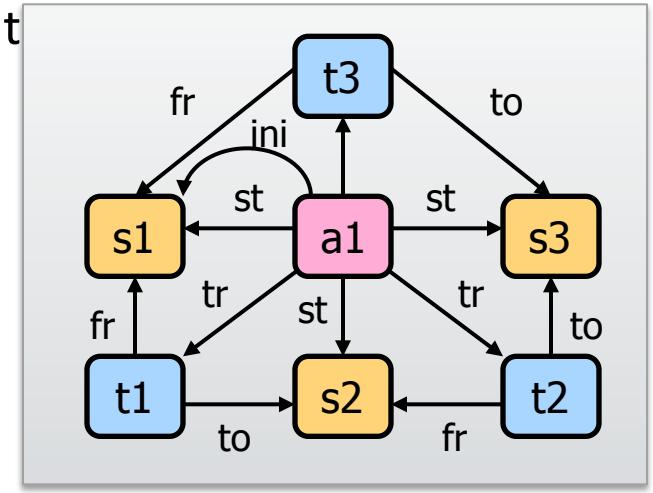
Osztály

Attribútum

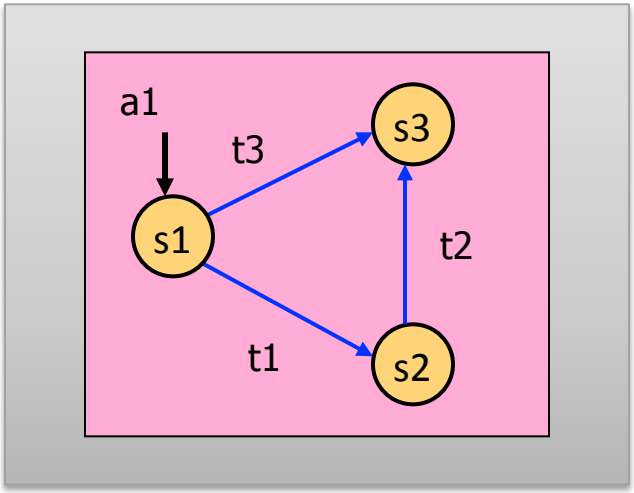
Metamodell

Metamodell szint

Modell szint

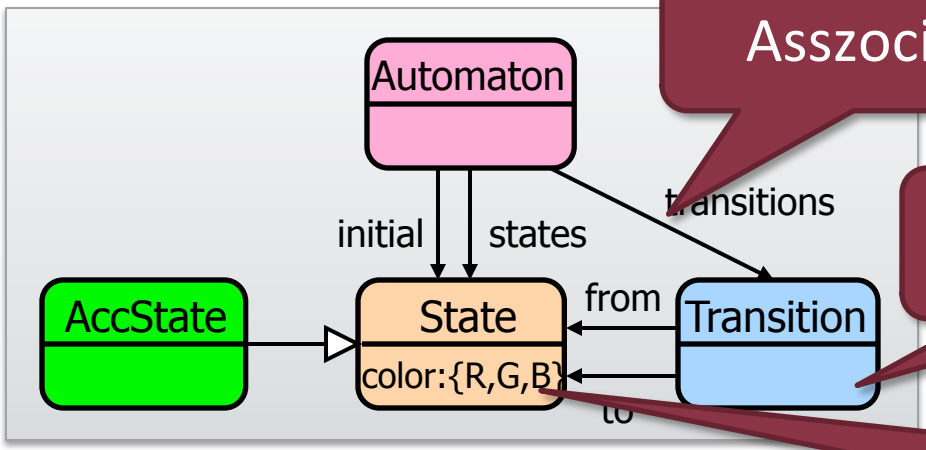


Absztrakt szintaxis



Konkrét szintaxis

Metamodellek és példányok



Asszociáció

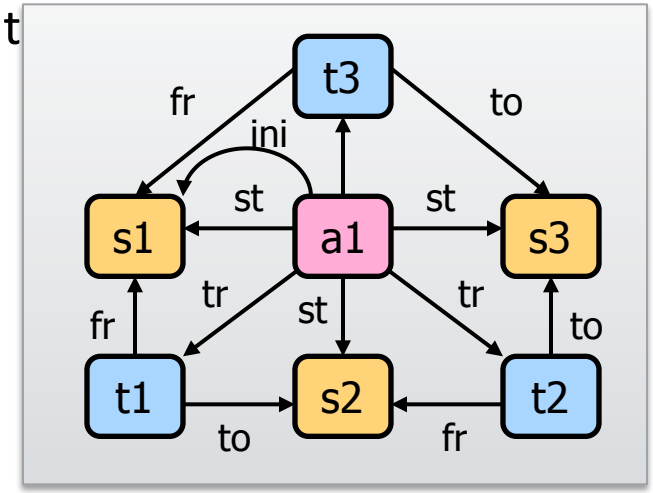
Osztály

Attribútum

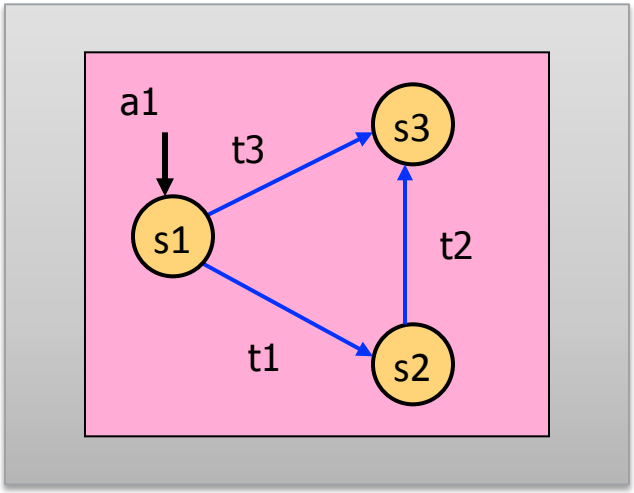
Metamodell

Metamodell szint

Modell szint

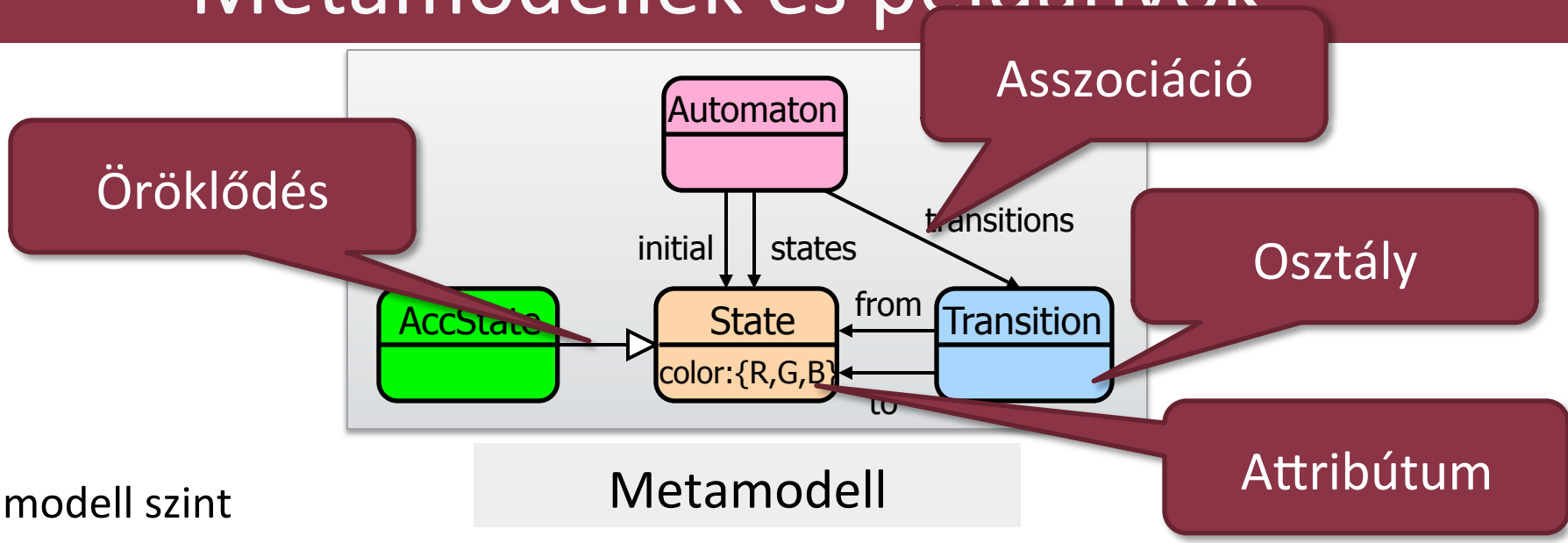


Absztrakt szintaxis



Konkrét szintaxis

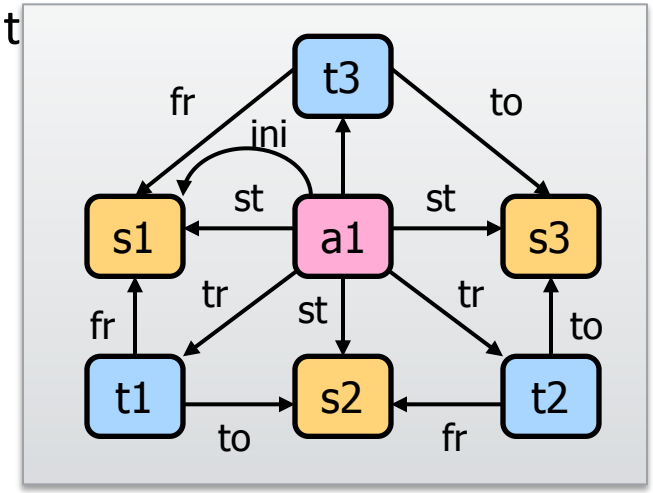
Metamodellek és példányok



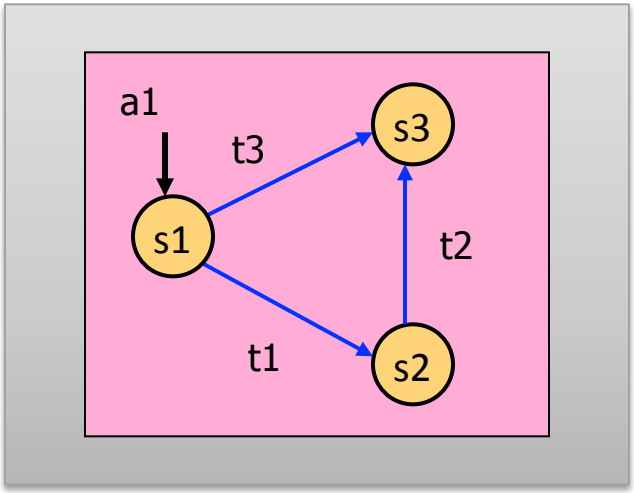
Metamodell szint

Metamodell

Modell szint

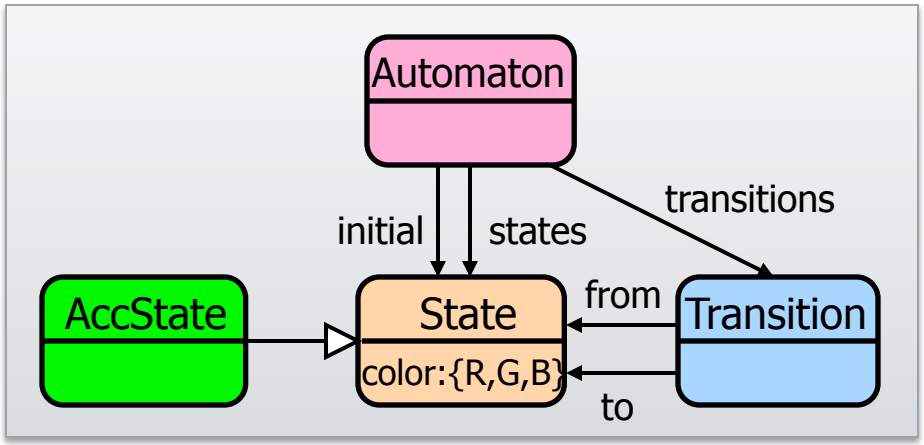


Absztrakt szintaxis



Konkrét szintaxis

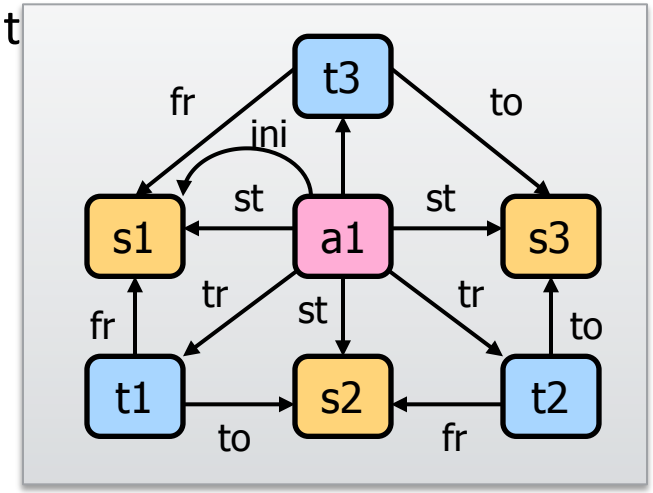
Metamodellek és példányok



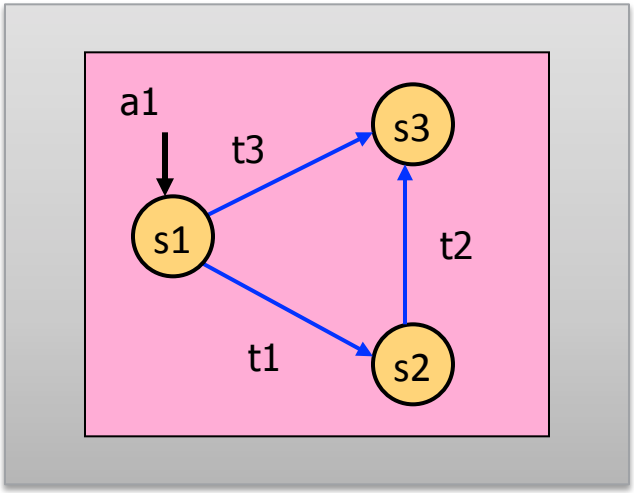
Metamodell

Metamodell szint

Modell szint

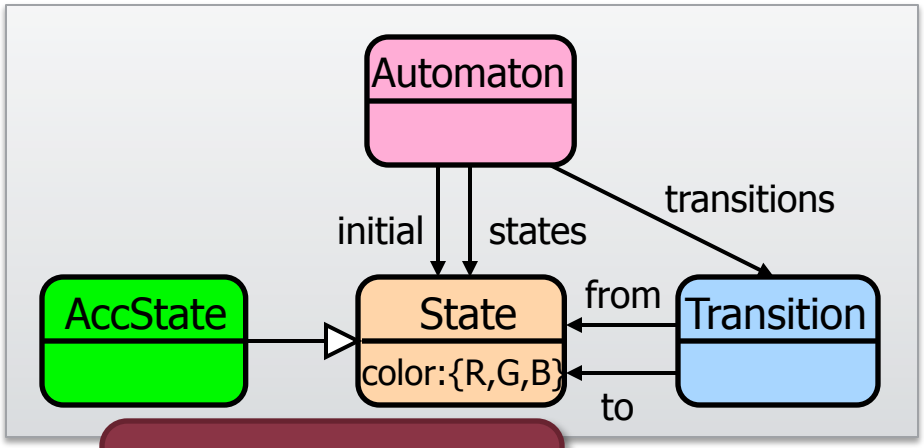


Absztrakt szintaxis



Konkrét szintaxis

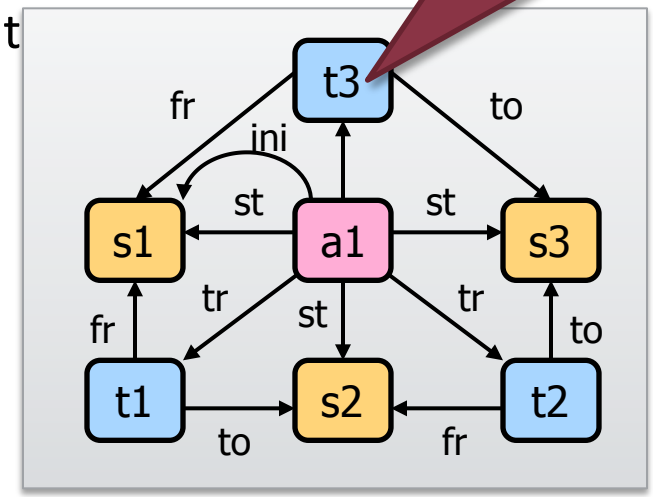
Metamodellek és példányok

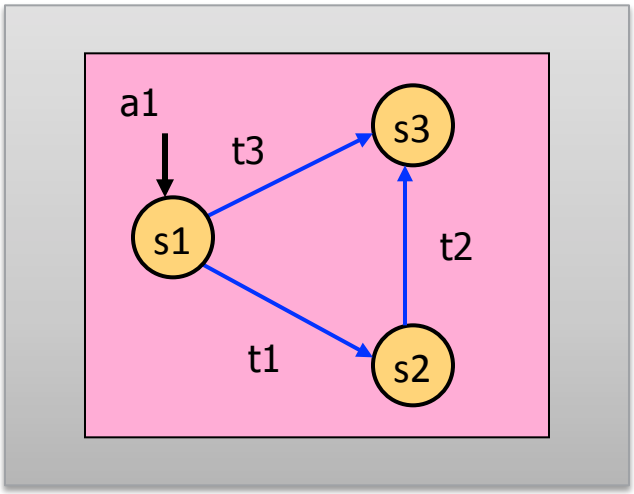


Objektum

Metamodell szint

Modell szint

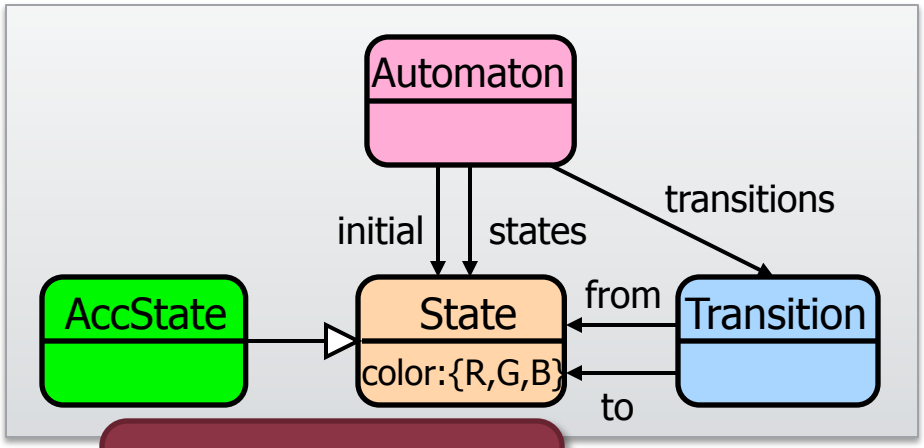




Absztrakt szintaxis

Konkrét szintaxis

Metamodellek és példányok

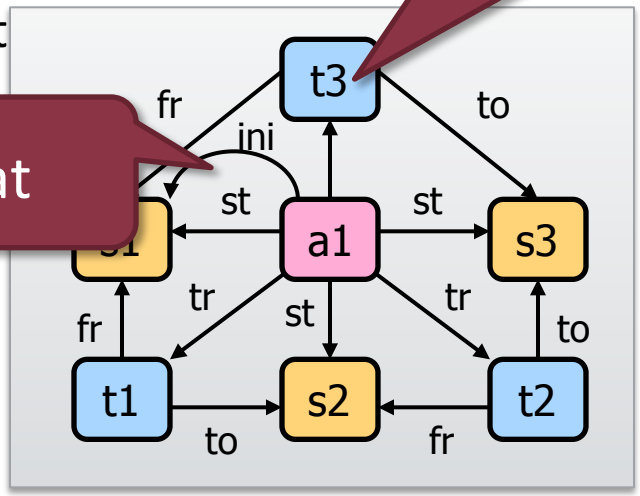


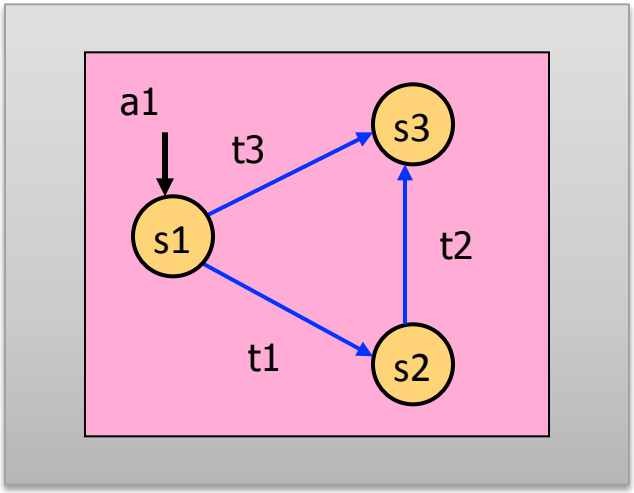
Objektum

Metamodell szint

Modell szint

Kapcsolat

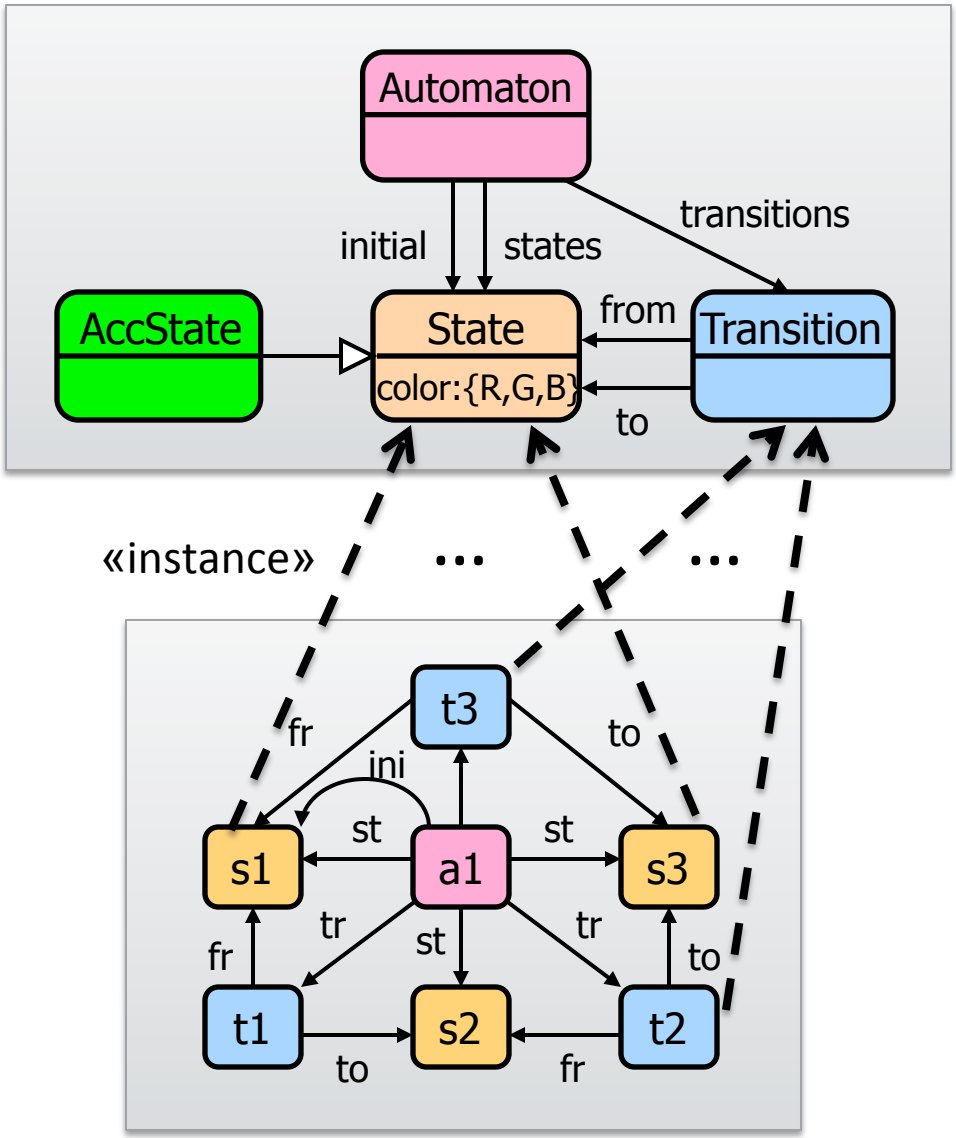




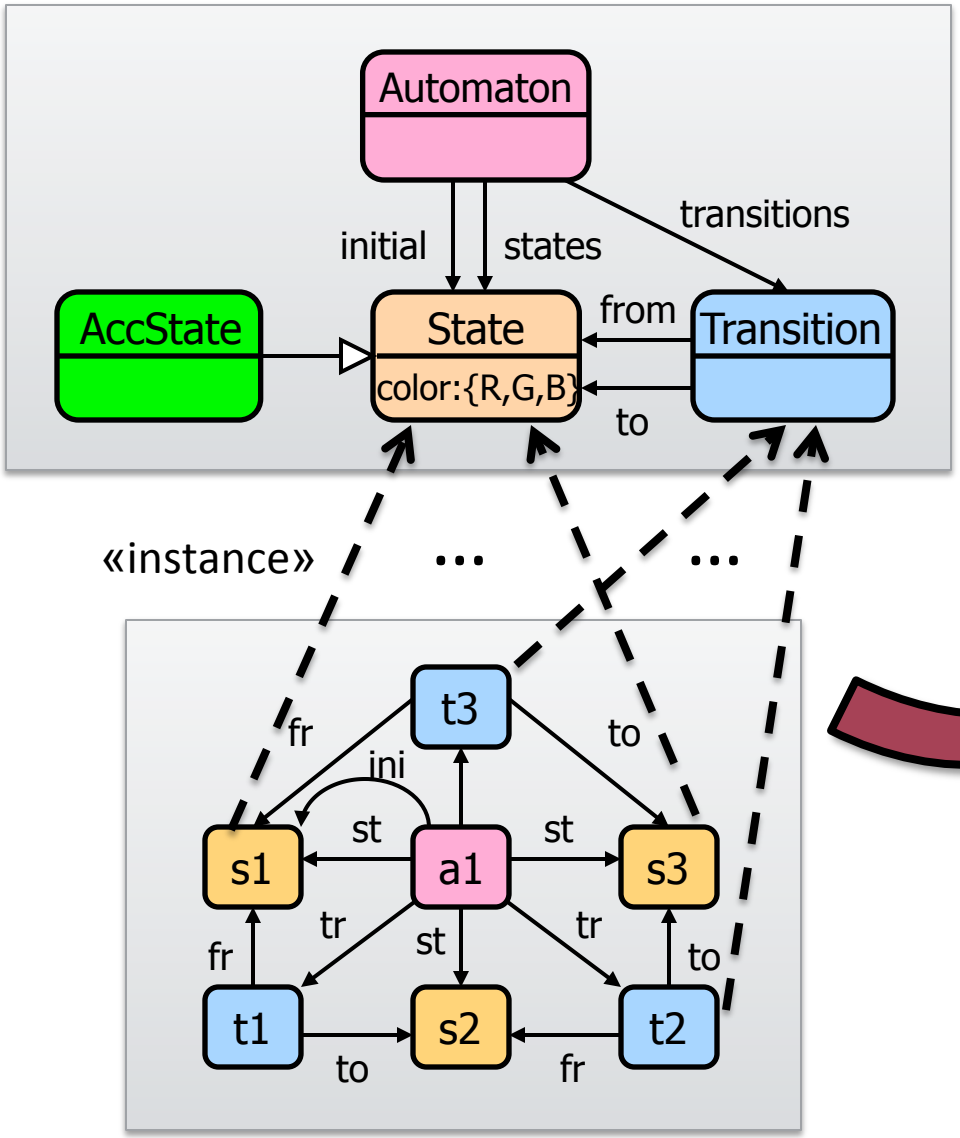
Absztrakt szintaxis

Konkrét szintaxis

Példányosítás



Példányosítás



Kapcsolat modellszint és metamodell szint között

Jólformáltsági szabályok

- Számossági kényszerek
  - Legfeljebb egy: 0..1
  - Sok: *
- Aggregáció/Tartalmazás
  - Minden modellelemnek legfeljebb egy szülő
- Nyelvspecifikus kényszerek
  - Pl. egyedi nevek
  - Pl. OCL-ben (Object Constraint Language) kifejezhető

Szemantika

- Szemantika: a nyelv fogalmainak jelentése
- Hogyan értelmezzük a modellt
- Statikus szemantika
  - Eddig erről volt szó
- Dinamikus szemantika
  - Lehetséges viselkedés
  - Állapotváltozások

Szemantika

- Fő megközelítések:
  - Denotációs
    - Egyik nyelv fogalmainak lefordítása a másik nyelvre
    - Fordított
  - Operációs
    - A nyelvi fogalmak viselkedésének modellezése
    - Interpretált
  - ~~Axiomatikus~~
    - ~~Logikai formulák~~
    - ~~Nehezen értelmezhető~~

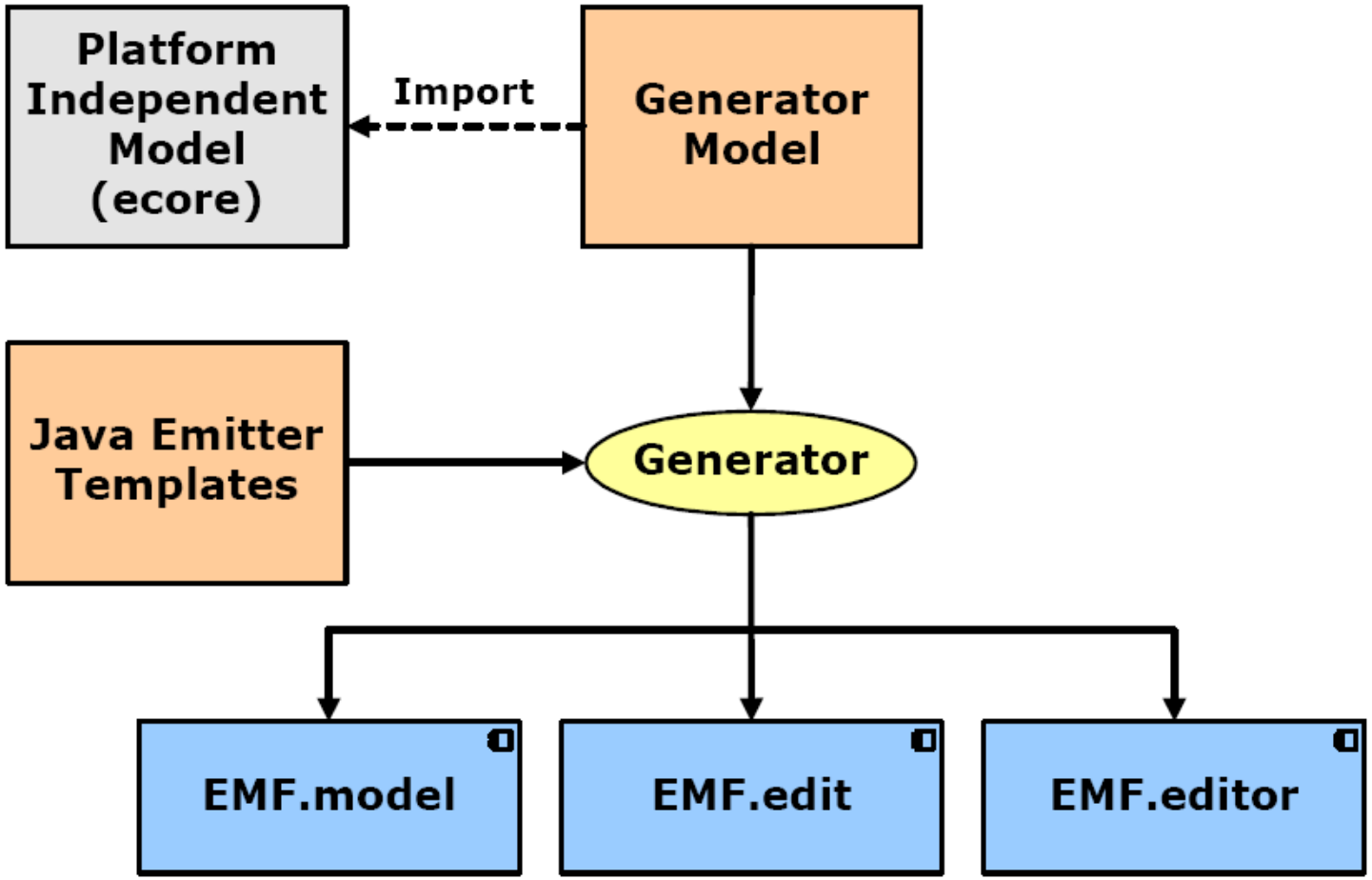
Eclipse Modeling Framework (EMF)

