



# Modellalapú szoftverfejlesztés

VIII. előadás

Szintaxis és  
Szemantika

Dr. Semeráth Oszkár

# Miből áll a vizuális szakterületi nyelv?

- Mire van szükség egy vizuális szakterületi nyelv definiálásakor?

- |                                |   |                       |
|--------------------------------|---|-----------------------|
| > Nyelv struktúrája            | } | → Absztrakt szintaxis |
| > <u>Kiegészítő kényszerek</u> |   |                       |
| > Megjelenítés                 |   | → Konkrét szintaxis   |
| > Struktúra jelentése          |   | → Szemantika          |

# Szintaxis és Szemantika

## I. Kényszerek

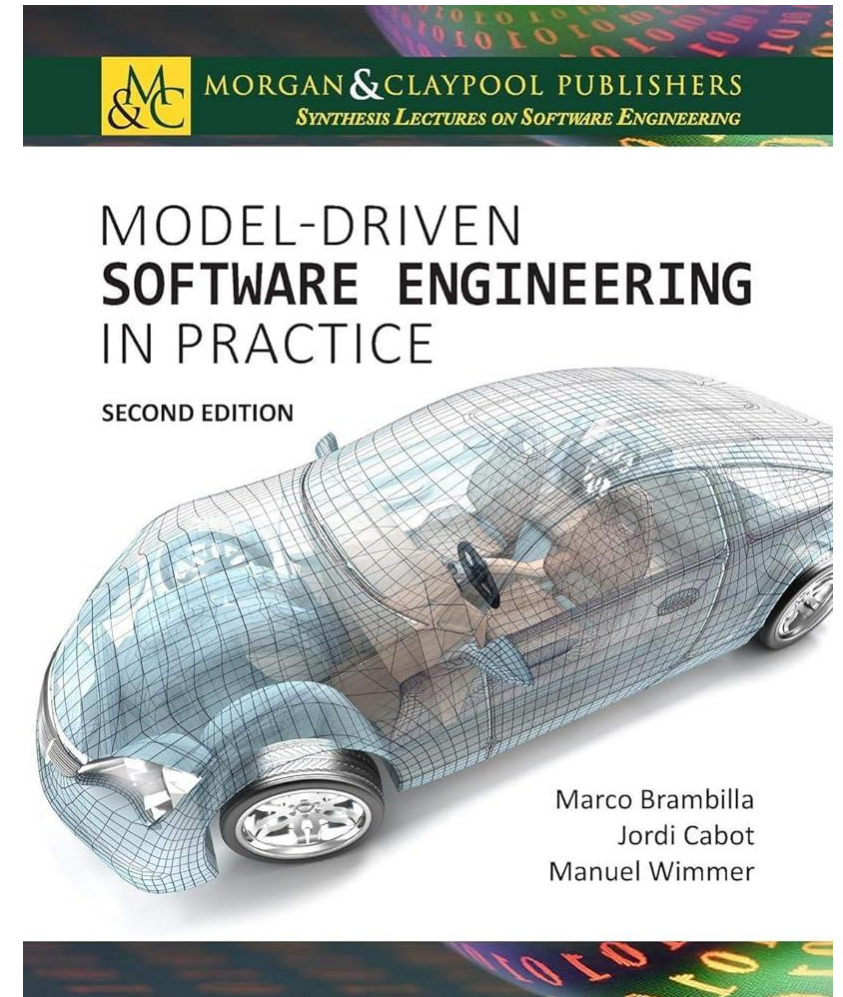
## II. Konkrét és Absztrakt szintaxis

## III. Szerkesztők

## IV. Szemantika



- Példák az alábbi könyvből jönnek:



# Kényszerek - Motiváció

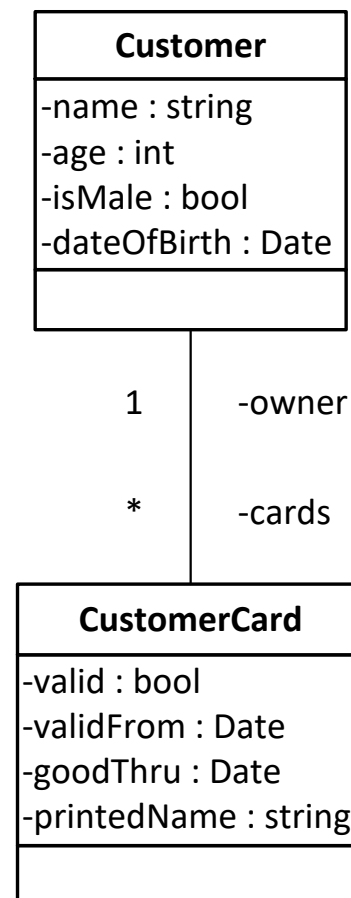
**Kihívás:** Mit tegyünk a modellünk tervezési szabályaival?

## ■ Példa: **objektum-orientált programozás**

- > Minden objektum saját maga konzisztenciájáért felel
- > “***name*** nem üres”
- > “***age*** nem lehet negatív”
- > “***printedName*** a kártyán tartalmazza a ***name*** attribútumot”

## ■ Példa: **szakterület-specifikus nyelvek**

- > Globális kényszerek a tipikusak, modellre vonatkozna a kényszerek
- > “***printedName*** egyedi”
- > “***Customer***-nek egy időben csak egy érvényes kártyája lehet”



# Kényszerek

- Mi a kényszer?
  - > *A kényszer egy megszorítás a metamodel egy vagy több elemén, értékén.*
- A struktúrális megadás kényelmes, bonyolult kényszereket azonban körülményes struktúraként megfogalmazni
  - > A hiányosság nem azért áll elő, mert a metamodellező nyelvet rosszul konstruáltuk meg!
- **Hogyan legyen megadható egy kényszer?**



# Object Constraint Language (OCL)

- Lehetővé teszi precíz UML/metamodellek definiálását
- OMG szabvány
- Tulajdonságai
  - > **Az OCL kényszerek deklaratívak:** azt adják meg mi helyes és nem azt, hogy mit kell tenni
  - > **Az OCL kényszereknek nincs mellékhatásuk:** az OCL kifejezések kiértékelése nem változtatja meg a rendszer állapotát
  - > **Az OCL kényszerek formális szintaxissal és szemantikával rendelkeznek:** értelmezésük egyértelmű és automatizálható
- Metamodelll bővítése = több lehetőség
- Kényszerek bővítése = kevesebb lehetőség
- Editorban szerkeszthető, OCL interpreterek

# Kontextus

- **Kontextus:** Az a modell elem, amire az OCL kifejezést definiálták

> osztály, interfész, adattípus, komponens, művelet, példány

pl nemüres név:

```
context Customer inv: self.name <> ''
```

Customer
-name : string
-age : int
-isMale : bool
-dateOfBirth : Date

- **Kontextus típusa:** annak az modellelemnek a típusa, amire a kifejezést kiértékelik

- Ha a kontextus egy típus, akkor a kontextus típusával megegyezik

```
context Customer inv:...
```



```
 $\forall x: \textit{Customer}(x) \Rightarrow \dots$ 
```

- **Kontextus példány:** a konkrét modellelem, amire kiértékeljük a kifejezést

> „self” kulcsszóval hivatkozunk



# Kifejezéstípusok kontextus szerint

- Invariáns

- > „*A magyarok személyi számának második-harmadik számjegye a születésük évét jelöli.*”

- Elő- és utófeltétel

- > „*Sötét van, mielőtt/miután felkel/lenyugszik a nap*”

