Modellalapú szoftverfejlesztés

XIII. előadás

Kitekintés

Dr. Mezei Gergely

Optimalizálás, obfuszkálás, kódgenerálás

I. Összefoglalás

II. Nyitott kérdések

A mai előadás

I. fejezet Miért?

II. fejezet Miről?

III. fejezet Hogyan?

1

Szöveges modellezés

Fordítóprogramok,
Nyelvfeldolgozás lépései.
Kódgenerálás,
Interpreterek

2

Grafikus modellezés

Szerkezet + megjelenítés,
Blockly, UML Profile,
Metamodellezés,
Szemantika

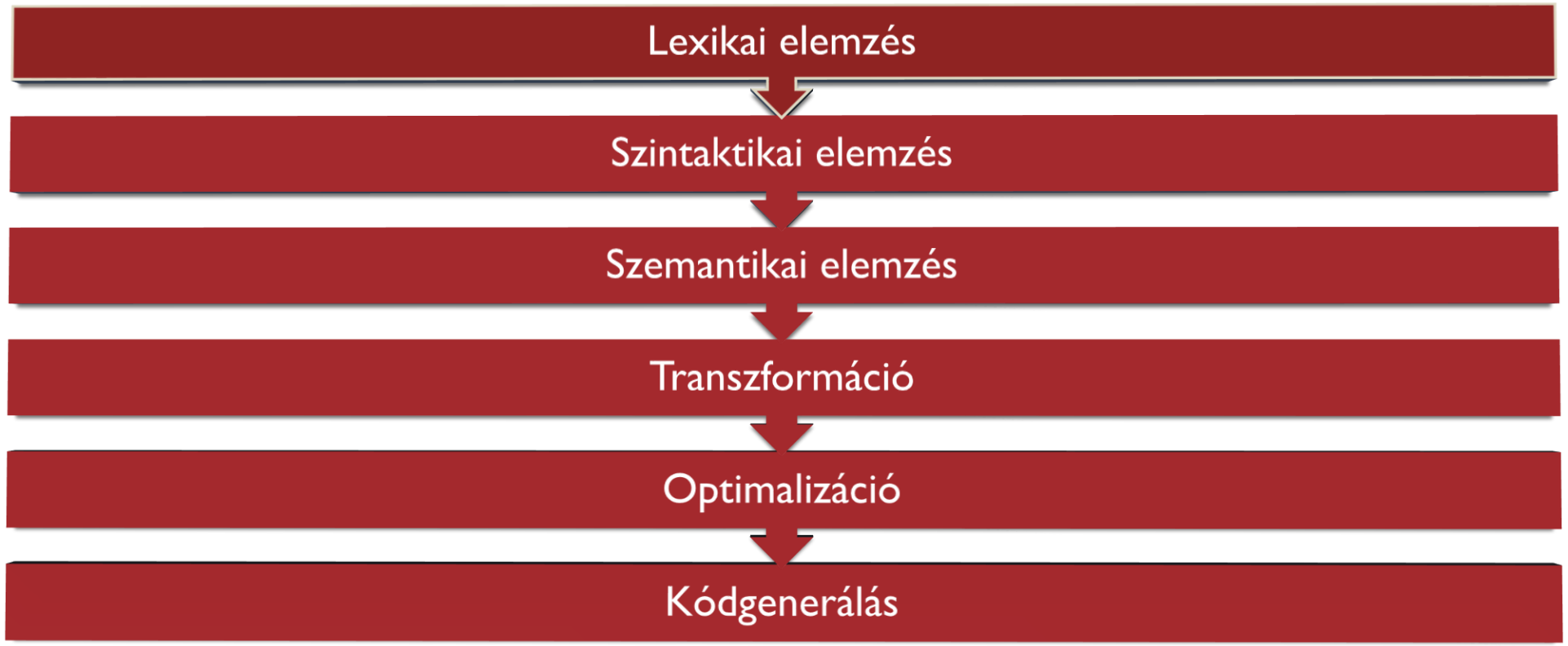
3

Modellfeldolgozás

Modellfeldolgozás,
Kódgenerálás,
Gráftranszformáció,
Modellalapú fejlesztés

MIRŐL VOLT SZÓ?

SZÖVEGES MODELLEZÉS



SZÖVEGES MODELLEZÉS

GRAFIKUS MODELLEZÉS

- Szöveges vs grafikus nyelvek
- Absztrakt szintaxis
  - UML Profile
  - Blockly
  - Metamodellezés – MOF
  - OCL
- Konkrét szintaxis
- Editorok
- Szemantika

GRAFIKUS MODELLEZÉS

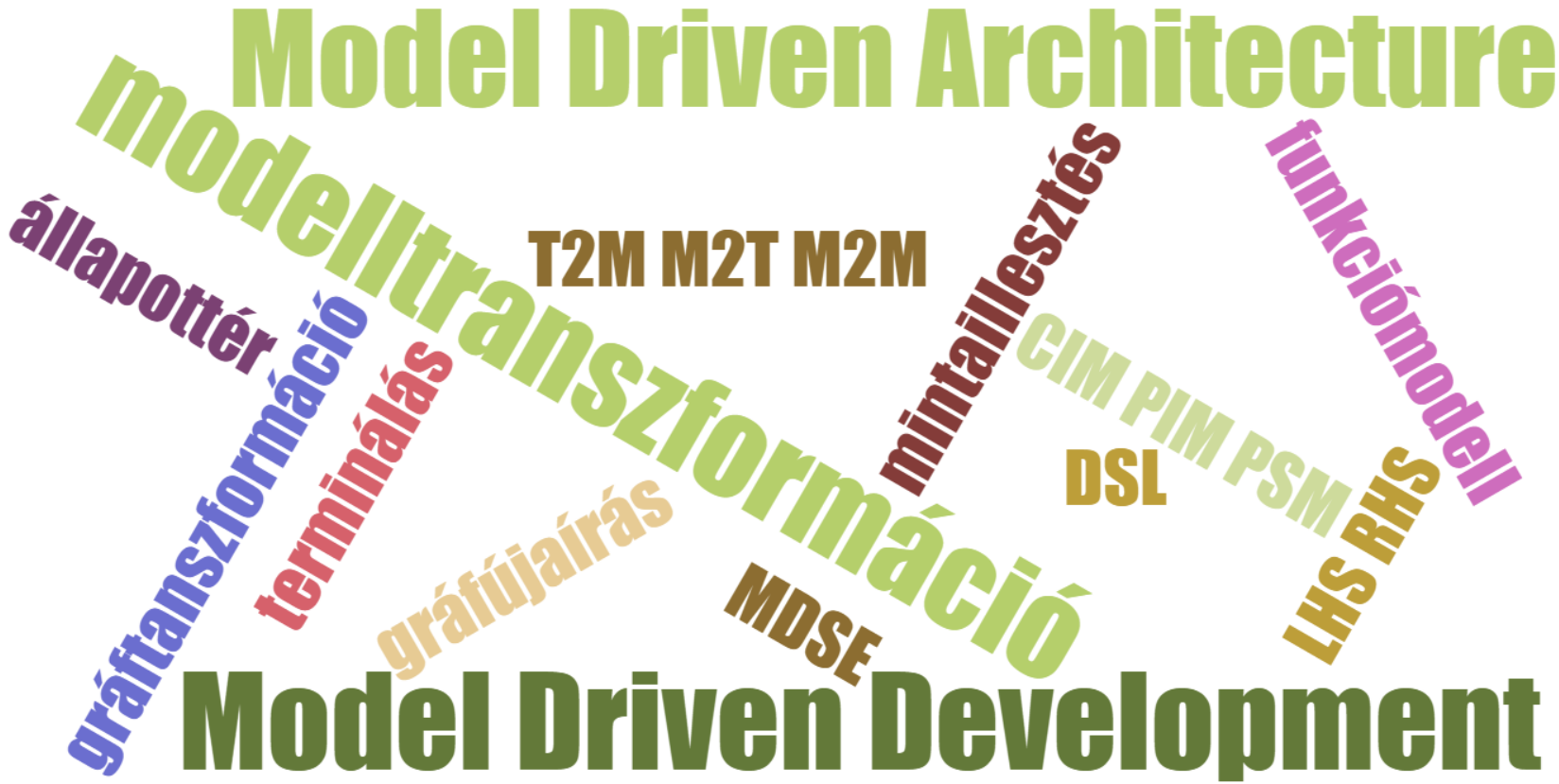
MODELLFELDOLGOZÁS

- M2M, T2M, M2T
- Traceability kezelés
- Vezérlés - szabályok, prioritás, control flow
- Imperatív módszerek (XSLT, Apache Velocity, Acceleo, Xtend, T4)
- Deklaratív: gráfminták (bal oldal, jobb oldal, mintaillesztés)
- Inkrementalitás gráftranszformációkban

MODELLALAPÚ MEGKÖZELÍTÉSEK

- MBE – MDA – MDD
- MDA: CIM – PIM – PSM
- Y modell (modellvezérelt tervezés)
- Funkciómodellezés
- Generatív programozás

MODELLFELDOLGOZÁS



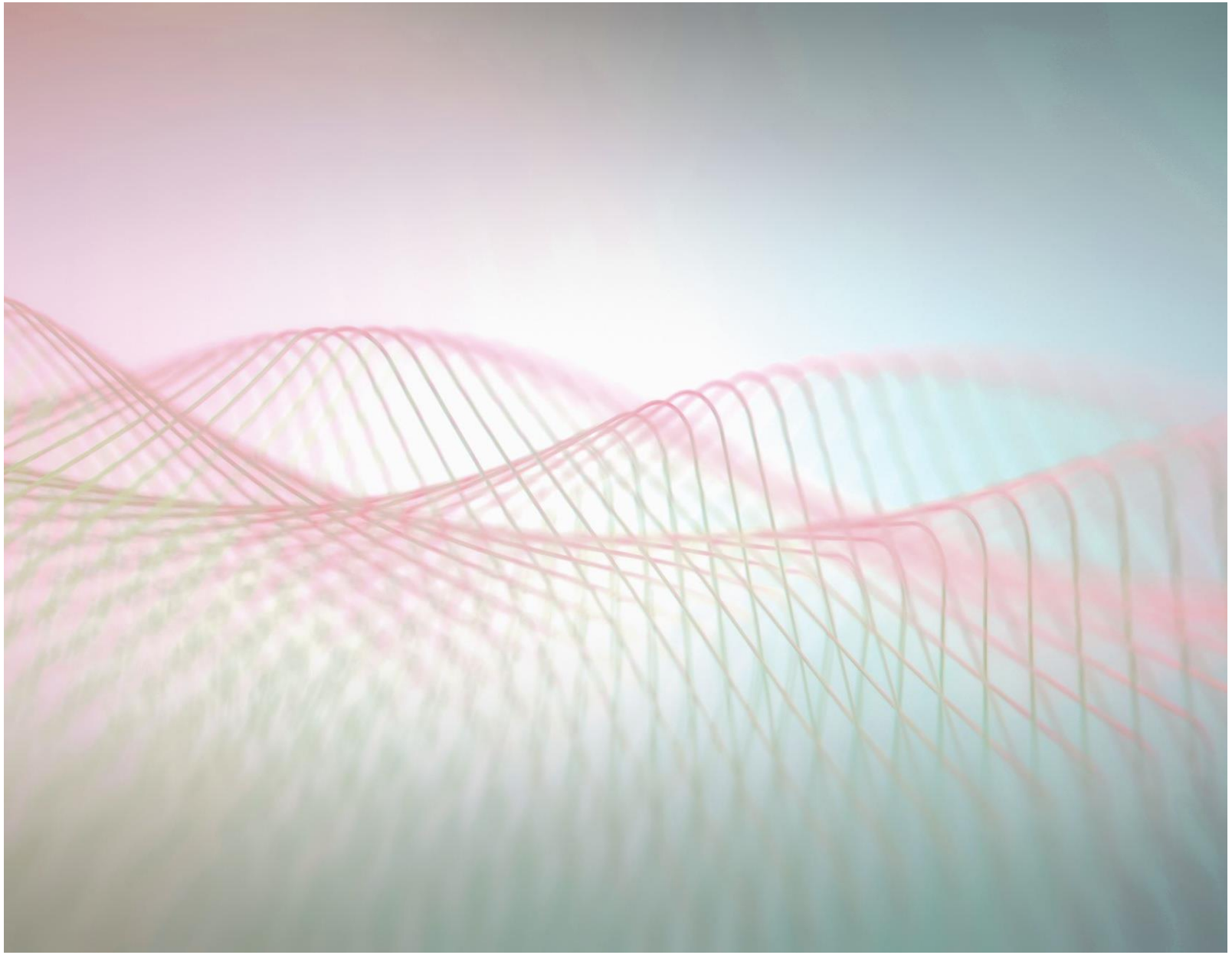
OPTIMALIZÁLÁS, OBFUSZKÁLÁS, KÓDGENERÁLÁS

