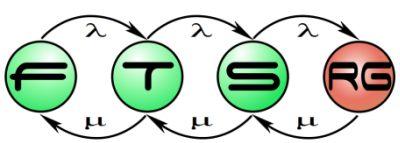
Szöveges editorok - Xtext



Szöveges editorok - Áttekintés

- Eddig
  1. Szöveges editorok alapjai
  2. Eclipse editorok készítése IMP technológiával
- Most
  1. Eclipse editorok készítése Xtext környezetben

Xtext

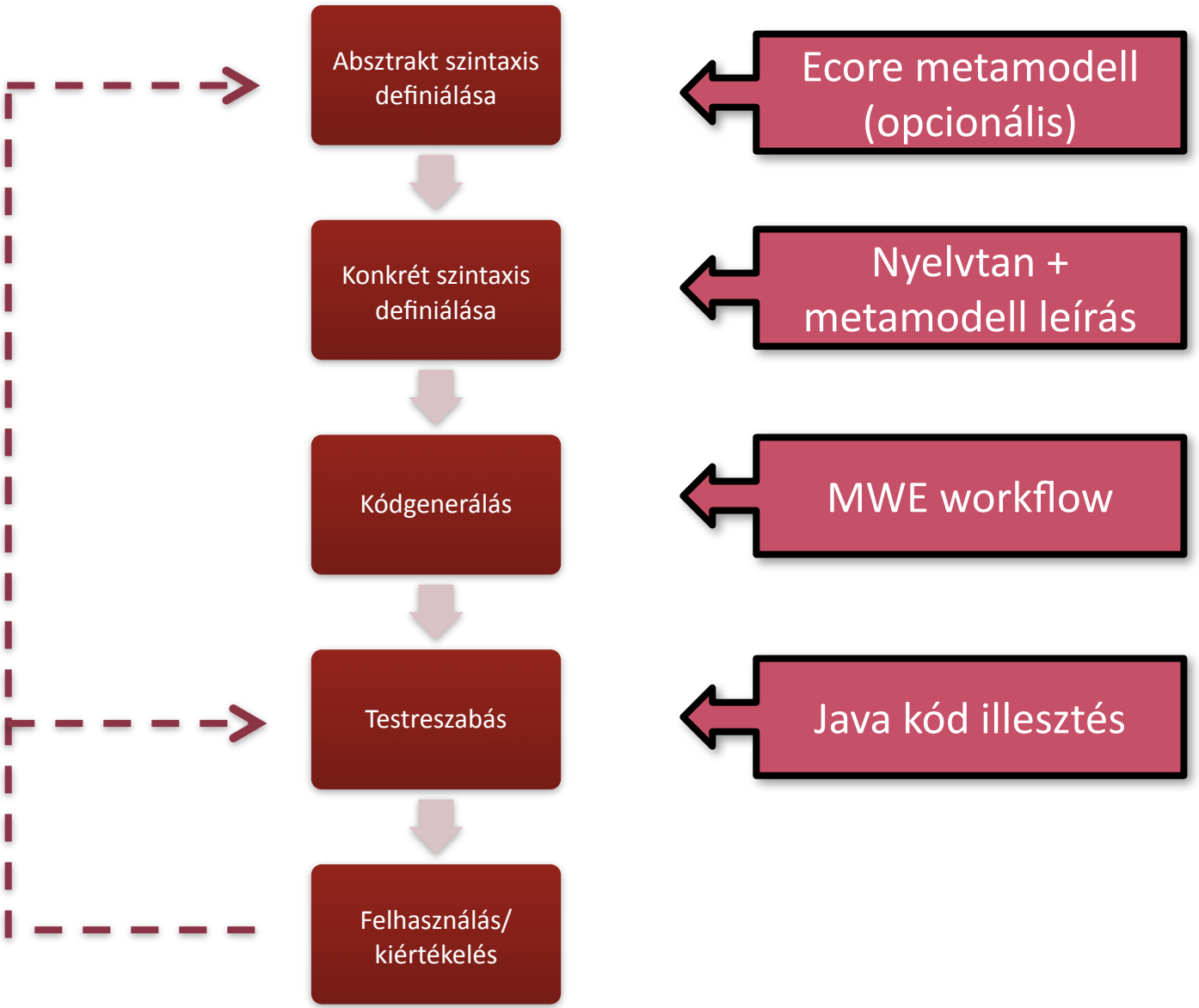
"A DSL for writing DSLs"