Metamodellezés Eclipse alatt

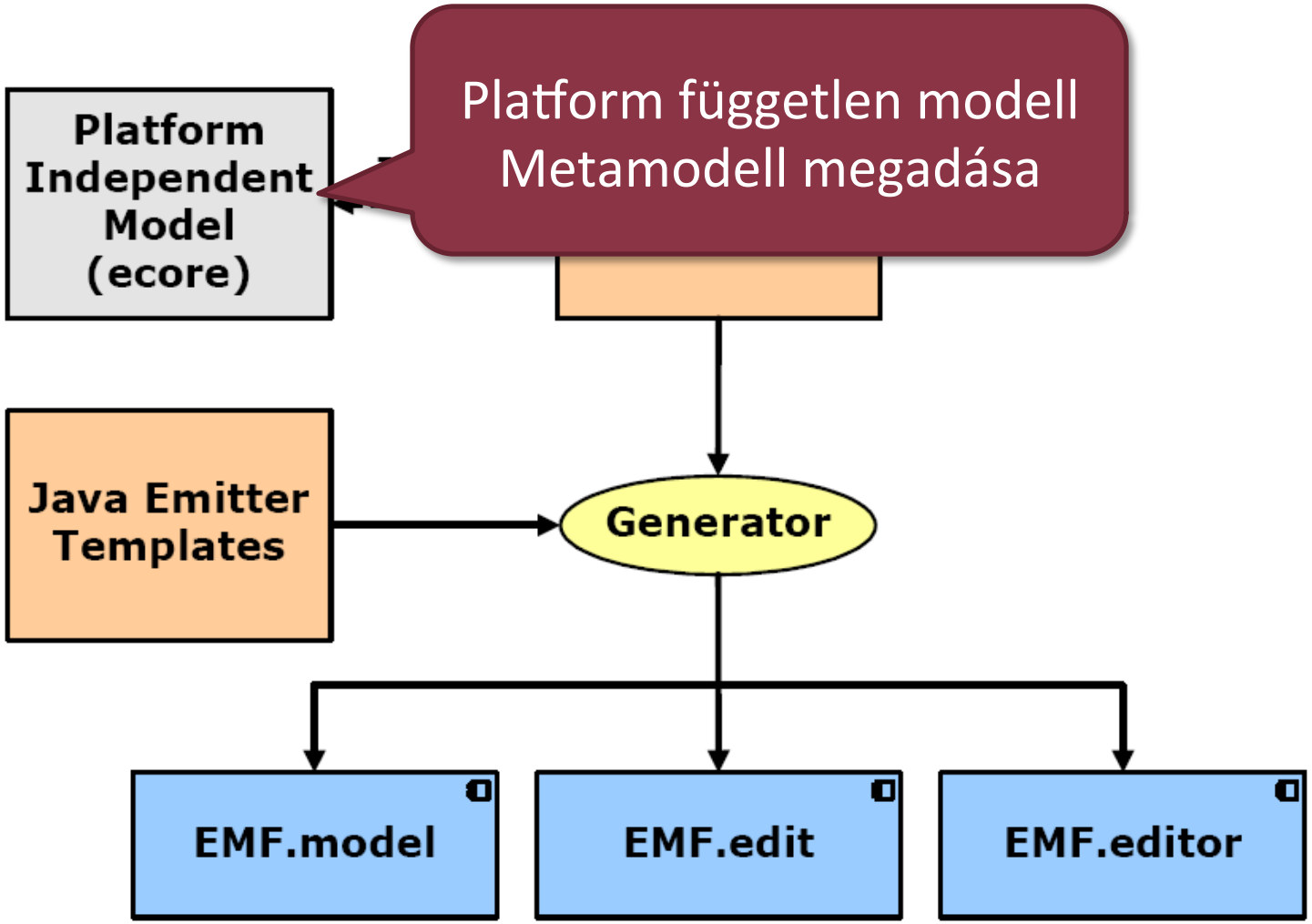
Eclipse Modeling Framework

- Modellezési komponens Eclipse alá
- Lehetőség domain-specifikus nyelvek definiálására
- Szerkesztés
  - Alapvető parancsok
  - Értesítés változásokról
  - Visszavonás
  - Alapvető editor komponens
- XML/XMI export-import támogatás
- Saját metamodellező mag

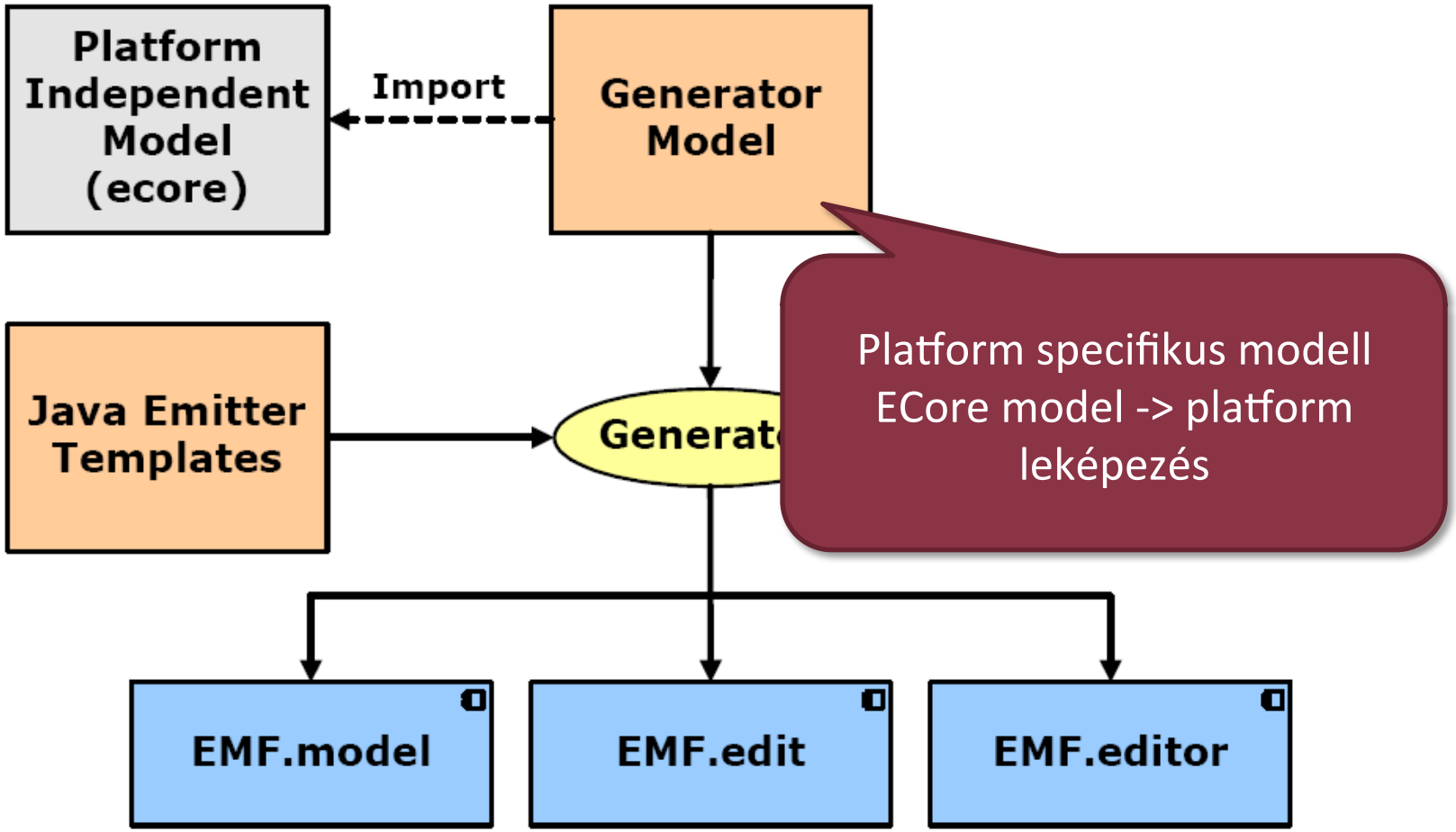
Az EMF eszközkészlet



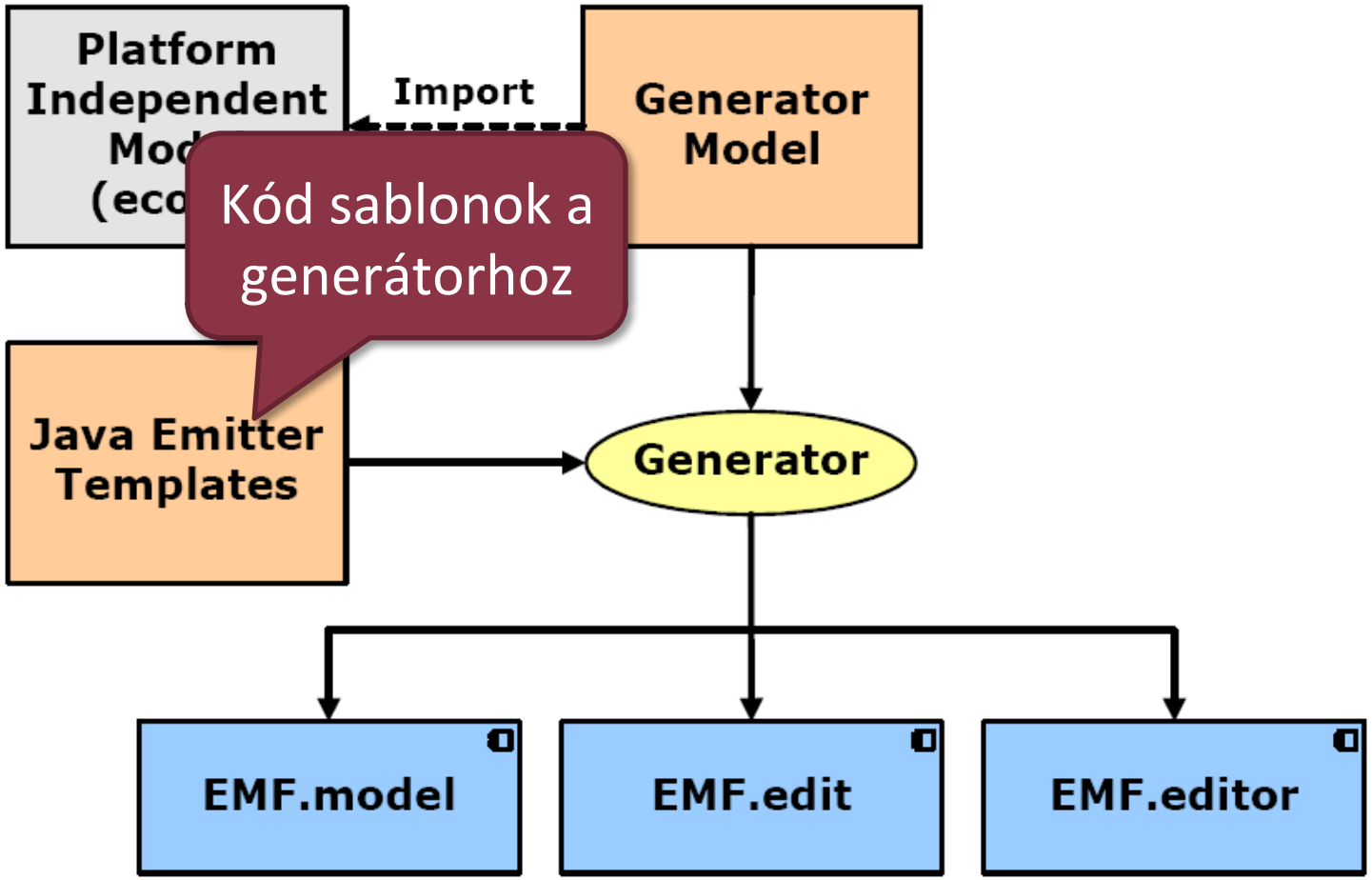
Az EMF eszközkészlet



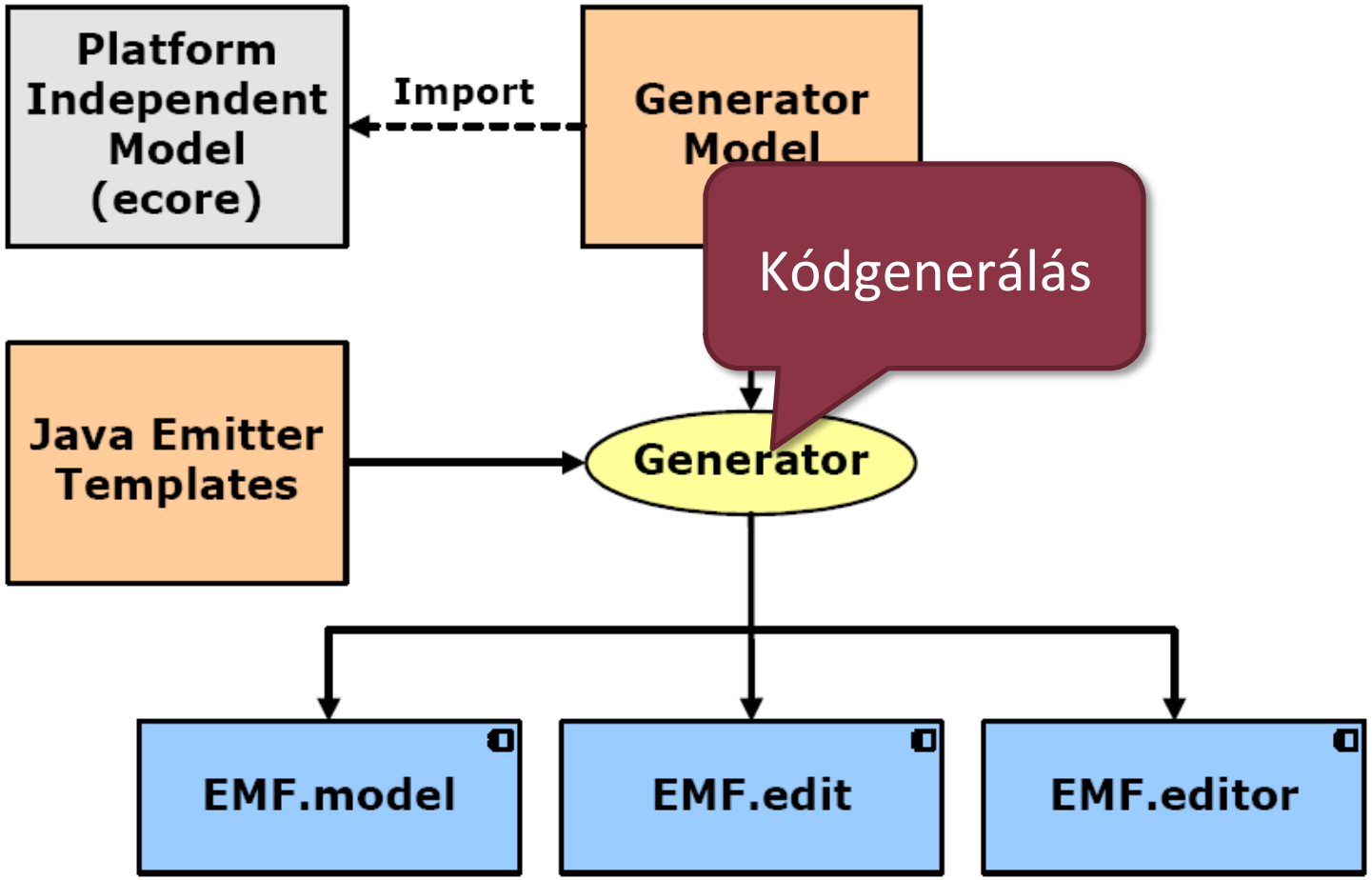
Az EMF eszközkészlet



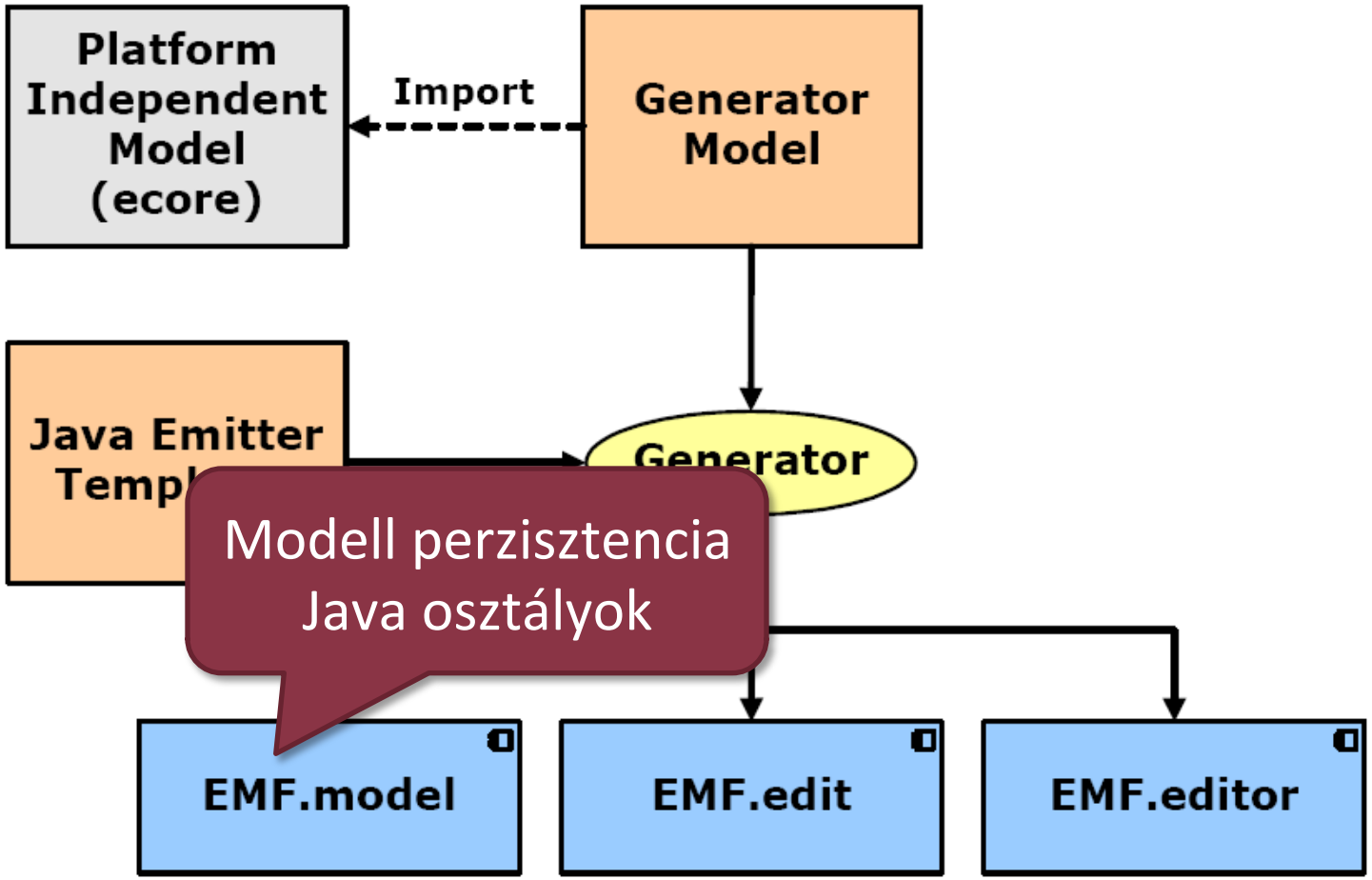
Az EMF eszközkészlet



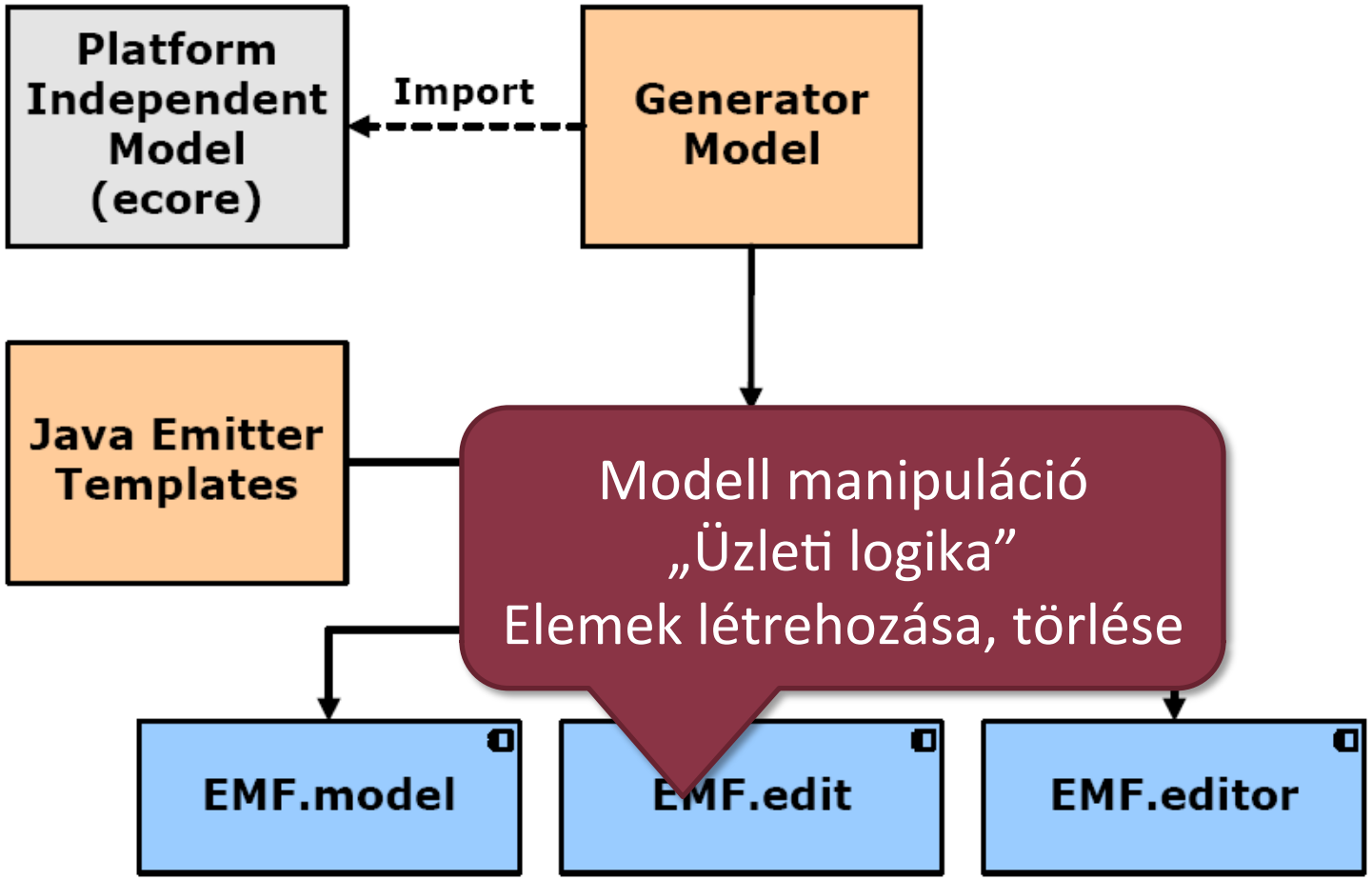
Az EMF eszközkészlet



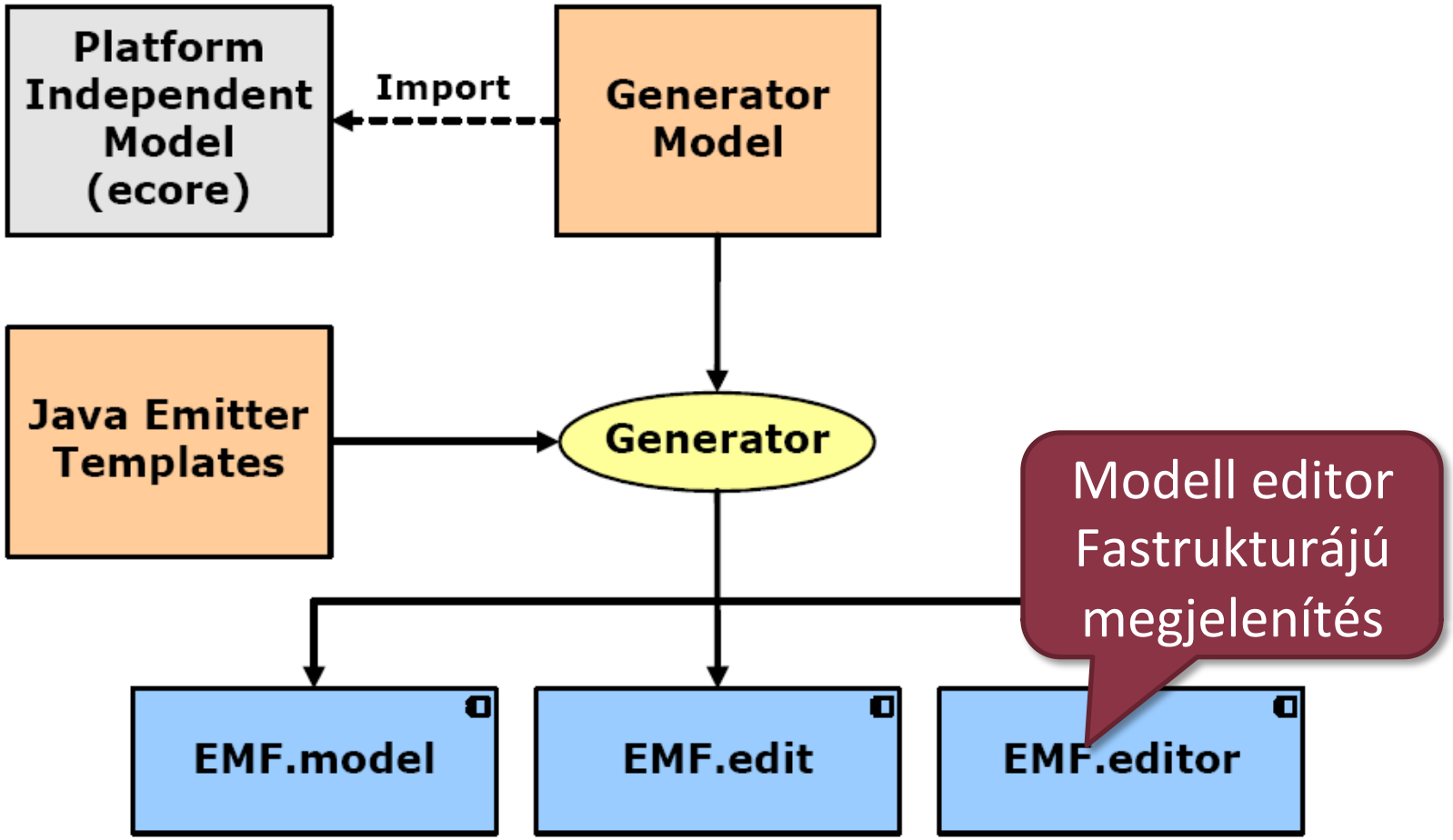
Az EMF eszközkészlet



Az EMF eszközkészlet



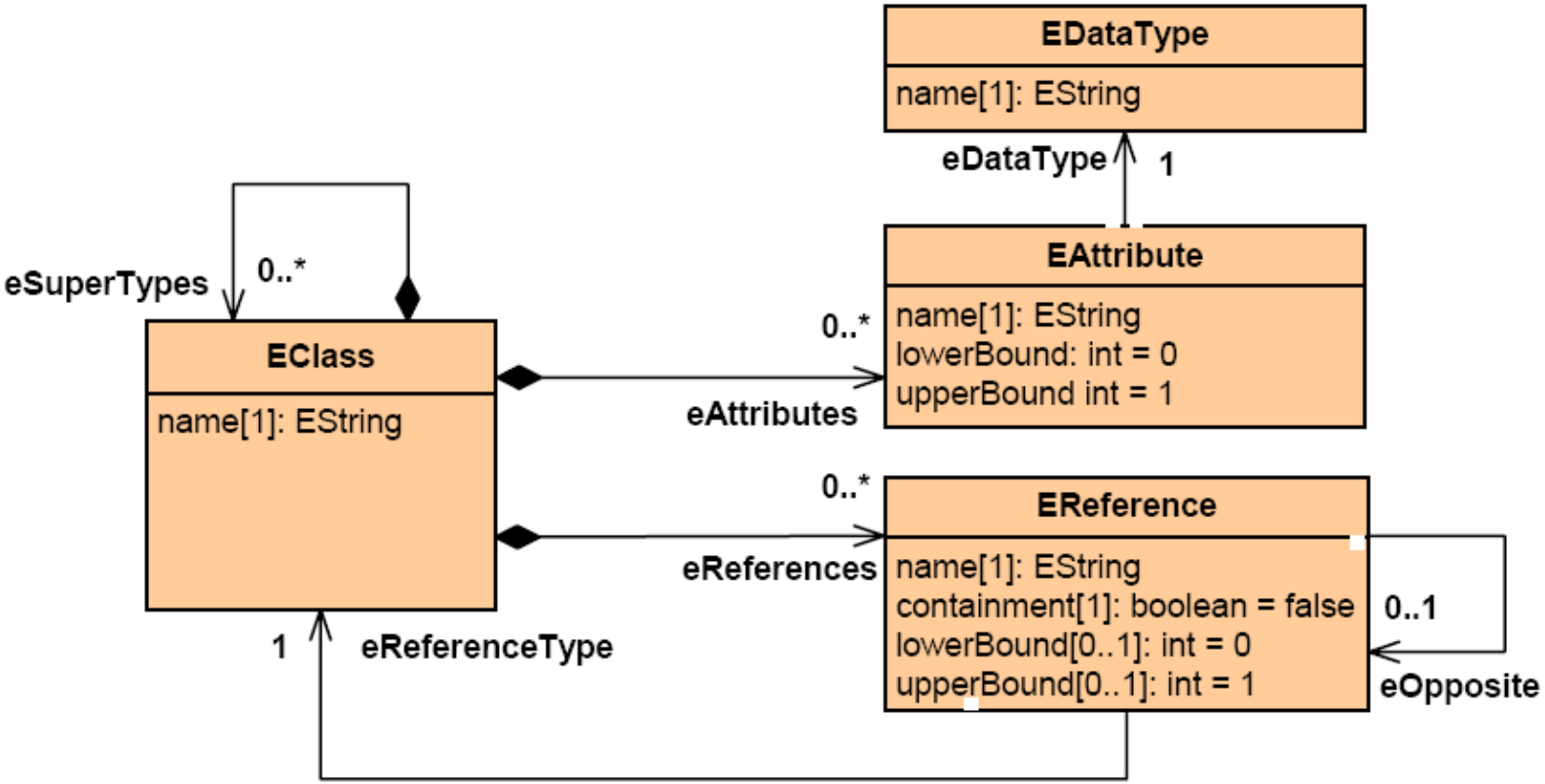
Az EMF eszközkészlet



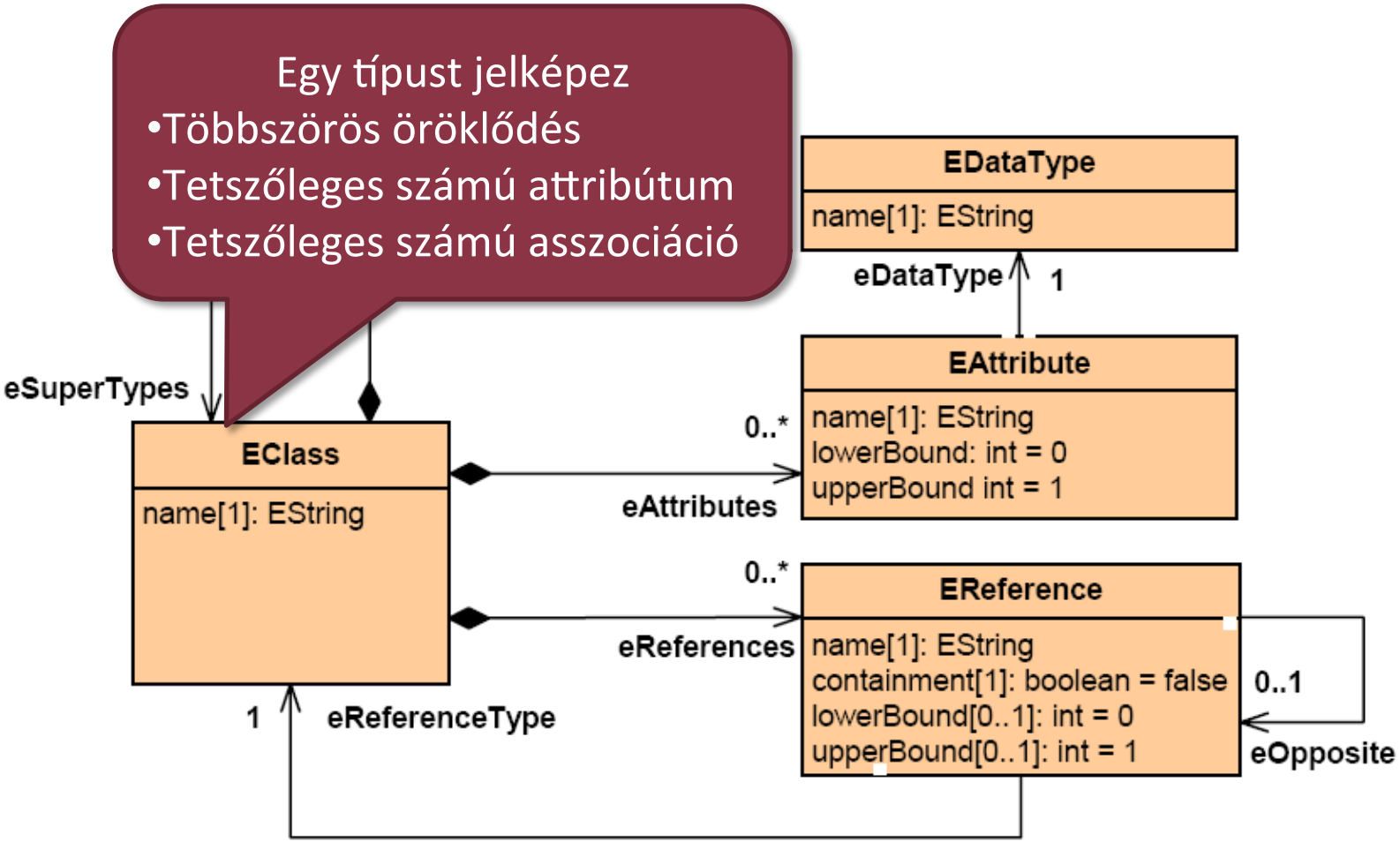
Ecore

- Az EMF metamodellező nyelve
  - Meta-nyelv: metamodellező nyelv
- A metamodellek platformfüggetlenek
  - További nyelv(ek) az implementáció-közeli modellezésre
- Strukturális modellek definíciójára szolgál