I. Összefoglalás

II. Nyitott kérdések

NYITOTT KÉRDÉSEK

- Régebbi témák
  - Nagy modellek kezelése – modellek hatékony particionálása
  - Nagy modellek feldolgozása – GPGPU alapú párhuzamosítás
  - Szemantikus web feldolgozása metamodell alapon
- Aktuális témák
  - Dinamikus, n-szintű metamodellezés
  - MPS-alapú DSL fejlesztés
  - MetaDSLx - .NET alapú DSL fejlesztőrendszer
  - Modellgenerálás



TÖBBSZINTŰ DINAMIKUS ÖNLEÍRÓ MODELLEZÉS

TÖBBSZINTŰ METAMODELLEZÉS

- Többszintű metamodellezés:
  a koncepciók evolúciója
  - Kezdetben van *valami*…
  - … amit több lépésben finomítunk….
  - … hogy aztán megkapjuk a konkrét *terméket*

Az én Black Thunderem
Ár: 260.000Ft

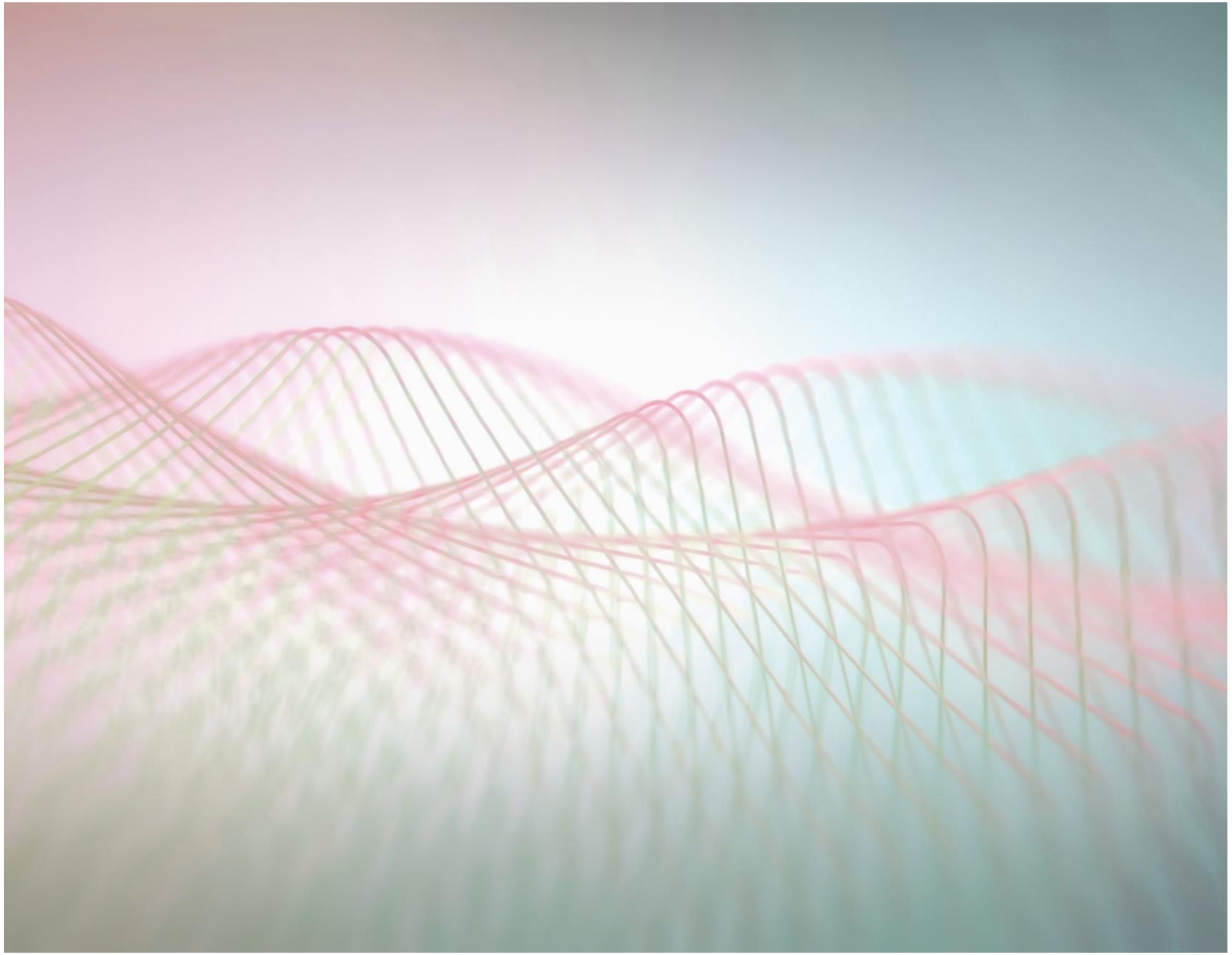
TÖBBSZINTŰ METAMODELLEZÉS

- Gyártási folyamat
  - Prototípusokon keresztül, sok lépésben
  - Lépésről lépésre finomítjuk a koncepciókat
    - Egy rendszerben: absztrakt és konkrét komponensek
  - "Élő" szakterületi nyelvek, agilis jellegű hozzáállás
    - Verziókövetés helyett példányosítása mentén elágazás
  - Validált finomítás

DINAMIKUS ÉS ÖNLEÍRÓ MŰKÖDÉS

- Modellezett műveletek
  - Maga a validációs logika is műveletekből áll!
  - A Bootstrapben írjuk le, mi a helyes működés
  - Teljes flexibilitás a modellezési paradigmában

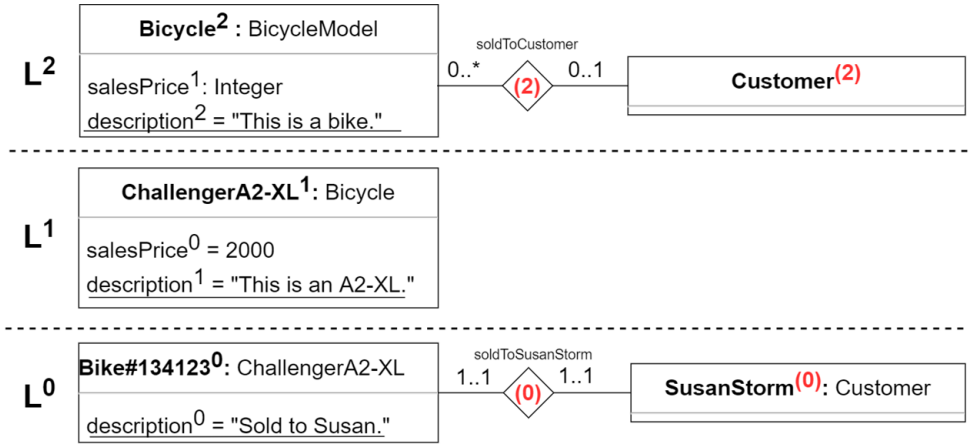
    (Pl. típusrendszer megadása)
  - Más modellező rendszereket is tud szimulálni
- Művelet = adat
  - Módosítható műveletek által
  - Önjavító kód? (Skynet?)

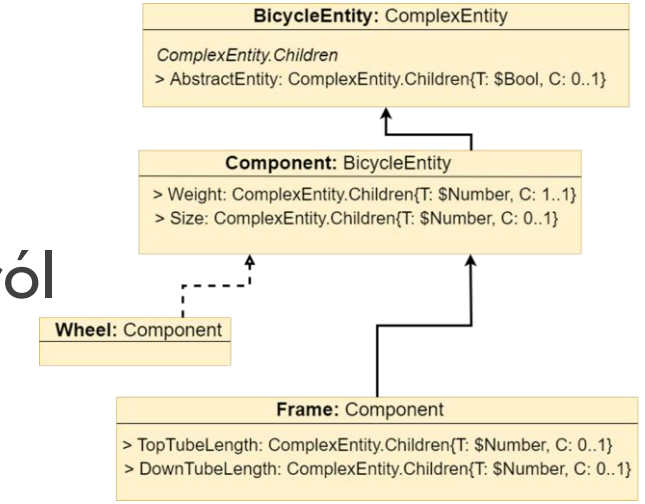


TÖBBSZINTŰ MODELLEZÉS EGYSÉGES LEÍRÁSA

TÖBBSZINTŰ MODELLEZÉS - SOKFÉLE MEGKÖZELÍTÉS

- Többszintű metamodellezés
  - Sokféle megközelítés létezik
  - Sokszor az alap fogalmakban sincs megegyezés
    - pl. mi az, hogy field, type, meta, stb.
- Cél
  - Szerkezet és szemantika egységes leírása
  - Egységes formalizmus, egységes leíró nyelv
- Egységes alapokon könnyebb objektíven beszélni fogalmakról
  - Különböző megközelítések objektívebb összehasonlítása





TÖBBSZINTŰ MODELLEZÉS - EGYSÉGES LEÍRÁS

- Hogyan tudjuk leírni a többszintű modelleket egységesen?
  - Szerkezet – a modell struktúrája
    - pl. címkézett irányított gráf segítségével
  - Szemantika – a modell jelentése, jólformáltsága
    - pl. logikai formulák segítségével
- Hogyan tudjuk ezt a gyakorlatban használni?
  - Szakterületi nyelv kidolgozása...
  - … a szerkezet és szemantika leírására
  - Gyakorlat-orientált szintaxis

TÖBBSZINTŰ MODELLEZÉS - EGYSÉGES LEÍRÁS (PÉLDÁK)

```
ModelElement: undef {}

Node: ModelElement {
    fields: Field[];
    isAbstract: bool;
}

Field: undef {
    value: base;
    type: base;
}
```

```
Edge: ModelElement {
  source: Node;
  target: Node;
  sourceMin: number;
  sourceMax: number;
  targetMin: number;
  targetMax: number;
}

Inheritance: Edge { }
```