Peter Friese, Itemis

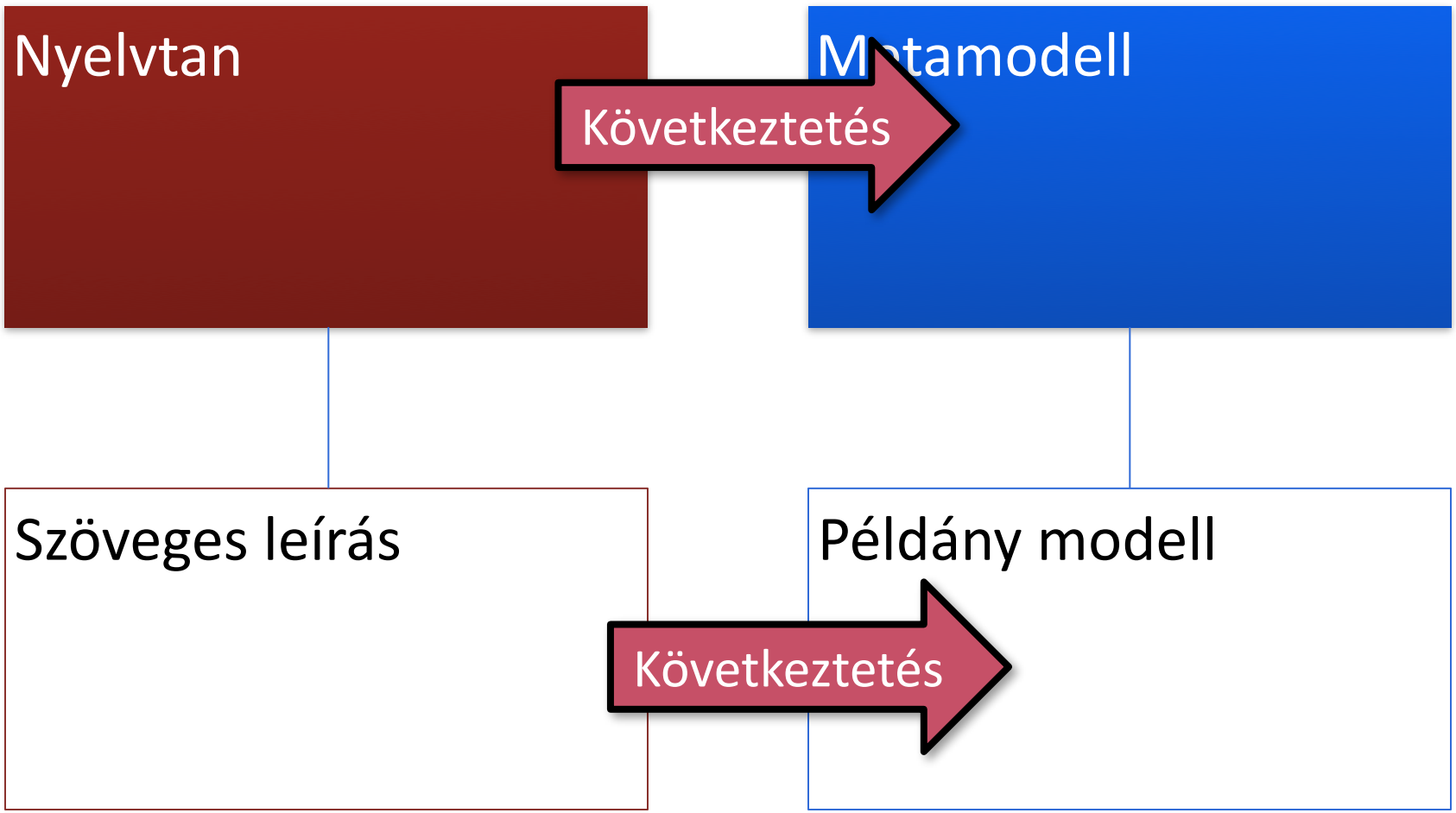
Xtext

- Cél
  1. Integrált szöveges editor
  2. EMF alapú AST
  3. Kódgenerálási támogatás
     - Editorhoz
     - Nyelvhez

Felhasználás menete



Nyelvek és modellek



Konkrét szintaxis megadása

Nyelvtan leírása

- Domain-specifikus nyelv (.xtext fájl)
  1. EBNF nyelvtanspecifikáció
  2. Kiegészítésekkel
     - További nyelvtan elemek
     - Metamodell generálás pontosítása

Konkrét szintaxis megadása

- Parser generátor
  1. Antlr
  2. (Packrat parser generátor)
- LL(*) nyelvtanok támogatása
  1. Jó hibajelző képesség
- Ecore metamodell automatikus származtatása

LL(*) nyelvtanok

- Két hete: LL(k)
  1. Balról jobbra olvasás
  2. Baloldali levezetés
  3. k karakter előretekintés
- LL(*)
  1. Balról jobbra olvasás
  2. Baloldali levezetés
  3. Korlátlan hosszú előretekintés
     - Véges automata használata

Elemzés menete

- Kiindulás
  1. Mondatszimbólum
- Lépések
  1. Szabályalkalmazások választása
     - Predict and match
     - Ehhez kell az előretekintés, hogy determinisztikus legyen
- Cél
  1. A tényleges mondatig eljutni

Probléma: balrekurzív nyelvek

```
Expression:
  Expression ‘*’ Expression |
  Expression ‘+’ Expression |
  INT;
```

- Levezetés automatikusan
  1. Expression
     -> Expression ‘+’ Expression
     -> (Expression ‘+’ Expression) ‘+’ Expression
     -> …

Ugyanez előjöhet több lépés után is!

Balrekurzív nyelvtanok

- Balrekurzív nyelvtanok
  1. Végtelen hosszú szabályalkalmazássorozat lehetséges
  2. Automatikusan az elemző nem tud választani
  3. Nem bal-elemezhetőek
- Megoldás
  1. Balrekurzió megszüntetése (left-factoring)

Balrekurzió megszüntetése

- További nem-terminálisok bevezetése
  1. Szabályok során balrekurzió elkerülése

Példa: nem balrekurzív műveletek

```
Addition:
 Multiplication ('+'
right=Multiplication)*;

Multiplication:
 NumberLiteral ('*'
right=NumberLiteral)*;

NumberLiteral:
 value=INT;
```

Mi változott még?

- Explicit műveleti sorrend!
  1. Az új nyelvtanban a szorzás magasabb precedenciájú
  2. Hozzá lehetne adni zárójelezést

Példa

```
Addition:
 Multiplication ('+' right=Multiplication)*;

Multiplication:
 Primary ('*' right=Primary)*;

Primary:
 NumberLiteral |
 '(' Addition ')';

NumberLiteral:
 value=INT;
```

Mi kell még

- Metamodell következtetés
  1. Kérdés: Miért nincs IMP-ben (LPG parser generátorban)?
- Visszaadott érték típusának megadása
- Átalakítási szabályok

Nincs metamodel inference IMP-ben, mert egy kötött formájú AST-t ad ki – mindig! Ha más kell, kézi transzformáció.