- Kezdeti érték

- > „*Az autó gyártásakor 0 km-t futott*”

- Származtatott érték

- > „*A végső érdemjegy a ZH és a vizsga átlaga*”

- Metódustörzs

- > „*A könyvtárban lévő könyvek száma a polcokon lévő könyvek számának összege*”

# Invariánsok

- *Metamodellelemre* definiált kényszer
- Egy logikai kifejezés, aminek a metamodellelem minden példányára, minden időpillanatban igaznak kell lennie

```
context <metaelem>
```

```
inv [<kényszernév>]: <logikai kifejezés>
```

# Invariáns példák

```
context Customer inv:  
  self.name <> ''
```

Attribútum kényszerek

```
context Customer inv:  
  age >= 18
```

Numerikus műveletek

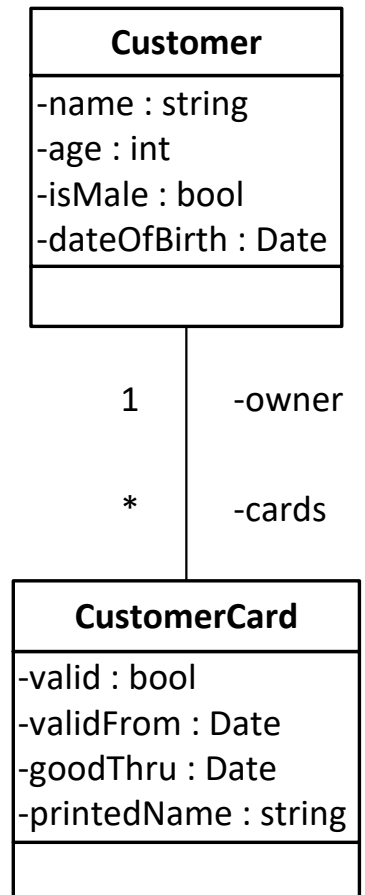
```
context CustomerCard inv checkDates:  
  validFrom.isBefore(goodThru)
```

Függvényhívás (pure)

```
context Customer inv:  
  Customer.allInstances() ->  
    forAll(c1, c2 | c1<>c2 implies  
      c1.name <> c2.name)
```

Logika ( $\forall, \wedge, \neg, \exists, \vee$ )

Date
+isBefore() : bool



# Elő- és utófeltételek

- *Műveletre* definiált kényszer
- A művelet hatására koncentrált algoritmustól vagy implementációtól függetlenül
  - > Előfeltétel: egy adott művelet elvégzése előtti utolsó időpillanatban igaz feltétel
  - > Utófeltétel: egy adott művelet elvégzése utáni első időpillanatban igaz feltétel

```
context <metaelem>::<művelet> (<paraméterek>)  
pre[<kényszernév>]: <logikai kifejezés>
```

```
context <metaelem>::<művelet> (<paraméterek>)  
post[<kényszernév>]: <logikai kifejezés>
```

# Elő- és utófeltétel példák

```
context LoyaltyAccount::isEmpty(): Boolean
```

```
post: result = (points = 0)
```

Visszatérési érték

```
context Customer::birthdayHappens()
```

```
post: age = age@pre + 1
```

Korábbi érték

```
context Service::
```

```
upgradePointsEarned(amount: Integer)
```

```
post: calcPoints() = calcPoints@pre() + amount
```

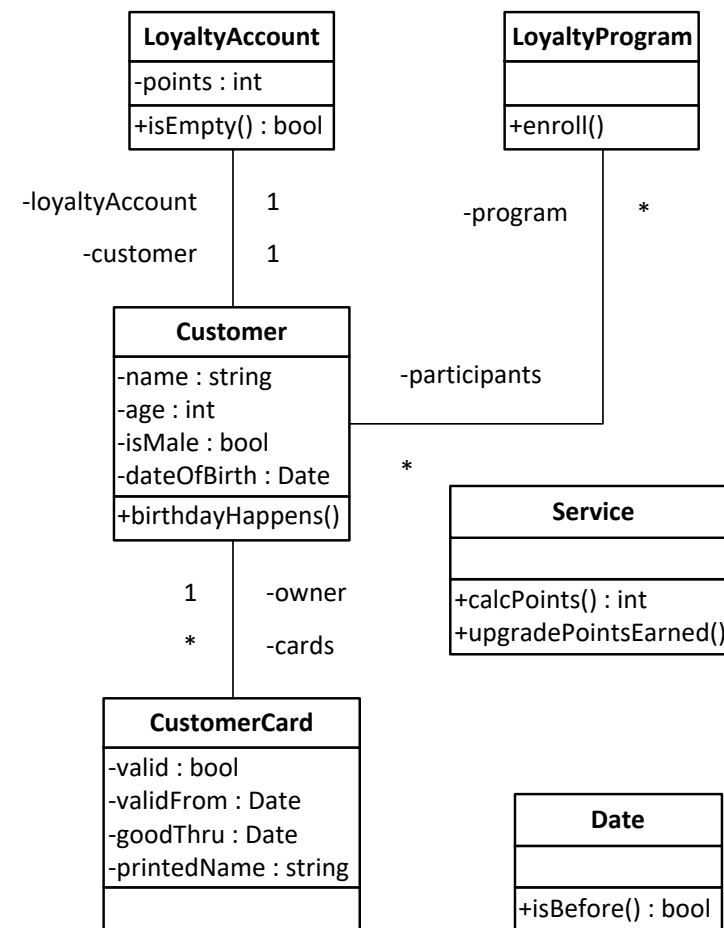
```
context LoyaltyProgram::enroll(c : Customer)
```

```
pre: c <> null && c.name <> ''
```

```
post: participants = participants@pre->
```

```
including(c)
```

paraméterek



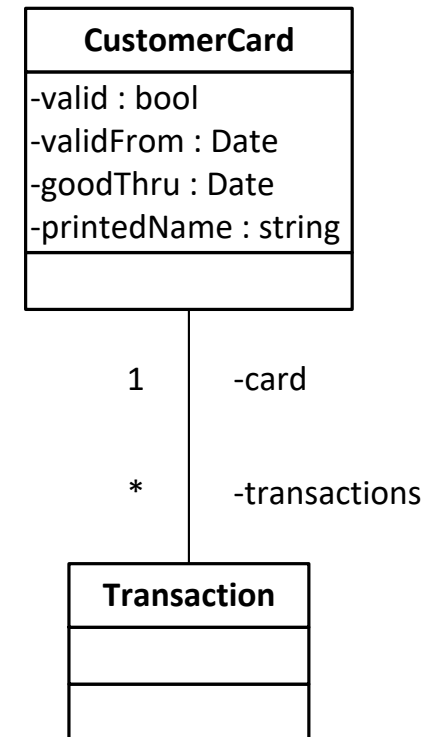
# Kezdeti érték

- *Attribútumra vagy asszociációra* definiált kényszer
- Egy érték, amit az attribútum vagy asszociáció vesz fel a kontextus példány létrejöttének pillanatában

```
context <metaelement>  
init: <logical expression>
```

```
context CustomerCard::transactions : Set(Transaction)  
init: Set{}
```