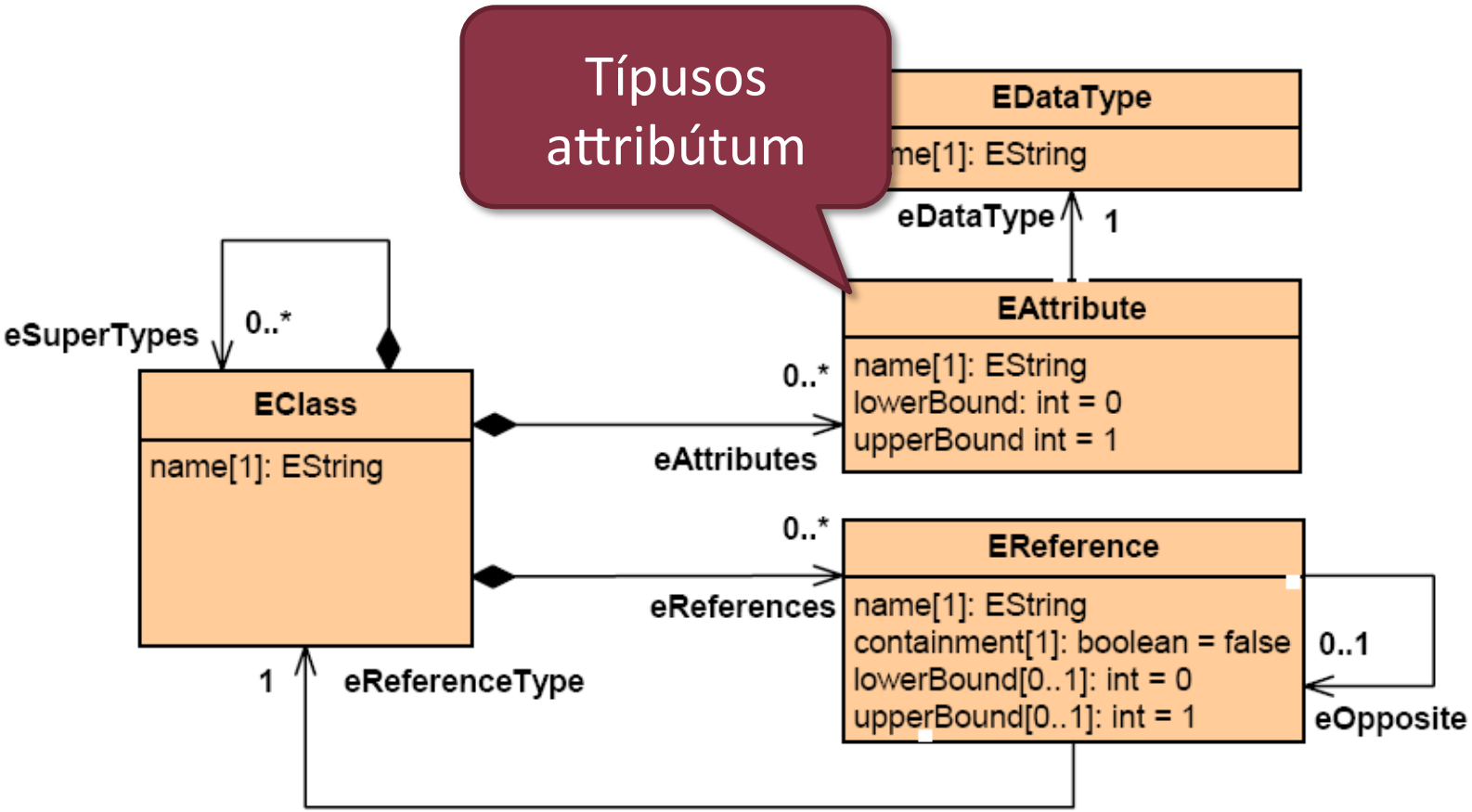
Ecore – A legfontosabb elemek



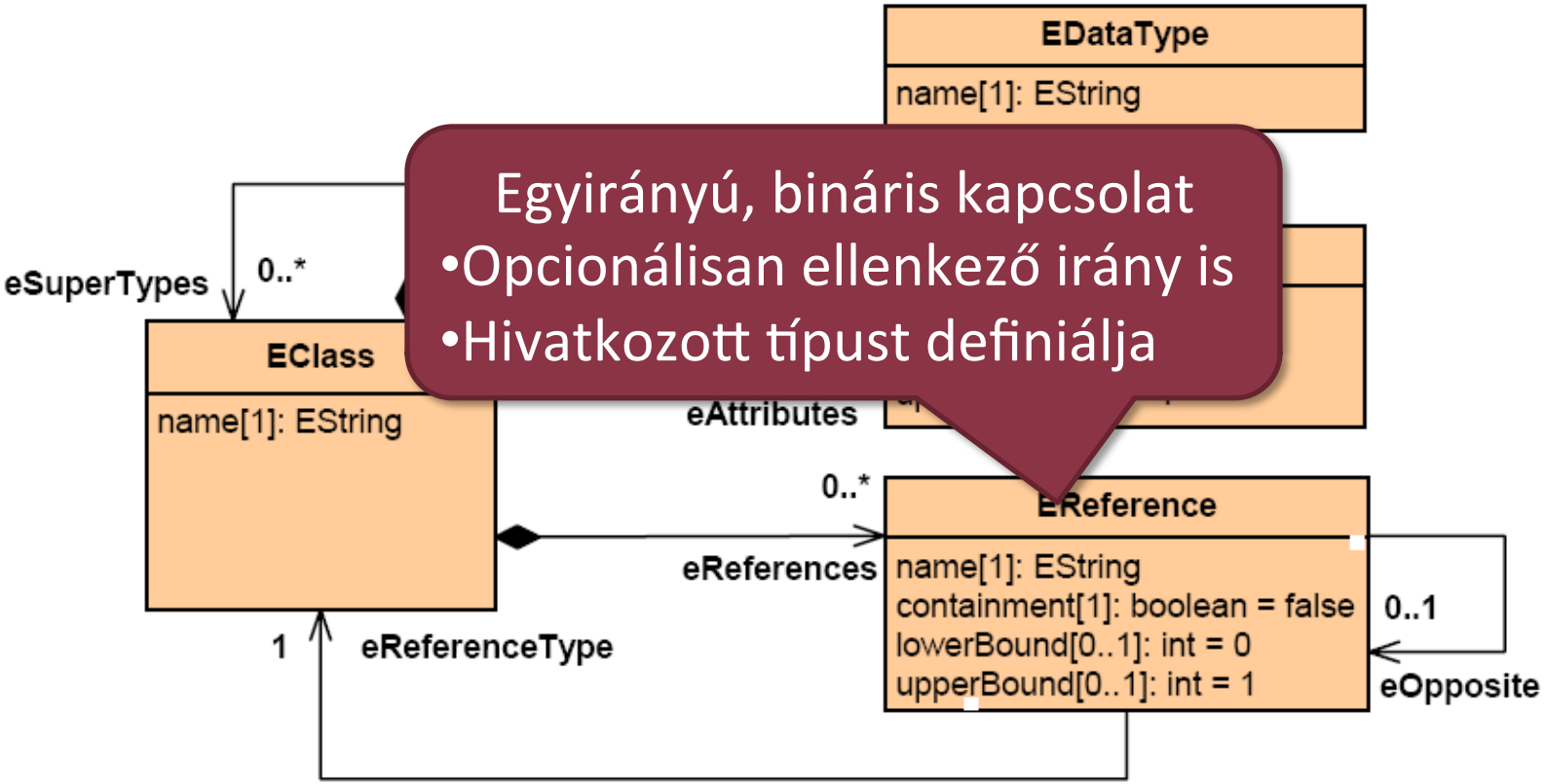
Ecore – A legfontosabb elemek



Ecore – A legfontosabb elemek



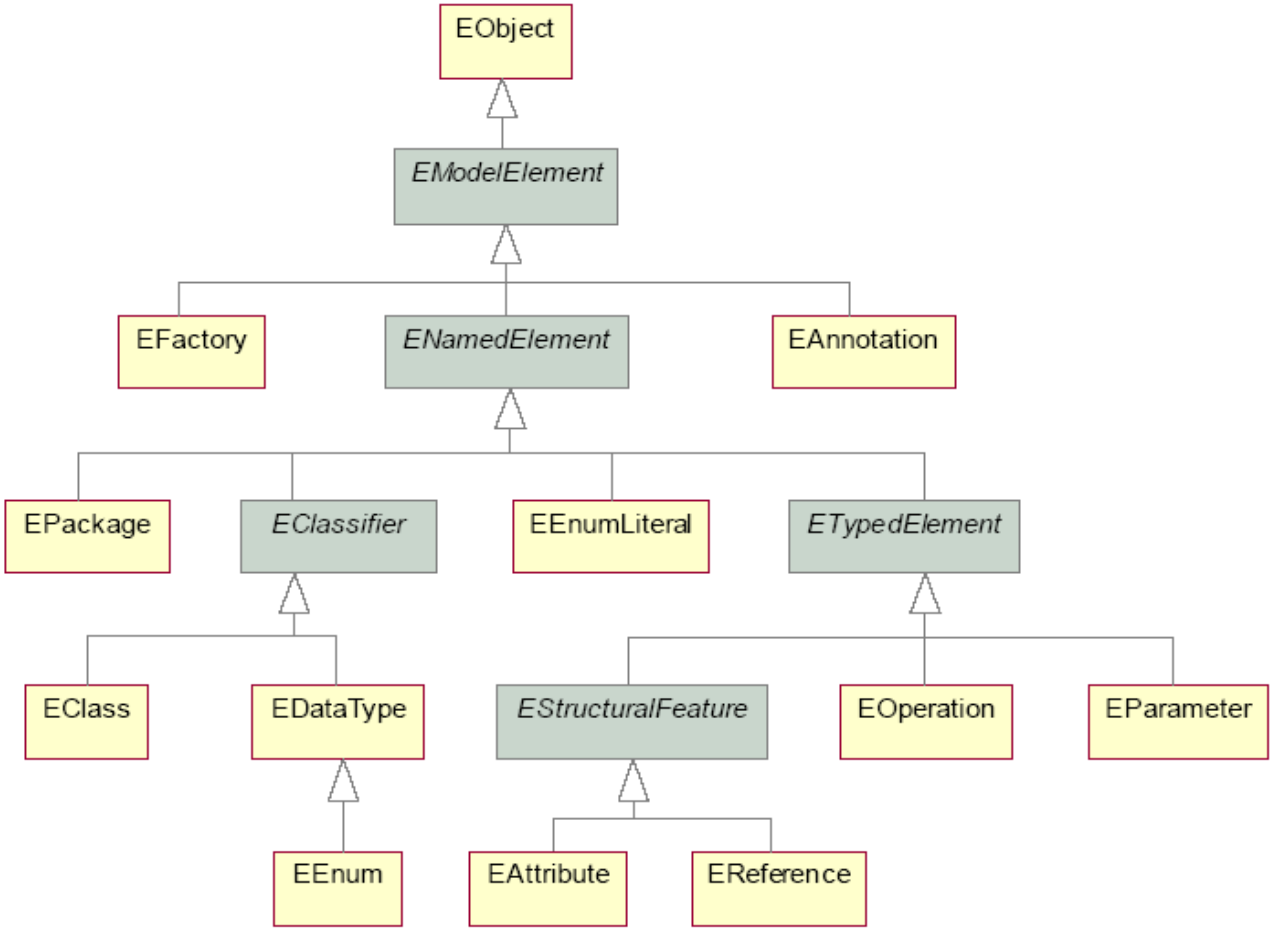
Ecore – A legfontosabb elemek



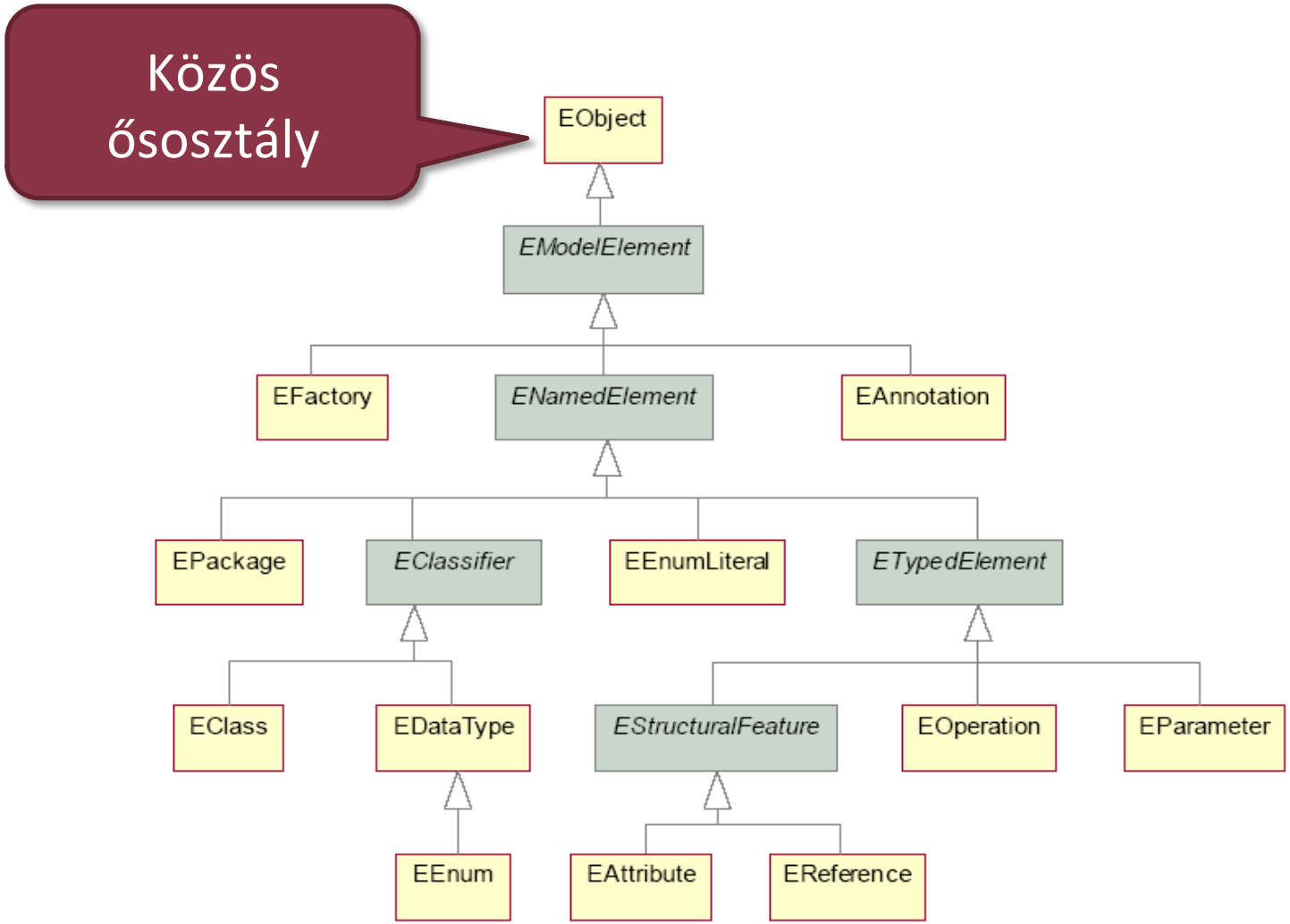
Ecore - Legfontosabb elemek

- EPackage
  - Osztályok halmaza
  - Egyszerre kezelendő
    - Fordításkor/kódgeneráláskor
    - Futásidőben regisztrálódik