Szerkezet

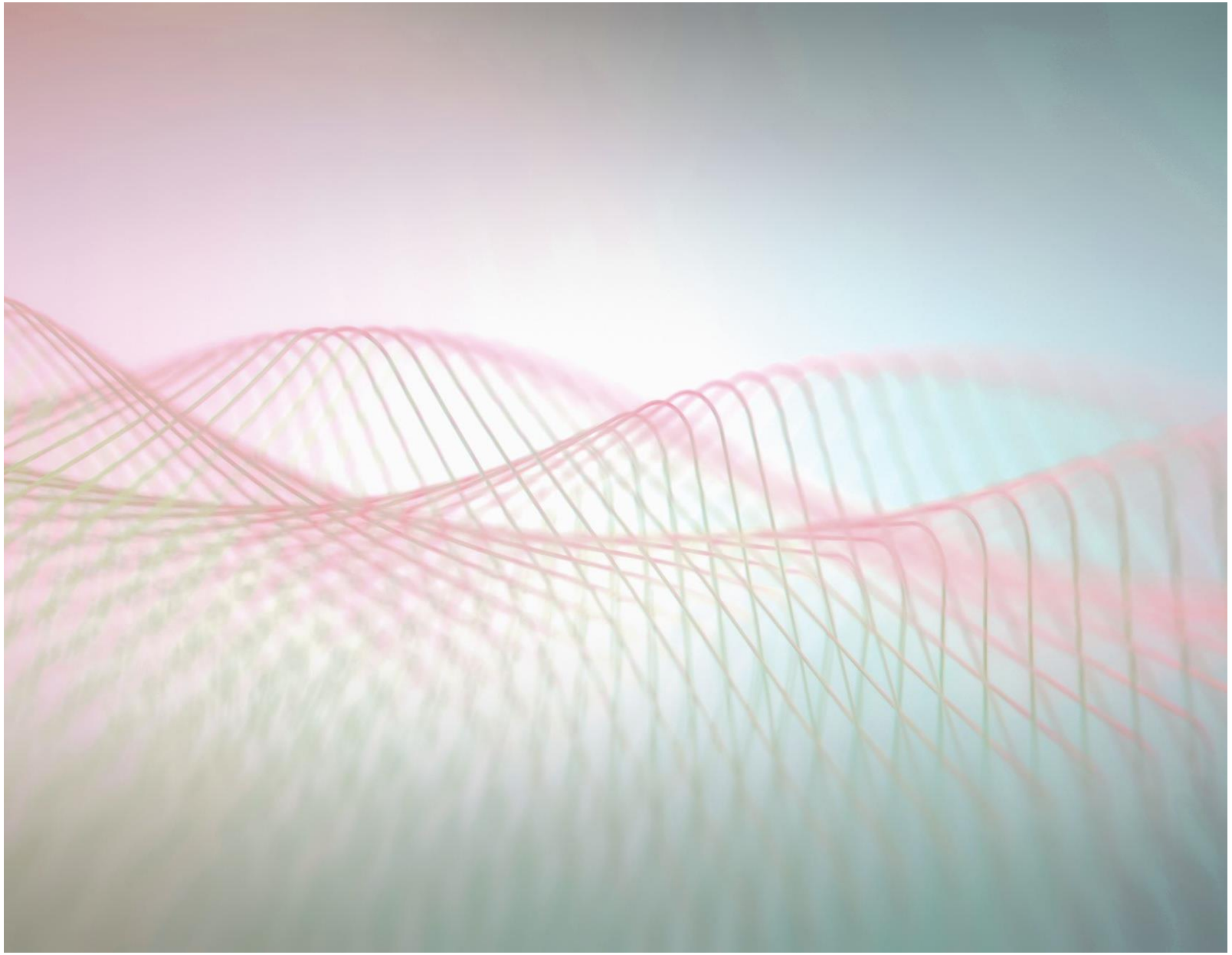
Szemantika

```
forAll node as Node:
  forAll edge as Edge:
   sum(edge2.sourceMin) >= edge.meta.sourceMin and
   sum(edge2.sourceMax) <= edge.meta.sourceMax
  where edge.meta=edge2.meta and edge2.source=node
where edge.source=node;
```

TÖBBSZINTŰ MODELLEZÉS - EGYSÉGES LEÍRÁS (PÉLDÁK)

```
NamedNode: Node {
 @potency = 0;
 @level = 2;
 isAbstract = true;
 fields {
  name: Field {
   @potency = 2;
   @level = 2;
   @nature = dual;
   type = string;
   }
  }
}
```

```
PT_NN_Inh: Inheritance {
 @potency = 0;
 @level = 2;
 source = ProductType;
 target = NamedNode;

 sourceMin = 1;
 sourceMax = 1;
 targetMin = 1;
 targetMax = 1;
}
```



KÓD- MODELL SZINKRONIZÁCIÓ MPS-BEN

KÓD-MODELL SZINKRONIZÁCIÓ MPS-BEN

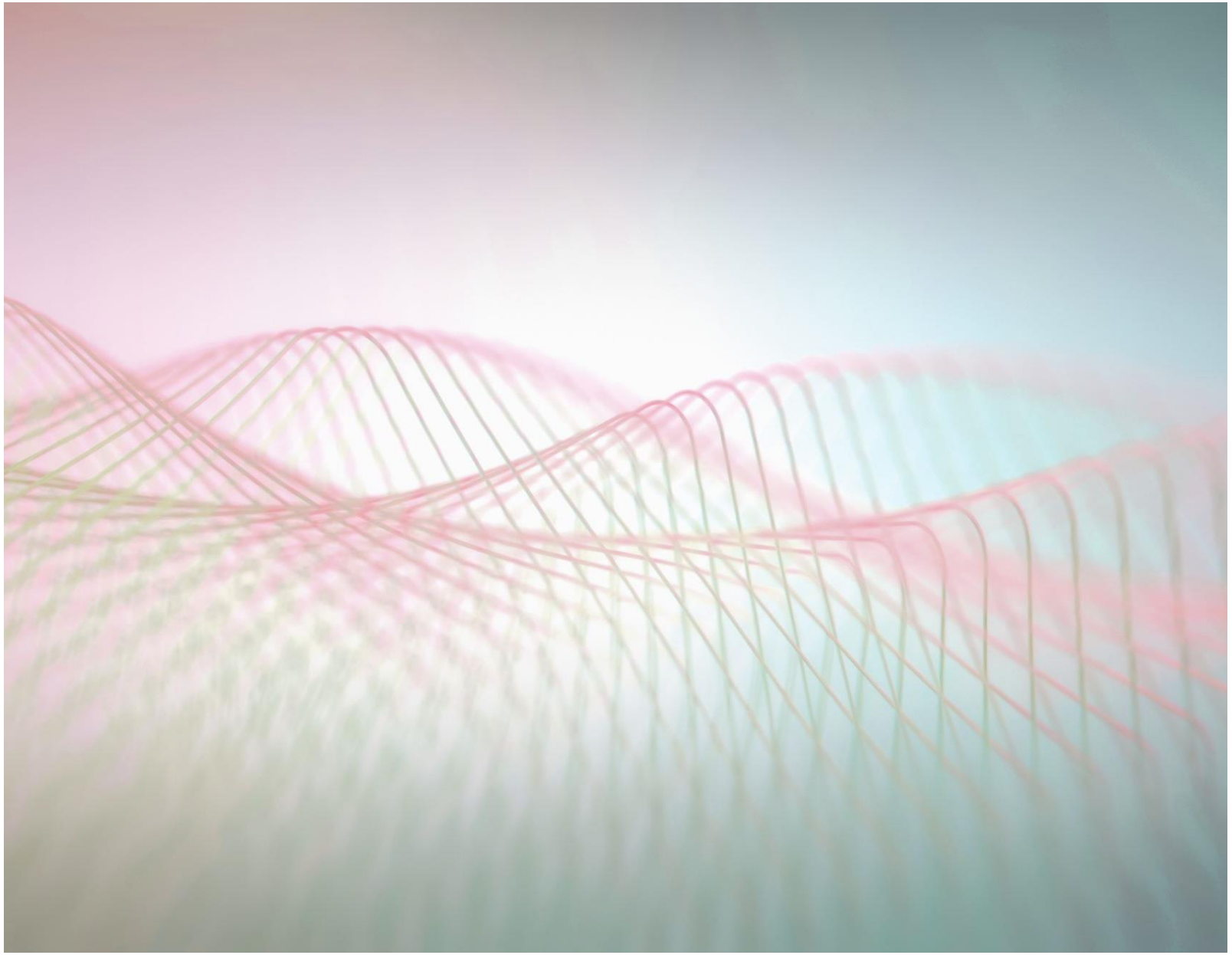
- Az MPS sajátossága a projekcionális szerkesztés
  - Előnyei:
    - Nincs szintaktikai hiba
    - Teljes autocompletion és syntax highlight
    - Nyelvkompozíció
    - Magas szintű programmodell
  - Hátrányai:
    - Másképpen kell benne programozni, mint egy szöveges szerkesztőben
    - Szövegesen tárolt forráskódot hogyan lehet átemelni MPS-be…?

KÓD-MODELL SZINKRONIZÁCIÓ MPS-BEN

- Szövegesen tárolt forráskódot hogyan lehet átemelni MPS-be…?
  - Jelenleg kézzel újra be kell vinni a kódot, emiatt:
    - Meglévő projektbe nehezen integrálható
    - MPS-el generált kódot nem lehet visszaolvasni később