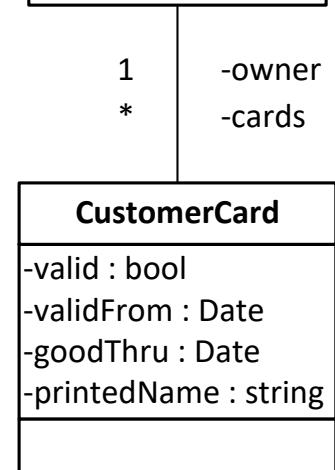
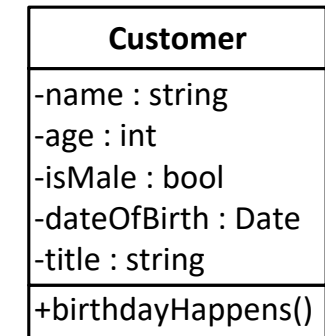
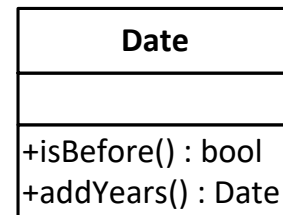
```
context CustomerCard::valid : Boolean  
init: true
```



# Származtatott érték

- *Attribútumra vagy asszociációra* definiált kényszer
- A származtatott elem nem önmagában létező érték, mindig más elemek segítségével definiálják

<b>context</b> CustomerCard::printedName <b>derive:</b> owner.title.concat(owner.name)
---





# Lekérdező műveletek törzse

- *Műveletre* definiált kényszer
- Vannak olyan műveletek, melyek egy adott értéket kérdeznek le, nincs egyéb mellékhatásuk
- Kényszerben rögzíthetjük, hogy pontosan mit is kell visszaadniuk

```
context LoyaltyAccount::getCustomerName() : String  
body: Membership.card.owner.name
```

```
context CustomerCard::getTransactions(from : Date, until: Date )  
                                     : Set(Transaction)  
body: transactions ->select (date.isAfter( from ) and  
                             date.isBefore( until ) )
```

```
context Department::getSalary(): int  
body: member.salary ->  
      iterate(member: Member; sum: Integer = 0 | sum + salary)
```

# Kényszerek és öröklés

- A kényszerek is öröklődnek
- Az **invariánst** a származott osztály örökli. A származott osztály szigoríthatja a kényszert, de nem enyhítheti.
  - > „**VIPCustomer** neve is egyedi kell legyen”
- Az **előfeltételt** lehet enyhíteni a származott osztály újradefiniált műveletében. Szigorítani nem lehet.
- Az **utófeltételt** lehet szigorítani a származott osztály újradefiniált műveletében. Enyhíteni nem lehet.
  - > „**VIPCustomer** elindíthat egy szolgáltatást akkor is, ha nincs hűségpontja.”
  - > „**VIPCustomer** születésnapján hűségpontot kap.”

# Szintaxis és Szemantika

## I. Kényszerek

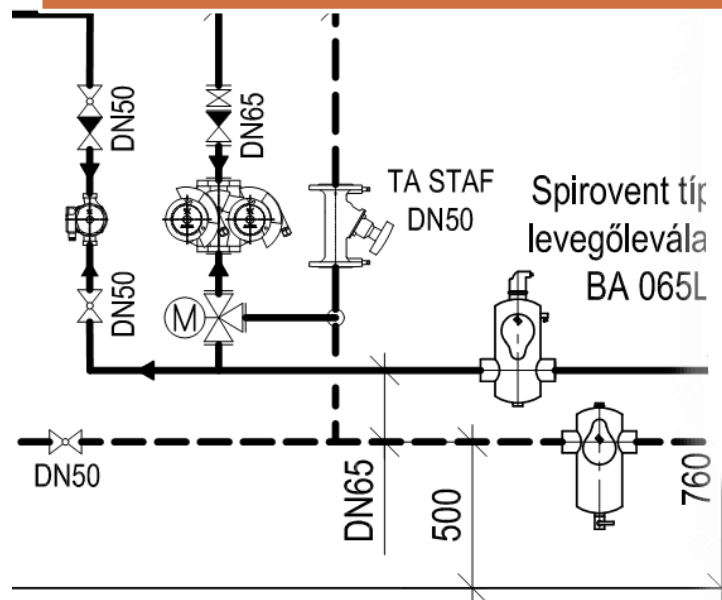
## II. Konkrét és Absztrakt szintaxis

## III. Szerkesztők

## IV. Szemantika



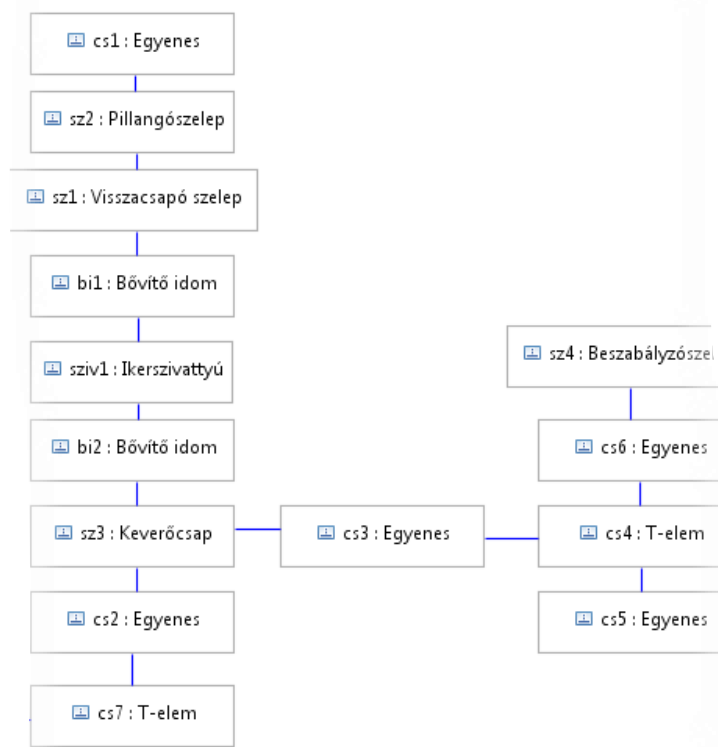
## Konkrét Szintaxis



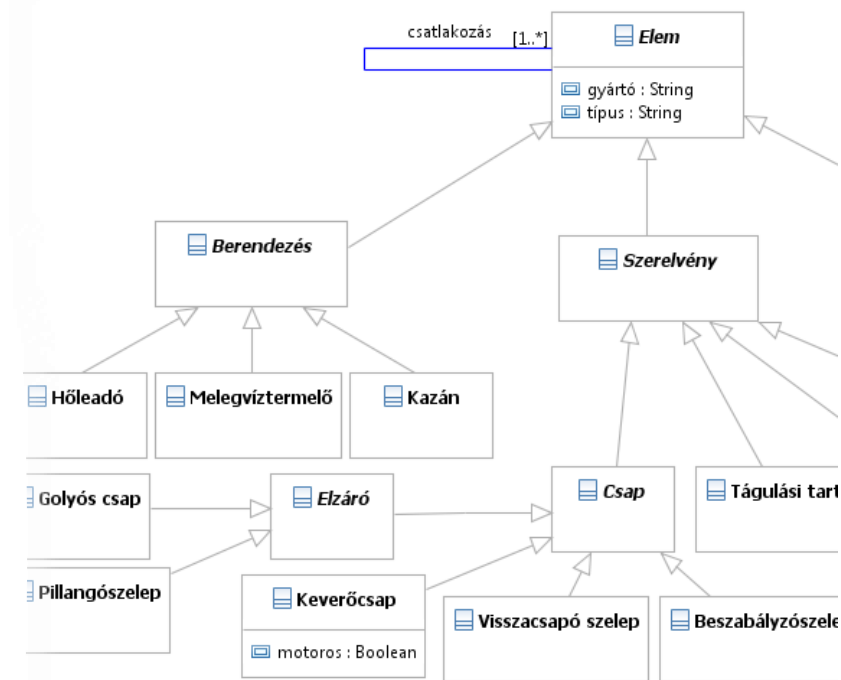
Honeywell  
keverőcsap  
DN50 K<sub>vs</sub> 40