Teljes Ecore hierarchia

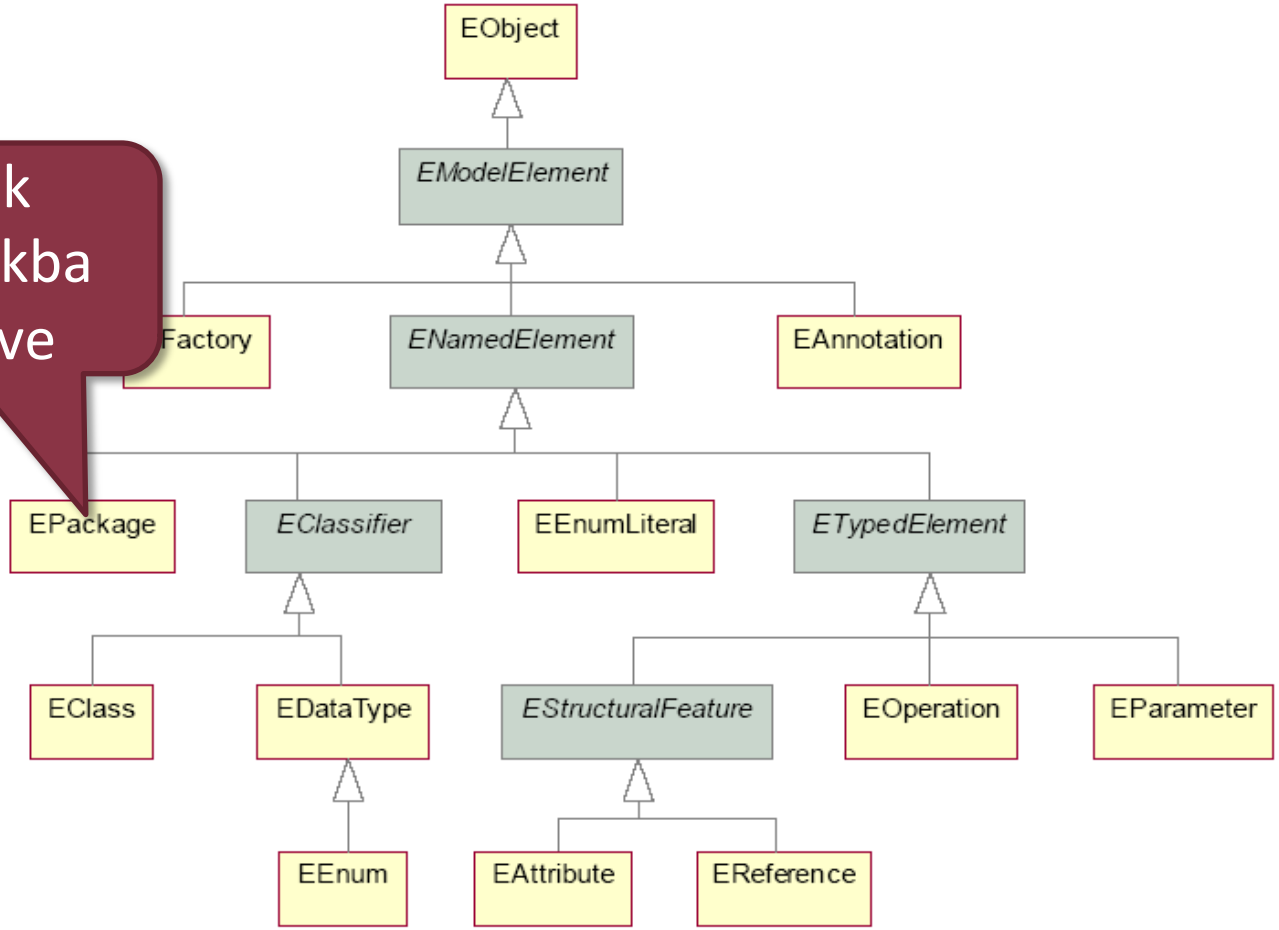


Teljes Ecore hierarchia

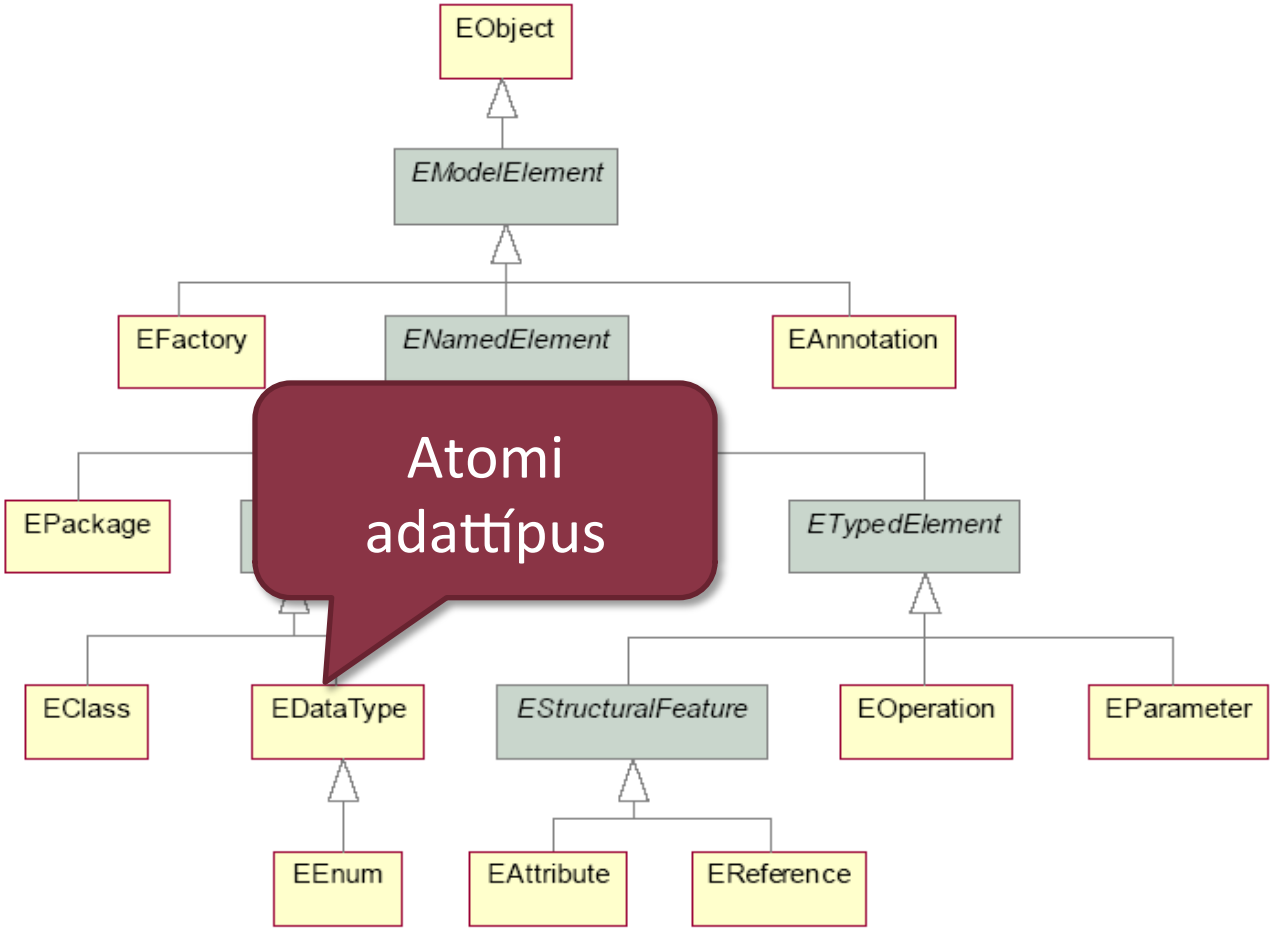


Teljes Ecore hierarchia

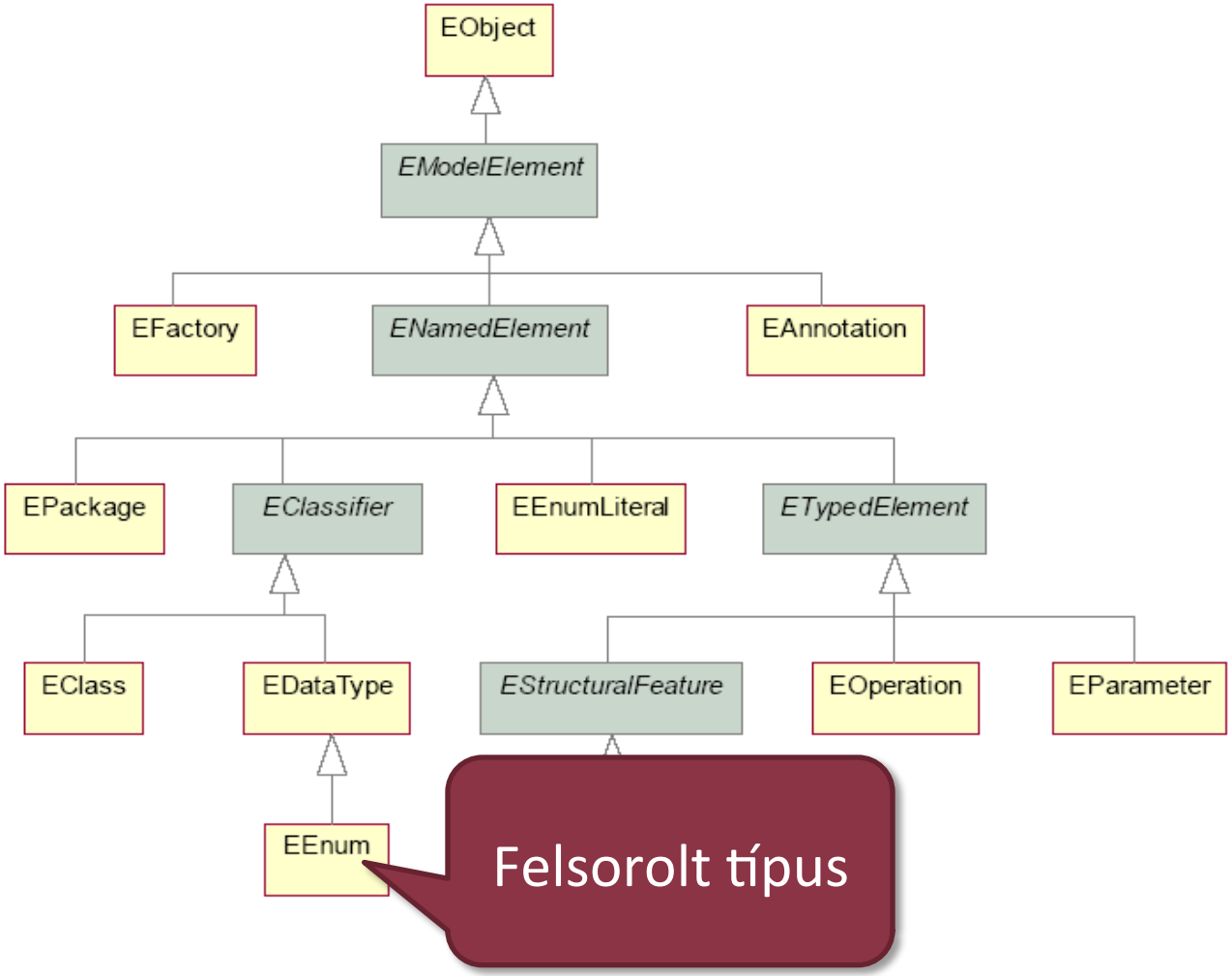
Típusok csomagokba szervezve



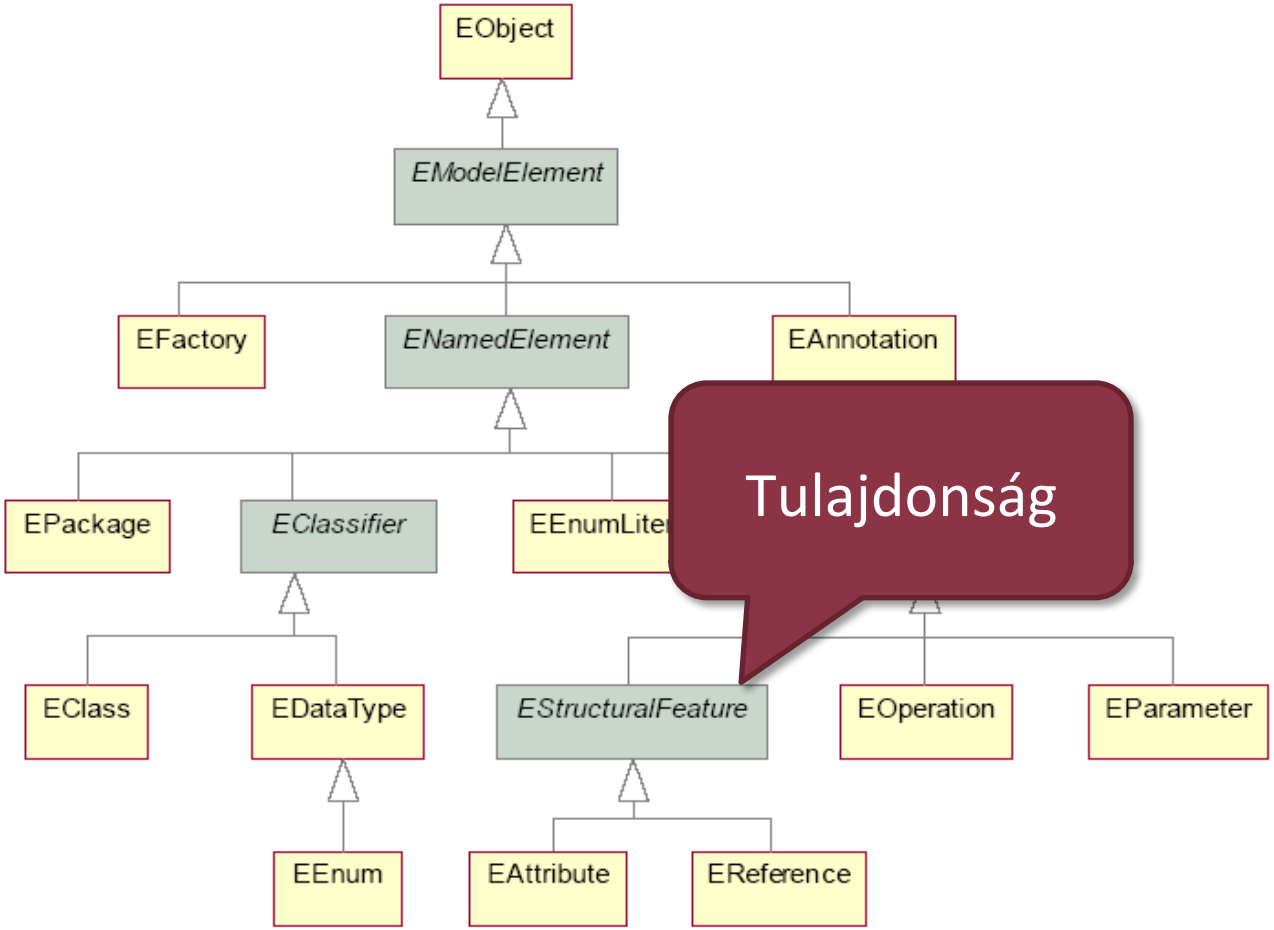
Teljes Ecore hierarchia



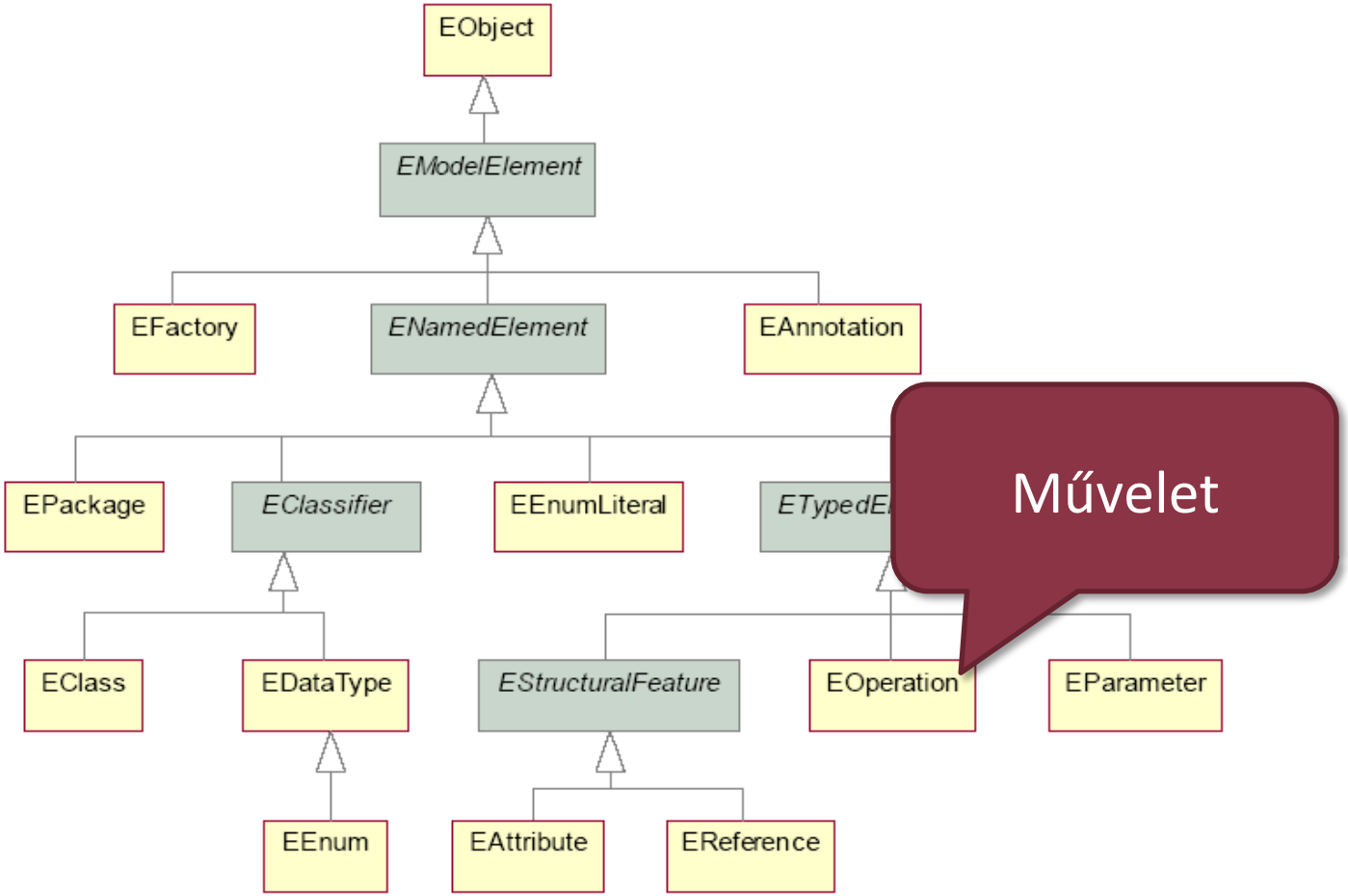
Teljes Ecore hierarchia



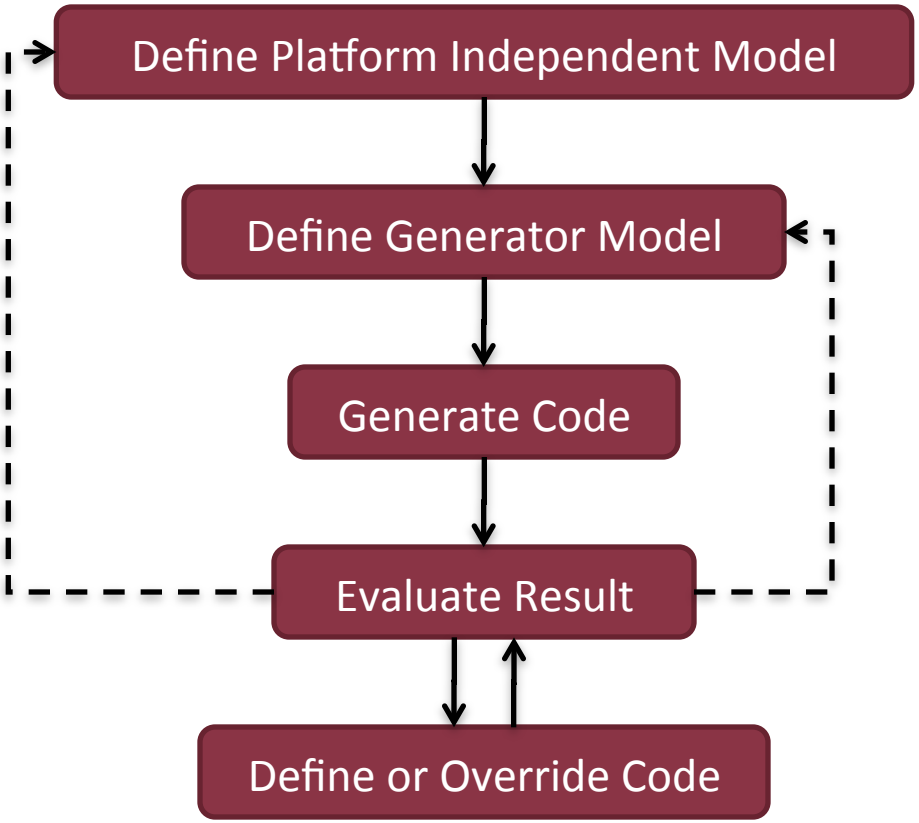
Teljes Ecore hierarchia



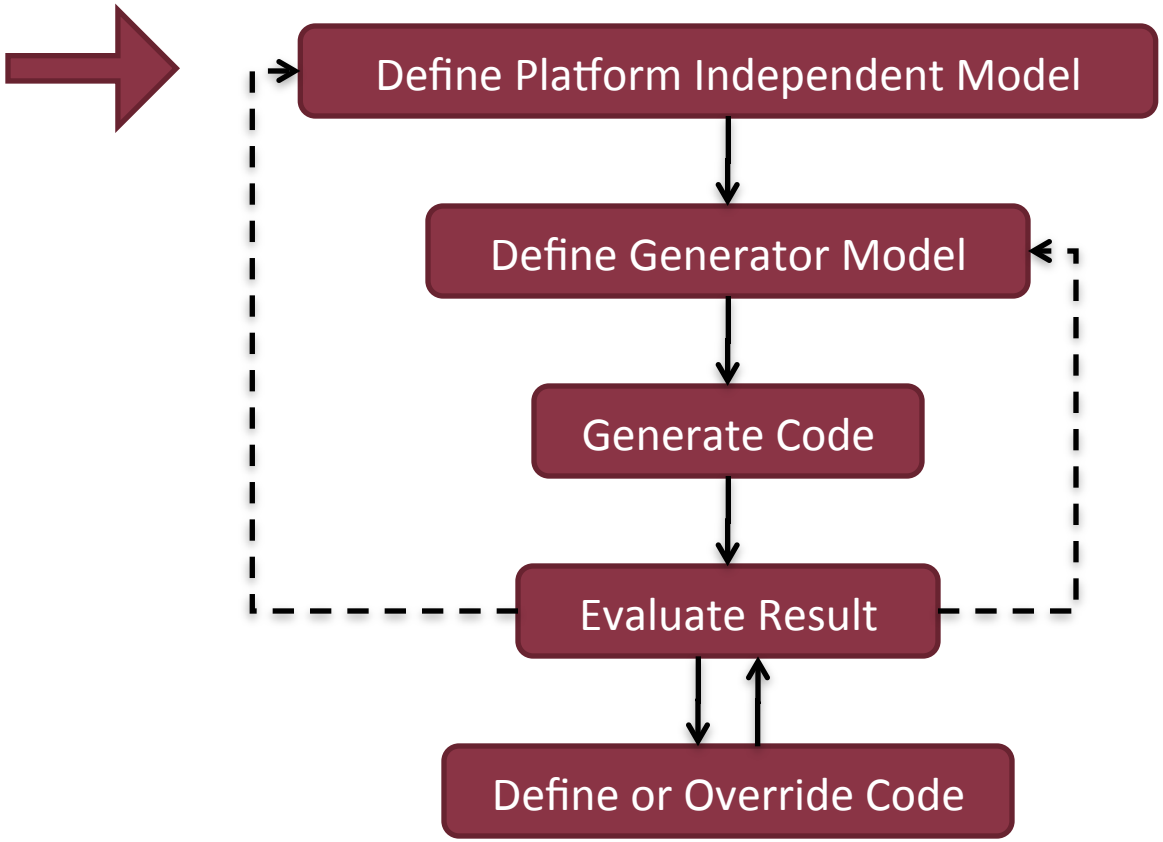
Teljes Ecore hierarchia



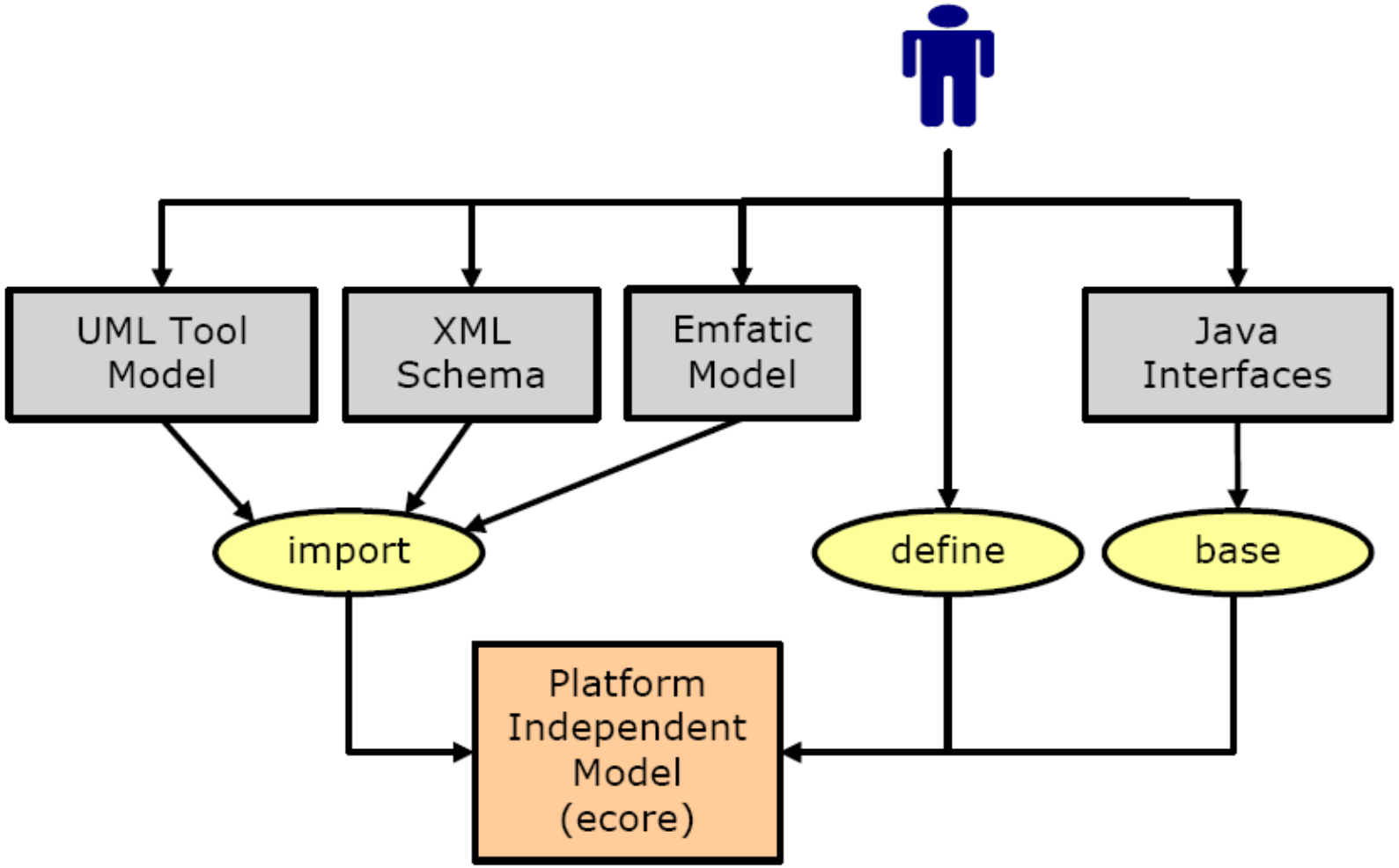
Az EMF felhasználása



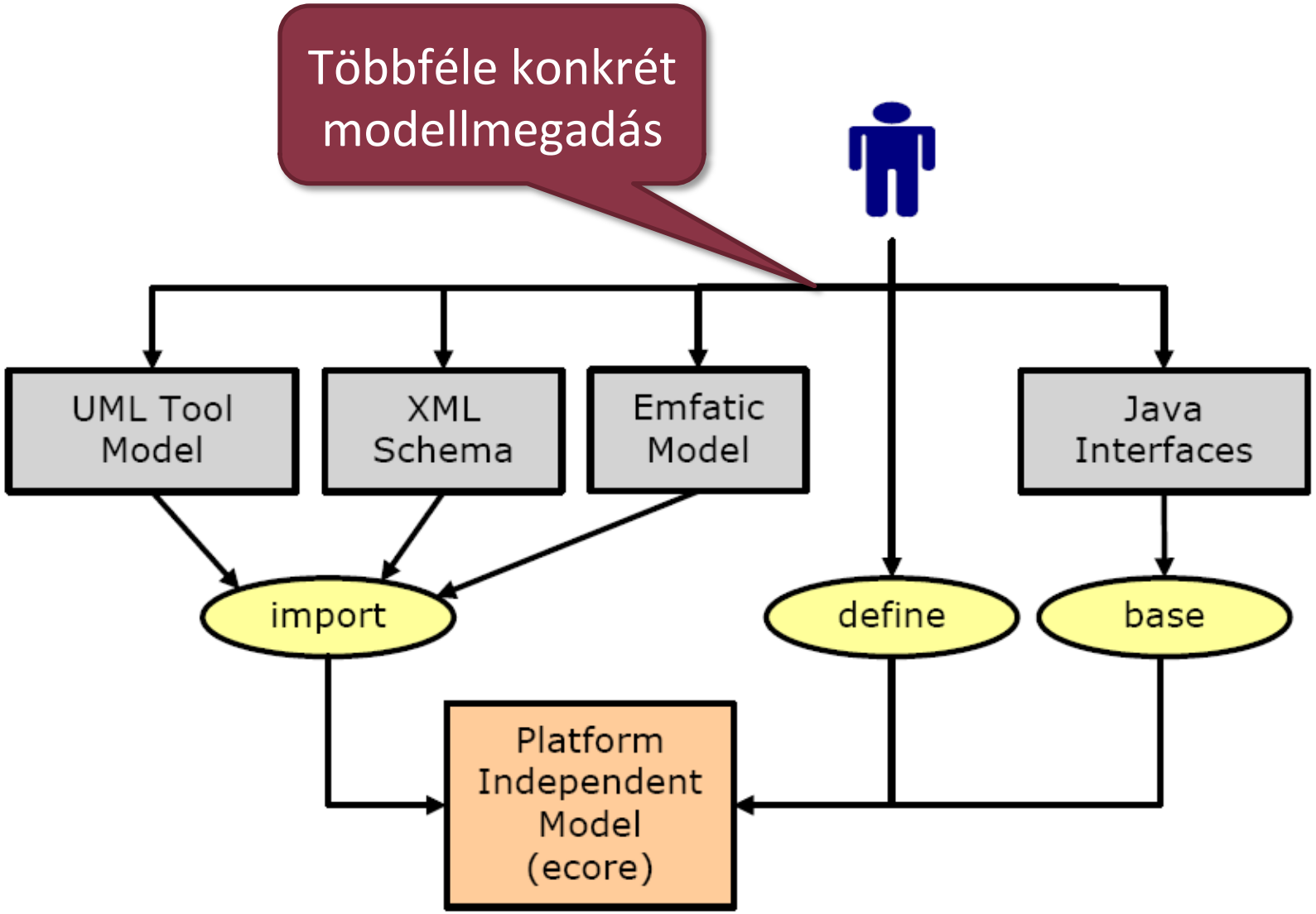
Az EMF felhasználása



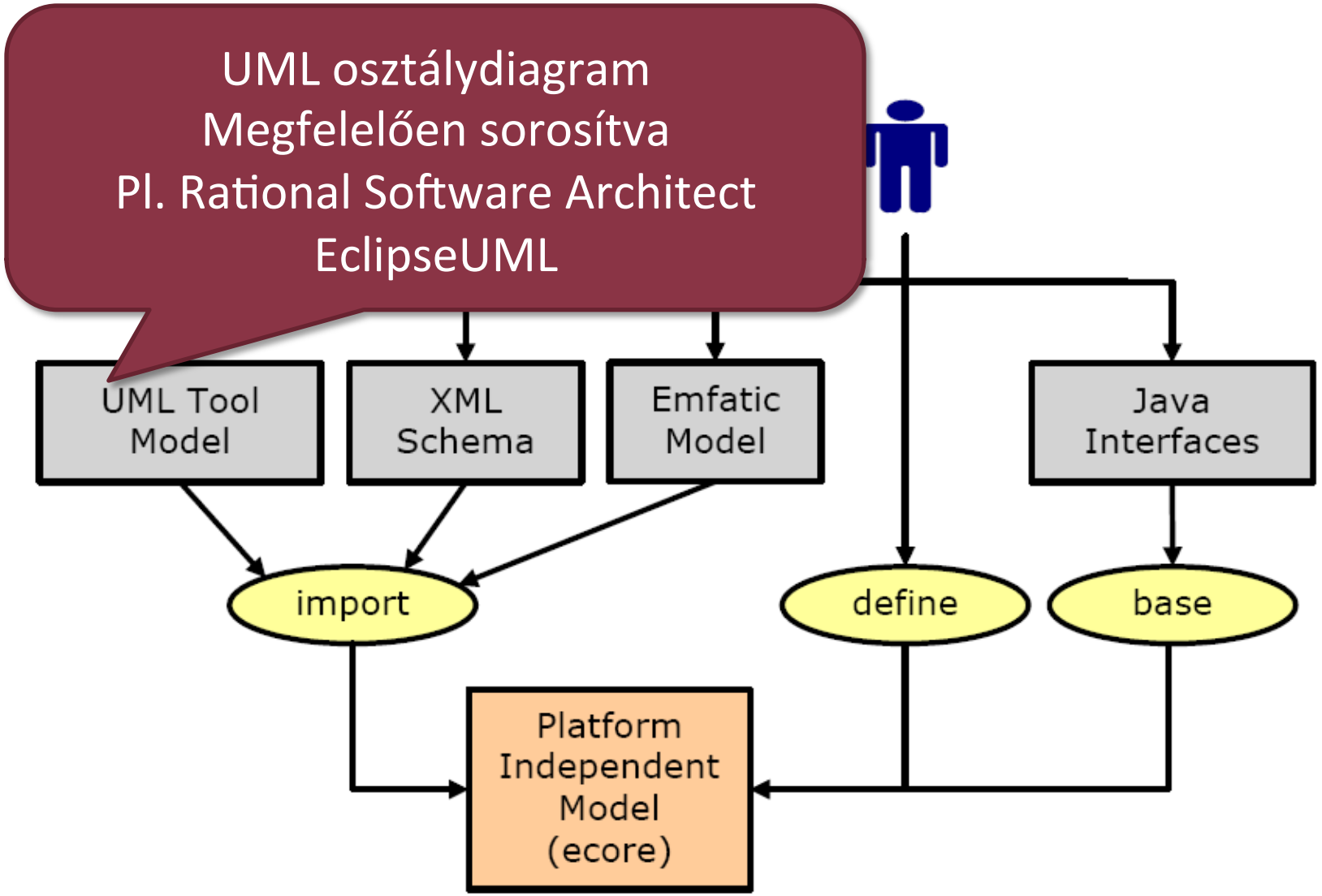
Ecore modell létrehozása



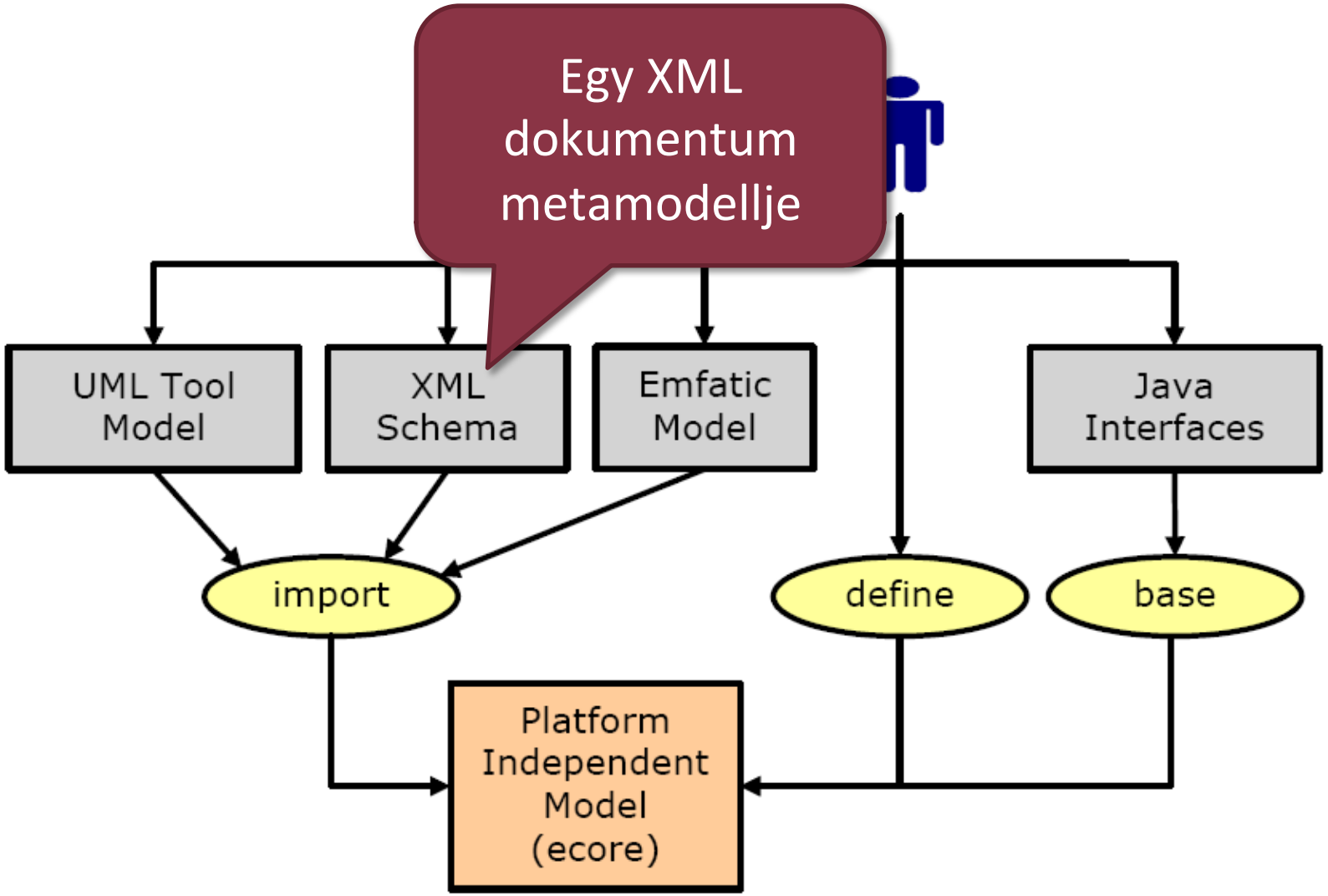
Ecore modell létrehozása



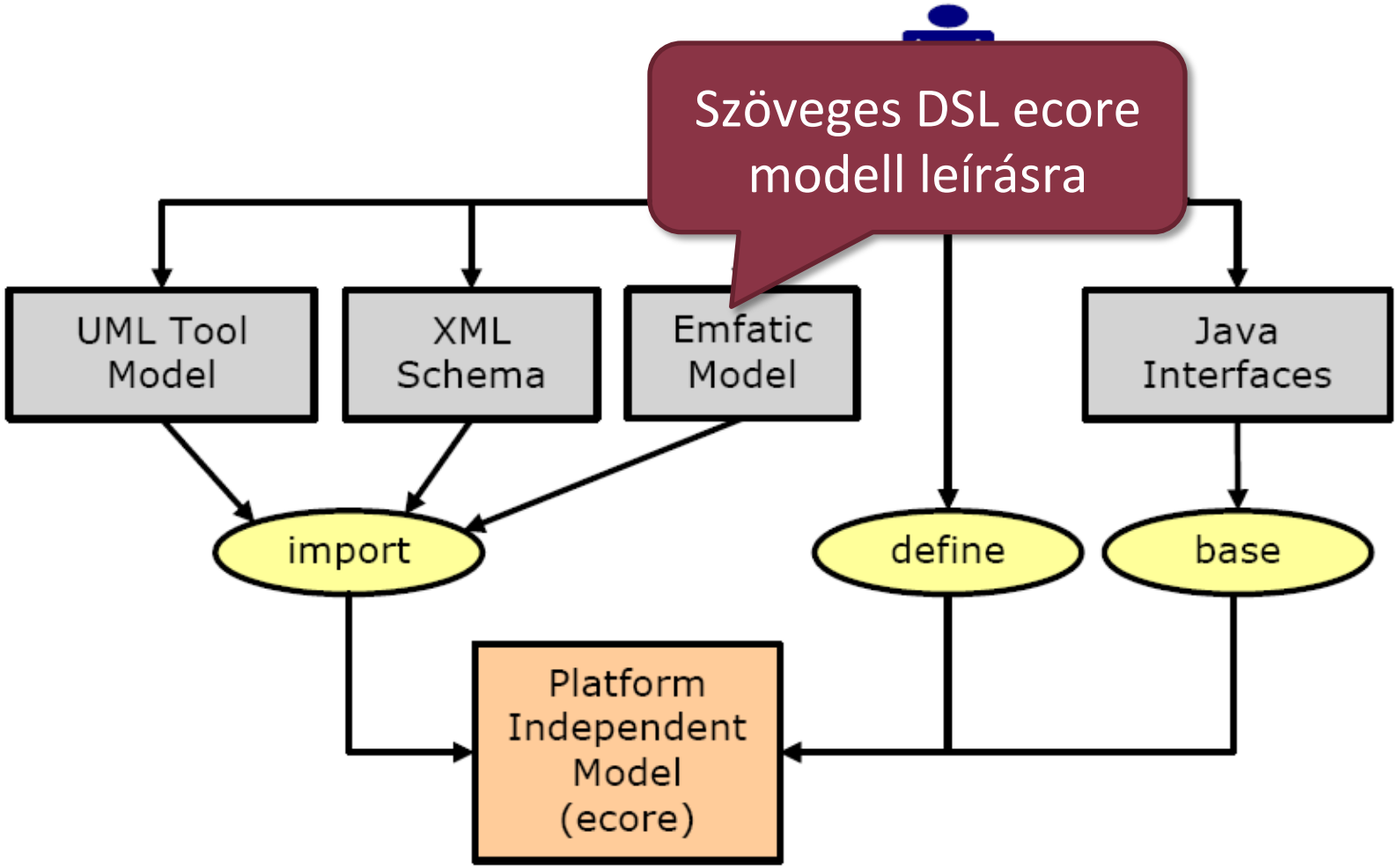
Ecore modell létrehozása



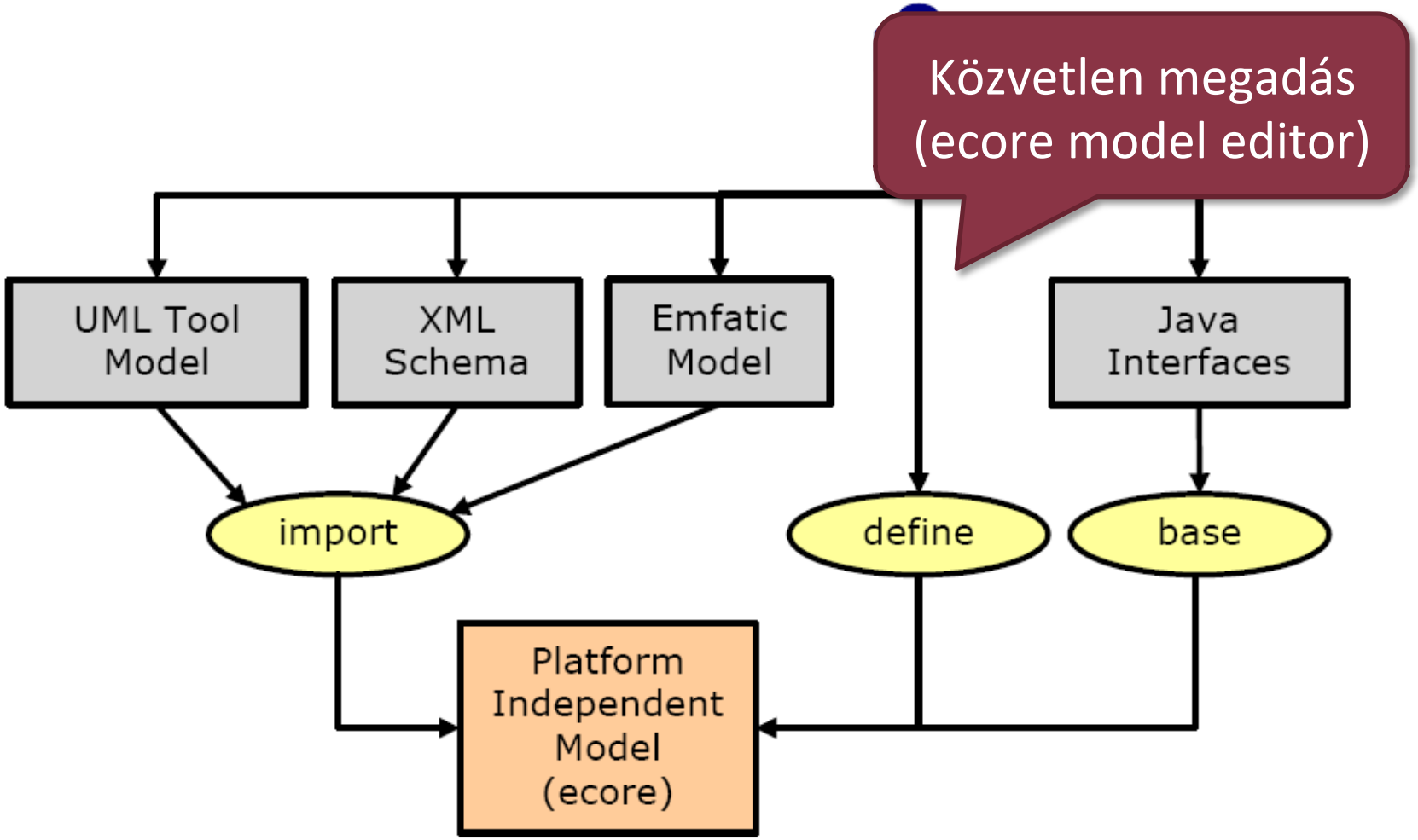
Ecore modell létrehozása



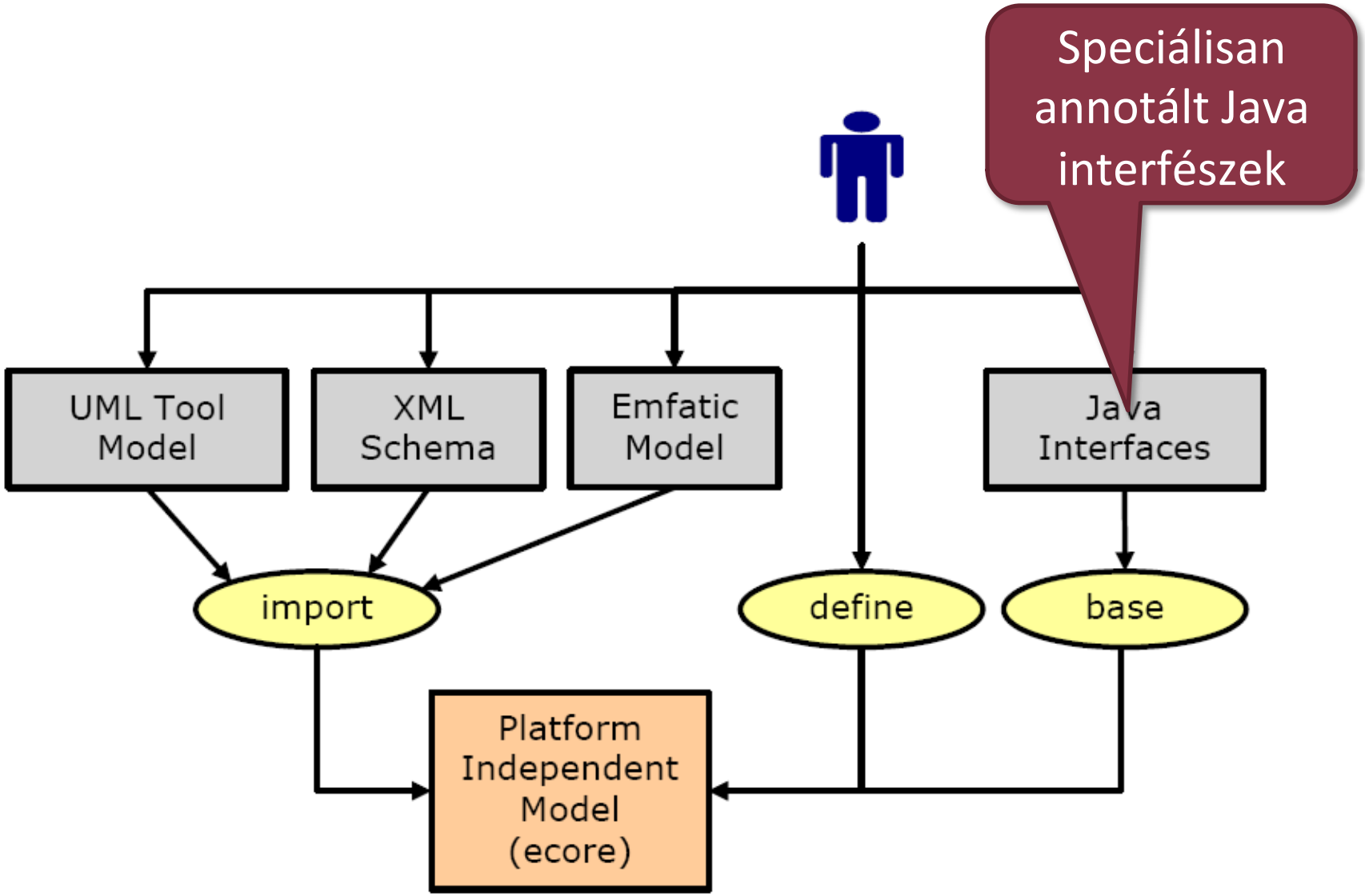
Ecore modell létrehozása



Ecore modell létrehozása

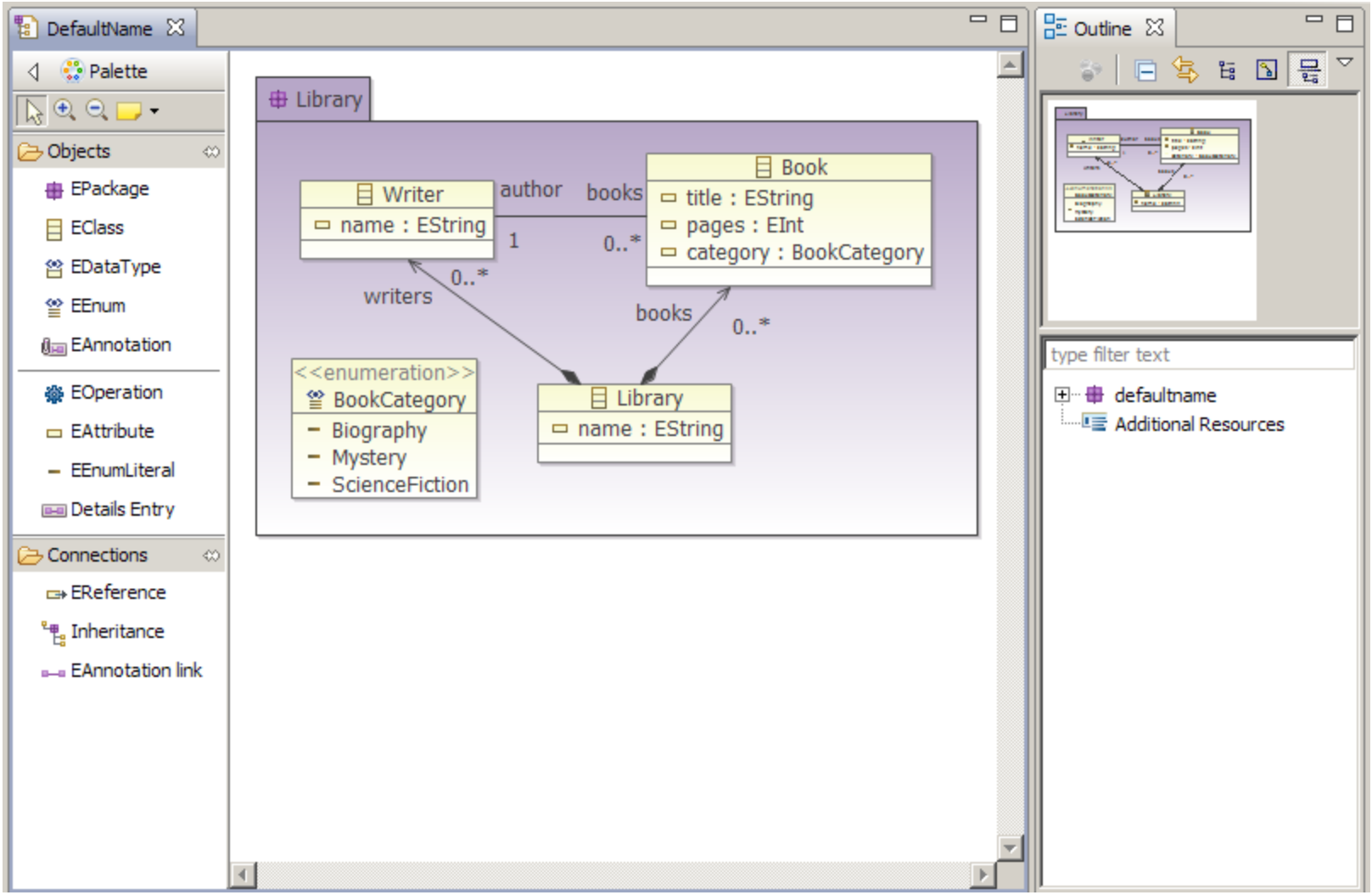


Ecore modell létrehozása

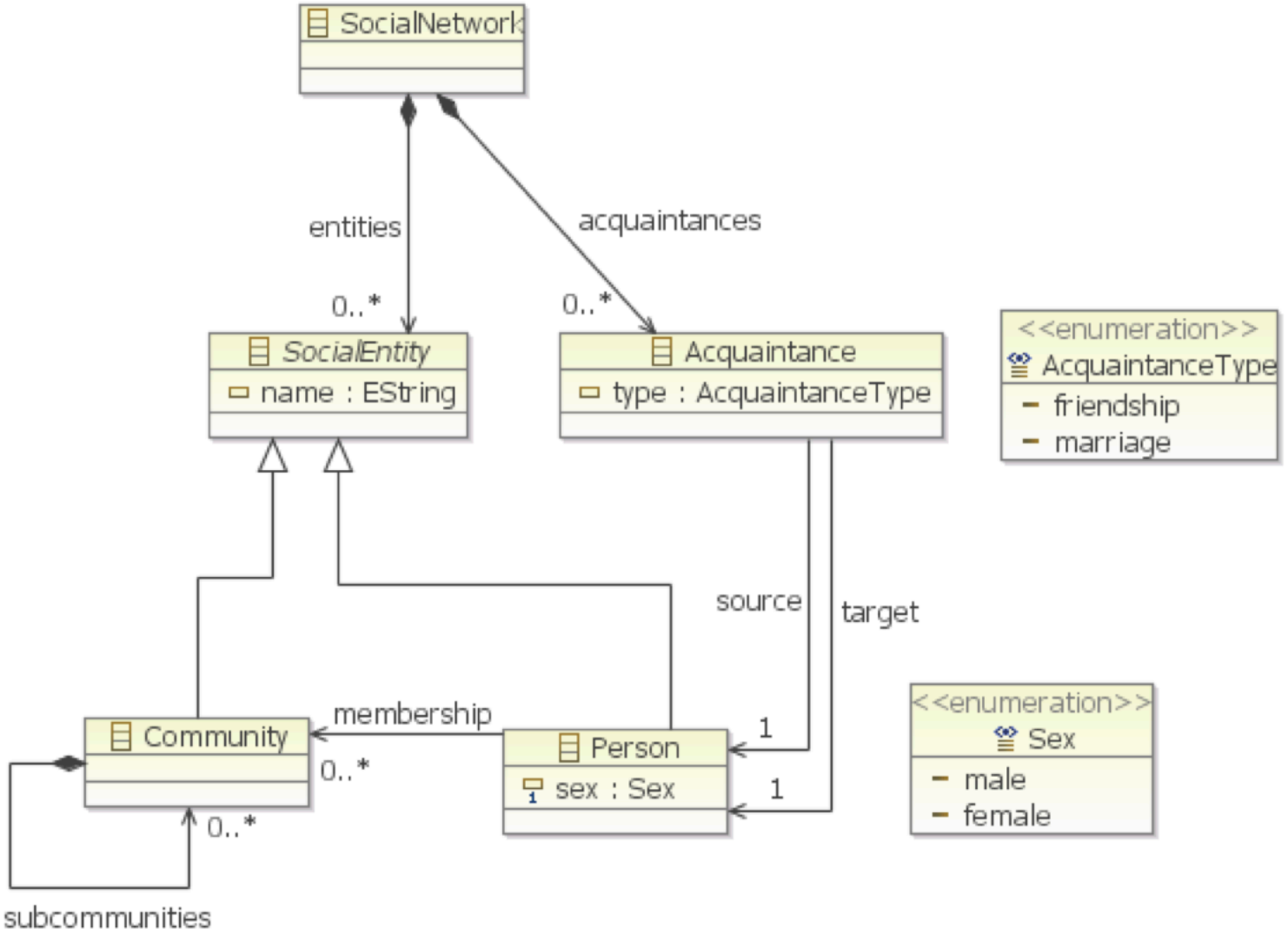


Ecore Tools: Ecore Diagram Editor

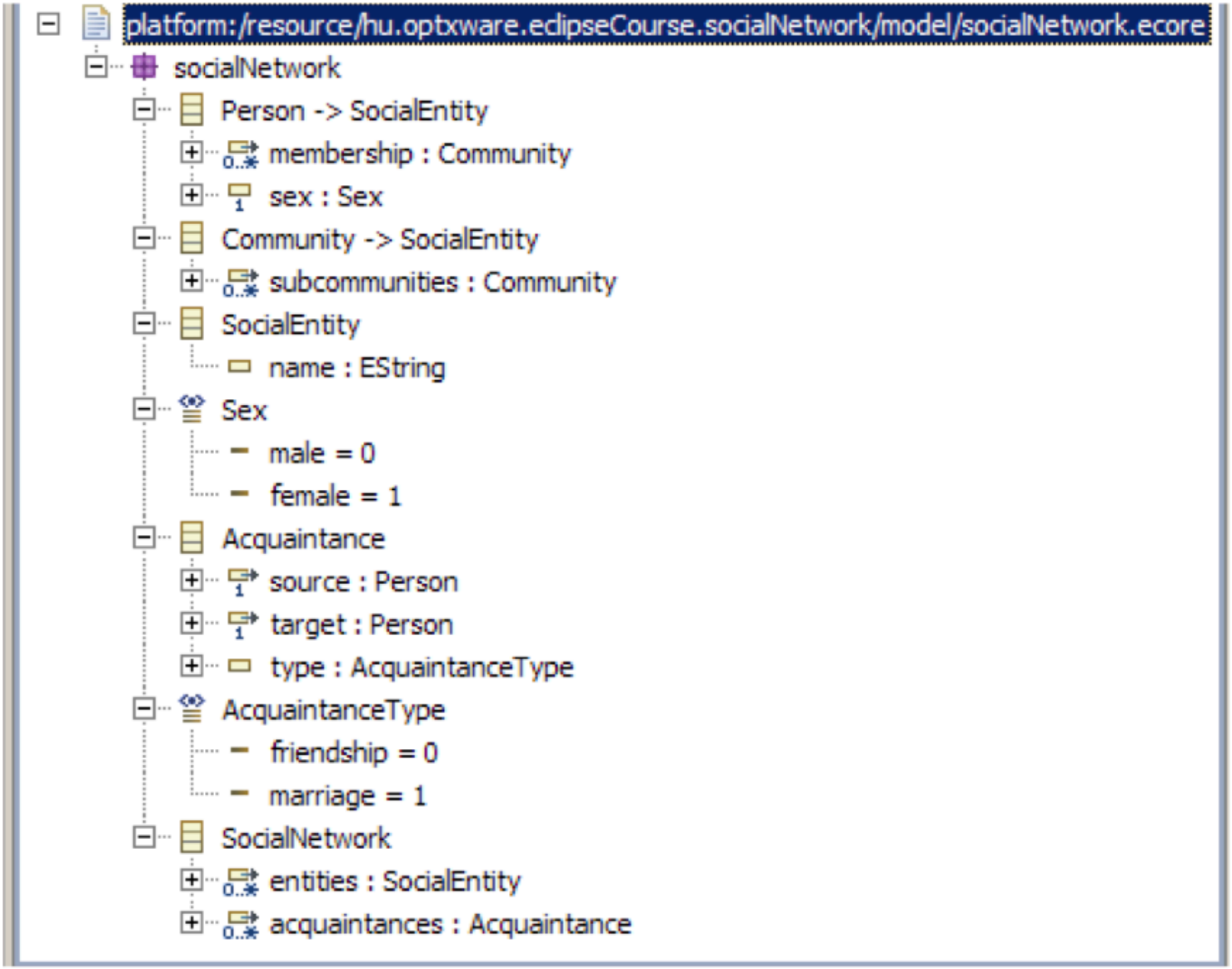
- Grafikus DSL EMF metamodellek számára



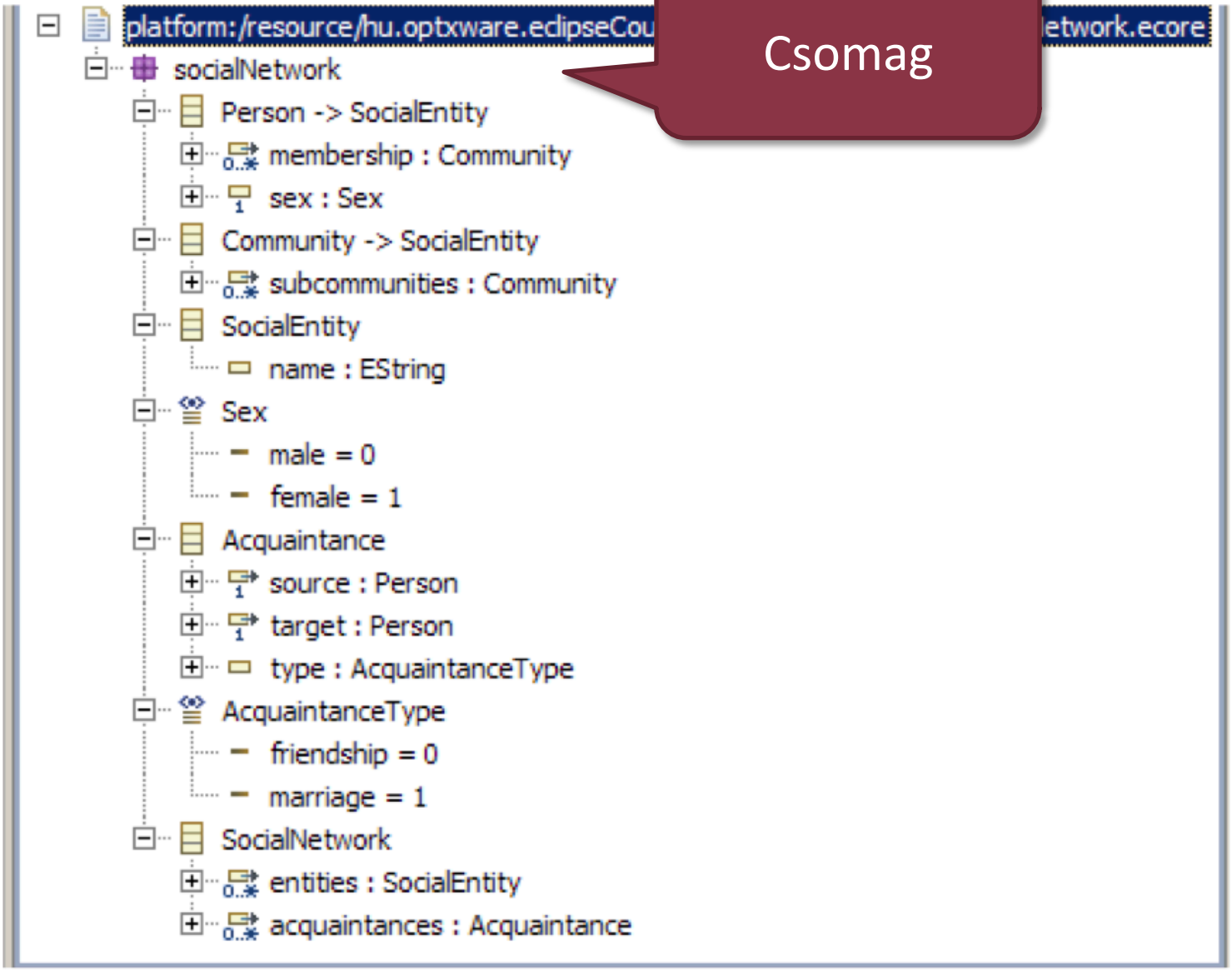
Példa: Közösségi háló



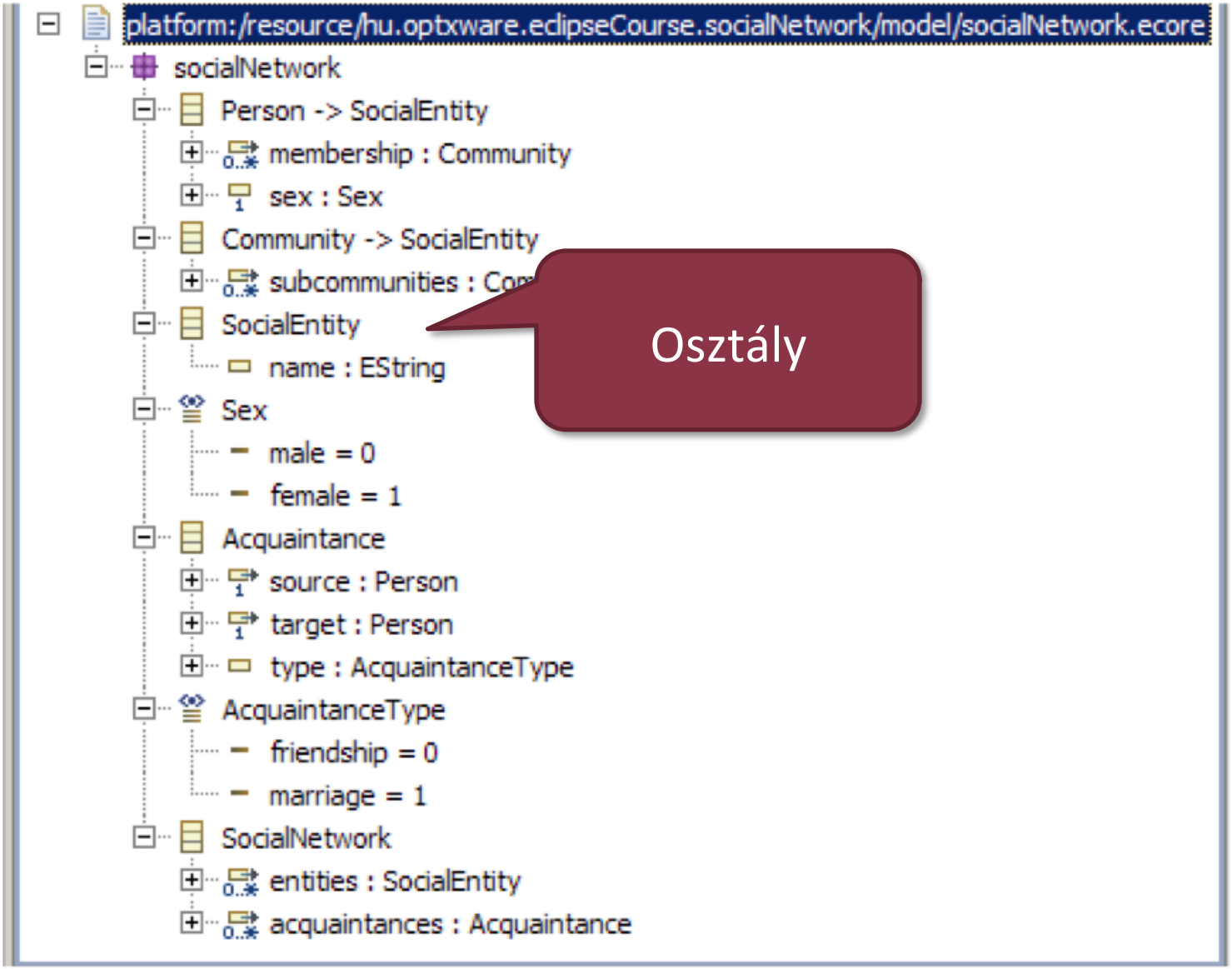
Példa: Közösségi háló metamodell