AST típusok

- Ha nem adjuk meg, egyszerűen rövidítésnek veszi:
  1. `NumberLiteral : … ;`
  2. `NumberLiteral returns NumberLiteral : …;`

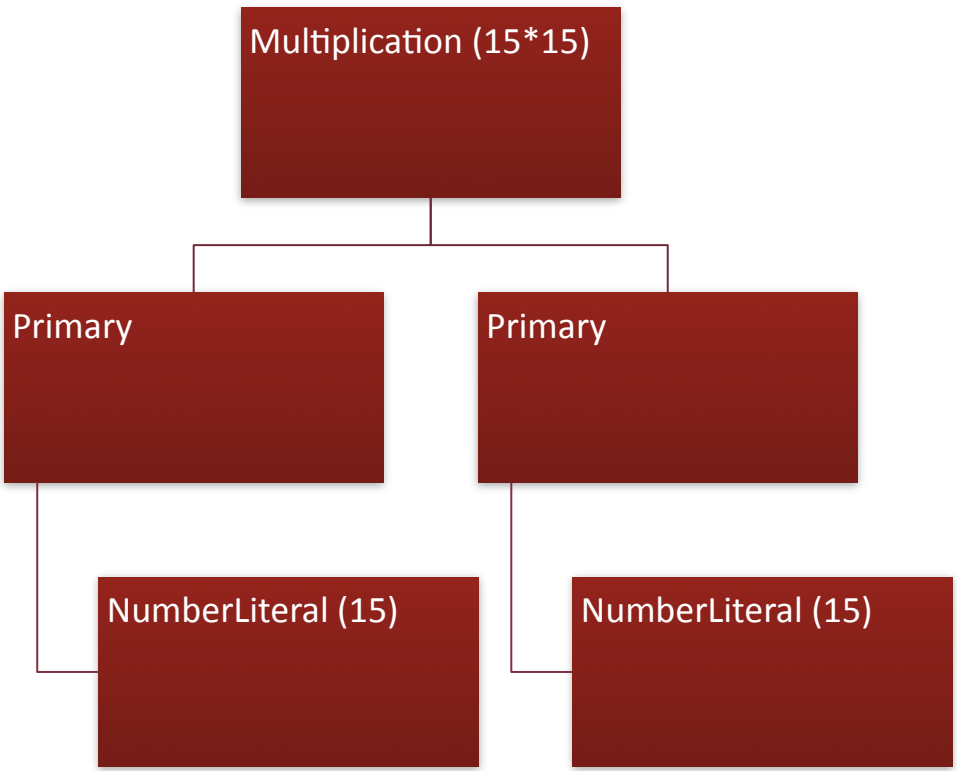
- Felhasználható arra, hogy több nyelvi elem azonos típust adjon vissza (rekurzív hierarchia)

Típusok

```
Addition returns Expression:
 Multiplication ('+' right=Multiplication)*;

Multiplication returns Expression:
 Primary ('*' right=Primary)*;

Primary returns Expression:
 NumberLiteral |
 '(' Addition ')';

NumberLiteral:
 value=INT;
```