KÓD-MODELL SZINKRONIZÁCIÓ MPS-BEN

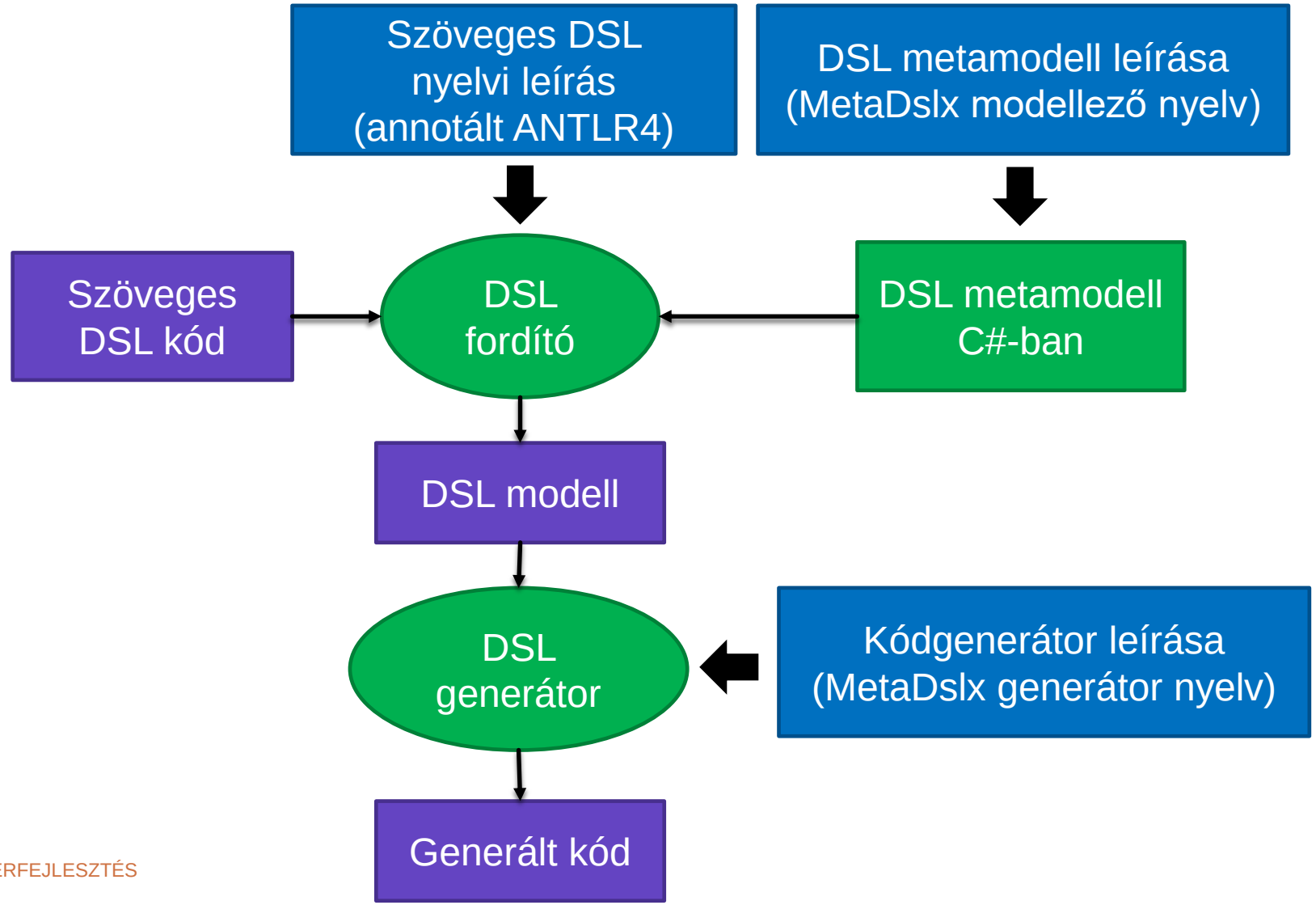
- A kód-modell szinkronizációnak nincsen elméleti akadálya
  - Egy hagyományos parserrel (pl. ANTLR) előállítható MPS modell
  - DSL kódból és generált forráskódból is
- Szinkronizációs probléma: ha változik az MPS-DSL, akkor hogyan követi az ANTLR nyelvtan?
- Megoldás:
  - MPS-DSL struktúrából ANTLR nyelvtan generálása (?)



METADSLX: DSL FEJLESZTÉSI KERETRENDSZER

DSL támogatás .NET alapon

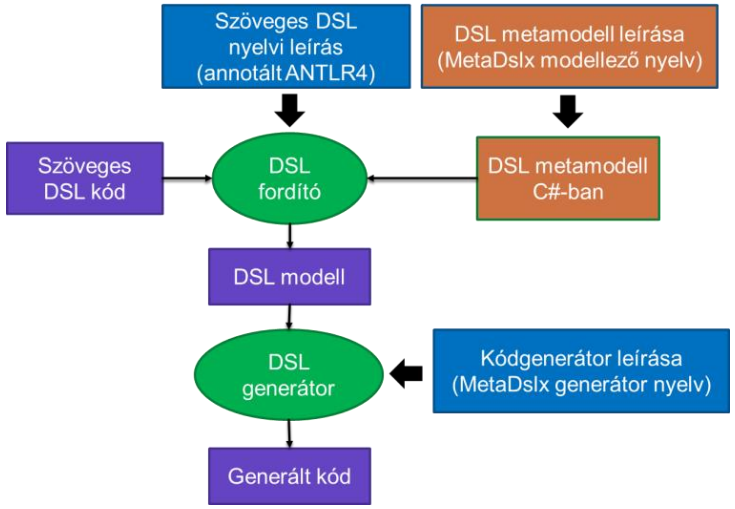
MetaDslx keretrendszer

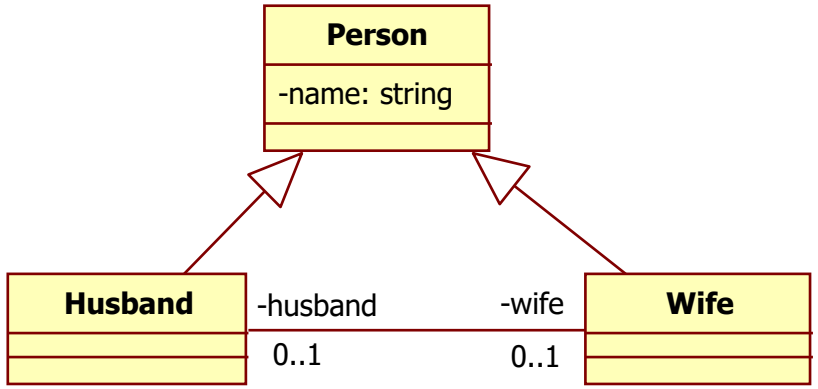


DSL metamodell definiálása (~ Xcore)

```
namespace Sample.Namespace
{
    metamodel MyLanguage(Uri= "http://example.org/mylang/1.0");

    abstract class Person
    {
        [Name]
        string Name;
    }

    class Husband : Person
    {
        Wife Wife;
    }

    class Wife : Person
    {
        Husband Husband;
    }

    association Husband.Wife with Wife.Husband;
}
```

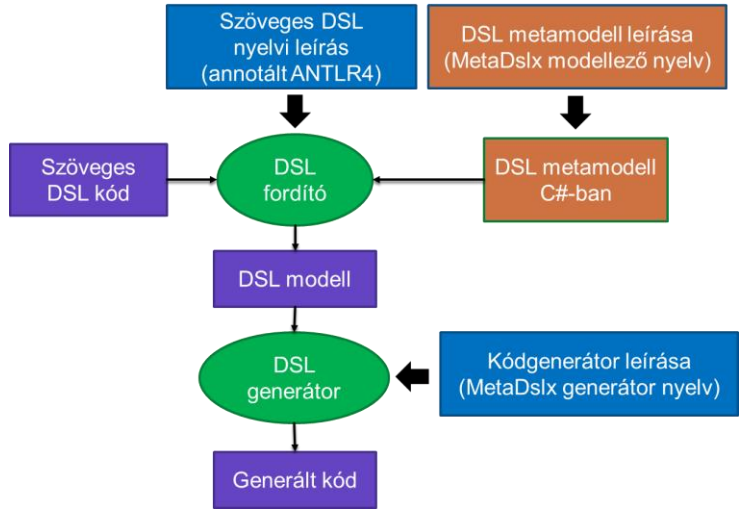
Szemantikai annotáció: később, a fordító számára lesz hasznos

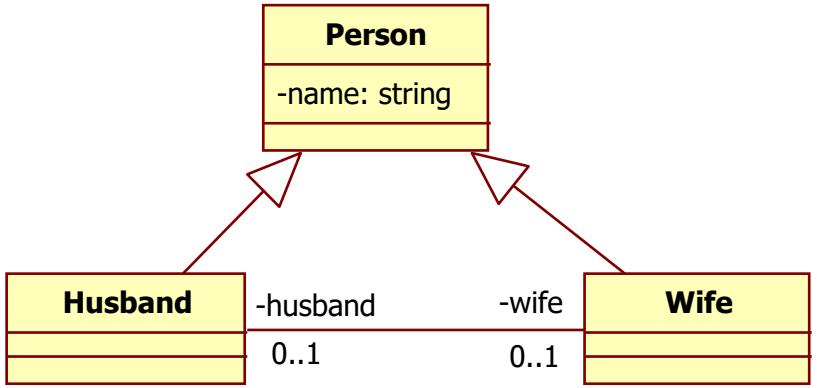




Modell használata C#-ból

```
// Módosítható modell felépítése a megszokott mutable szintaxissal:
MutableModel model = new MutableModel();
MyLanguageFactory factory = new MyLanguageFactory(model);

// Objektumok létrehozása:
HusbandBuilder husband = factory.Husband();
husband.Name = "Joe";
WifeBuilder wife = factory.Wife();
husband.Wife = wife; // Automatikusan beállítja: wife.Husband = husband;

// ----------------------------------------

// Konvertálás immutable modellé:
ImmutableModel imodel = model.ToImmutable();

// Immutable objektumok megszerzése:
Husband ihusband = husband.ToImmutable();
Wife iwife = wife.ToImmutable();

// Az alábbi feltételek teljesülnek:
// ihusband.Wife == iwife
// iwife.Husband == ihusband
```

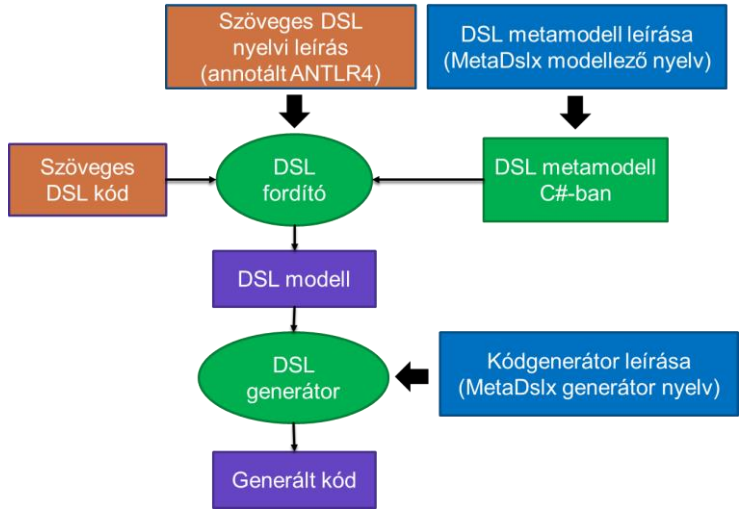




ANTLR szabályok – a konkrét szintax (~Xtext)

Szöveges DSL példa:

```
family Smith
{
    husband MrSmith;
    wife MrsSmith;
}
```

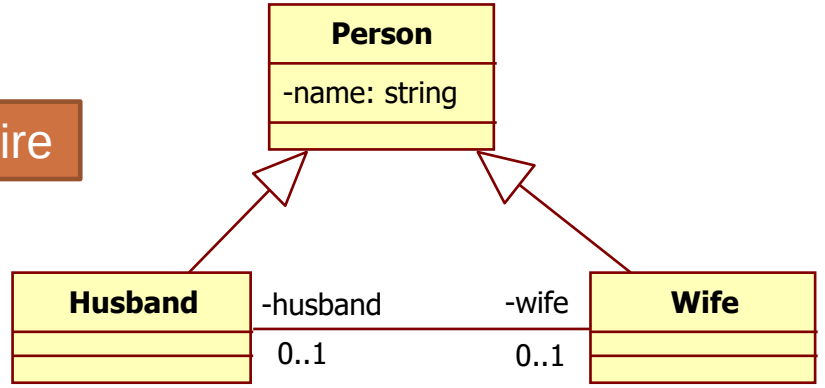


Nyelvtani szabányok (ANTLR4 szintaxis + annotációk):

```
Main: 'family' Identifier '{' Husband Wife '}';

$NameDef(Husband)
Husband: 'husband' $Name Identifier ';';

$NameDef(Wife)
Wife: 'wife' $Name Identifier ';';

Identifier: [a-zA-Z]*;
```