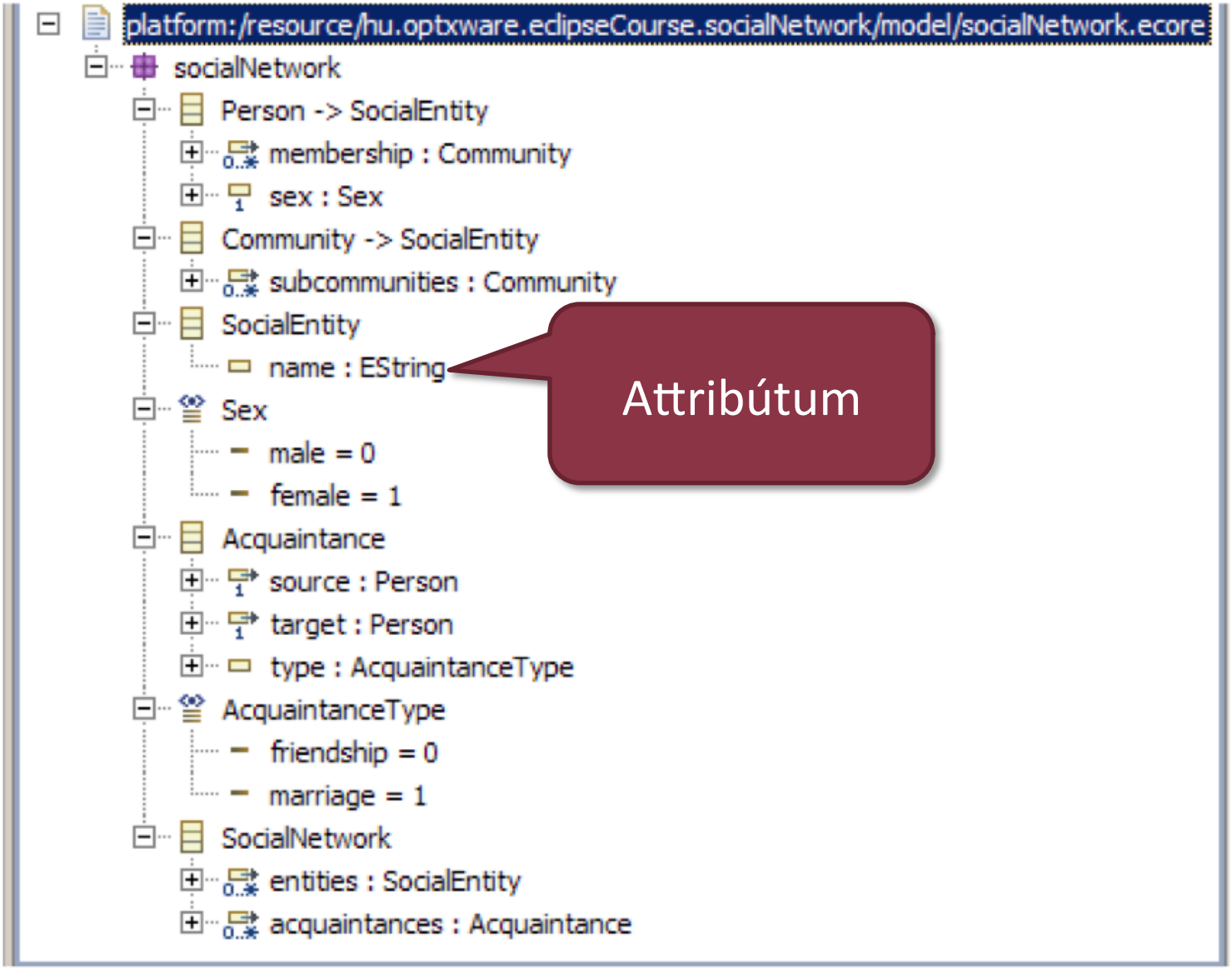
Példa: Közösségi háló metamodell



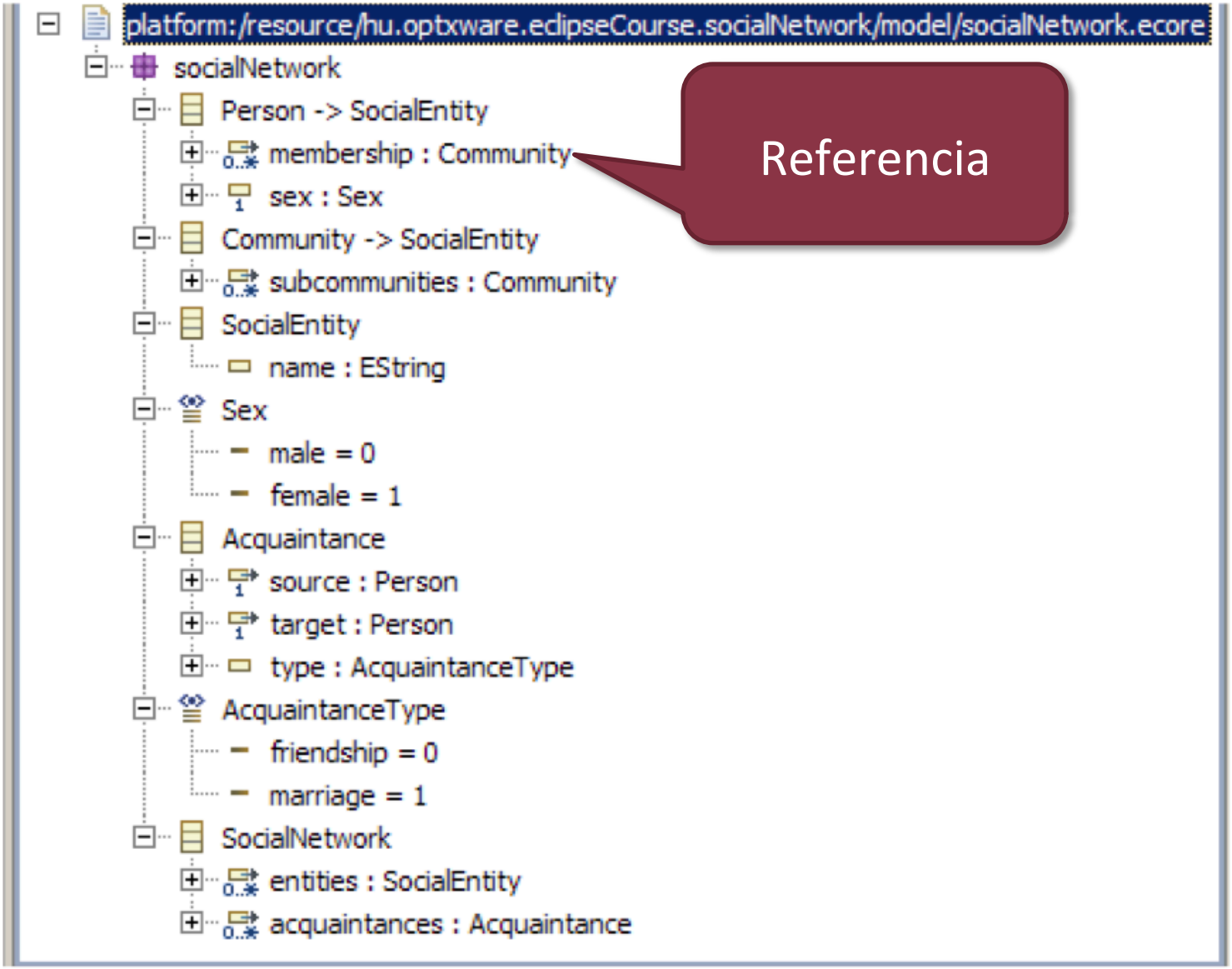
Példa: Közösségi háló metamodell



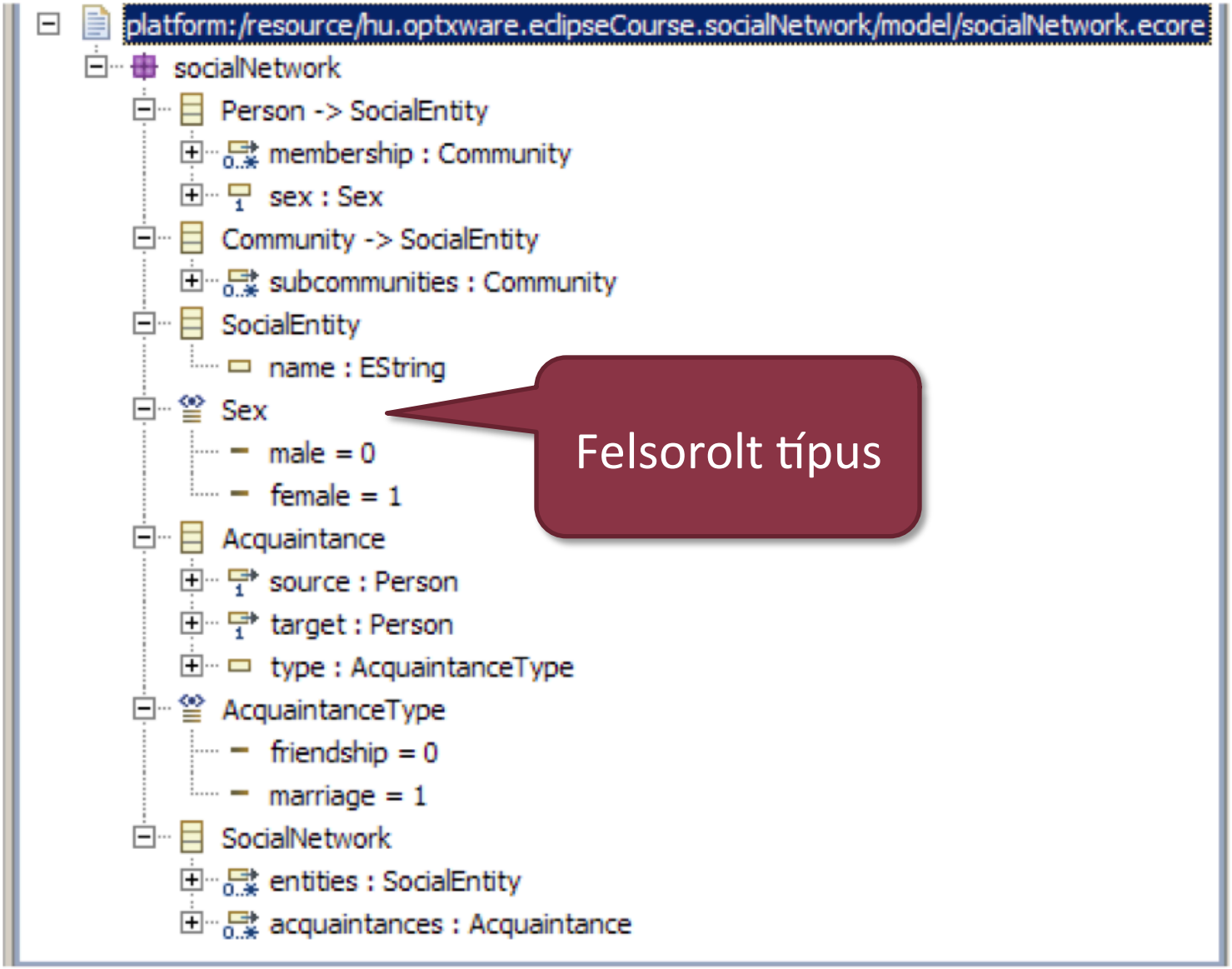
Példa: Közösségi háló metamodell



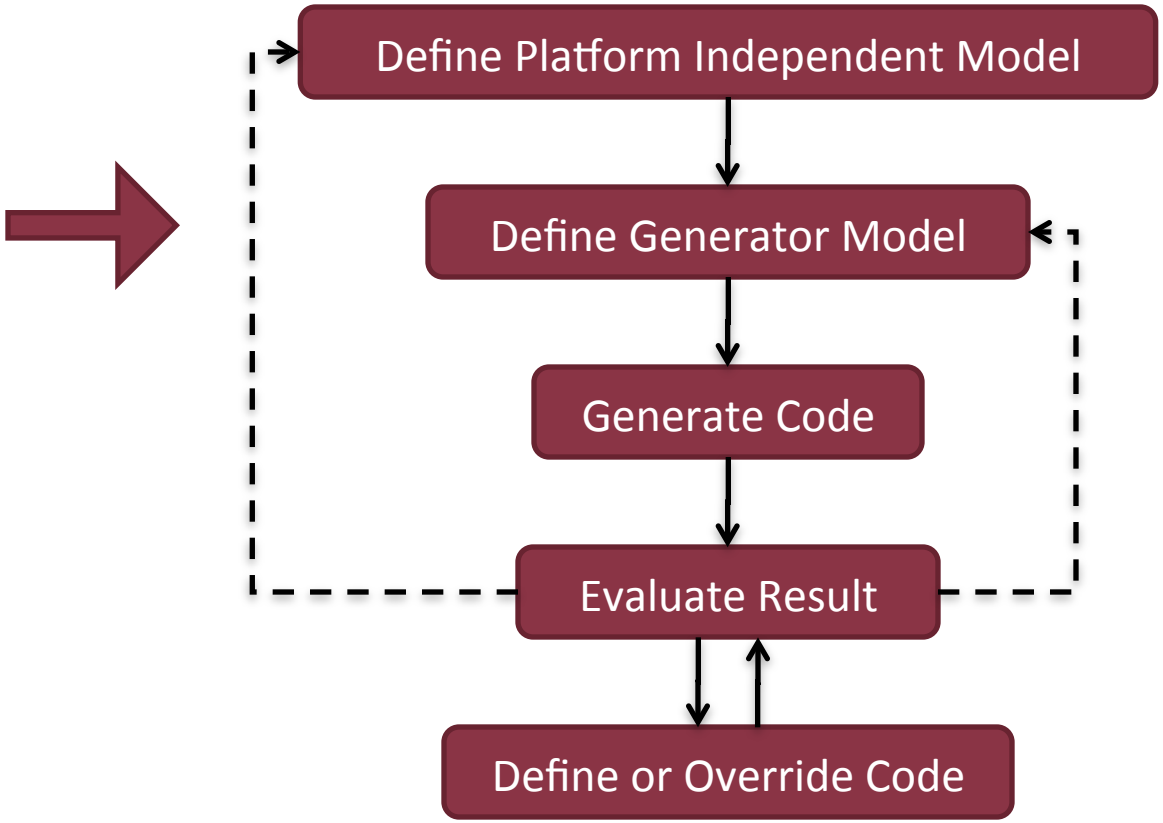
Példa: Közösségi háló metamodell



Példa: Közösségi háló metamodell



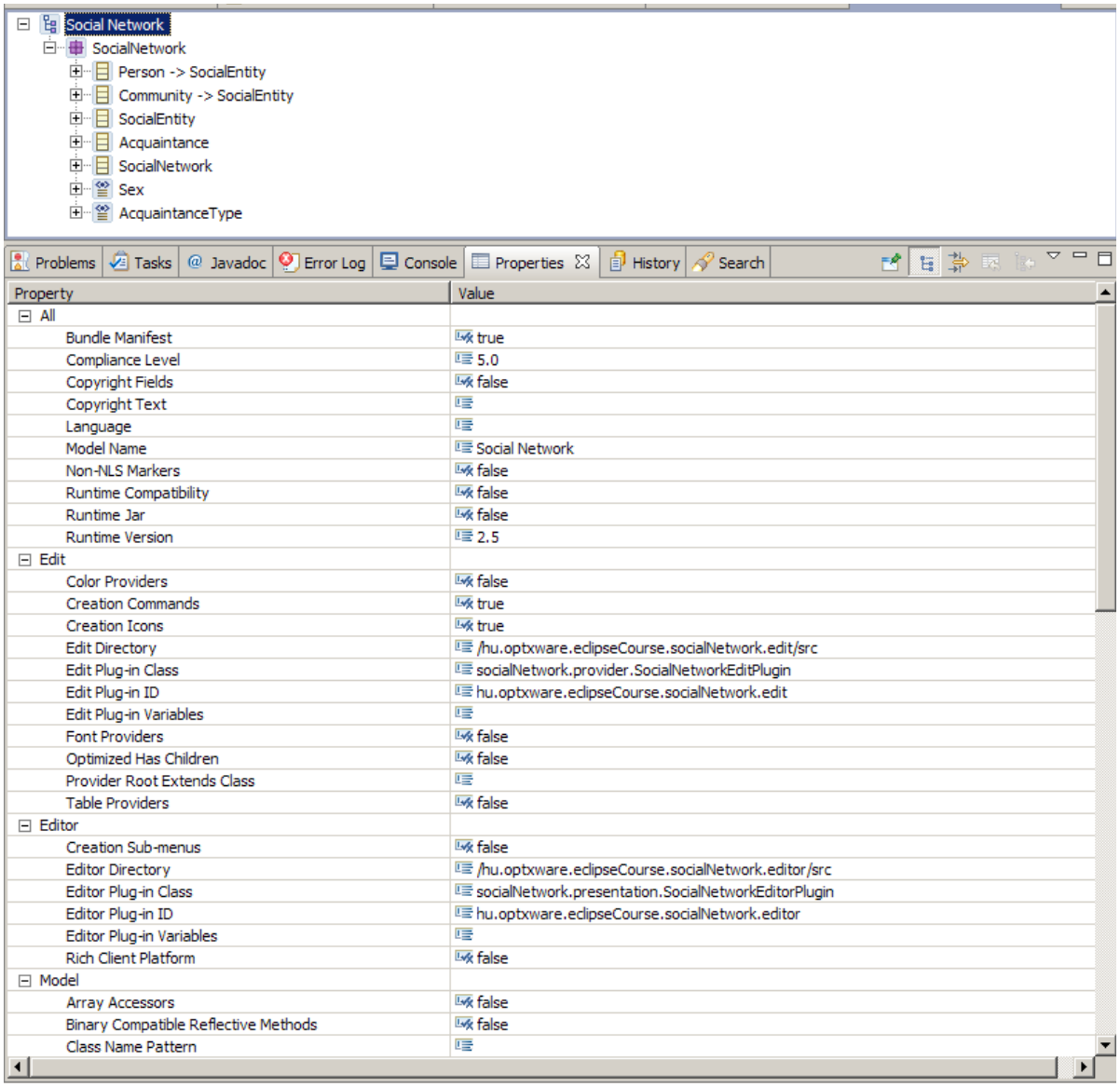
Az EMF felhasználása



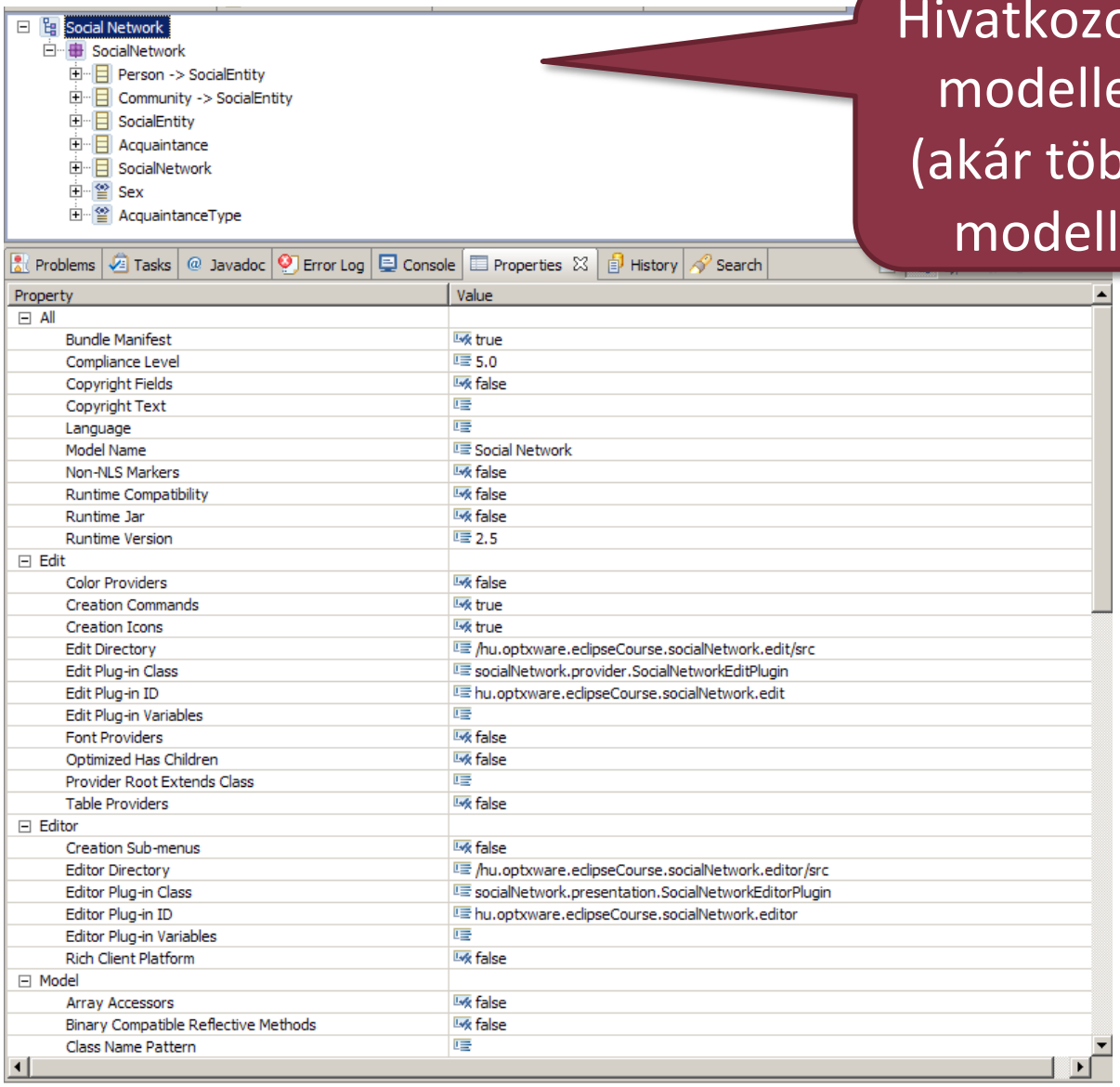
Generátor modell

- Cél:
  - Kódgenerálási beállítások megadása
- Szintén EMF modell
  - Hivatkozik a saját Ecore modellünkre
  - Más metamodell!
- Kódgenerálási beállítások
  - Java verzió (pl. Java 5 esetén enumok használata)
  - Package/projektnevek
  - …

Generátor modell

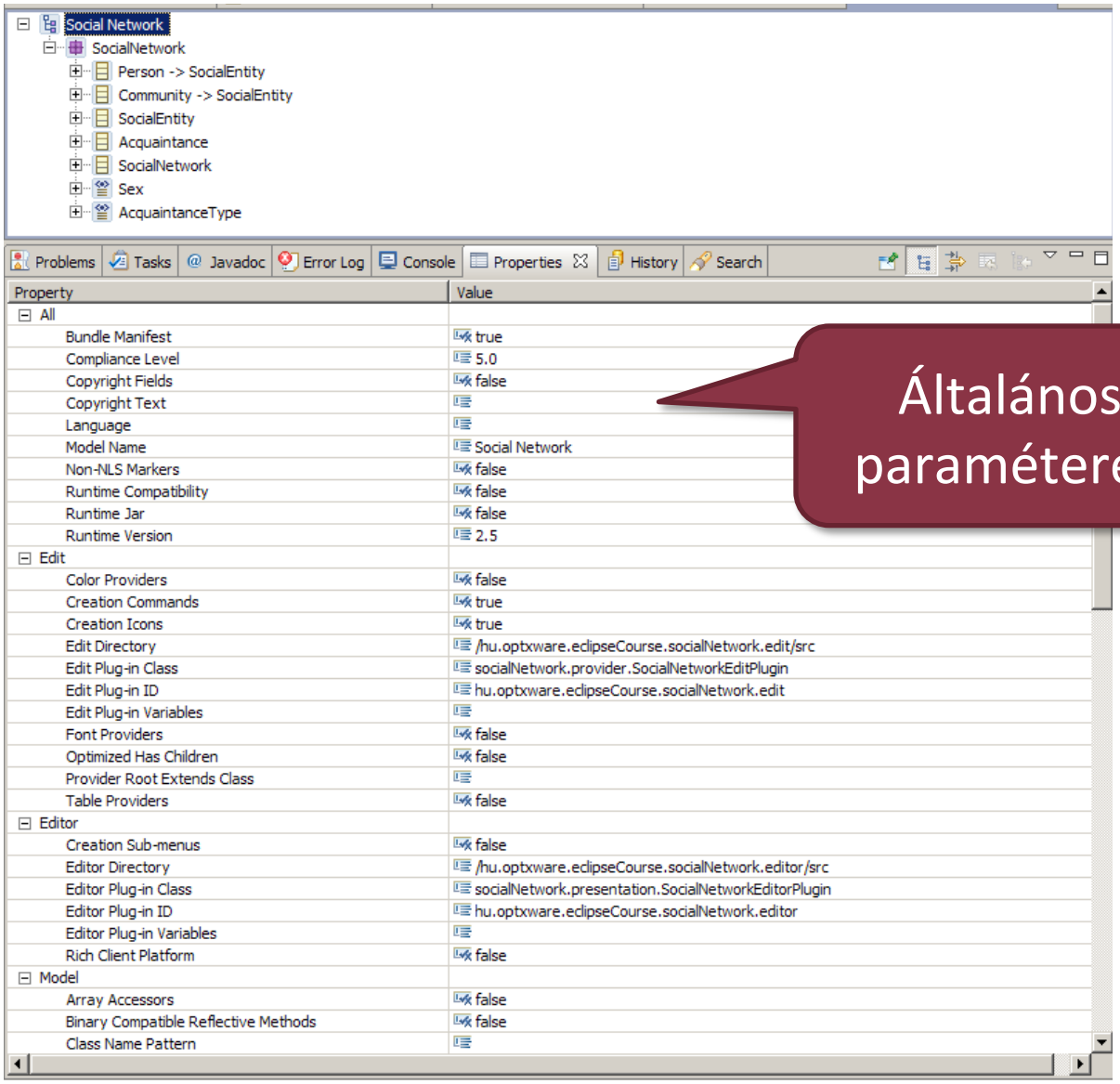


Generátor modell



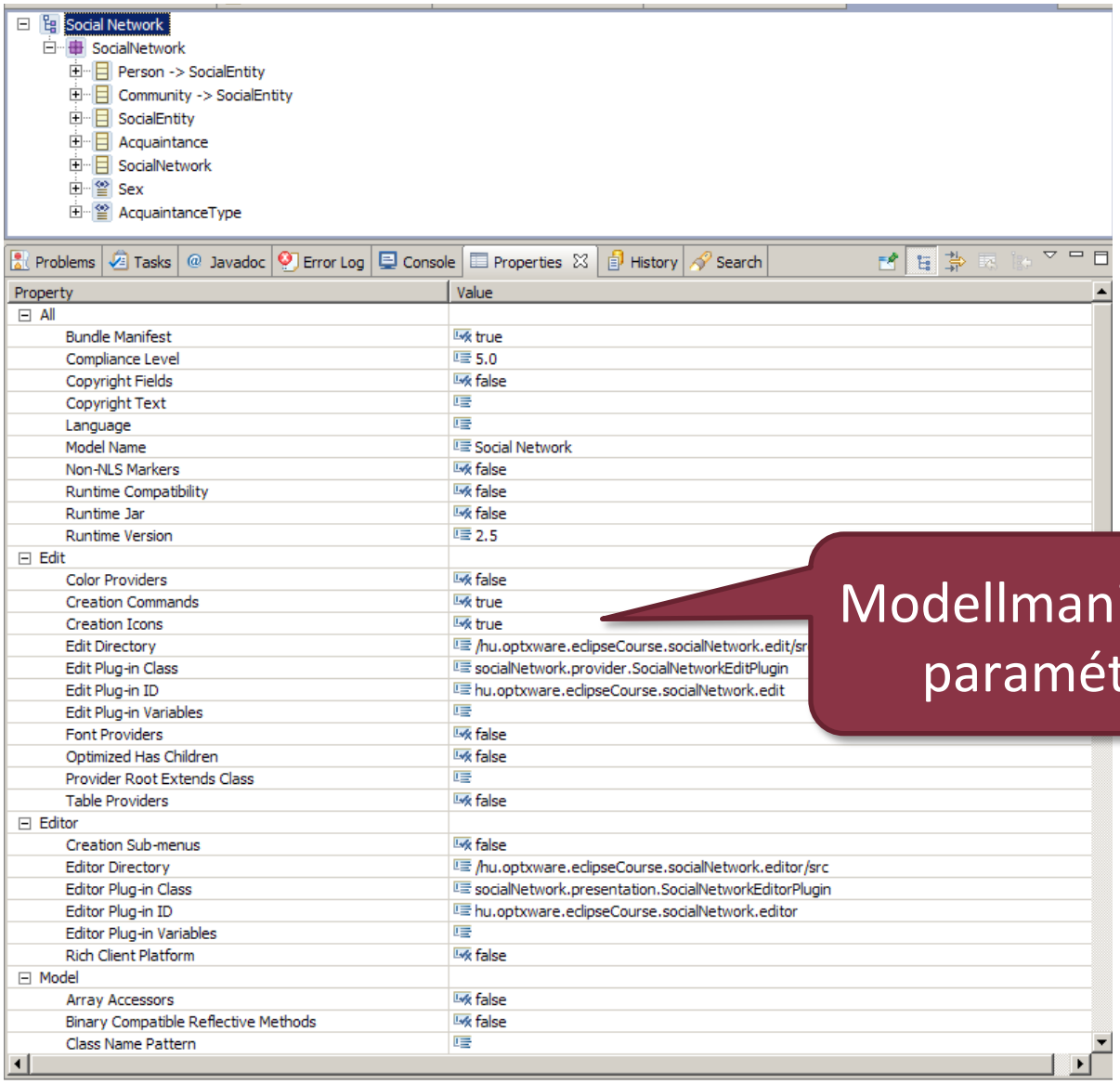
Hivatkozott Ecore modellelemek (akár több Ecore modellből is)

Generátor modell



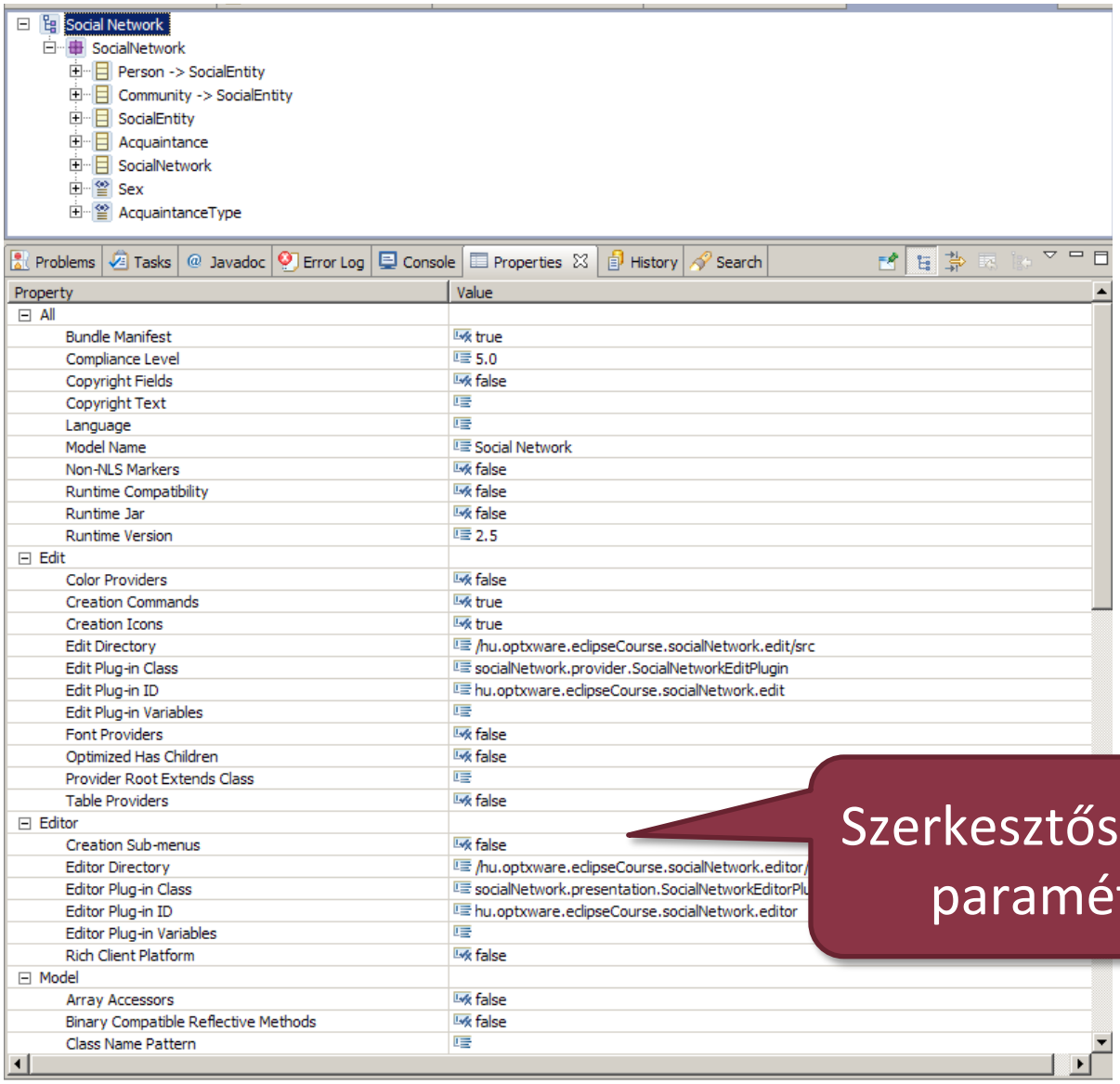
Általános paraméterek

Generátor modell



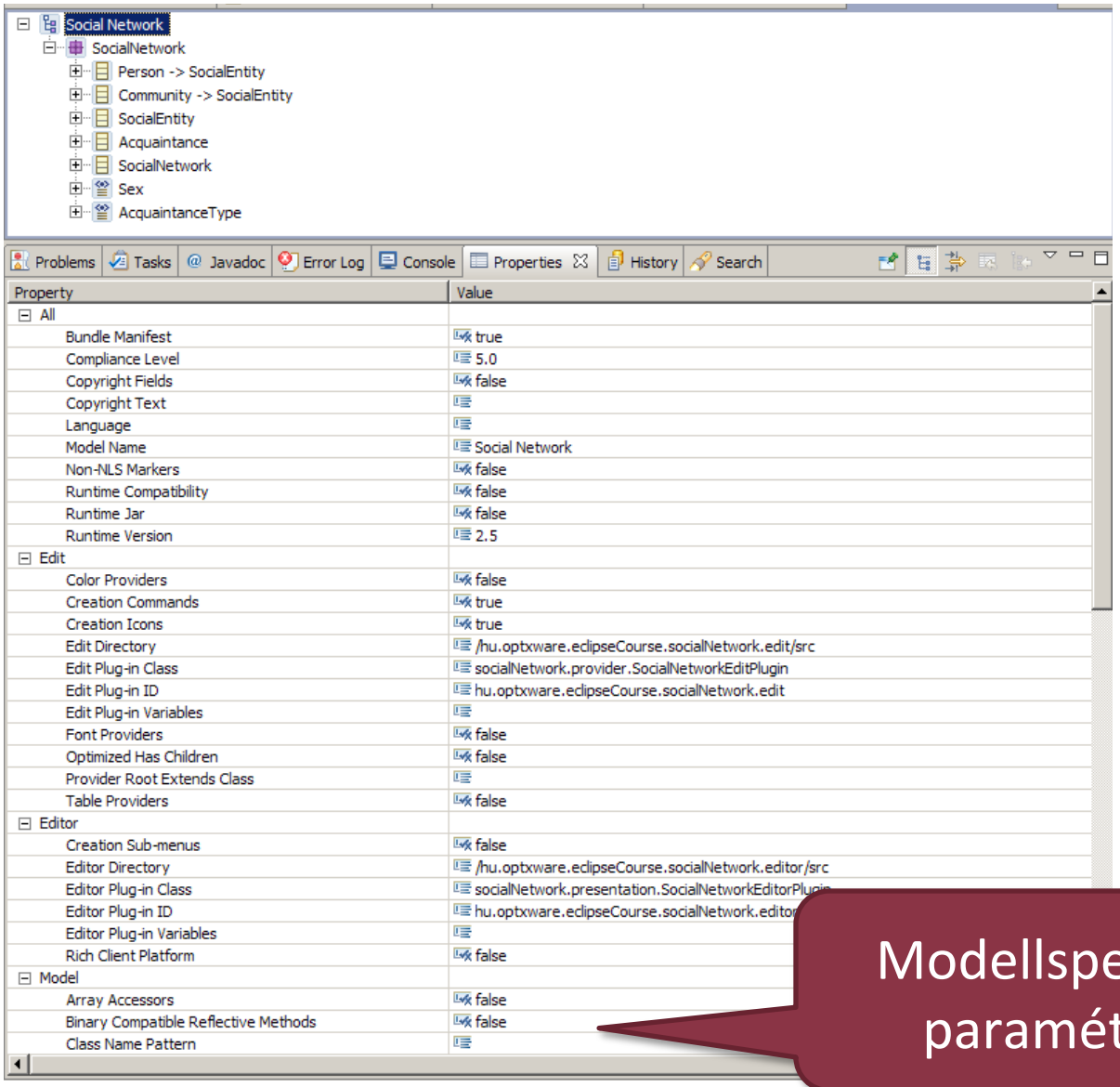
Modellmanipuláció paraméterei

Generátor modell



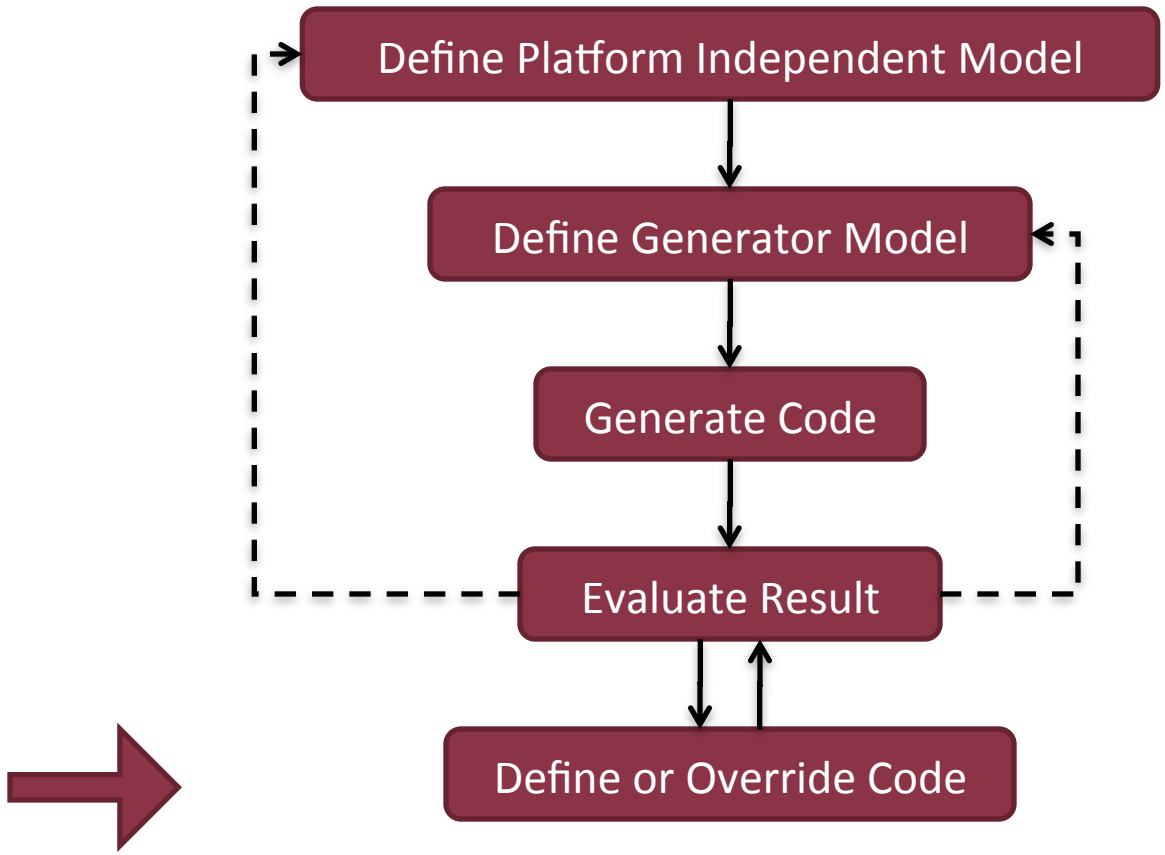
Szerkesztőspecifikus paraméterek

Generátor modell



Modellspecifikus paraméterek

Az EMF felhasználása



A generált EMF komponensek

EMF.Editor

- Egyszerű fa alapú szerkesztő

EMF.Edit

- GUI adatforrások
- Parancsok

EMF.Model

- Modellkezelő réteg