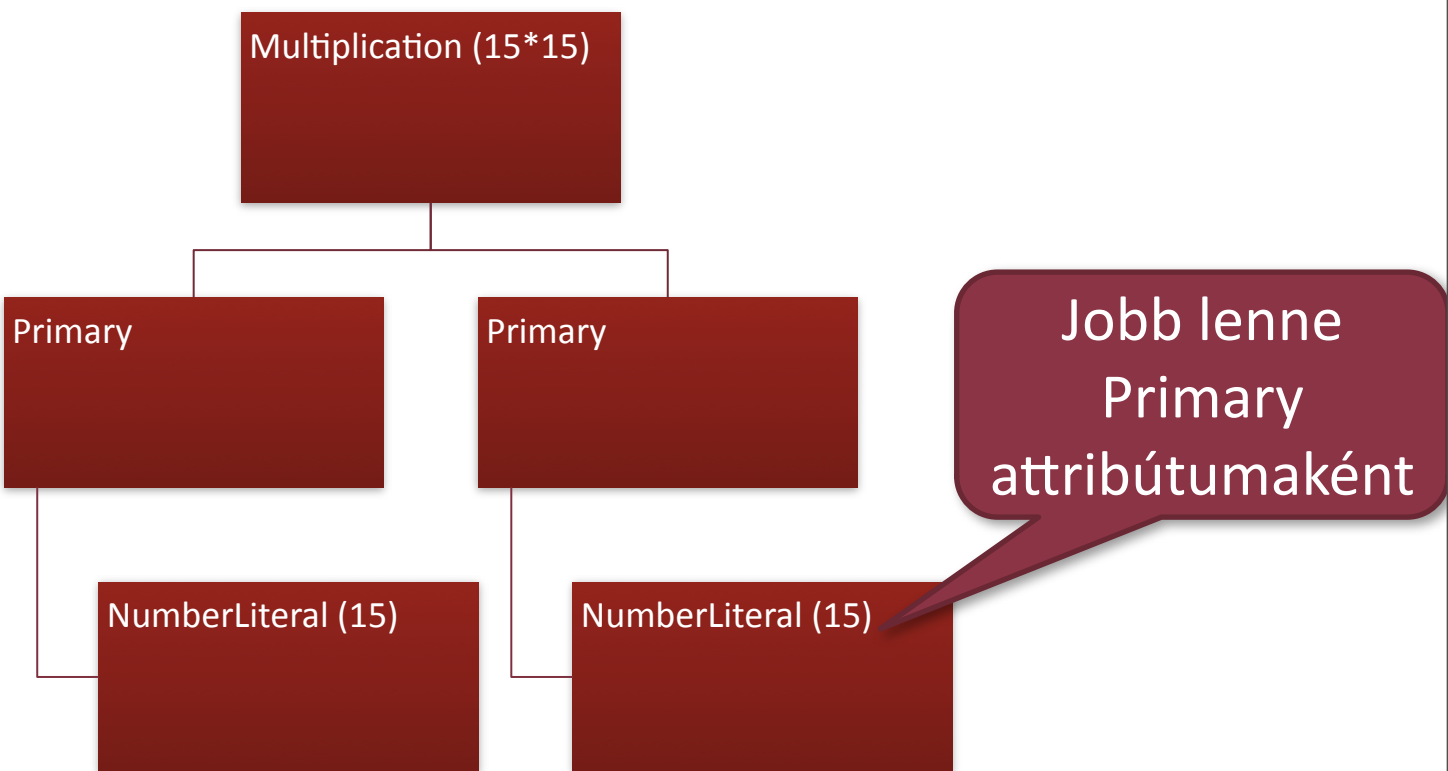
Átalakítási szabályok

- Nyelvtan helyenként nem jól olvasható
  1. Nem-rekurzív nyelvtan hoz ilyet



Átalakítási szabályok

- Nyelvtan helyenként nem jól olvasható
  1. Nem-rekurzív nyelvtan hoz ilyet

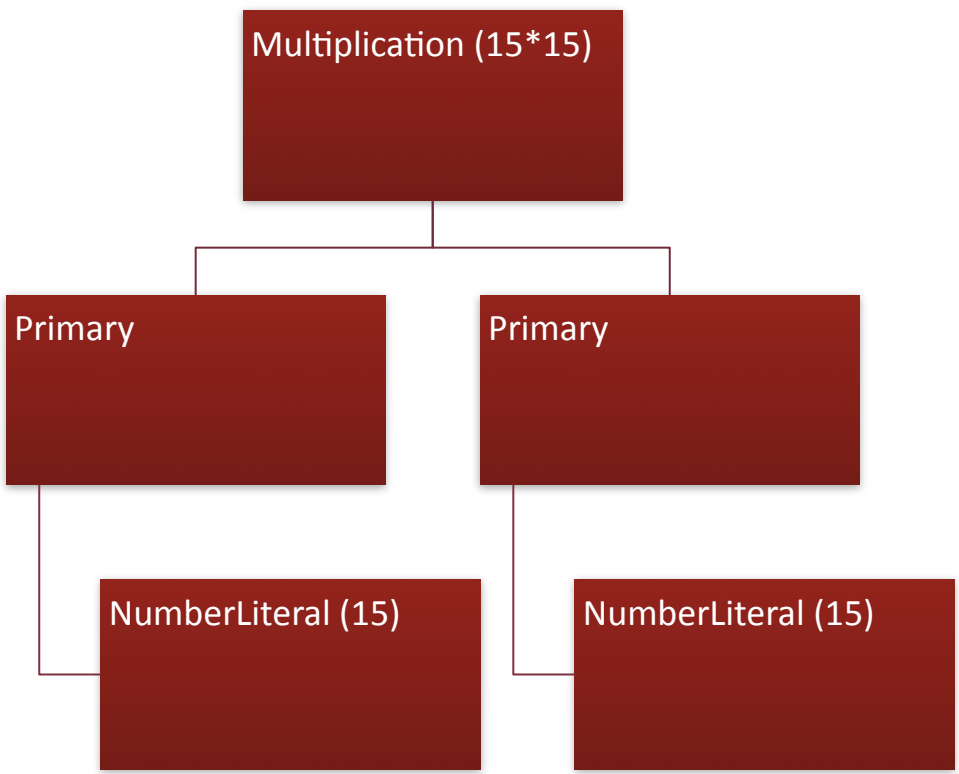


Átalakítási szabályok

- Hivatkozásokhoz szabályok illeszthetőek
  1. Metamodell következtetést befolyásolják
  2. Egyszerű átalakításokra elegendőek
  3. DE: bonyolítják a nyelvtant
- Tipikus szabály
  1. Megkapunk egy node-ot, és felhasználjuk

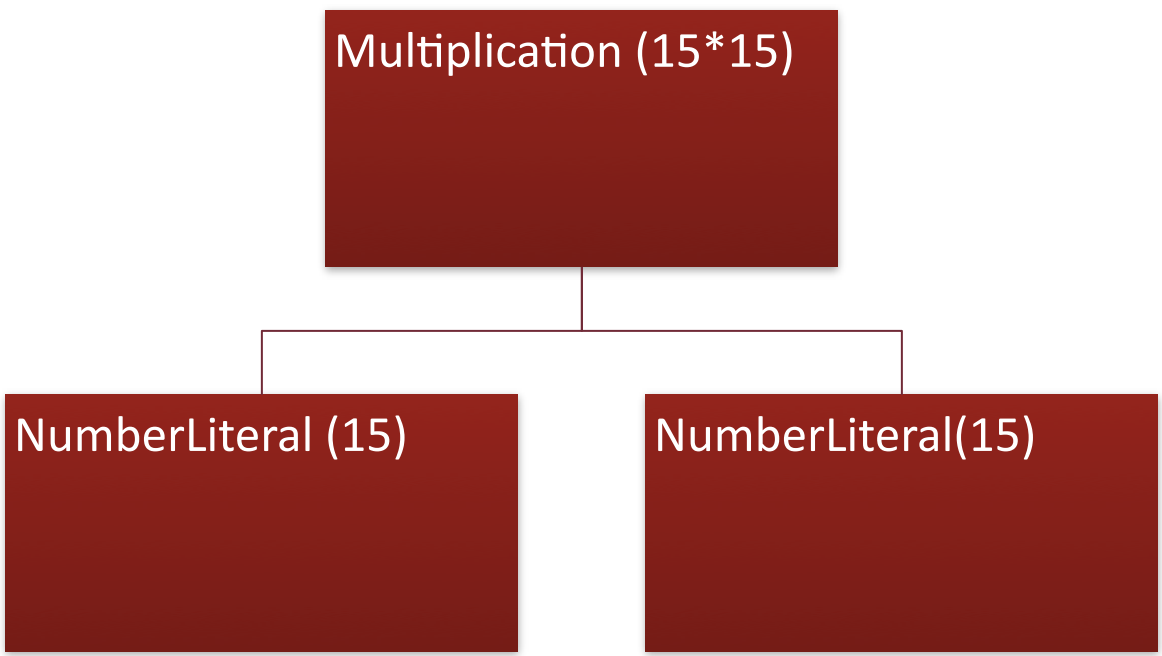
Átalakítási szabályok

```
Multiplication returns Expression:
  Primary ({Multiplication.left=current} '*'
right=Primary)*;
```