Spirovent típusú  
iszapleválasztó  
BE 065L

## Absztrakt Szintaxis



## Metamodel



## Definíciók áttekintése

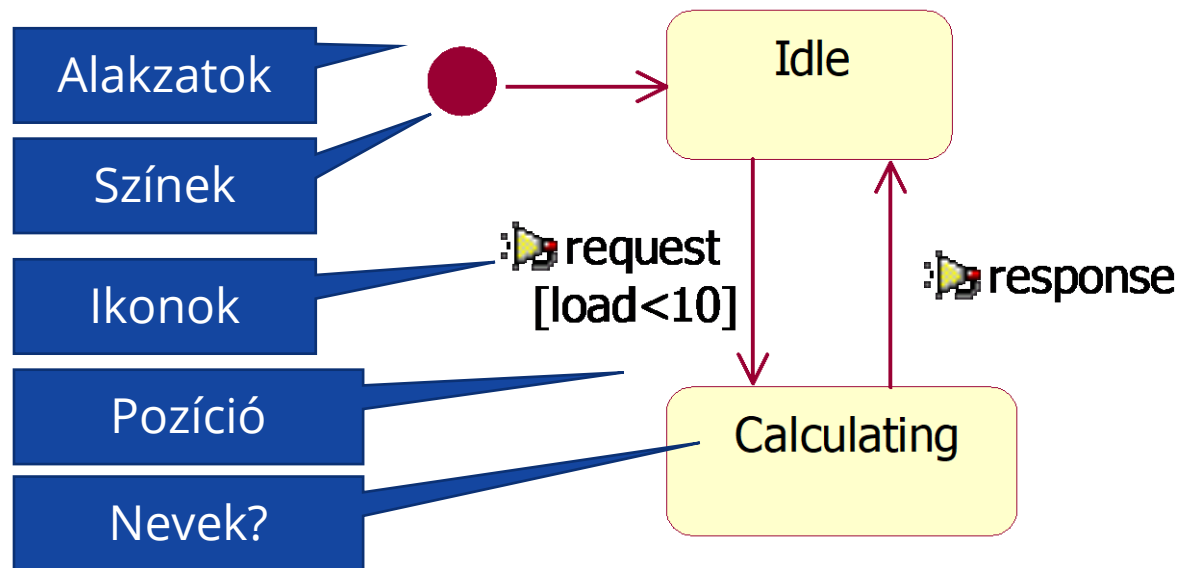
- **Cél:** válasszuk szét a **reprezentációtól független és függő** részeit.
- **Definíció [Absztrakt szintaxis]:** Olyan (absztrakt) adatstruktúra, amely a modell reprezentációtól független részét írja le.
- **Definíció [Konkrét szintaxis]:** A modell reprezentáció-specifikus része.
- Használják modellekre és modellező nyelvekre (modellezőeszközökre) is.

*„A Yakindu modellezési nyelv konkrét szintaxisában lekerekített téglalapok jelölik az állapotokat.”*

# Konkrét szintaxis

- Mi része a konkrét szintaxisnak?

## Grafikus szintaxis



## Szöveges szintaxis

```
String state = "idle";  
request() {  
    if (state == idle &&  
        this.load<10)  
        state = "calculating";  
}  
response() { /* ... */ }
```

Formázás  
(java)

Kulcsszavak

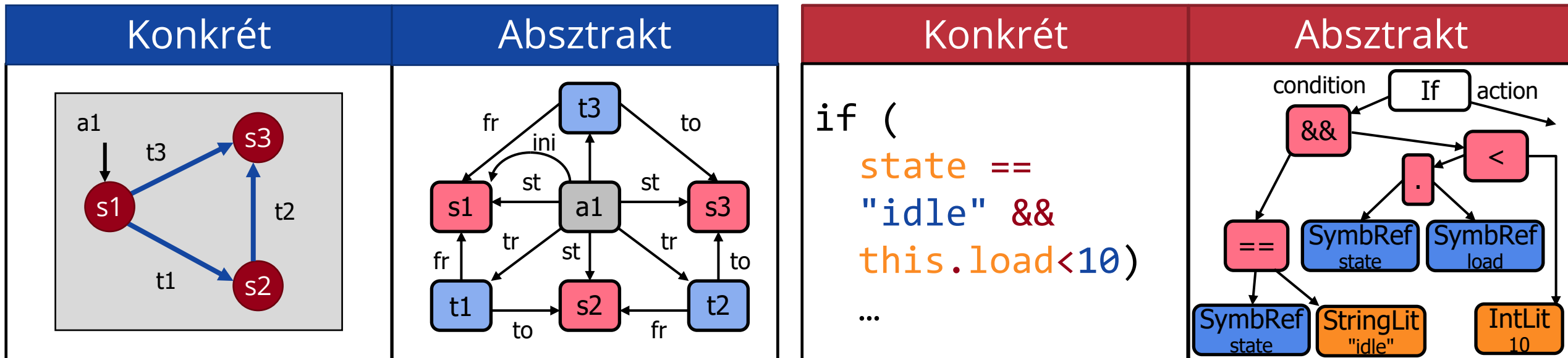
Nyelvtan?

Komment?

Syntax  
highlight?

# Absztrakt szintaxis

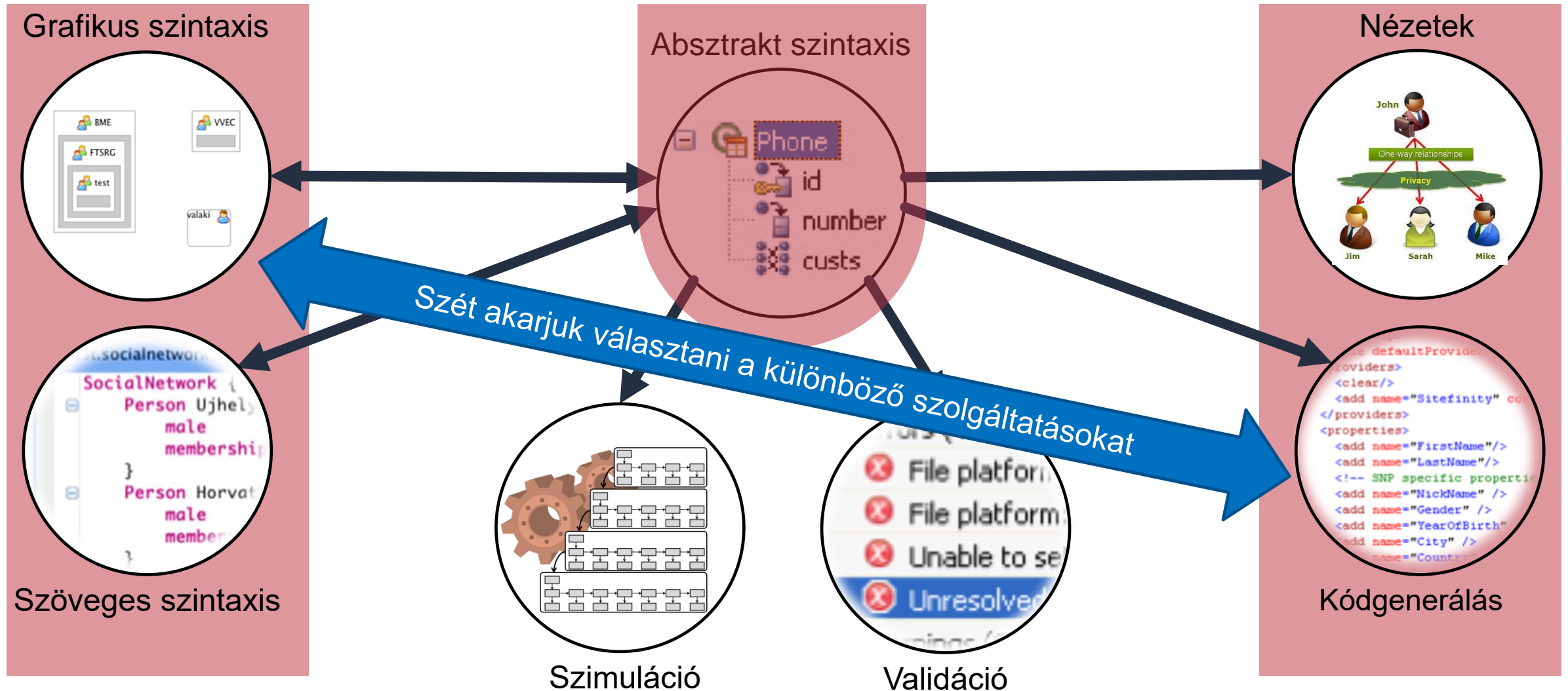
- Hogyan reprezentáljuk a modellek absztrakt szintaxisát?