Szemantikai annotációk

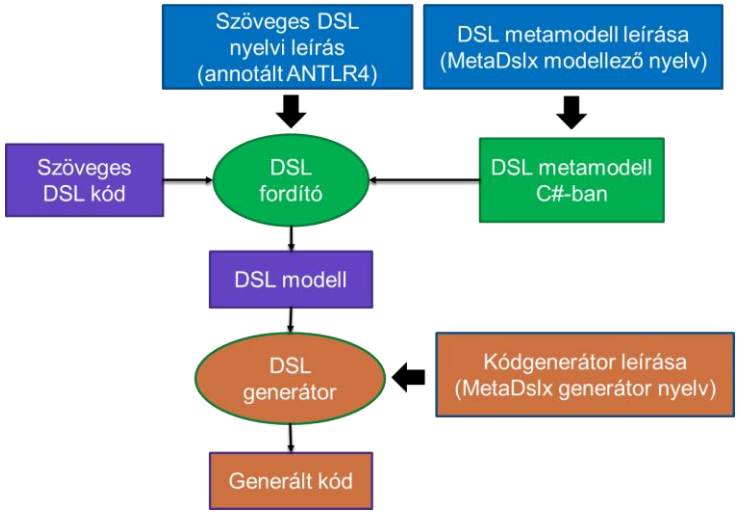
Hivatkozás a metamodell elemeire



Kódgenerálás (~Xtend)

Generátor:

```
namespace MGenTutorial;
generator HelloGenerator for object;

template SayHello(string name)
Hello, [name]!
end template
```



Használat C#-ból:

```
namespace MGenTutorial
{
    class Program
    {
        static void Main(string[] args)
        {
            HelloGenerator generator = new HelloGenerator();
            string output = generator.SayHello("World");
            Console.WriteLine(output);
} } }
```

Kimenet:

```
Hello, World!
```

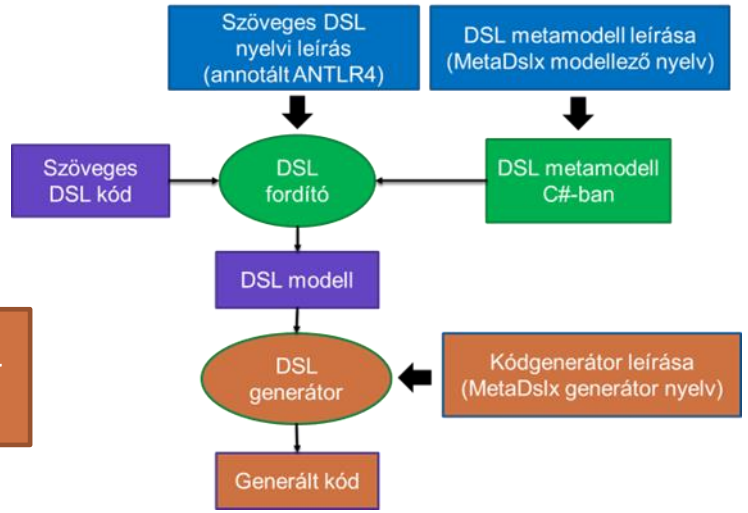
Kódgenerálás

Generátor:

```
namespace MGenTutorial;
generator FamilyGenerator for IEnumerable<Person>;

template GenerateFamily(string name)
family [name]
{
    [GenerateMembers()]
}
end template

template GenerateMembers()
[loop(Instances->h:typeof(Husband))]
husband [h.Name];
[end loop]
[loop(Instances->w:typeof(Wife))]
wife [w.Name];
[end loop]
end template
```

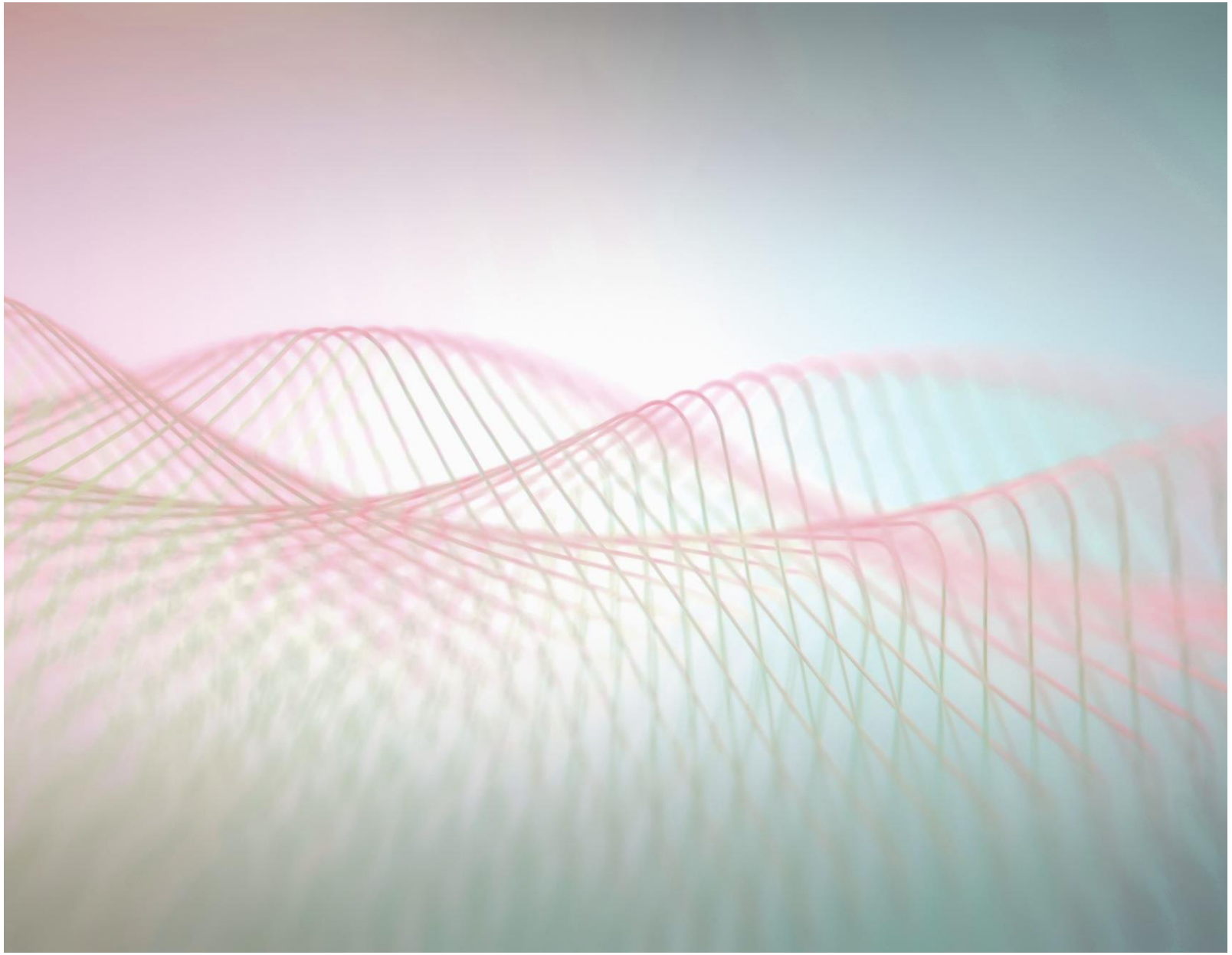
Másik sablon vagy függvény meghívása (megőrzi a bekezdést)

Modell kényelmes bejárása: loop (LINQ-szerű bejárás)



Egy lehetséges kimenet:

```
family Smith
{
    husband MrSmith;
    wife MrsSmith;
}
```



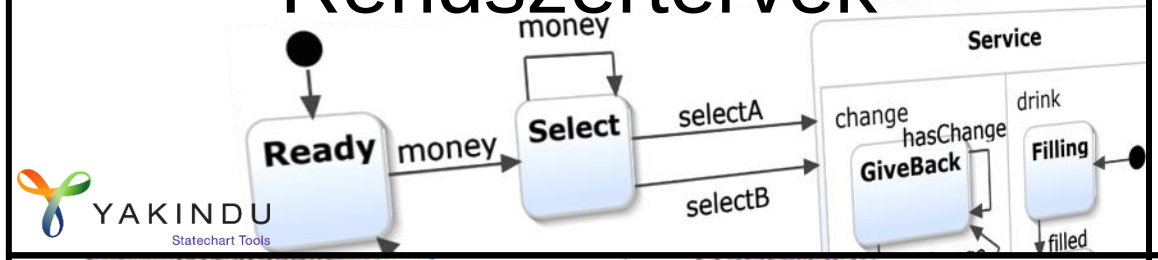
Automatikus modellgenerálási technikák

És a **CoReDiSc** kihívás

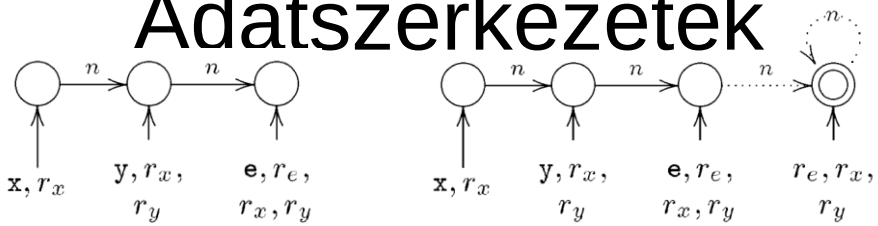
Modellezés gráfokkal

- Gráf alapú modelleket széles körben alkalmaznak az informatikában

Rendszertervek

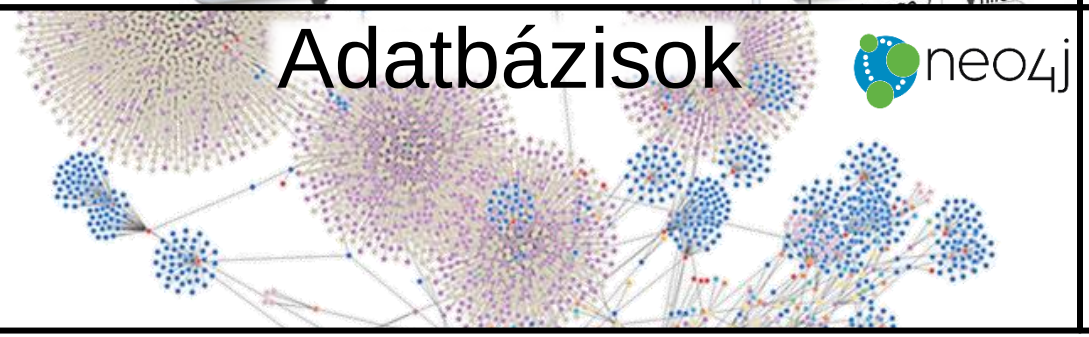


Adatszerkezetek

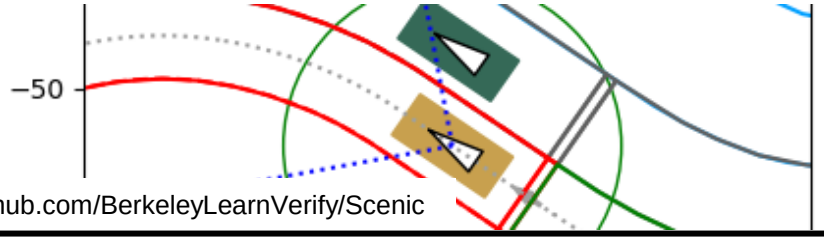


Sagiv, M., Reps, T., & Wilhelm, R. (2002). Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *24*(3), 217-298.