Átalakítási szabályok

```
Multiplication returns Expression:
  Primary ({Multiplication.left=current} '*'
right=Primary)*;
```



Példa

```
Addition returns Expression:
 Multiplication ({Addition.left=current} '+'
right=Multiplication)*;

Multiplication returns Expression:
 Primary ({Multiplication.left=current} '*'
right=Primary)*;

Primary returns Expression:
 NumberLiteral |
 '(' Addition ')';

NumberLiteral:
 value=INT;
```

További nyelvi elemek

- Felsorolás (EEnum)

```
enum Sex returns Sex:
      male = 'male' | female = 'female';
```

- Szabad sorrendű elemek

```
Modifier:
static?='static'? & final?='final'? &
visibility=Visibility;


enum Visibility:
PUBLIC='public' | PRIVATE='private' |
PROTECTED='protected';
```

További nyelvtan elemek

- Until token
  1. terminal ML_COMMENT : '/*' -> '*/';
- Wildcard
  1. FOO : 'f' . 'o';
- Tartomány
  1. terminal INT returns ecore::EInt: ('0'..'9')+;
- ... (dokumentáció)

Kódgenerálás

Kódgenerálás

- Van egy nyelvtanunk
  1. +metamodell információk
- Kell egy szerkesztő
  1. Kódgenerálás
     - Ecore metamodell generálás
     - Ecore generátor modellből kód generálás
     - Parser generálás
     - Editor komponensek generálása
     - …

- Bonyolult folyamat
  1. Lépésekre bontható
  2. Újabb DSL
     - Modellezési munkafolyamat leírására
     - Modeling Workflow Engine (MWE)

Az Xtext workflow

- Generátor fragmensek adhatóak meg
  1. Ecore generátor
  2. Ecore importer
  3. Antlr generátor hívása
  4. …
  5. Egyesével paraméterezhető
- Jó alapértelmezett generátort kapunk
  1. Ritkán kell hozzányúlni

Példa

```
component = Generator {
        pathRtProject = runtimeProject
        pathUiProject = "${runtimeProject}.ui"
        projectNameRt = projectName
        projectNameUi = "${projectName}.ui"
        language = {
                uri = grammarURI
                fileExtensions = file.extensions

                // Java API to access grammar elements (required
by several other fragments)
                fragment = grammarAccess.GrammarAccessFragment
{}

                // generates Java API for the generated
EPackages
                fragment = ecore.EcoreGeneratorFragment {
                // referencedGenModels = "uri to genmodel
```

Példa - Folytatás

```
                    // a custom ResourceFactory for use with EMF
                    fragment =
resourceFactory.ResourceFactoryFragment {
                            fileExtensions = file.extensions
                    }

                    // The antlr parser generator fragment.
                    fragment =
parser.antlr.XtextAntlrGeneratorFragment {
                    //  options = {
                    //              backtrack = true
                    //      }
                    }
…
```

Generátor futása után

- Kapunk egy működő szerkesztőt
  1. Forráskód színezés
  2. Outline
  3. Content assist
  4. Hivatkozásfeloldás
  5. …
- Mindenből alapértelmezett implementáció

Testreszabás

Testreszabás

- Szolgáltatásokkal bővíthető
  1. **Scoping: jobb linkfeloldáshoz**
  2. **Formatting: automatikus megjelenítés**
  3. **Validáció: ellenőrzés**
  4. Content Assist: precízebb segítség
  5. Labeling: feliratozás (JFace Label Provider)
  6. Outline: Outline view testreszabása
  7. Quick Fix: javítások kezelése

Scoping

- Feladat:
  1. Változóknak van elérhetősége (scope)
  2. Ez nyelvfüggő
  3. Alapértelmezés: csak lokális referenciák elérhetőek!

Scoping szolgáltatás

- Feladat
  1. Referencia és hivatkozható elemek összekapcsolása
- Implementáció
  1. Többféle megközelítés támogatott
     - Explicit import
       - EMF resource-ok kijelölése
     - Lokális scope
       - Fájlokon belüli scope megadására
     - **Deklaratív**
       - **Metódusnév alapján történő szűrés**
     - …

Deklaratív scoping

- AbstractDeclarativeScopeProvider osztály
  1. Kétféle generikus metódus
     - `IScope scope_<RefDeclaringEClass>_<Reference>(ContextType> ctx, EReference ref)`
     - `IScope scope_<TypeToReturn>(<ContextType> ctx, EReference ref)`
  2. Megfelelő metódusok felvehetőek
     - Hasonló az LPG visitorokhoz
     - De nem statikus összerendelés (ősosztályban nincsenek meg a megfelelő metódusok!)