- Absztrakt szintaxis:** tipikusan egy gráf alapú struktúra.

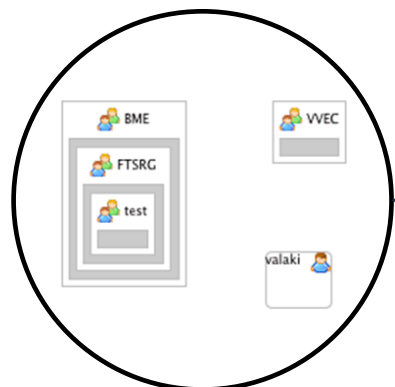


# Hogyan válasszuk szét a konkrét és absztrakt szintaxist?

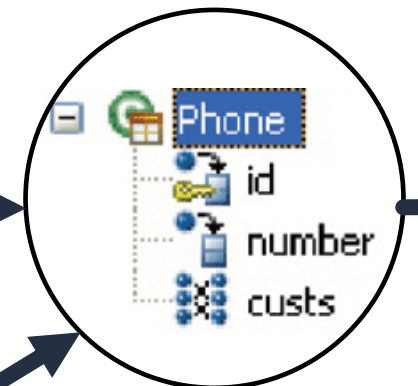


# Változtatunk a konkrét szintaxisban, de az absztraktban nem

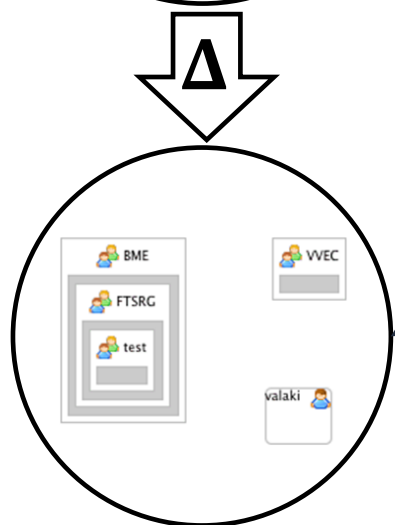
Grafikus szintaxis



Absztrakt szintaxis

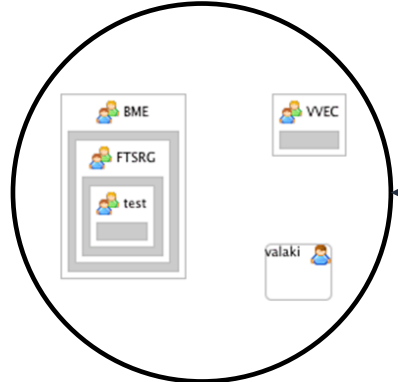
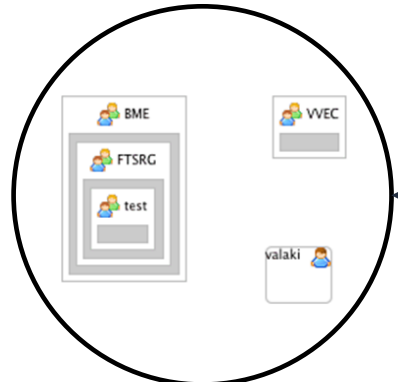


Kódgenerátor

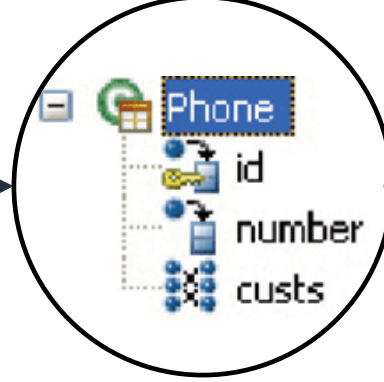
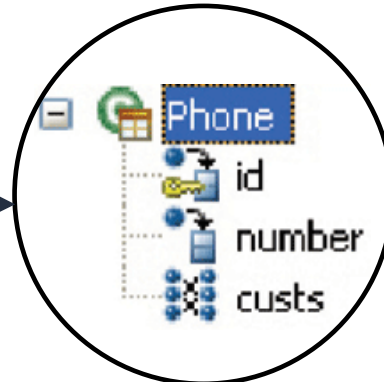