- Perzisztencia
- Reflektív API

A generált EMF komponensek

EMF.Editor

- Egyszerű fa alapú szerkesztő

EMF.Edit

- GUI adatforrások
- Parancsok

EMF.Model

- Modellkezelő réteg
- Perzisztencia
- Reflektív API

EMF.Model

- Az Ecore modell teljes implementációja
- Hatékony perzisztenciakezelés
  - XMI technológia
- A modell és a kód közel van egymáshoz
  - Előre tudjuk, mit kapunk
  - Általában nem szükséges módosítani

EMF.Model

- Lehetséges kiegészítések
  - Saját fájlformátum támogatás
    - Parser
    - Okos szerkesztő
    - Xtext projekt így működik
  - Extra információk beszúrása a generált forrásfájlokba
    - Kerülendő
    - Inkább legyen az Ecore modell része

Reflektív és generált API

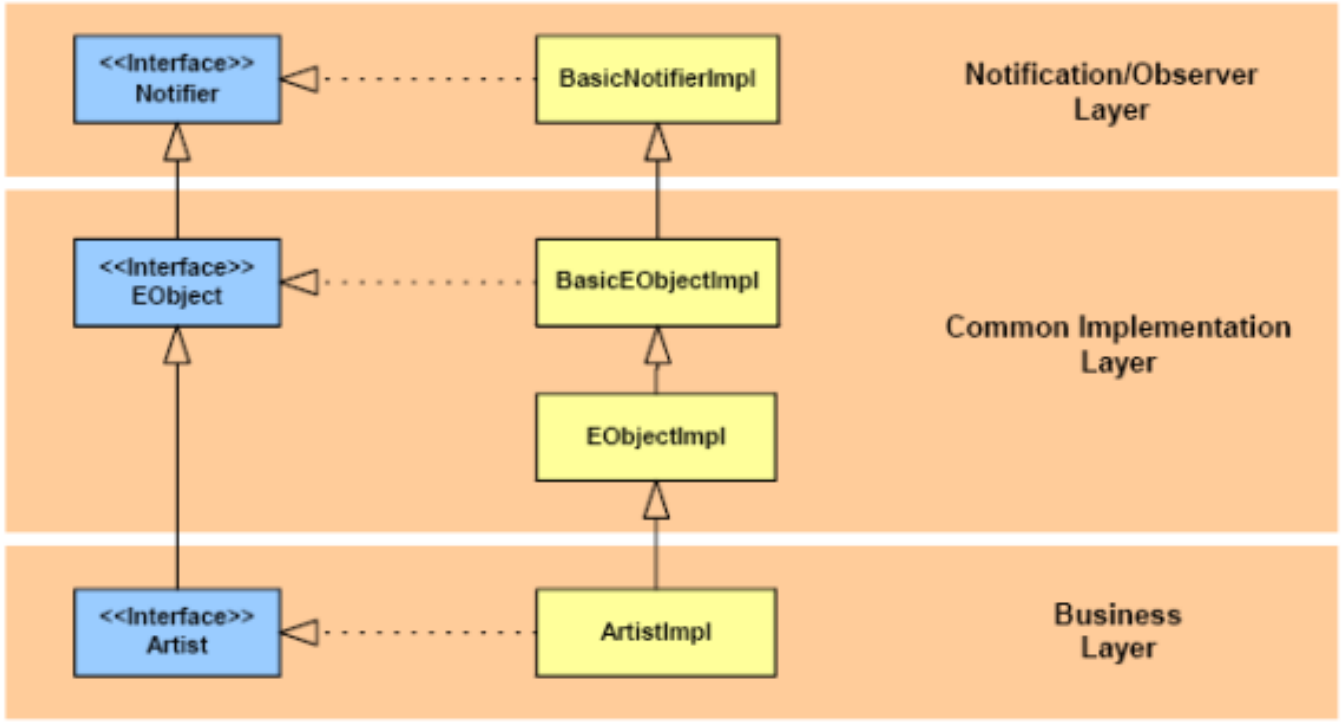
- Generált API
  - Típushelyes
  - Egyszerű getter/setter megoldás
  - Általában ezt célszerű használni
    - Félrecímzések, stb. elkerülhetőek
    - Optimalizálható konkrét metamodellre!
- Reflektív API
  - Stringekkel paraméterezhető
    - Ld. még: dinamikus nyelvek, Java reflektív API
  - Egységes kezelhetőség metamodellek között is

Generikus vs generatív

- Generatív megoldás
  - Kódot generálunk a modell alapján
  - Minden esetre lehet eltérő kódot használni
    - Pl. modell-specifikus tree editor, később GMF editor
- Generikus megoldás
  - Általános, minden esetre kiterjedő
    - Pl. sorosítás, Reflective Tree Editor