Deklaratív scoping

- Metódusok
  1. Gyakorlatilag felsorolják az elérhető elemeket
  2. Paraméterek:
     - Hivatkozás forrása (kontextus, ill. deklaráló elem)
     - Hivatkozás típusa (EClass)

Validáció

- Feladat
  1. További feltételek ellenőrzése a beolvasott modellen
  2. Megközelítés
     - Tetszőleges EMF alapú eszköz működik (pl. OCL)
     - Xtext-specifikus Java-kódon alapuló validátor

Java validátor

- Vázat generál egyszerű mintával
  1. Minden ellenőrzési metódushoz @Check annotáció
  2. Metódus paramétere egy elem az AST-ből
- Hiba esetén az ősosztály error/warning metódusát kell meghívni

Validátor példa

```
@Check
public void noNameCollision(Community entity) {
  noNameCollision(entity, entity.eContainer().eContents
(), SocialNetworkPackage.Literals.SOCIAL_ENTITY__NAME);
}


private void noNameCollision(EObject eObject,
List<EObject> siblings, EStructuralFeature nameFeature)
{
  String name = (String) eObject.eGet(nameFeature);
  for (EObject sibling : siblings) {
                  if(name.equals(sibling.eGet
(nameFeature)) && eObject != sibling) {
                          error("Duplicate name",
nameFeature.getFeatureID());
```

Formatter

- Cél:
  1. Kód automatikus formázása
     - Nem látható karakterek illesztése
- Mikor kell
  1. Felhasználó kéri
  2. EMF modell dinamikus módosításakor

Formatter

- Szabályok
  1. Mire vonatkozik?
  2. Mit hajt végre?
- Ez Java utasításokkal megadható

Mire vonatkozik

- after(token): token után
- before(token): token előtt
- around(token) ⇔ after(token) és before(token)
- between(token1, token2): token1-et követő token2
- bounds(token1, token2) ⇔ after(token1) és before (token2)
- range(token1, token2): token1-től token2-ig

Utasítások

- setIndentationIncrement
- setIndentationDecrement
- setLinewrap
- setSpace
- setNoSpace

Formázás példa

```
protected void configureFormatting(FormattingConfig c) {
  SocialNetworkGrammarAccess access =
(SocialNetworkGrammarAccess) getGrammarAccess();

  SocialNetworkElements sne = access.getSocialNetworkAccess();
  c.setLinewrap(1, 1, 1).after(sne.getLeftCurlyBracketKeyword_2());
  c.setLinewrap(1, 1, 1).before
        (sne.getRightCurlyBracketKeyword_5());
  c.setLinewrap(1, 1, 1).after(sne.getEntitiesAssignment_3());
  c.setLinewrap(2, 2, 2).before
        (sne.getAcquaintancesAcquaintanceParserRuleCall_4_0());
  c.setIndentationIncrement().after
        (sne.getLeftCurlyBracketKeyword_2());
  c.setIndentationDecrement().before
```