Adatbázisok



Tesztelrendezések



https://github.com/BerkeleyLearnVerify/Scenic

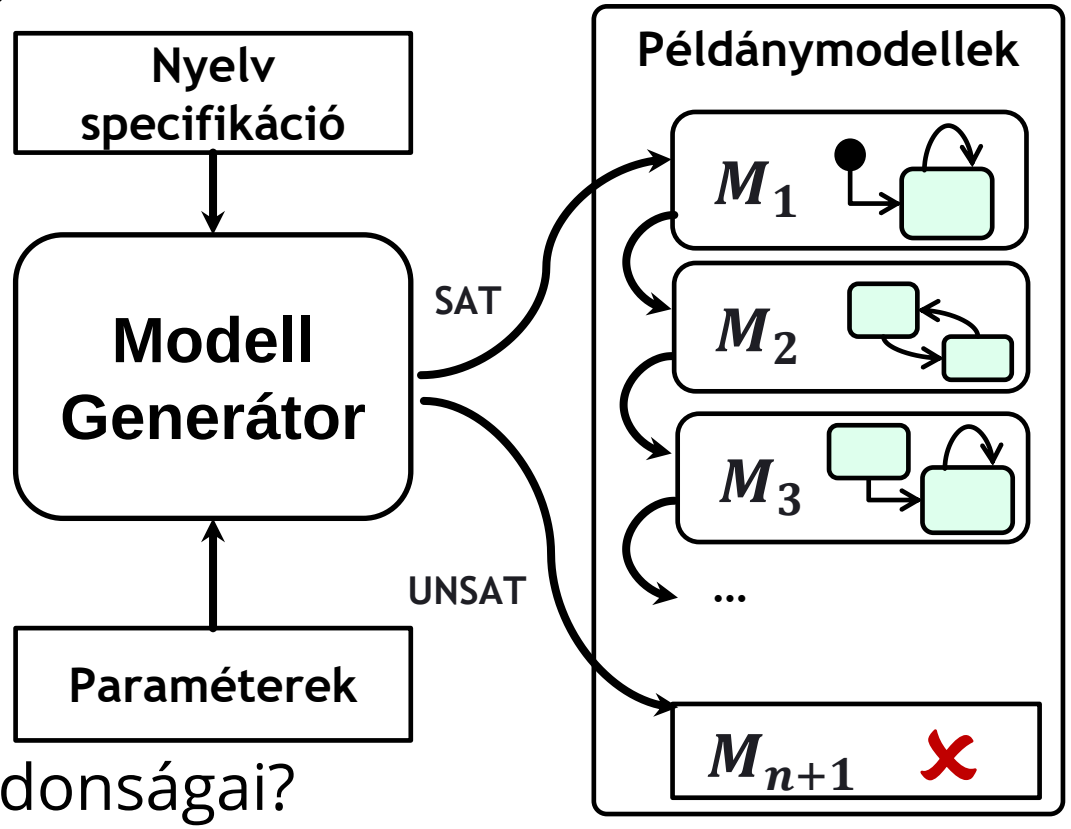
- Tesztelés, teljesítménymérés, tervezésitér bejárás: gráfokat használ.

Cél: (Konzisztens | Realisztikus | Diverz | Skálázható) generálás!

Modellgenerátorok felépítése

- Hogy néz ki egy ideális modellgenerátor?

1. **Bemenet:** Nyelv specifikációja
   ennek a nyelvnek a modelljeit állítja elő
2. **Bemenet:** Paraméterek
   leírhatjuk, milyen modelleket szeretnénk
3. **Kimenet:** Modellek
   modelleknek sorozata
4. **Kimenet:** Ellentmondásosság
   ha nincs mára feltételeknek megfelelő modell, ezt bizonyítjuk



- Melyek egy ideális modellgenerátor tulajdonságai?
- Milyen eszközök állnak rendelkezésre? → BME-n fejlesztett **Graph Solver**

Konzisztencia

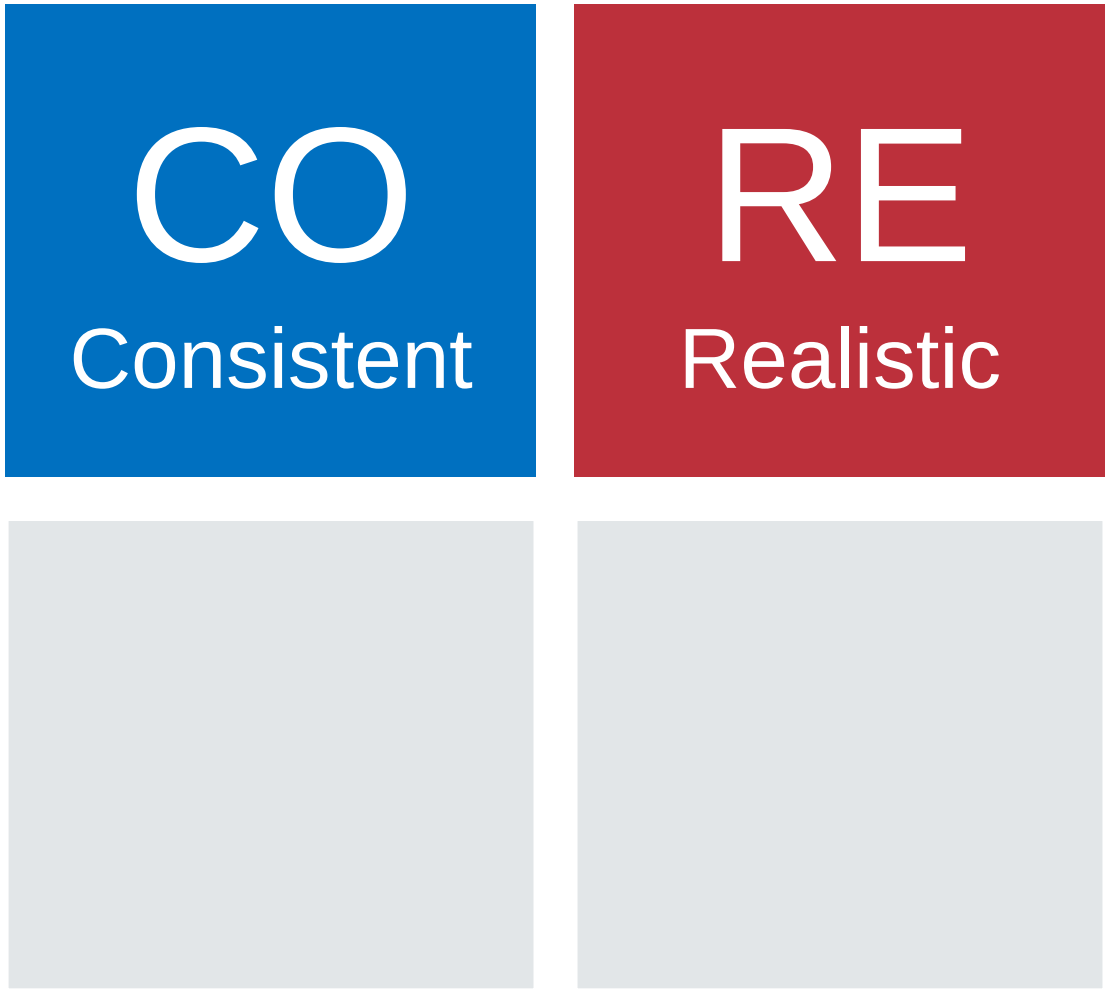
- Egy modellgenerátor **konzisztens**, ha minden generált modell teljesíti az összes jólformáltsági kényszert.
- Egy modellgenerátor **teljes**, ha (előbb-utóbb) képes minden szabályos modell előállítására.

- A jólformáltsági kényszerek összetett logikai kifejezések (OCL vagy **gráfminta**)

- **Graph Solver:** Gráfalapú konzisztens és teljes logikai következtető.

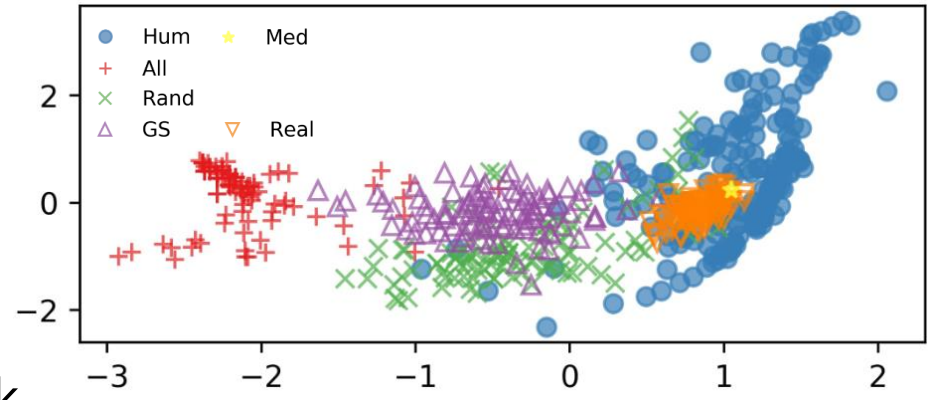
Realisztikusság

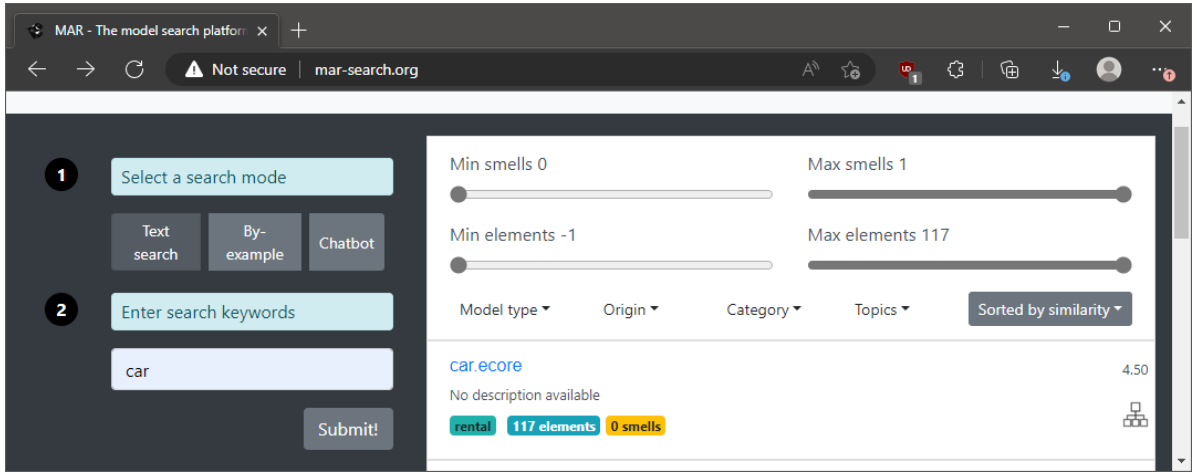
- Egy modellgenerátor realisztikus, ha a generált modelleket nem lehet megkülönböztetni az igaziaktól.