# Változtatunk az absztraktban

Grafikus szintaxis



Absztrakt szintaxis



Kódgenerátor



**Cél:** úgy szétválasztani a konkrét és absztrakt szintaxist, hogy a modellezési szolgáltatások ne interferáljanak

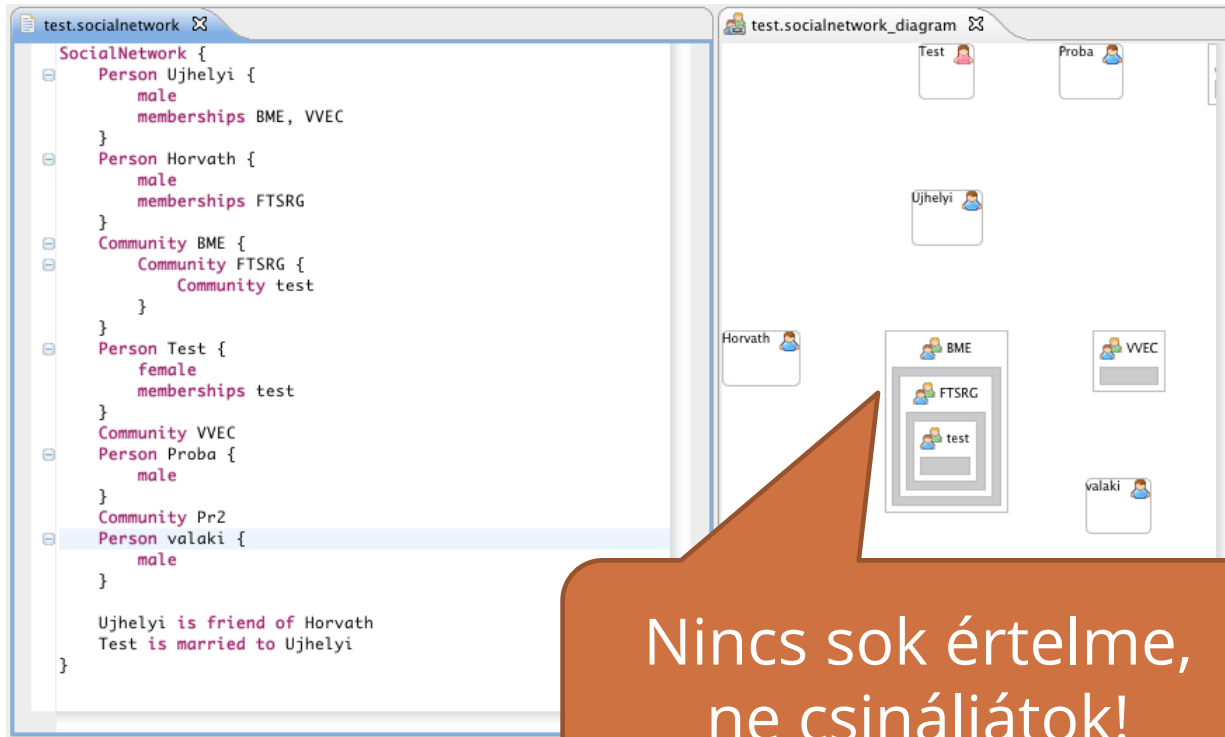
# Szintaxisok multiplicitása

- 1 absztrakt szintaxis → sok szöveges és grafikus jelölés
  - > Emberi szemmel olvasható és írható szöveges vagy grafikus szintaxis
  - > Szöveges szintaxis továbbításhoz vagy tároláshoz (tipikusan XML)
  - > UML esetében a diagramok általában csak parciális nézetek
- 1 absztrakt modell → sok konkrét szintaxis!
  - > Whitespace, diagram elrendezés
  - > Kommentek
  - > Szintaktikus cukor
- 1 szemantikus interpretáció → sok absztrakt modell
  - > p.l. UML2 Attribútum vs. egyirányú Asszociáció

# Szöveges + Grafikus

## Ugyanaz a modell, kettő szintaxis

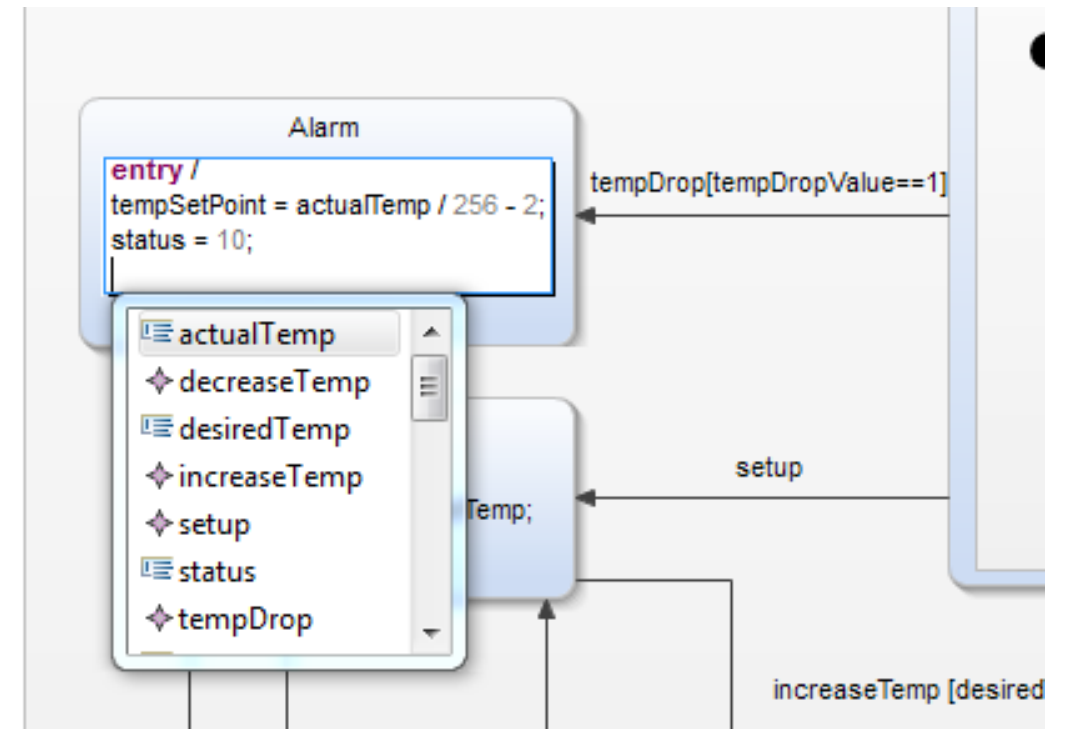
- Szöveges szerkesztő + grafikus nézet ☺
- Xtext Generic Viewer



Nincs sok értelme,  
ne csináljátok!