EMF modellek kezelése

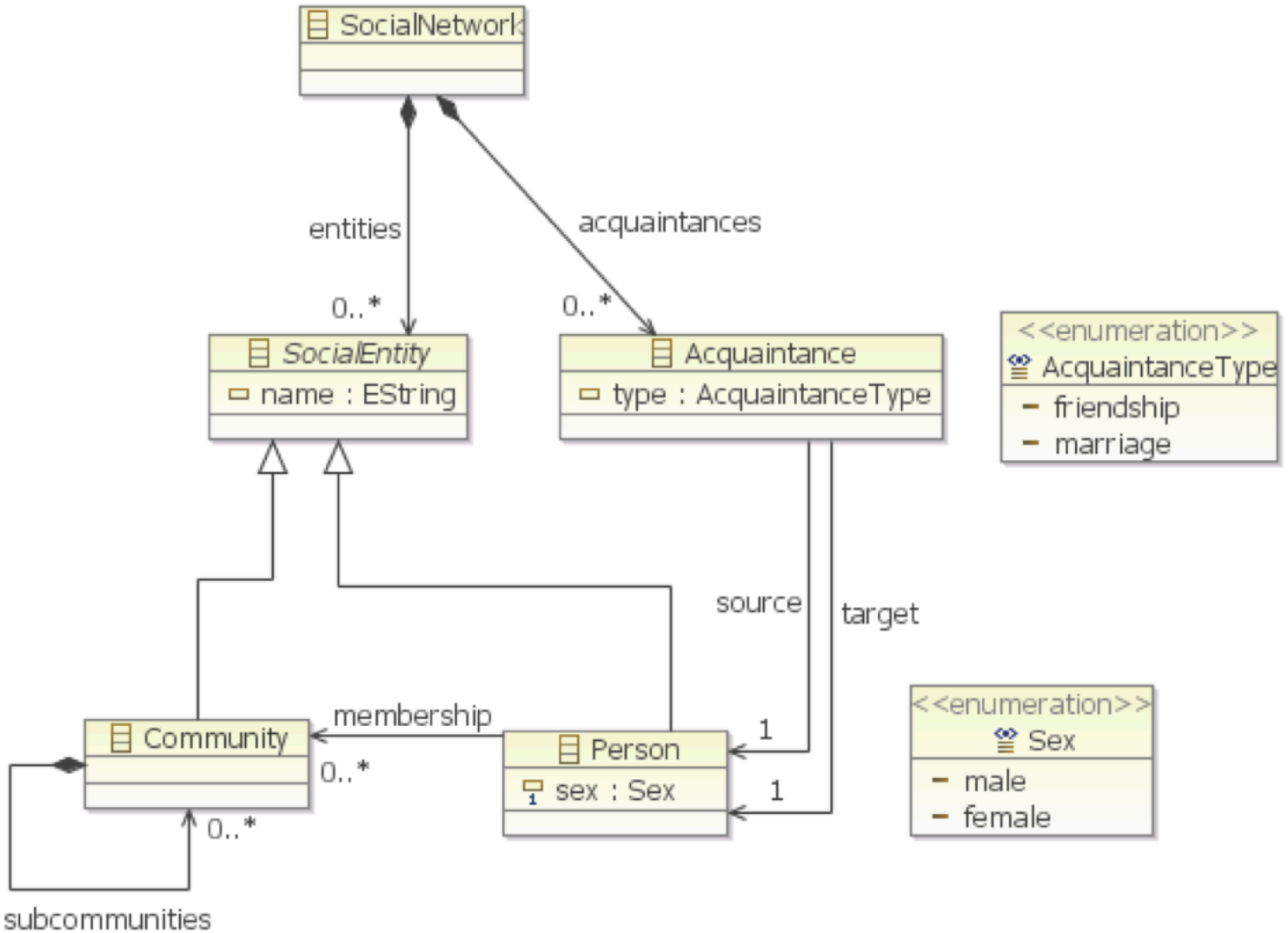
EMF modellek felhasználása

- Kétféle felhasználás
  1. Létező Ecore modellhez szöveges szintaxis
  2. Hivatkozás létező elemekre, de új modell

Létező Ecore modell

- Varázsló
  1. Importálja a modellt
  2. Generál egy nyelvtant hozzá
     - Viszonylag bőbeszédű szöveget vár
     - De kiindulásnak jó

Példa: Kapcsolati háló



Példa: Nyelvtan

```
grammar hu.bme.mit.demo.socialnetwork with
org.eclipse.xtext.common.Terminals

import "platform:/resource/hu.bme.mit.demo.socialnetwork/src-gen/hu/bme/
mit/demo/SocialNetwork.ecore"

import "http://www.eclipse.org/emf/2002/Ecore" as ecore

SocialNetwork returns SocialNetwork:
        {SocialNetwork}
        'SocialNetwork'
        '{'
                ('entities' '{' entities+=SocialEntity ( "," entities
+=SocialEntity)* '}' )?
                ('acquaintances' '{' acquaintances+=Acquaintance ( ","
acquaintances+=Acquaintance)* '}' )?
    '}';



SocialEntity returns SocialEntity:
        SocialEntity_Impl | Person | Community;
```

Példa: Konkrét szintaxis

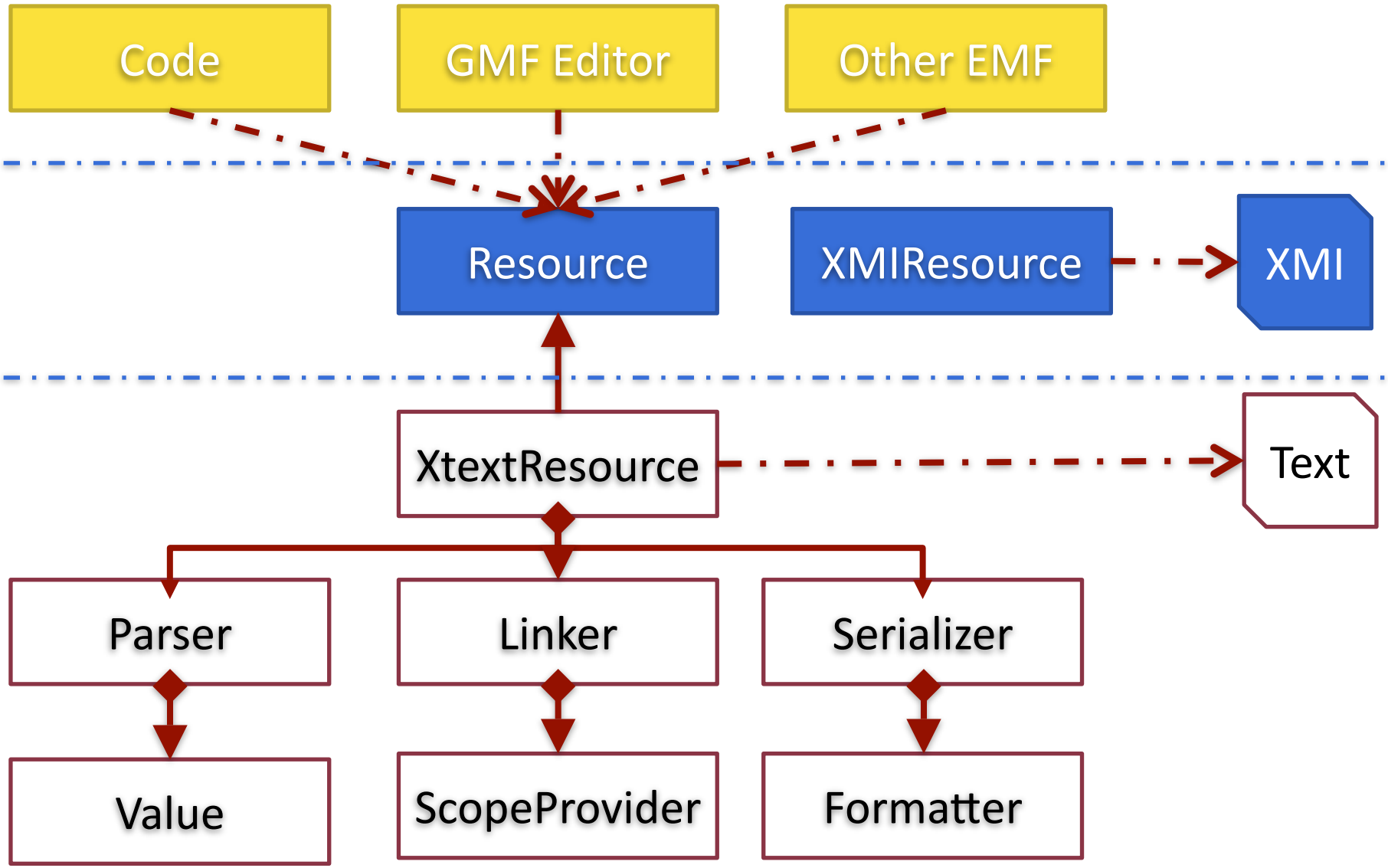
```
SocialNetwork {
        entities {
                Community community1 {
                        children {
                                Community subcommunity11 {
                                },
                                Community subcommunity12 {
                                }
                        }
                },
                Person person1 {
                        membership ( subcommunity11 )
                },
                Person person2 {
                        sex male membership ( subcommunity12 )
                }
        }
        acquaintances {
```