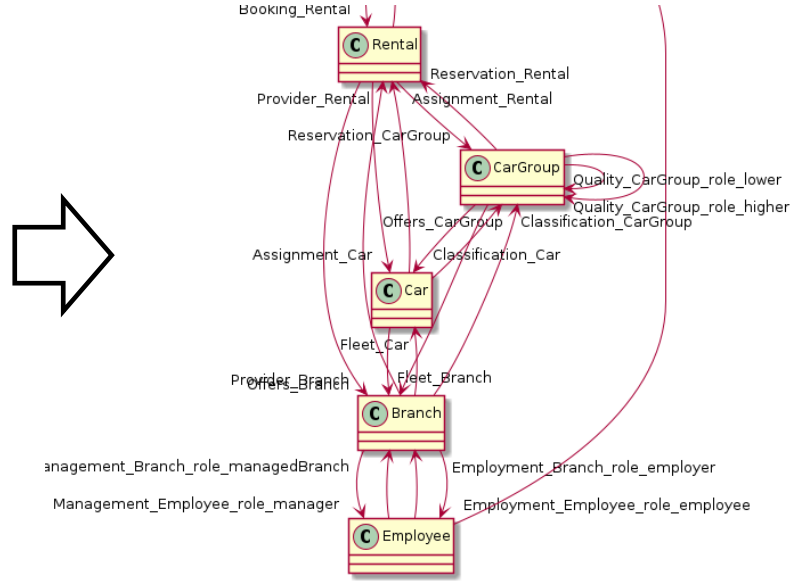


Realisztikusság

- Mitől lesz realisztikus a modell? Hogyan?
  - **Graph Solver:** hálózattudományi metrikák (fokszám),
  - Generálás során mérjük + szabályozzuk a metrikákat
- Honnan szerezzünk valódi modelleket?
  - A modellek fontos ismereteket tartalmaznak → titkosak.
  - Léteznek modell repository-k, amelyek indexelik a modelleket (http://mar-search.org/)



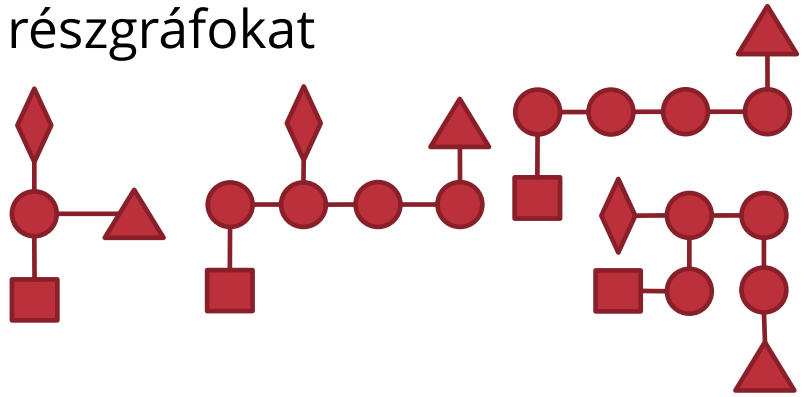




Diverzitás

- Modellek nem szimmetrikusak.
- A generált modellek szignifikánsan különböznek egymástól.

- Hogyan mérjük a diverzitást?
  - Pl számoljuk össze a különböző részgráfokat



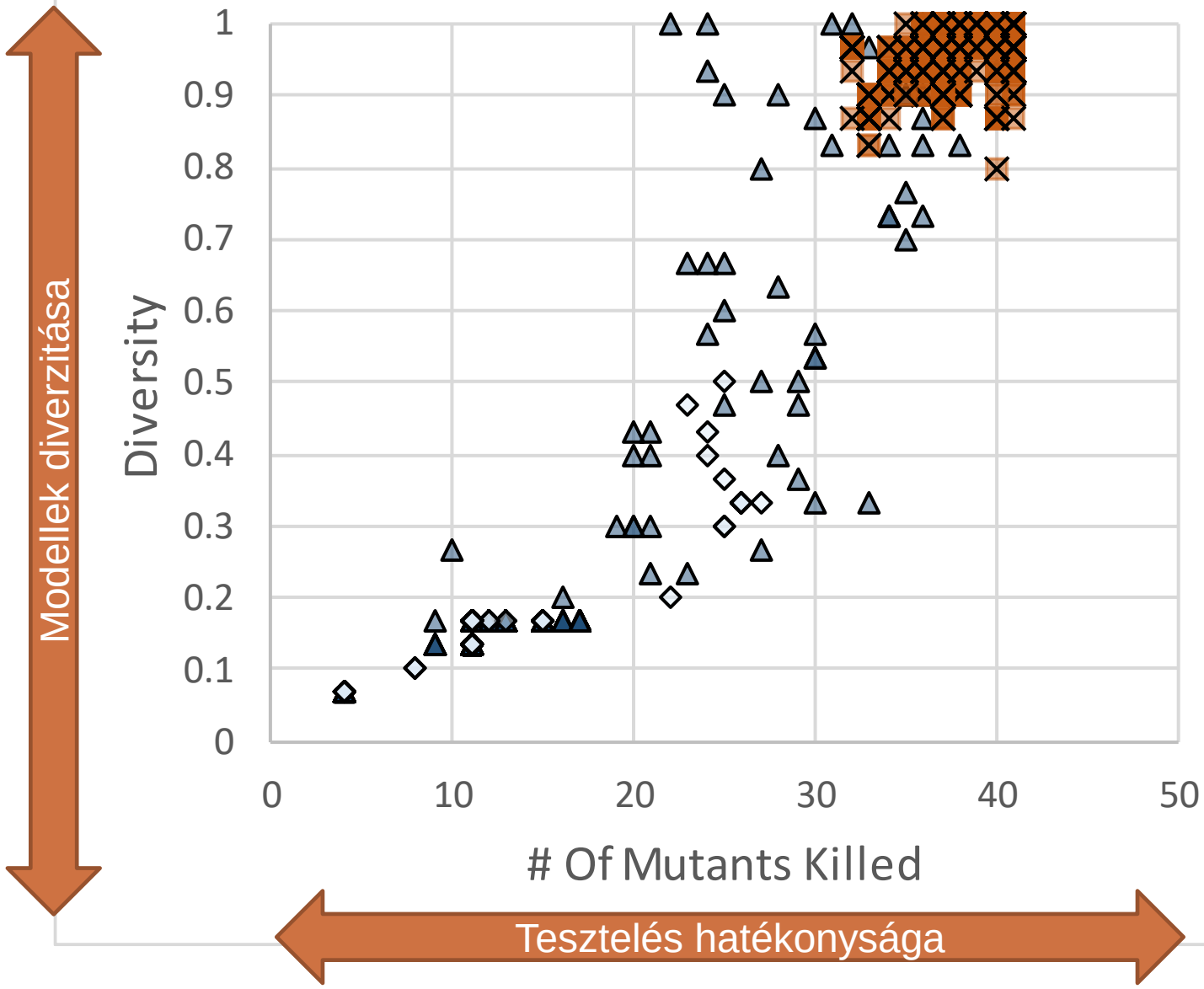
  - Minél több fajta, annál diverzebb!

→ jobb tesztek

CO
Consistent

RE
Realistic

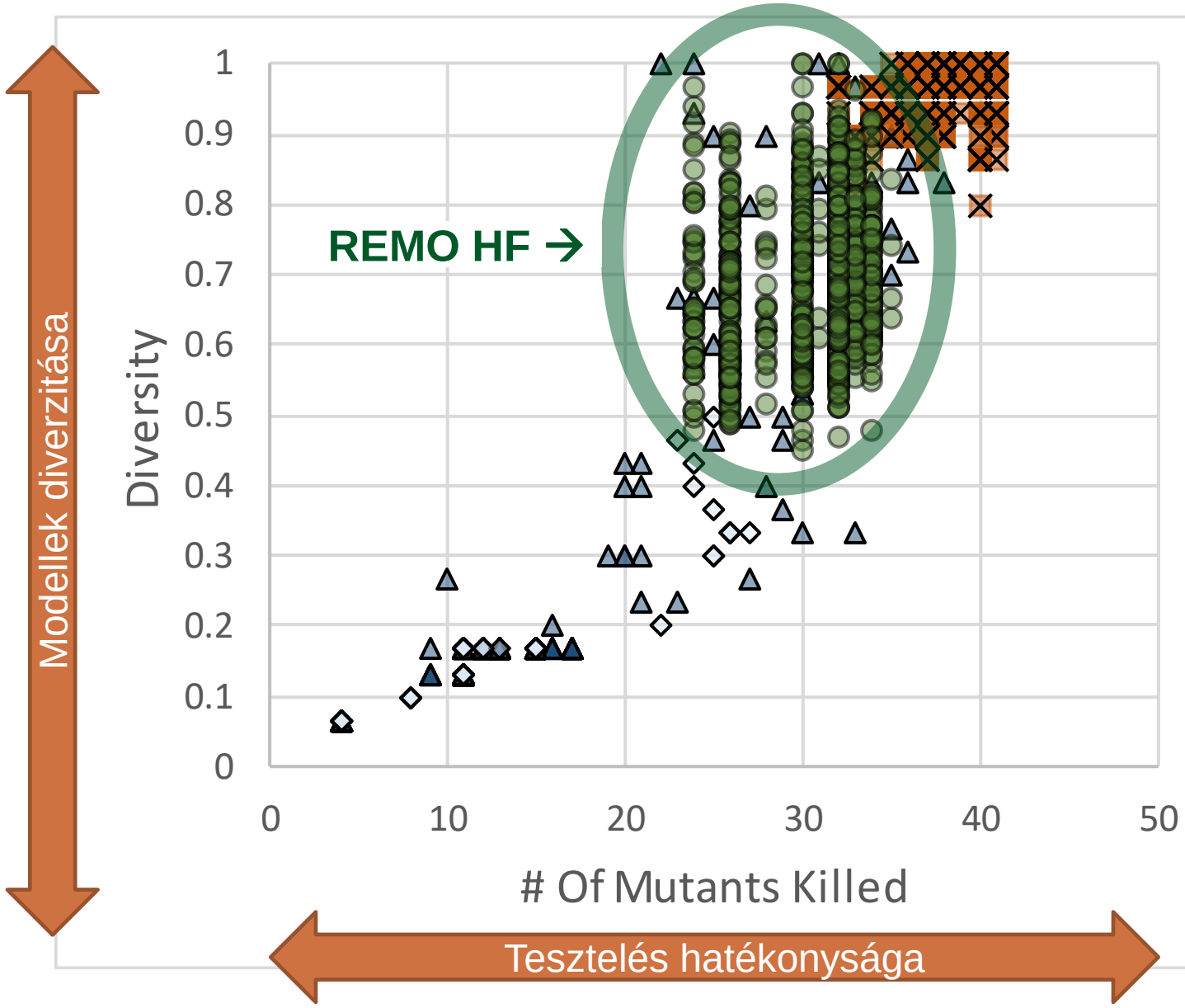
DI
Diverse



Hogyan viszonyul
- a **diverzitás**
- a **teszteléshez**?

YAKINDU
Statechart Tools

Különböző generátorokkal előállítottunk modelleket Yakindu-hoz, megmértük, hány mesterségesen beillesztett hibát detektálnánk, ha a modell tesztbemenetként használnánk.



Hogyan viszonyul
- a **diverzitás**
- a **teszteléshez**?