## Modell különböző aspektusai

- Diagram szöveges mezőkkel
- Beágyazott Xtext támogatás



# Szintaxis és Szemantika

## I. Kényszerek

## II. Konkrét és Absztrakt szintaxis

## III. Szerkesztők

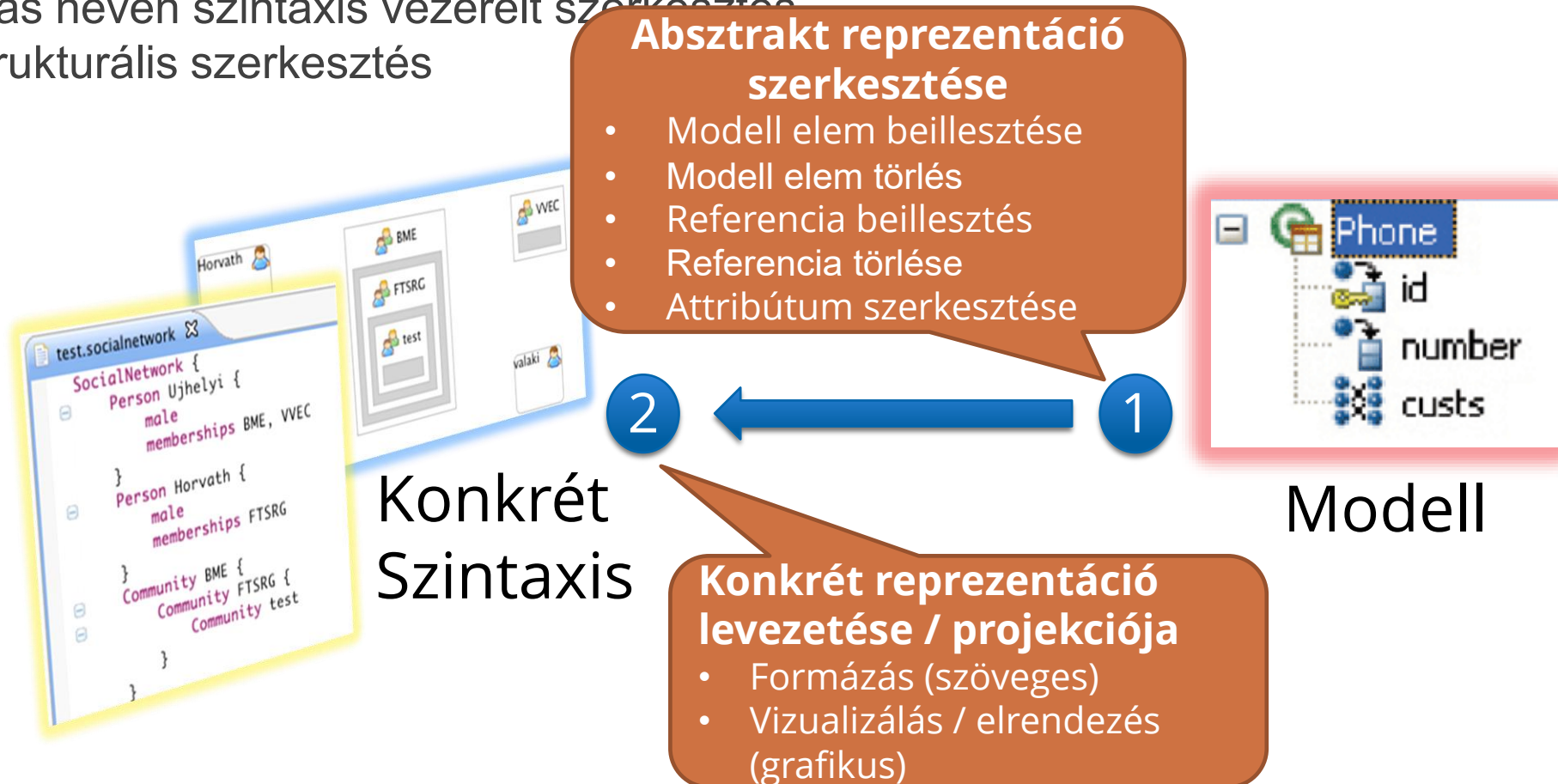
## IV. Szemantika



# Projekciós vs Nyers szerkesztés

## ■ 1. Munkafolyamat: **projekciós szerkesztés**

- > Más néven szintaxis vezérelt szerkesztés  
strukturális szerkesztés





# Projekciós vs Nyers szerkesztés

## ■ 2. Munkafolyamat: **nyers szerkesztés** (szöveges szintaxissal)

> Más néven forrás szerkesztés



Konkrét  
Szintaxis

### Konkrét reprezentáció szerkesztése

- Karakter(ek) beillesztése
- Karakter(ek) törlése
- Karakter(ek) cseréje

1



2



Modell

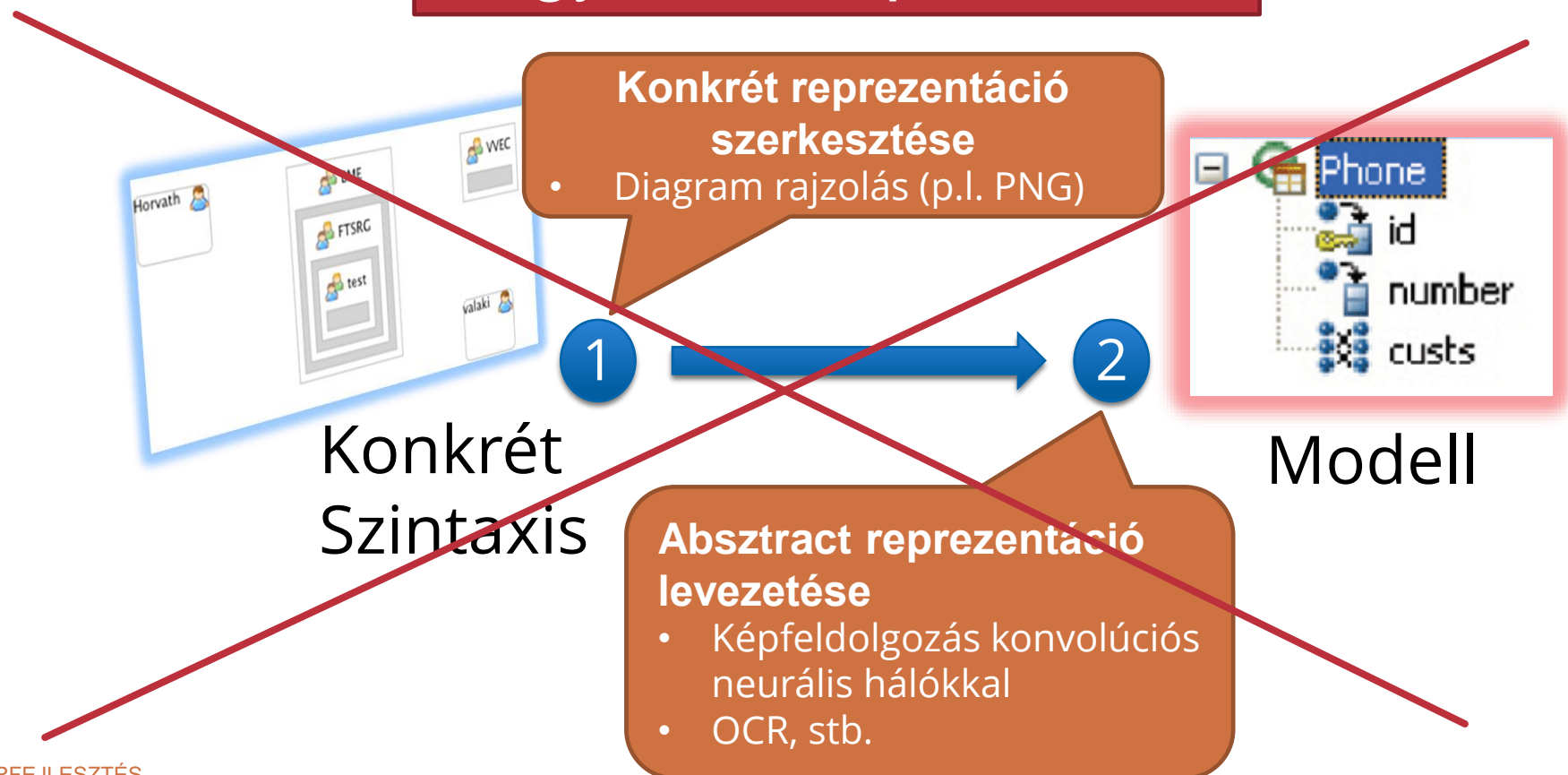
### Absztrakt reprezentáció levezetése

- Szöveges formátum feldolgozása

# Projekciós vs Nyers szerkesztés

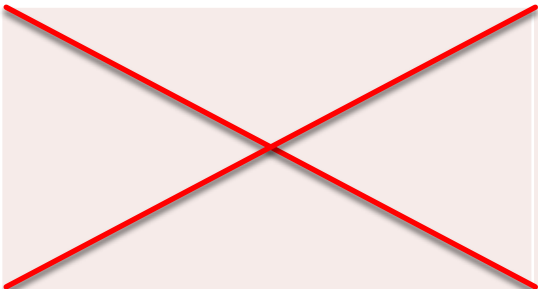



- 2. Munkafolyamat: **nyers szerkesztés** (grafikus szintaxissal)