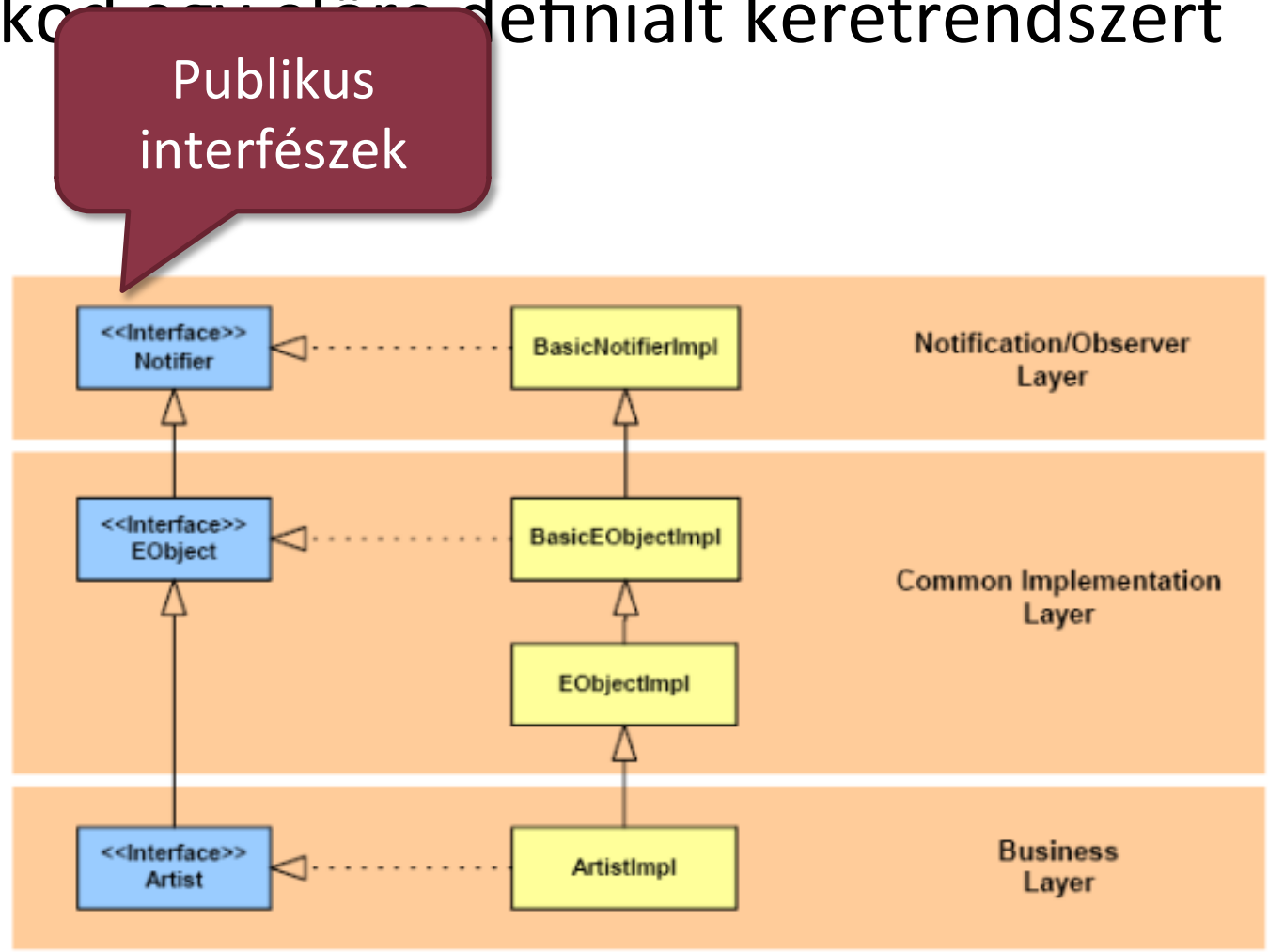
Generált struktúra

- A generált kód egy előre definiált keretrendszert terjeszt ki



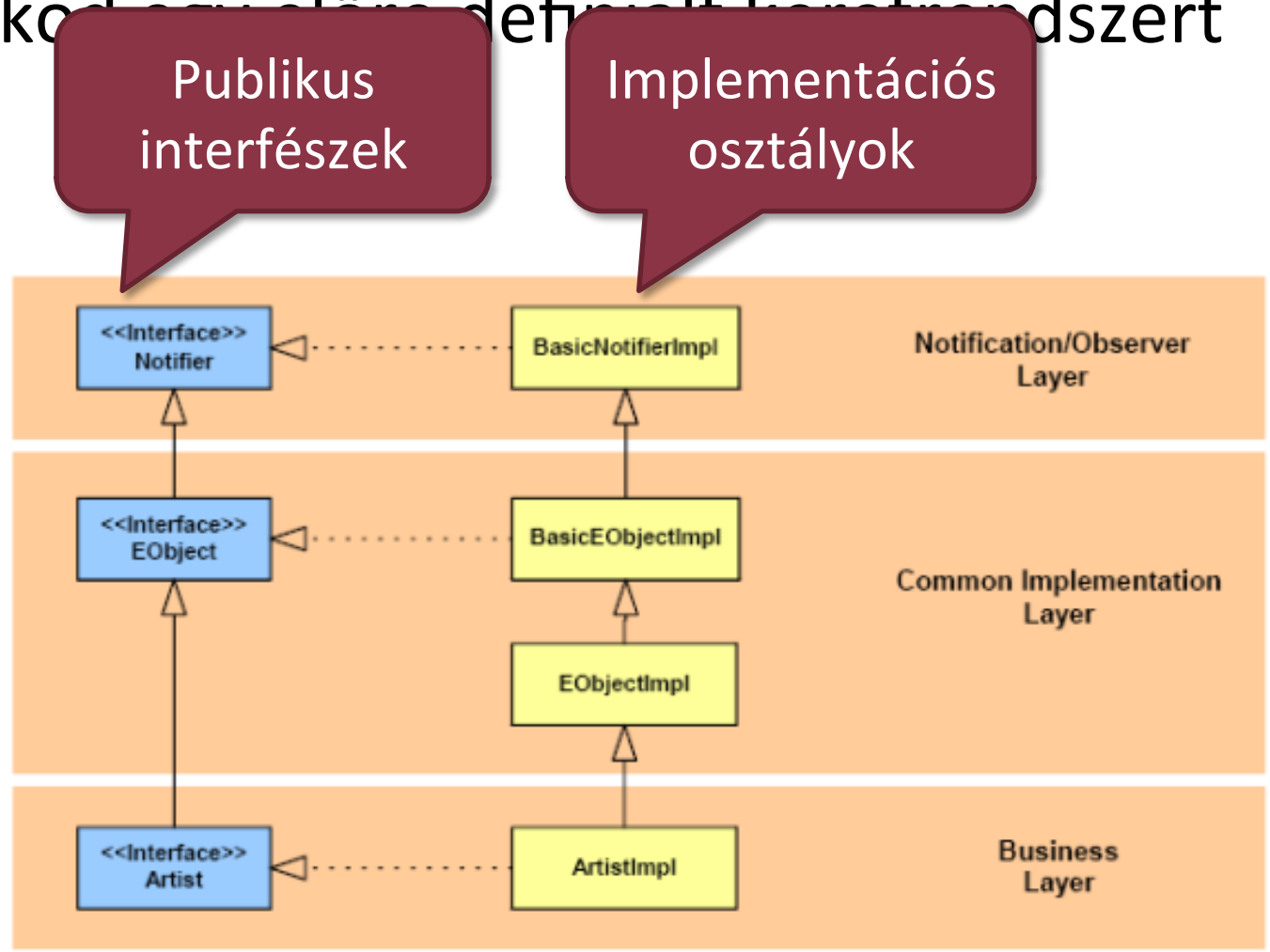
Generált struktúra

- A generált kód egy előre definiált keretrendszert terjeszt ki



Generált struktúra

- A generált kód egy előre definiált keretrendszert terjeszt ki



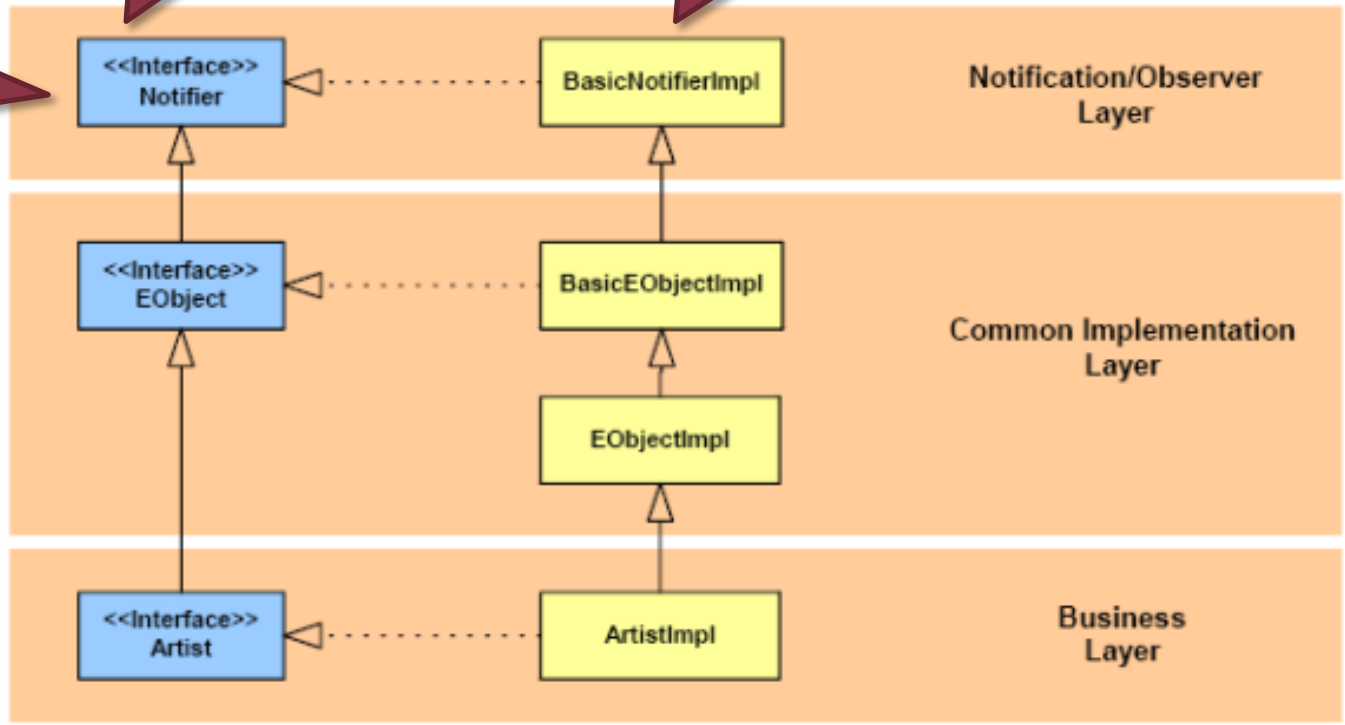
Generált struktúra

- A generált kód egy előre definiált keretrendszert terjeszt ki

Publikus interfészek

Implementációs osztályok

Általános értesítési mechanizmus



Generált struktúra

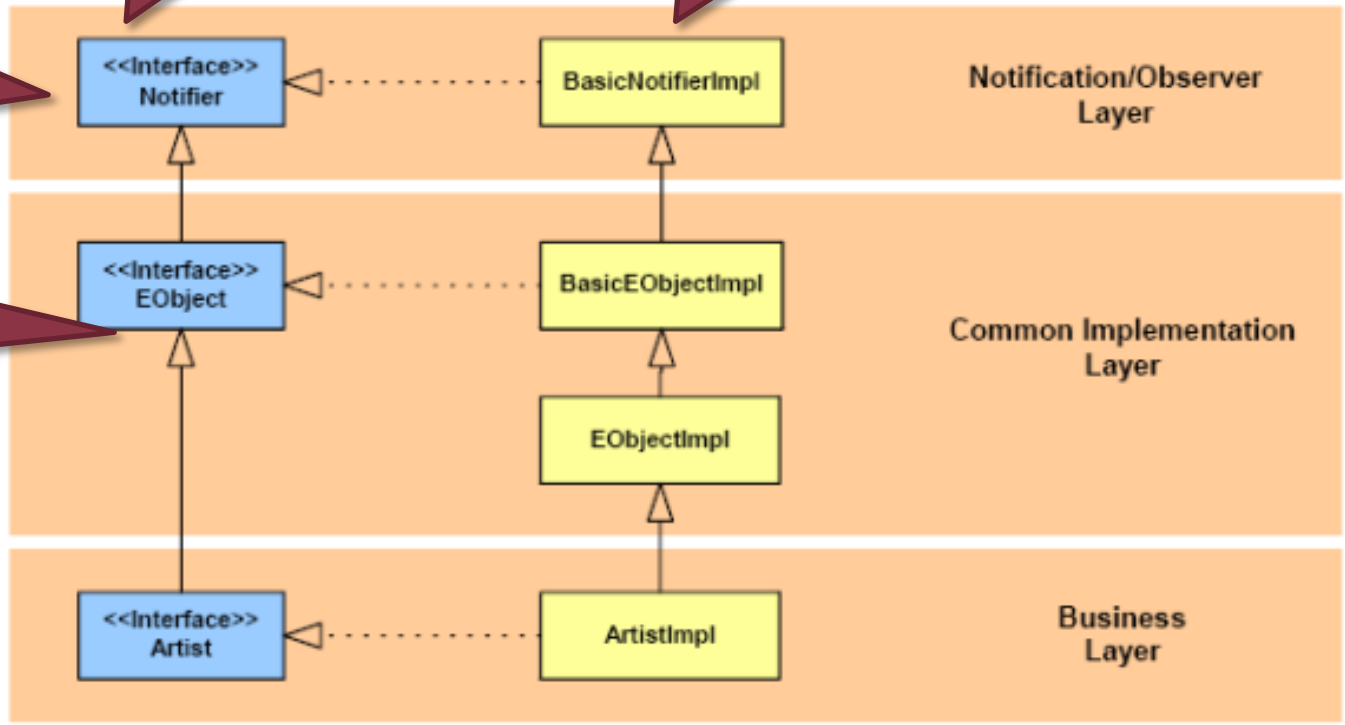
- A generált kód egy előre definiált keretrendszert terjeszt ki

Publikus interfészek

Implementációs osztályok

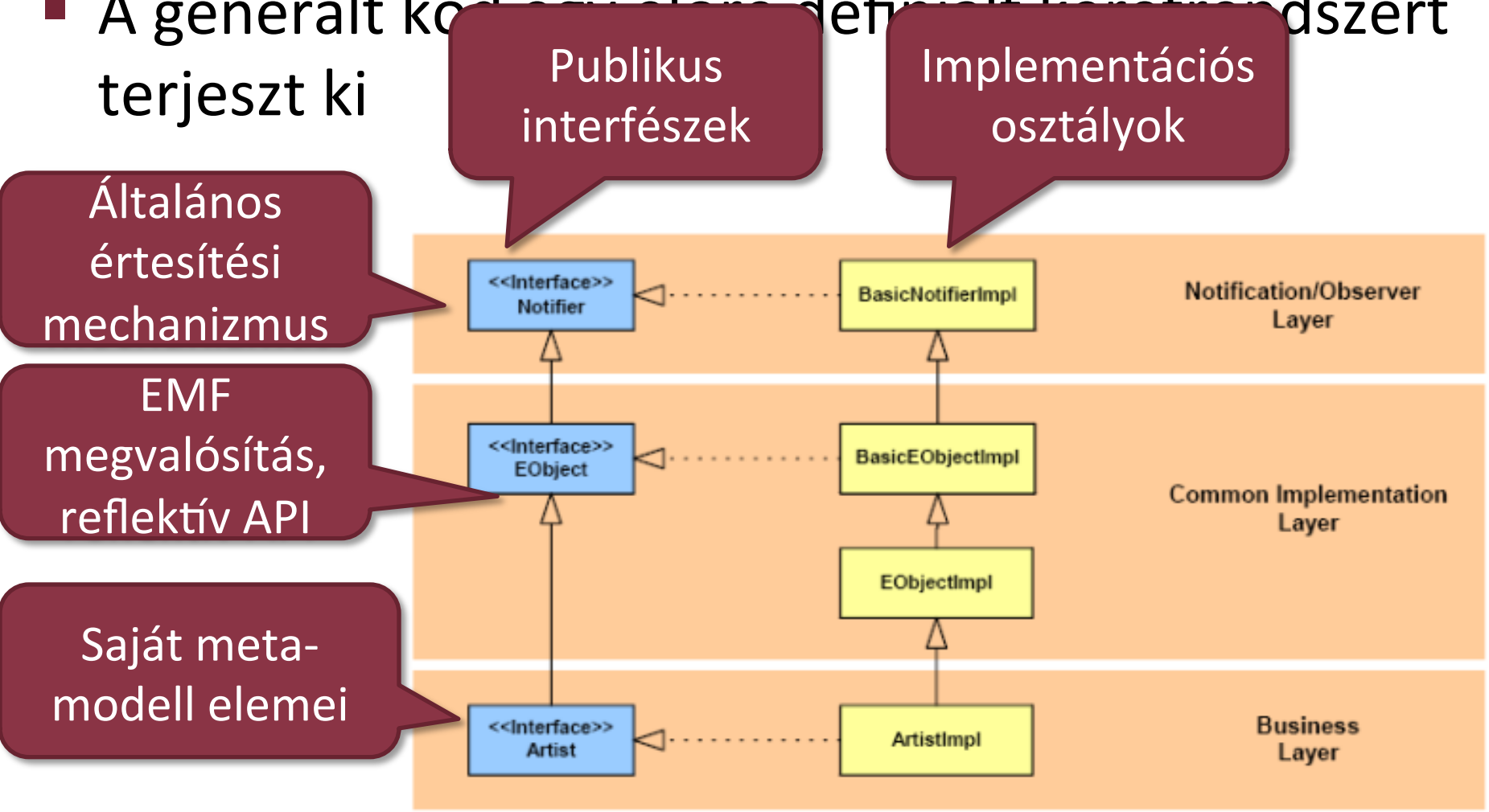
Általános értesítési mechanizmus

EMF megvalósítás, reflektív API



Generált struktúra

- A generált kód egy előre definiált keretrendszert terjeszt ki



EClass implementáció

- Attribútumok és referenciák kezelése
  - EAttribute: getter és setter metódusok
  - EReference: multiplicitásfüggő
    - “many”: szerkeszthető kollekció, hozzá egy getter metódus
    - “one”: getter metódus az értékre
- Kezdőértékek tárolása és inicializálása

EReference implementáció

- Hasonló az EAttribute-hoz
- A típus itt egy másik objektum lesz
- Néhány kiegészítés szükséges, ha nem tartalmazás jellegű reláció van
  - Az objektum más tárban is lehet
    - Referencia-feloldás
  - Inverz referenciák kezelése
    - Ellenkező oldal frissítése

EOperation implementáció

- Az Ecore modellben metódusokat is megadhatunk
- Nincs támogatás a szemantika definiálására
- Ötlet
  - Az Ecore modellben definiáljuk a metódus
    - Nevét
    - Paramétereit, típusukat
    - Visszatérési értékének típusát
  - Implementáljuk Java-ban
- A kódgenerátor csak kódvázat generál

EOperation implementáció

- Ha definiáltunk egy metódust
  - Hozzáadja az interfészhez
  - Egy üres implementációt készít az implementáló osztályban
- Első generálásnál az EMF egy Exception-t dobó implementációt készít
- Át kell venni a felügyeletet
  - Beállítjuk a @generated taget NOT-ra
  - Implementáljuk a metódust

EOperation implementáció

- H
  -
  -
- E
  ir
- Át kell venni a felügyeletet
  - Beállítjuk a @generated taget NOT-ra
  - Implementáljuk a metódust

```
public class Ximpl extends EObjectImpl implements X {

        /**
         * @generated NOT
         */
        void f() {
                // Provide the implementation
        }

}
```

EFactory implementáció

- EMF objektumpéldányosításhoz
  - Közvetlen példányosítás tilos!
- Példány elérése:
  - `<csomagnév>Factory.eINSTANCE`
  - `<csomagnév>Package.eINSTANCE .get<csomagnév>Factory`
- Minden osztály példányosítására metódus
  - `<osztálynév> create<osztálynév>`
- Vagy típusnév szerint

EPackage implementáció

- Példány elérése: <csomagnév>Package.eINSTANCE
- Minden osztályhoz metódus
  - `EClass get<osztálynév>`
- Minden osztály minden tulajdonságához metódus
  - `EStructuralFeature get<osztálynév>_<tulajdonságnév>`

Reflexió

- eClass()
  - Minden üzleti objektum megkaphatja a saját EClass reprezentációját
  - Hasonló a Java getClass() hívásához
    - eGet/eSet/eIsSet/eSet/eUnset()
  - Tulajdonságok generikus kezelése
- eContainer/eContents()
  - Hierarchia
  - Szülő/gyerekek visszaadása tartalmazások mentén

Reflektív API

- Platform biztosítja
- Generikus szolgáltatásokhoz
  - Sorosítás
  - Értesítés
  - Switch megoldásokhoz

Értesítés

- Minden modellobjektum támogatja az értesítésküldést
  - Observer minta
  - Event objektumok az értesítésben
  - Genmodellben állítható, hogy mi vált ki értesítést
- A megvalósítást az EMF biztosítja
  - Tipikusan sose akarjuk módosítani…

Értesítés

- Feliratkozás
  - `eAdapters().add(Adapter)`
  - Alapértelmezetten nem rekurzív!
- Értesítésküldés
  - `eNotify(Notification)`
  - Ritkán kell kézzel

EMF modellek sorosítása

- EMF modellek tárolása: erőforrásokban (resource)
- Egy objektumhoz rendelhetünk egy Resource példányt
  - `eResource()` adja vissza
- Beépített implementáció: `XMIResourceImpl`
  - Betöltés/mentés XMI formátumú fájlokból/fájlokba

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model manipulation here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, tr
try {

      for (EObject root : res.getContents()) {
            //TODO Model manipulation here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

ResourceSet példányosítás

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getReso
try {

      for (EObject root : res.getContents()) {
            //TODO Model manipulation here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

URI: modell elérési út

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getConten
            //TODO Model manipulation he
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

getResource:
lusta betöltés
on-demand

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model manipulation here
      }

      res.save(new HashMap<String, St
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Lehet több modellgyökér is! Típushelyességet ellenőrizni kell!

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model manipulation here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Resource mentése

EMF modell feldolgozás sablonja

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model manipulation here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Ne feledjük a kivételkezelést!

URI séma

- Erőforrás helyének meghatározása
  - URI segítségével
  - Alapvetően URI osztály statikus metódusaival
- Beépített támogatás (bővíthető)
  - File (java.io.File)
    - `URI uri = URI.createFileURI("/path/to/file")`
  - Eclipse workspace file (IFile)
    - `URI.createPlatformResourceURI("projectName/foldername/filename", true)`
  - Fájl egy Eclipse plug-inből
    - `URI.createPlatformPluginURI("pluginName/folderName/filename", true)`

Sorosítás és tartalmazási hierarchia

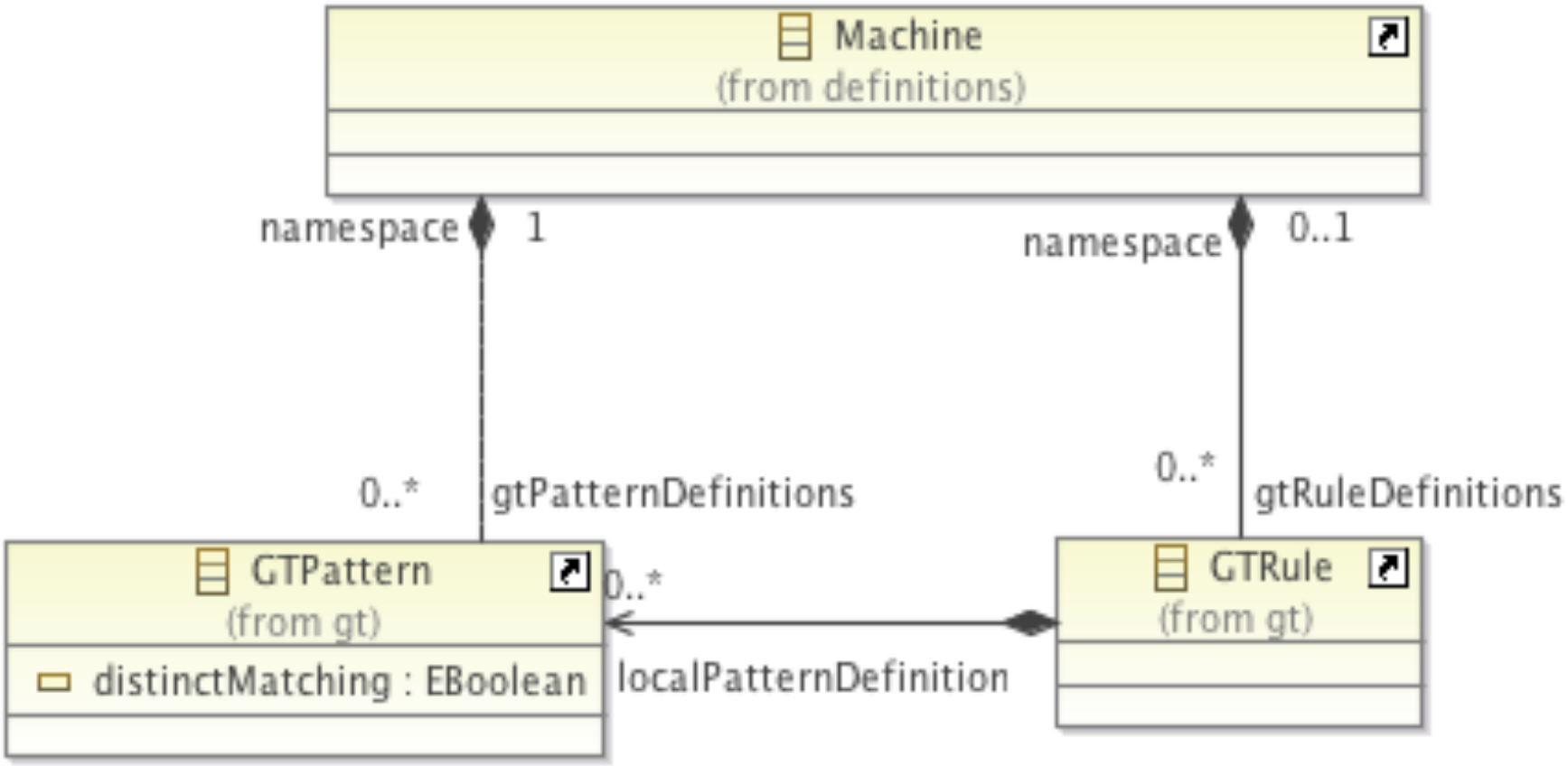
- Tartalmazási hierarchia kritikus
  - Metamodell szinten definiálandó
    - Fel kell sorolni a tartalmazás referenciákat
  - Garantáltan körmentes
    - A struktúra mentén elmenthető a tartalom
    - Kör mentén nem sorosítható a tartalom
  - Hibás struktúra esetén **sorosítási hibák**
    - Modellelem nincs benne a fastruktúrában
    - Kör van a fastruktúrában

Ellenőrző kérdések

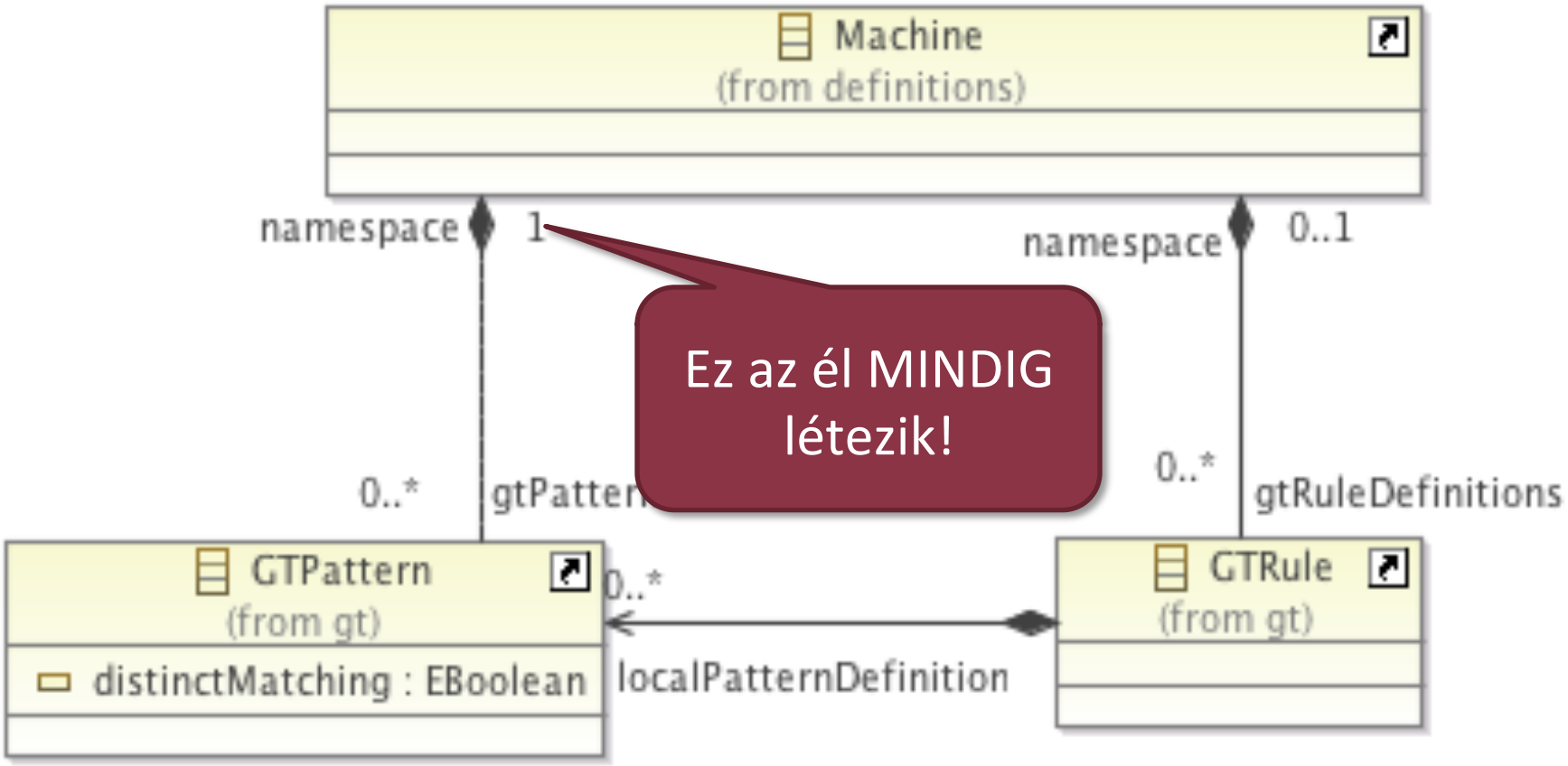
Ellenőrző kérdések

1. **Indulhat**-e ki egy **osztály**ból több tartalmazás típusú él?
2. **Indulhat**-e ki egy **objektum**ból több tartalmazás típusú él?
3. **Érkezhet**-e egy **osztály**ba több tartalmazás típusú él?
4. **Érkezhet**-e egy **objektum**ba több tartalmazás típusú él?
5. Válaszok: 1) Igen, 2) Igen, 3) **Igen**, 4) Nem

Mi a hiba a metamodellben?



Mi a hiba a metamodellben?