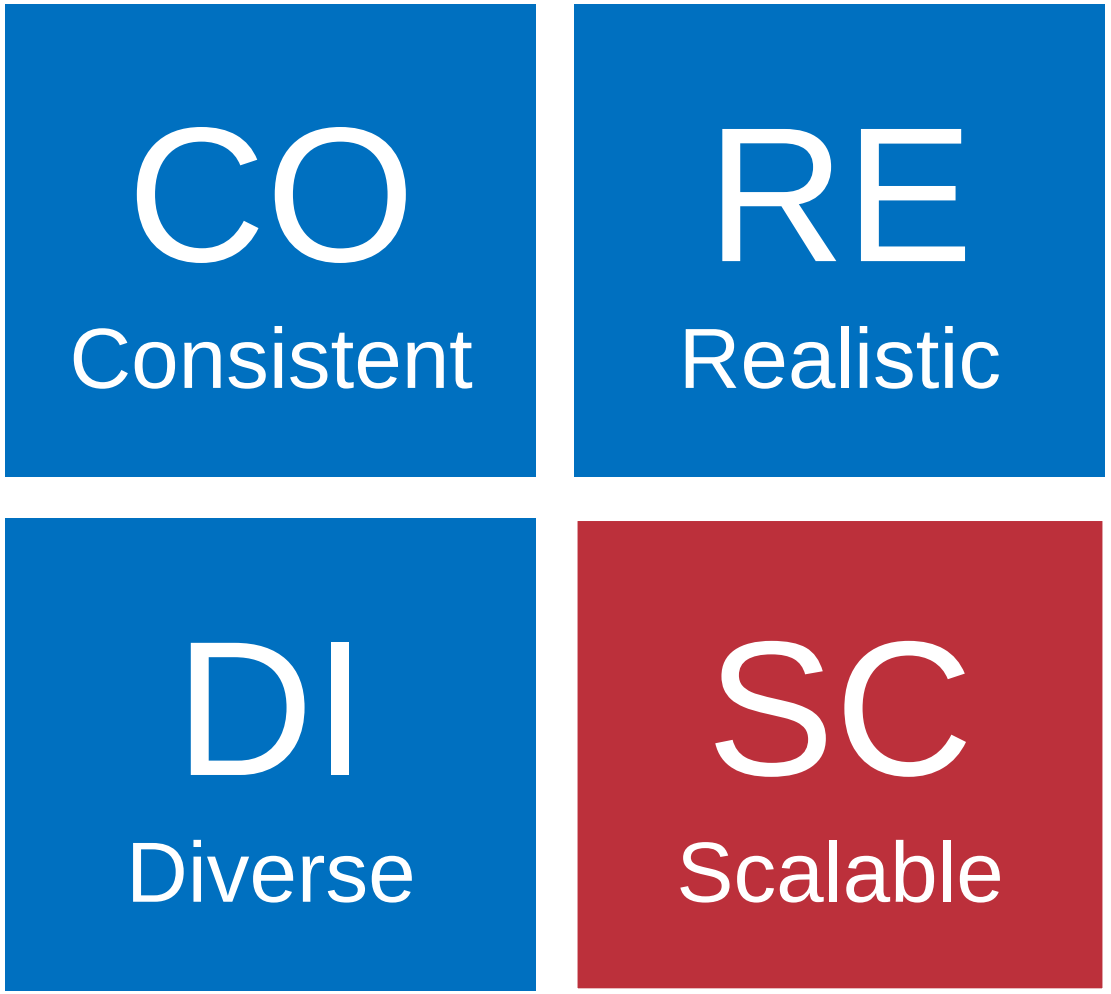
YAKINDU
Statechart Tools

Különböző generátorokkal előállítottunk modelleket Yakindu-hoz, megmértük, hány mesterségesen beillesztett hibát detektálnánk, ha a modell tesztbemenetként használnánk.

- **Tanulság 1:**
  SAT < Human < Graph Solver
- **Tanulság 2:**
  Korreláció

Skálázhatóság

- Egy modellgenerátor **skálázható:**
  - ha képes nagy modelleket előállítani
  - ha képes sok modellt előállítani
- **Co/Re/Di** teljesítése **nehéz**
- Skálázhatósági kihívások **(Sc--)**
- Vagy kompromisszumok **(Co-- Re-- Di--)**



Pár skálázhatósági mérés

Maximális modellméret pár DSL-en

	Largest model (#Objects)		
	Graph Solver	Sat4J	MiniSat
FAM+WF	6250	58	61
FAM-WF	7000	87	92
Yak+WF	1000	–	–
Yak-WF	7250	86	90
FS	4750	87	89
Ecore	2000	38	41

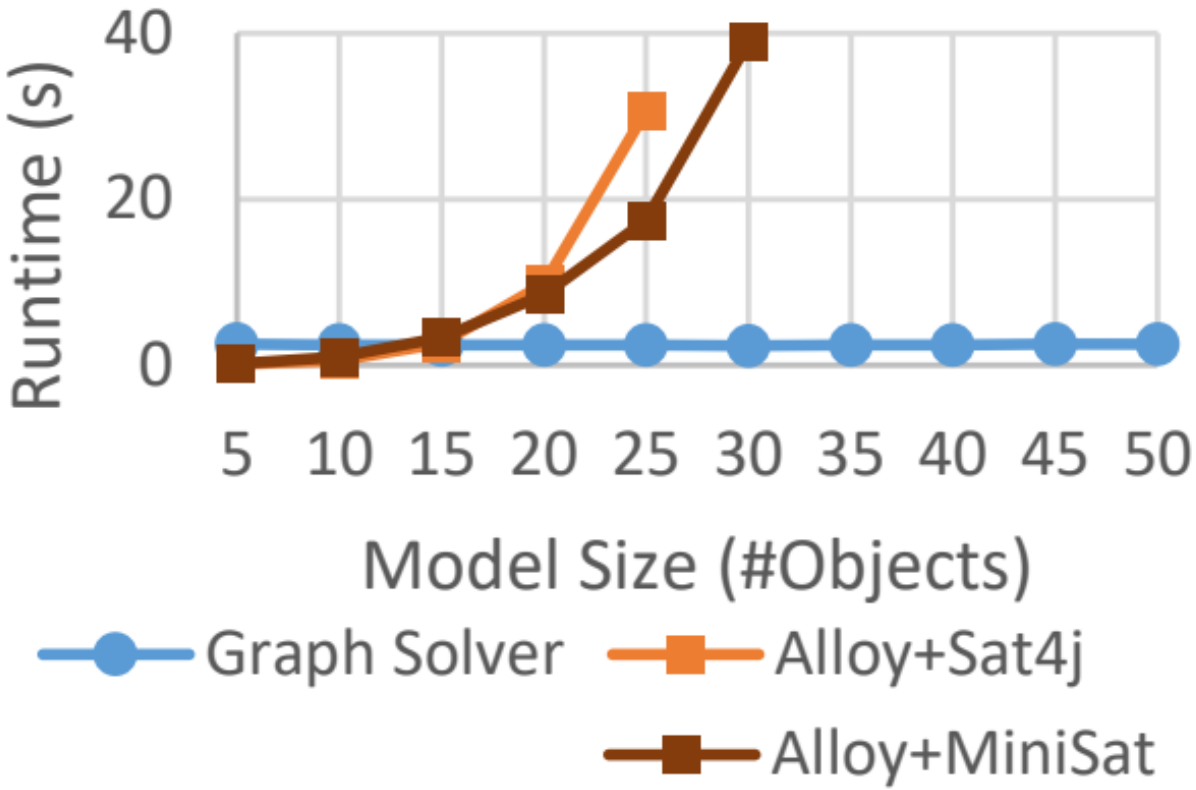
FAM: Industrial, Avionics

FS: File System example of

Yakindu: Industrial, Statemachine

Ecore: Metamodelling language

Példa futásidő összehasonlítás

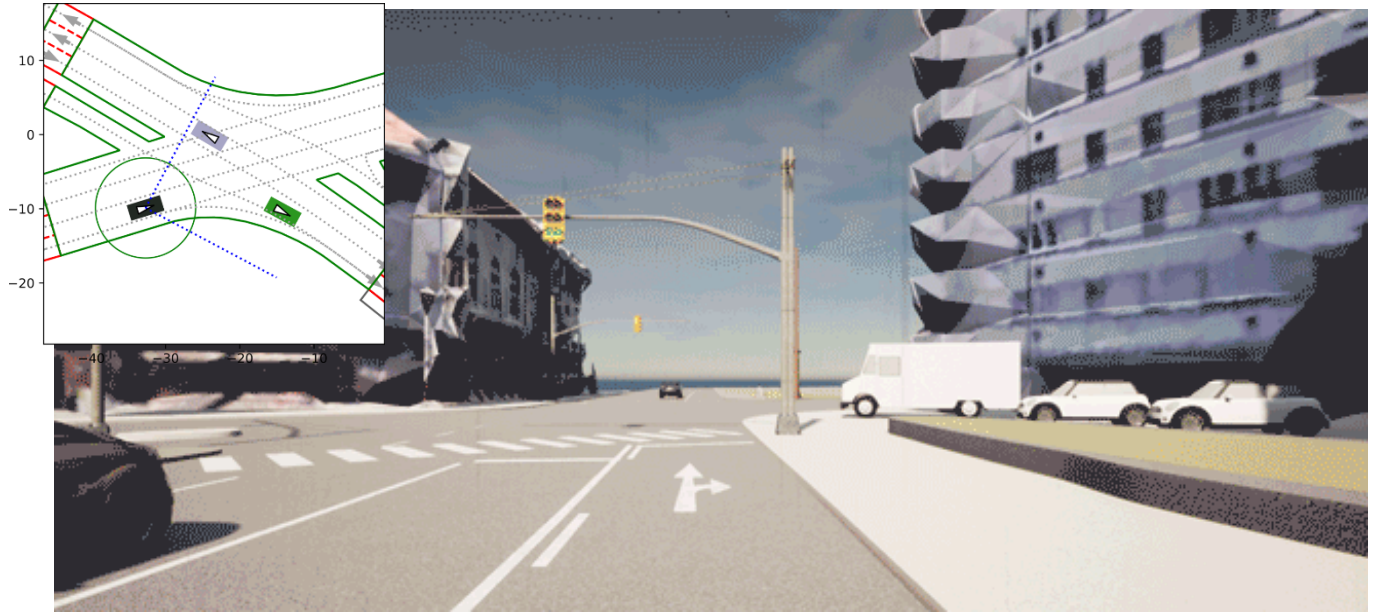


Konzisztens|Realisztikus|Diverz|Skálázható generátorok

- Milyen feladatra megoldásánál milyen generátorra van szükségünk?
  - Tesztgenerálás
  - Teljesítménymérés
  - Reprezentatív mintapéldák generálása
  - Igazságos házi feladatok generálása
  - Plágiumkeresés
  - Optimalizálás (pl legértékesebb architektúra generálása)

Összefoglalás

- Modellgenerálás fontos és nehéz feladat!
- Többféle tulajdonságú gráfgenerátorok: **Konzisztens**|**Realisztikus**|**Diverz**|**Skálázható**
- Egyetemen is fejlesztünk ilyen eszközöket, amely számos feladatra használjuk, pl:
  - Rendszermodellezés HF
  - Vasúti rendszerek tesztelése
  - Repülőgép modellező eszköz tesztelése
  - **Önvezető járművek tesztelése**



- Videó: https://youtu.be/fUopeDFIUKA
- Open source eszköz, VIATRA és EMF integráció https://github.com/viatra/VIATRA-Generator

Köszönöm a figyelmet!