Hivatkozás létező Ecore modellekre

- Két adat kell
  1. Ecore fájl
     - import direktíva nyelvtanban
  2. Generátor modell
     - MWE ne generáljon
     - Ecore generátor fragmens
       - Ne tartalmazza az importált csomagokat

Integráció EMF technológiákkal

Integráció az EMF technológiákkal



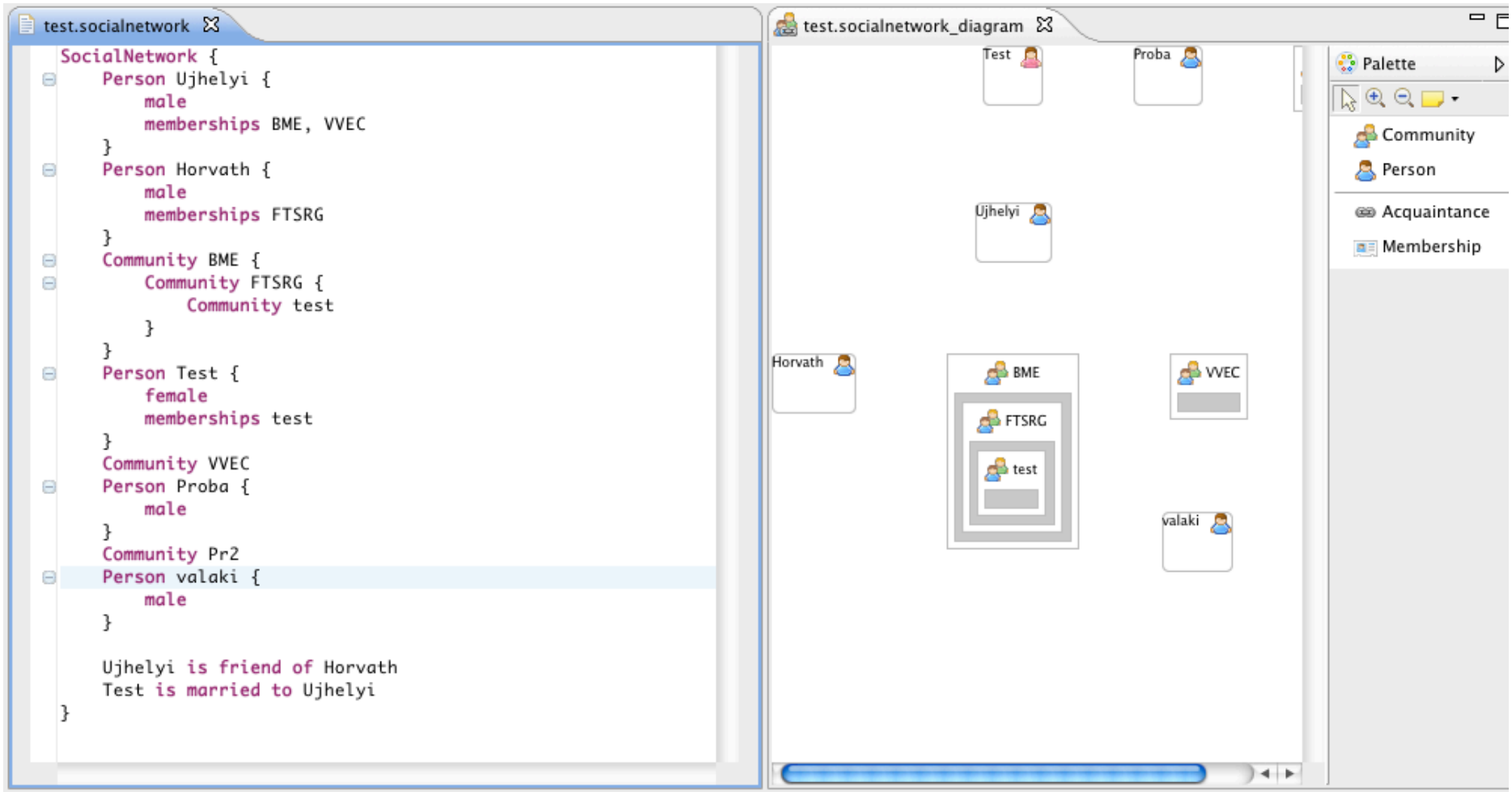
Integráció

- Xtext
  1. Saját EMF resource
  2. Ezen keresztül kapcsolódik
     - Nem kell foglalkozni vele, hogy ki sorosítja

GMF editor

- Domain modell beállítható Xtext erőforrásra
  1. Mentéskor automatikus szinkronizáció
- Oda kell figyelni a sorosíthatóságra!
  1. GMF nem tud a nyelvtanról
  2. Nem kell túl sok mindenre figyelni

GMF és Xtext editor



Kódgenerálás

- Generator projekt
- Xtend alapú kódgenerátor
- Adatok
  1. Be: EMF modell
  2. Ki: Kód
- Vezérlés: MWE

Összefoglalás

- Kiforrott technológia
  1. Jó alapértelmezések
  2. Kényelmes felhasználás
- Probléma
  1. Korlátozott metamodell következtetés
  2. Balelemzőt kell használni