A generált EMF komponensek

EMF.Editor

- Egyszerű fa alapú szerkesztő

EMF.Edit

- GUI adatforrások
- Parancsok

EMF.Model

- Modellkezelő réteg
- Perzisztencia
- Reflektív API

A generált EMF komponensek

EMF.Editor

- Egyszerű fa alapú szerkesztő

EMF.Edit

- GUI adatforrások
- Parancsok

EMF.Model

- Modellkezelő réteg
- Perzisztencia
- Reflektív API

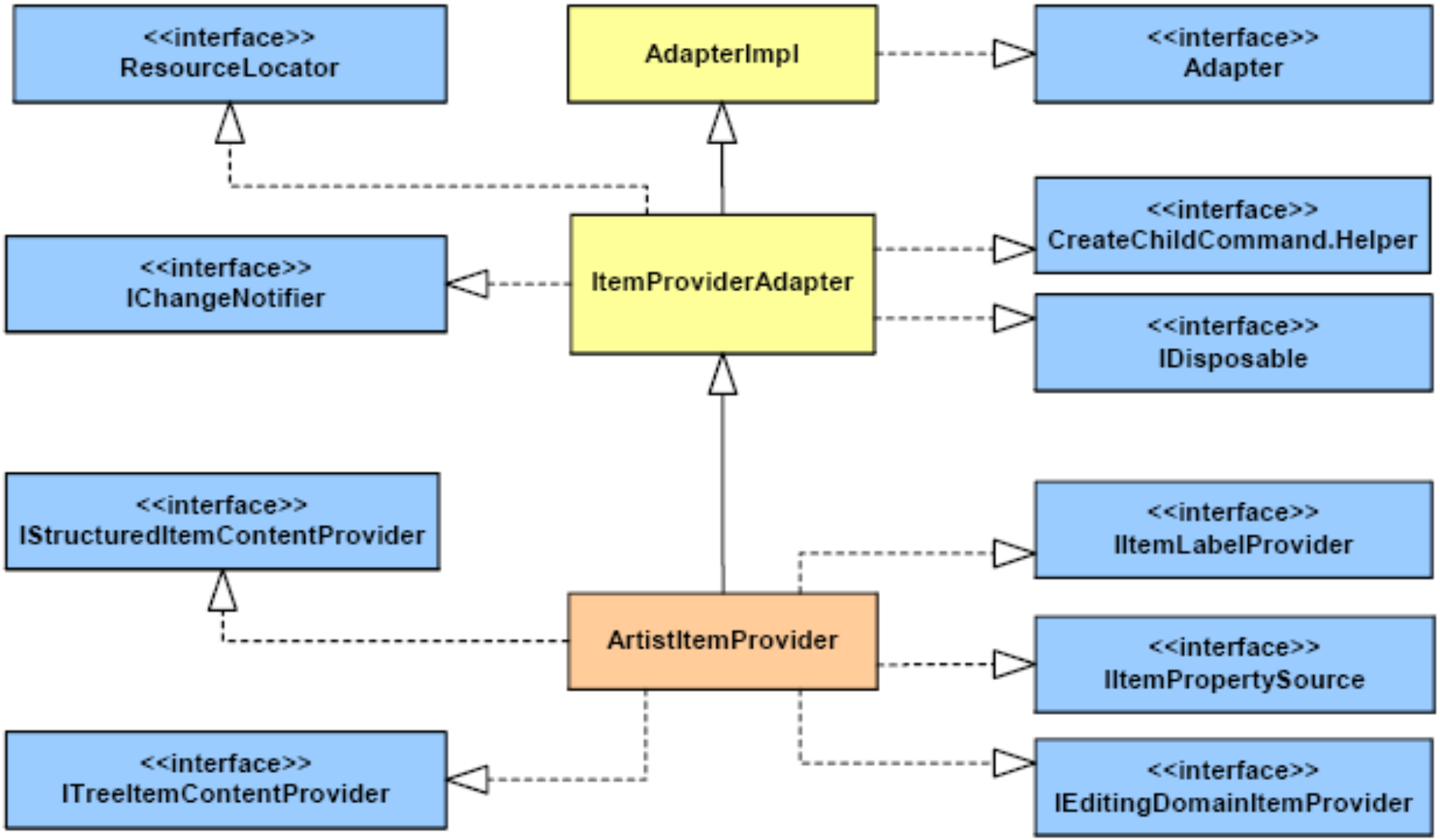
EMF.Edit

- Szerepe
  - A GUI és a modell szétválasztása
  - GUI független akciók implementálása
- Nagyobb eséllyel módosítjuk
  - Módosítjuk az element providert
  - Új parancsokat adunk hozzá

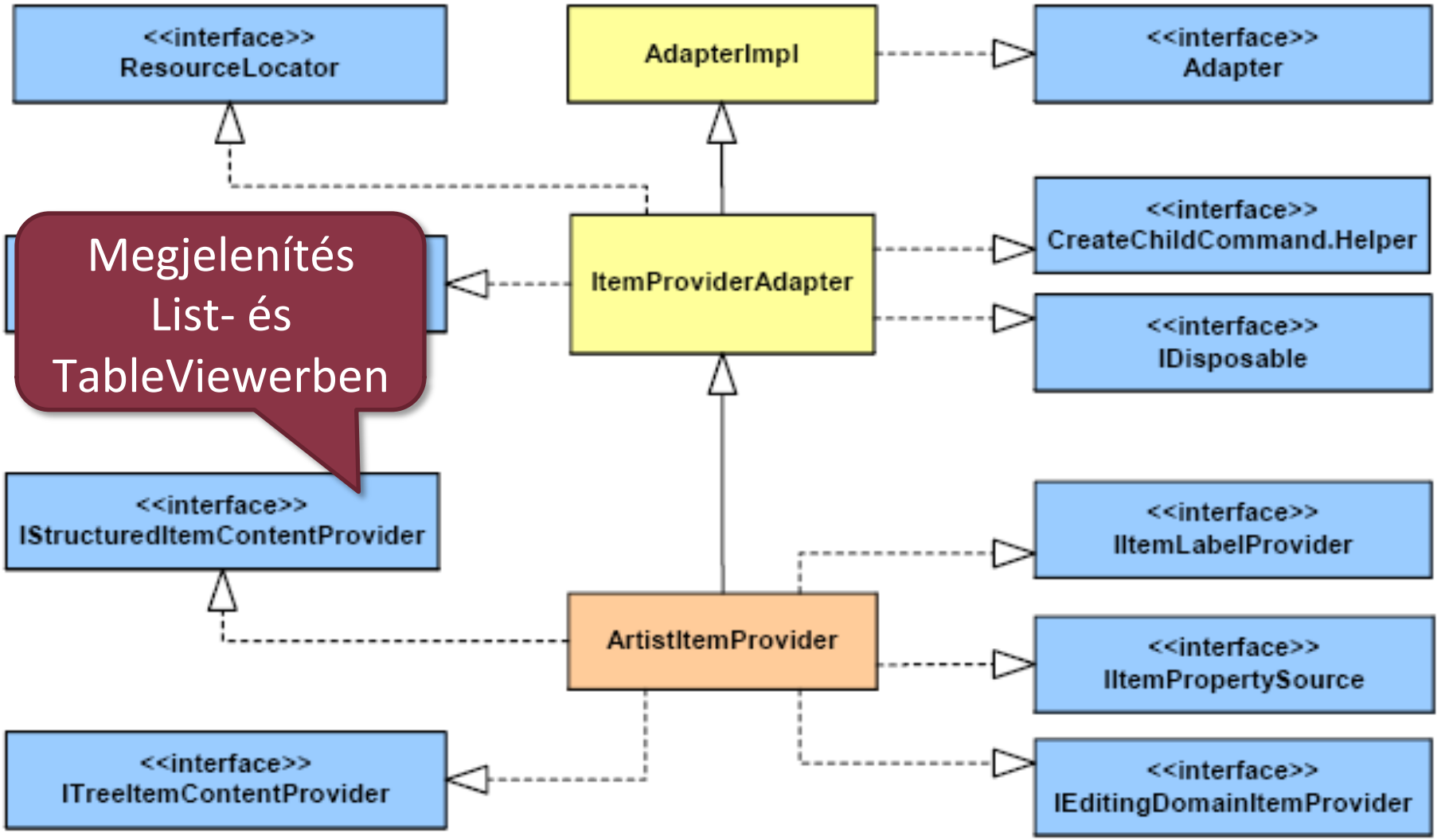
Generator minta

- Minden modell objektumhoz
  - Egy adapter jön létre
    - Neve: ItemProvider
    - Pl. ArtistItemProvider
- A providerek közös őse:
  - org.eclipse.emf.edit.provider.ItemProviderAdapter
    - Alapértelmezett implementáció alap funkcionalitáshoz
    - Egy részét definiáljuk felül

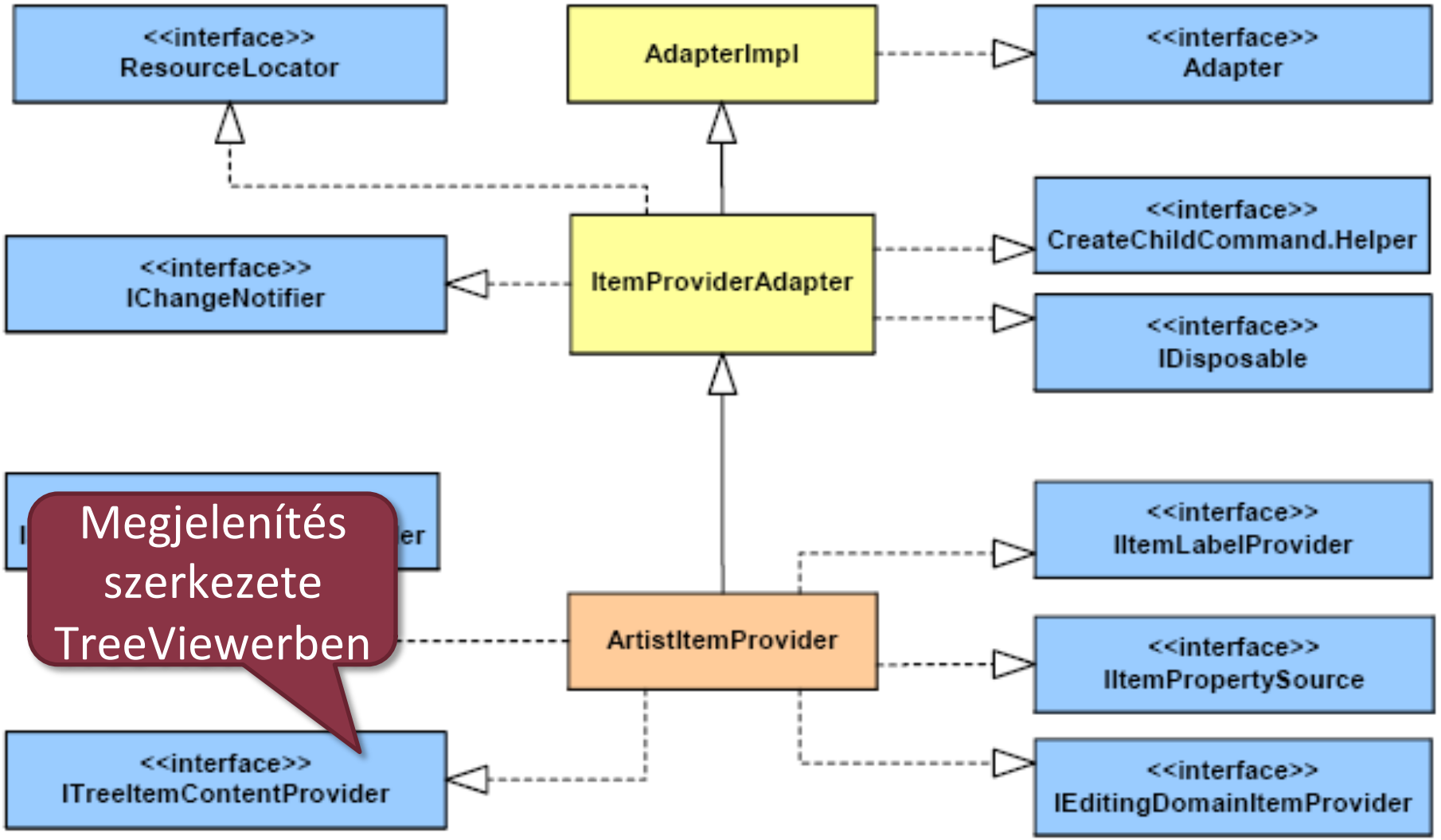
Generator struktúra



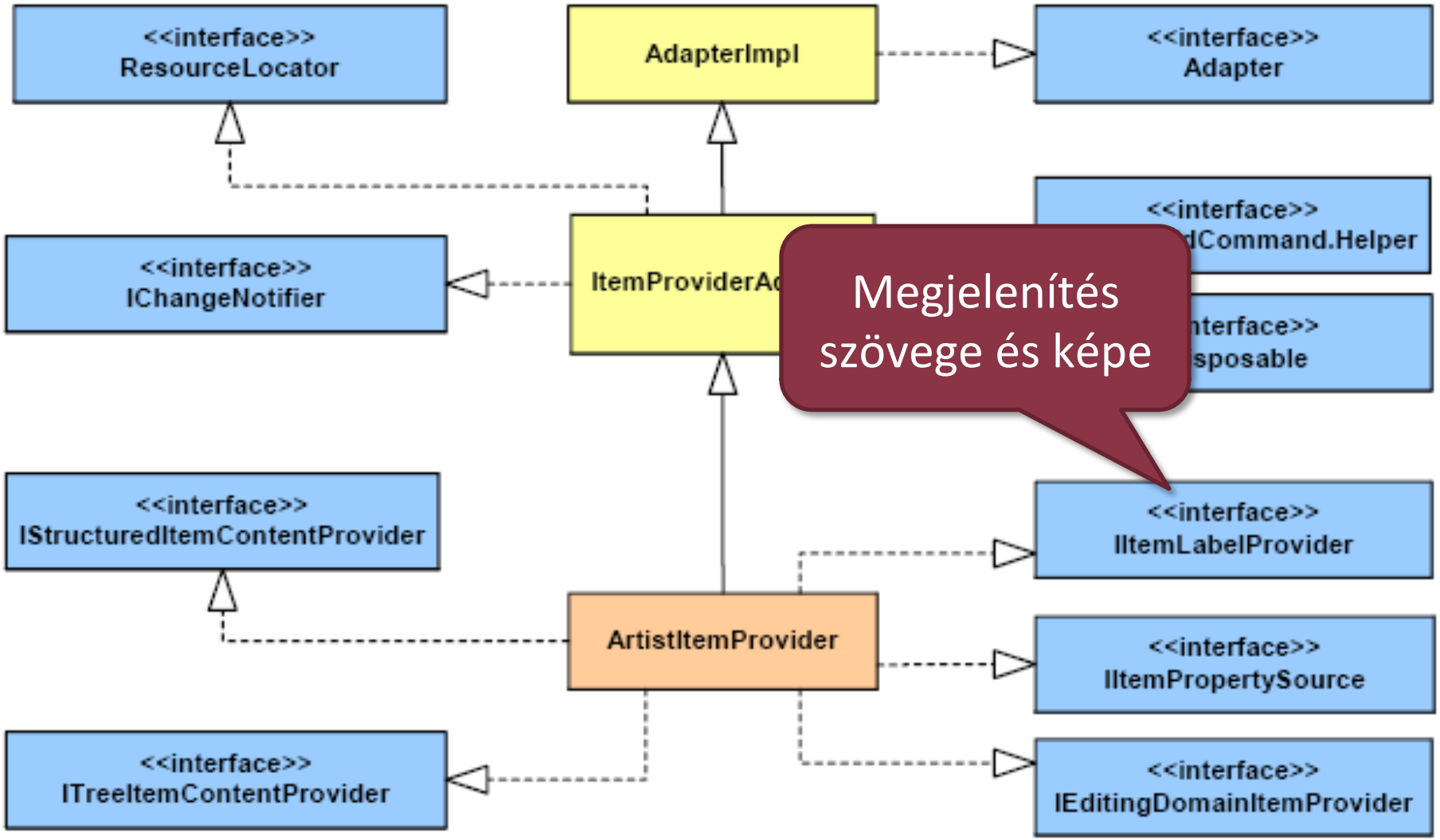
Generator struktúra



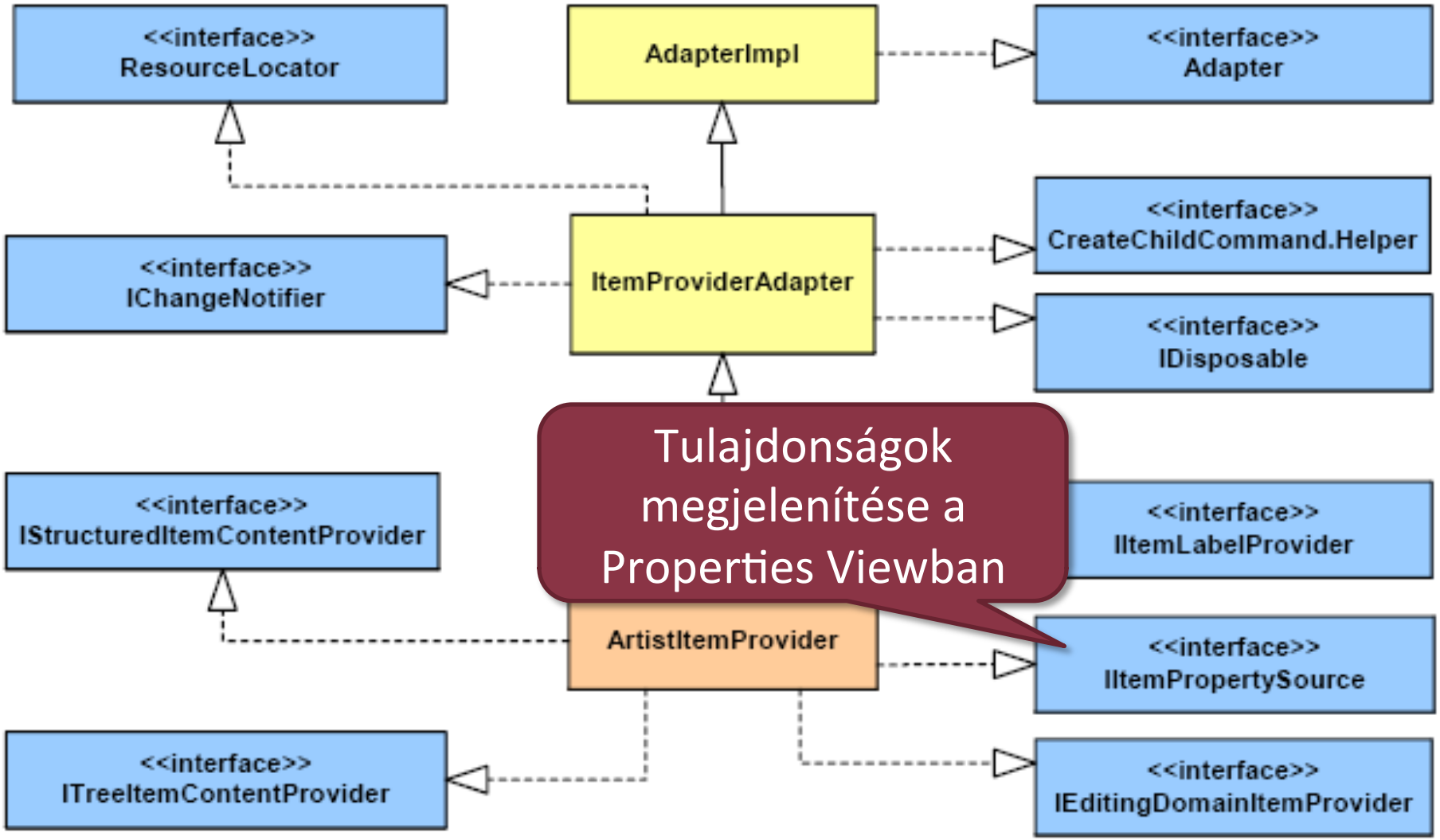
Generator struktúra



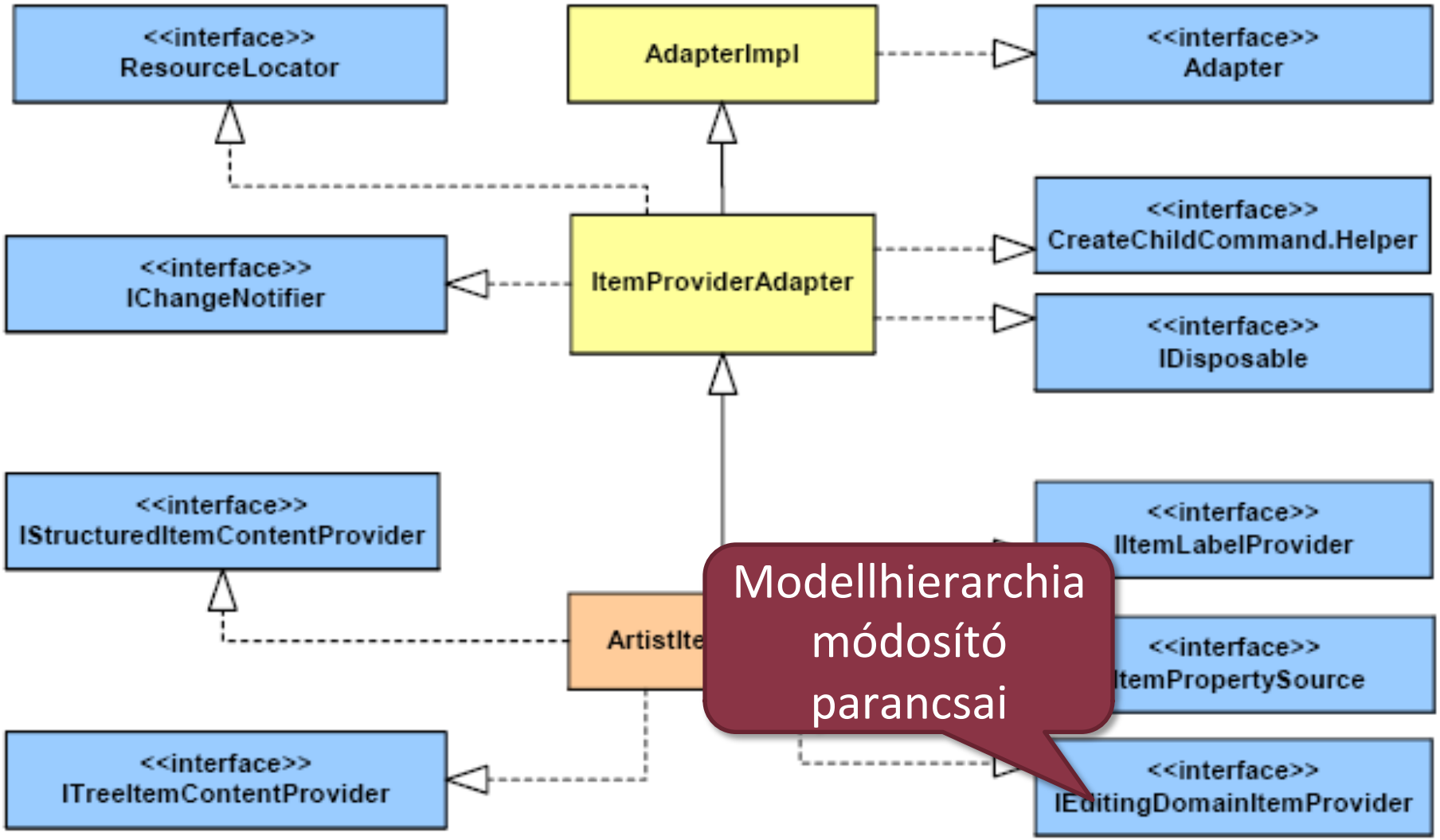
Generator struktúra



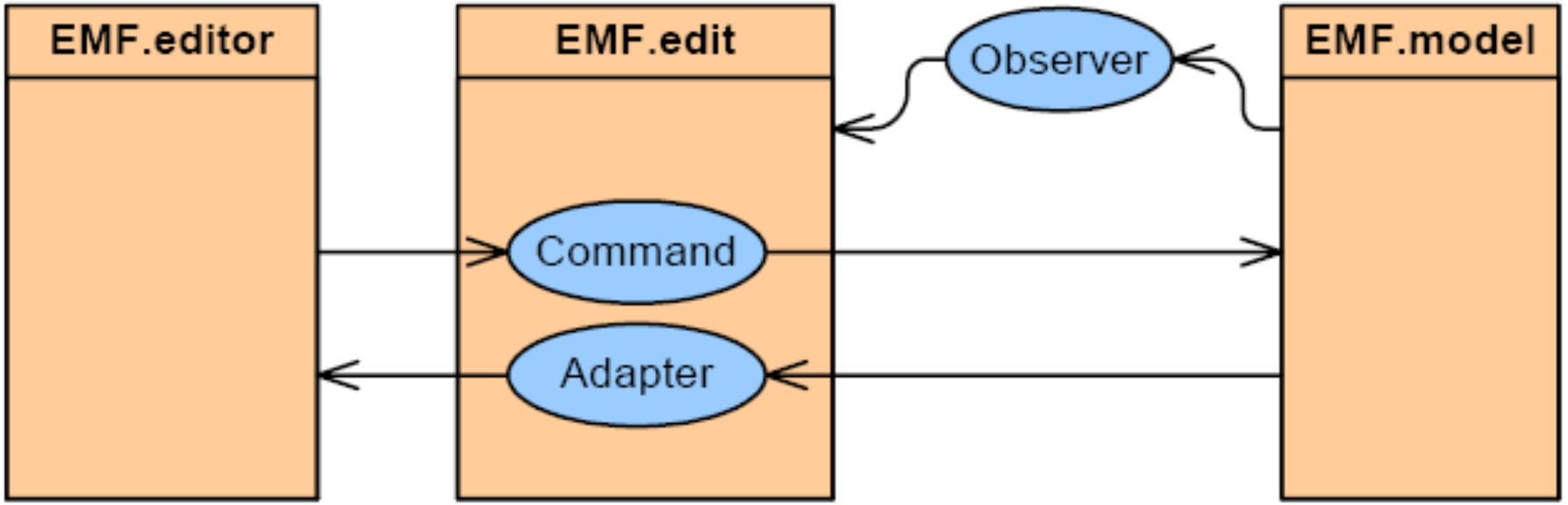
Generator struktúra



Generator struktúra



EMF.Edit és a tervezési minták



Címkék változtatása

- A testreszabás tipikus példája a címkék változtatása
  - A genmodel-ben megadhatjuk, hogy egy objektum mely attribútuma jelenjen meg címkeként
  - Mi van, ha többből akarjuk összerakni?
- Változtatni kell a ItemProvider.getText-en
  - Töröljük/NOT-ra állítjuk a @generated taget
  - Saját implementációt írunk

Ikonok változtatása

- Másik tipikus példa az ikonok megváltoztatása
- A genmodel egy egyszerű ikont rendel minden elemhez
  - Az elemek az edit projekt icons/full könyvtárában találhatóak
    - obj16: konkrét osztályokhoz
    - ctool16: gyerekelem-létrehozási parancsokhoz
- A legegyszerűbb a fájlok lecserélése saját ikonra
- Ha logika alapján kell: ItemProvider.getImage felülírása

EMF.Edit parancsok

- Minden módosítás parancsokon (command) keresztül történik
  - Menü akció
  - Attribútum változtatás
  - Drag-n-drop
- Undo/redo támogatás
- A keretrendszer generált és már létező kód keverékét használja
  - EMF Common Command Framework (CCF)
  - EMF.Edit által generált parancsok

Parancsok használata EMF.Edit-ben

- A parancsok használata a Template Method mintára épül
- Az ItemProvider implementálja a `createCommand()` metódust, a kéréseket továbbítja protected metódusaiknak
  - `createAddCommand()`
  - `createRemoveCommand()`
  - …
- A módosítás során a protected metódusok egyikét módosítjuk általában

Parancsok az EMF.Edit-ben: Példa

```
public class SetArtistNameCommand extends SetCommand {

        public SetArtistNameCommand(EditingDomain domain, Eobject owner,
                EStructuralFeature feature, Object value) {
                super(domain, owner, feature, value);
        }

        public void doExecute() {
                Artist artist = (Artist)this.owner;
                Logger.log(MessageFormat.format(
                        "Name of artist changed from {0} to {1}",
                        artist.getName(), value.toString()));
                super.doExecute();
        }

}
```

- Egyszerű példa: loggolás hozzáadása
- Bonyolultabb példák esetén az összetett parancsok módosítása is szükséges lehet

A generált EMF komponensek

EMF.Editor

- Egyszerű fa alapú szerkesztő

EMF.Edit

- GUI adatforrások
- Parancsok

EMF.Model

- Modellkezelő réteg
- Perzisztencia
- Reflektív API

A generált EMF komponensek

EMF.Editor

- Egyszerű fa alapú szerkesztő

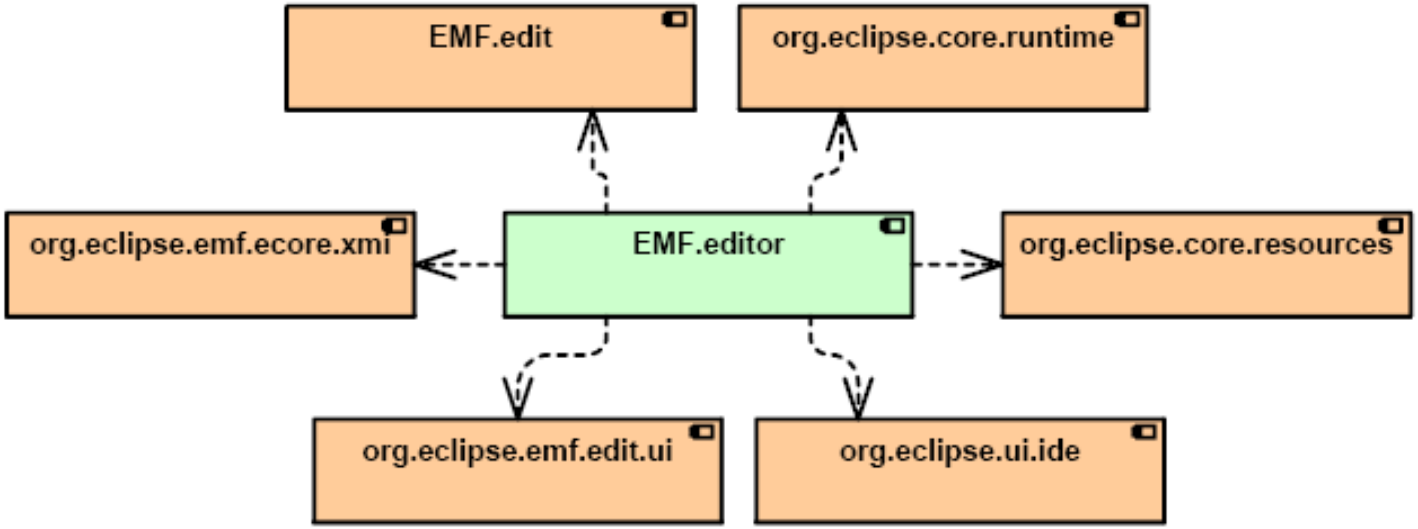
EMF.Edit

- GUI adatforrások
- Parancsok

EMF.Model

- Modellkezelő réteg
- Perzisztencia
- Reflektív API

EMF.Editor

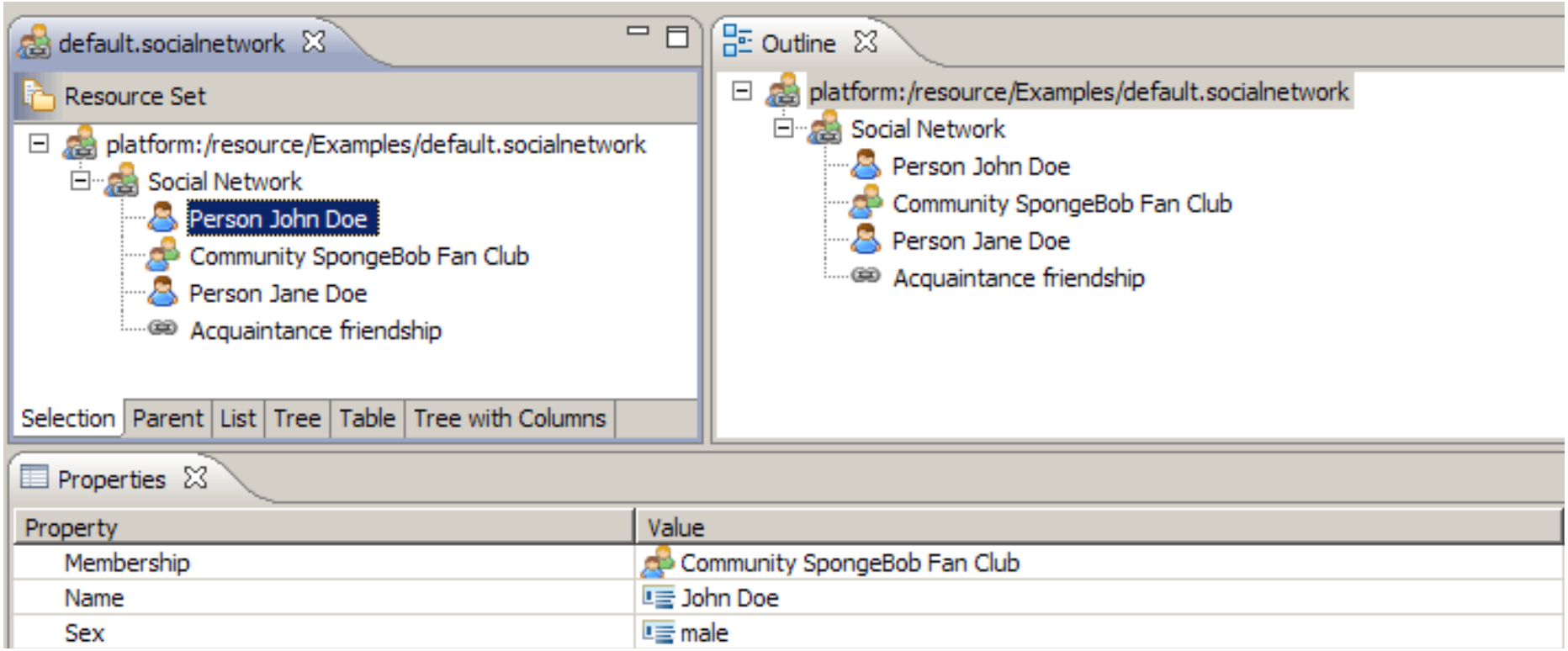


- Az EMF.Editor generálja az SWT/JFace kódot a grafikus editorhoz
- Két fő opció
  - Meghagyjuk alapértelmezetten
  - Újraimplementáljuk teljes egészében

Mi generálódik?

- Editor (JFace alapú)
  - Fastruktúra
  - Események – akciók összekötése
  - Workbench elemek beállítása
- Menük
- Varázsló (új modell…)
- Plugin activator

A generált szerkesztő



EMF.Editor - Összefoglalás

- A generált editor elsődlegesen tesztelésre való
  - Fastruktúra nehezen átlátható
  - Alapszintű műveletek
- Alternatív megoldások
  - GMF: GEF technológiára alapuló grafikus editor EMF modellekhez
  - Xtext: szöveges editor EMF modellekhez

Kitekintés: Xcore

Xcore: Új metamodellkezelési lehetőség

- Szöveges metamodell-leíró nyelv
  - Ecore gyengeségeire felismerték -> új technológia
  - Származtatott attribútumok/metódusok
    - Xtext/Xbase alapon
- Nagyrészt kompatibilis a régi módszerrel
  - Runtime közös
  - Legenerálható
- Részletek:
  - http://wiki.eclipse.org/Xcore

Xcore példa

*Library.xcore

```
package org.example.library

class Library
{
    String name
    contains Book[] books opposite library
    contains Writer[] authors opposite library
}

class Book
{
    String title
    int pages
    container Library library opposite books
    refers Writer[] authors
}

class Writer
{
    String name
    container Library library opposite authors
    refers Book[] books opposite
}
```

authors

Ctrl+Space to show shortest proposals

Kép forrása: http://wiki.eclipse.org/Xcore

Xcore példa

```
class Library
{
    String name
    contains Book[] books opposite library
    contains Writer[] authors opposite library
    op Book getBook(String title)
    {
        for (Book book : books)
        {
            if (title == book.title) return book
        }
        return null
    }
}

class Book

class Writer
```

Kép forrása: http://wiki.eclipse.org/Xcore

EMF - Összefoglalás

EMF

- Általános modellező keretrendszer
  - Többféle bemenet
  - Sorosítási és szerkesztési támogatás
- Kódgenerálás
  - Testreszabható
  - Jó alapértelmezések
- Sokan használják
  - Sokféle Eclipse-es projekt alaptechnológiája
  - Jól alkalmazható különféle területeken

További EMF alapú technológiák

- Validation
  - Kényszerek megadása és ellenőrzése
- Query
  - Magas szintű lekérdezések futtatása
- Compare
  - Modellek strukturális összehasonlítása (pl. verziókezeléshez)

- Teneo
  - EMF modellek perzisztenciája relációs adatbázisba
- SDO
  - Service Oriented Architecture megvalósítása EMF alapon
- CDO
  - Elosztott, kliens-szerver EMF modellek