**Nagyon nem praktikus**

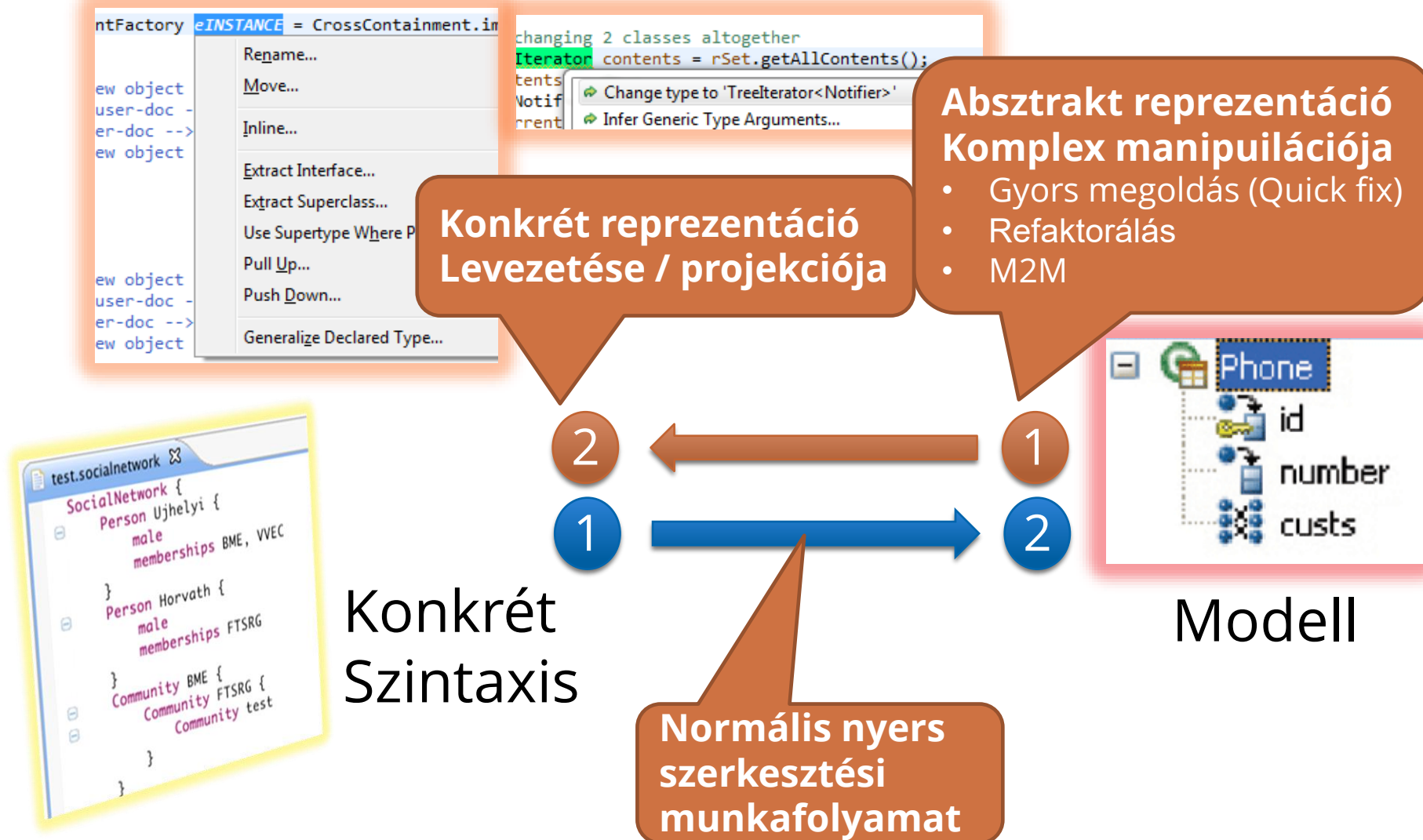


# Projekciós vs Nyers szerkesztés

„Tulajdonság mátrix” + példák

	Grafikus szintaxis	Szöveges szintaxis
Nyers szerkesztés		Tipikus 
Projekciós szerkesztés	Tipikus 	Ritka 

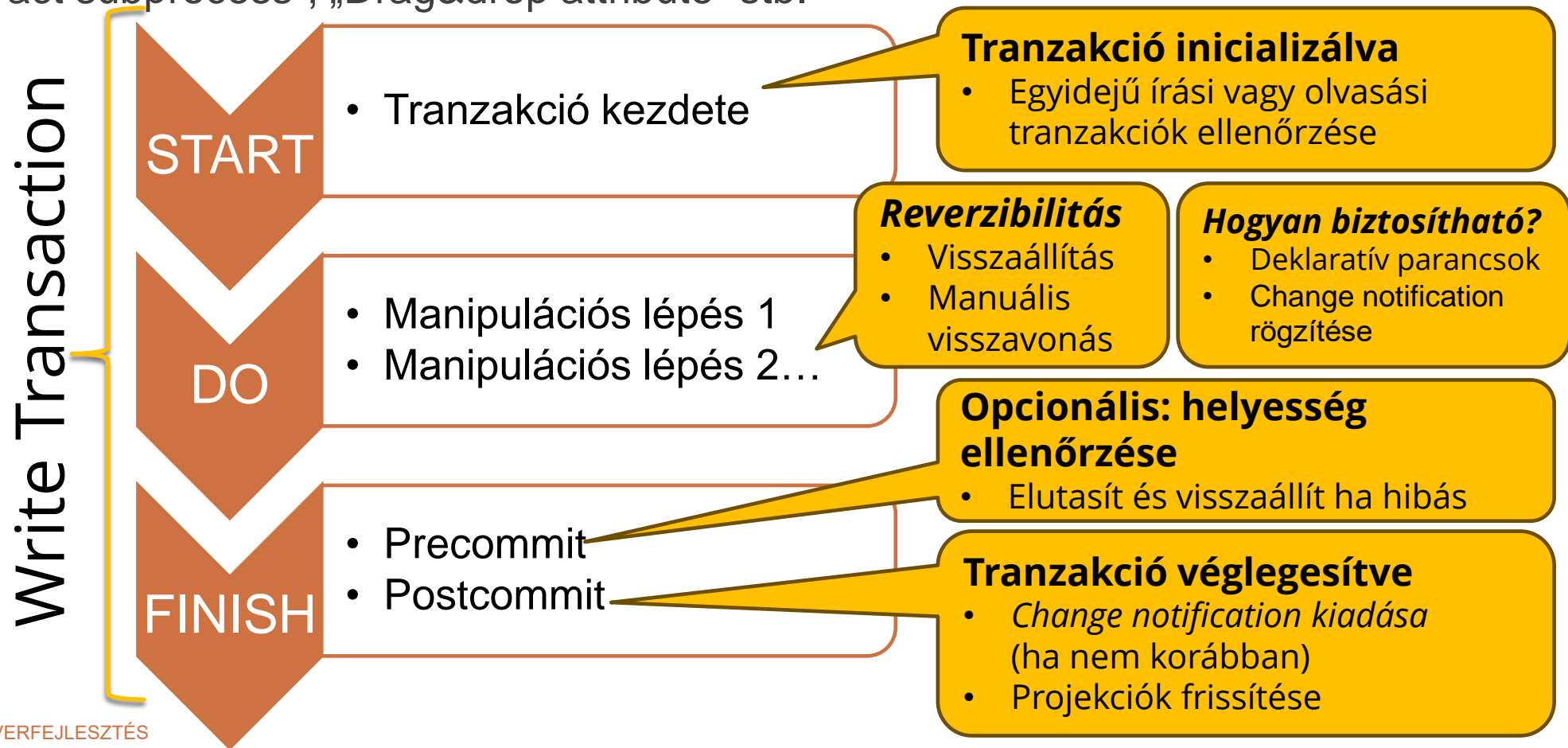
# Vegyes munkafolyamat



# Tranzakciók a projekciós szerkesztésben

## ■ Komplex manipulációs szekvencia egyetlen műveletként

> „Extract subprocess”, „Drag&drop attribute” stb.



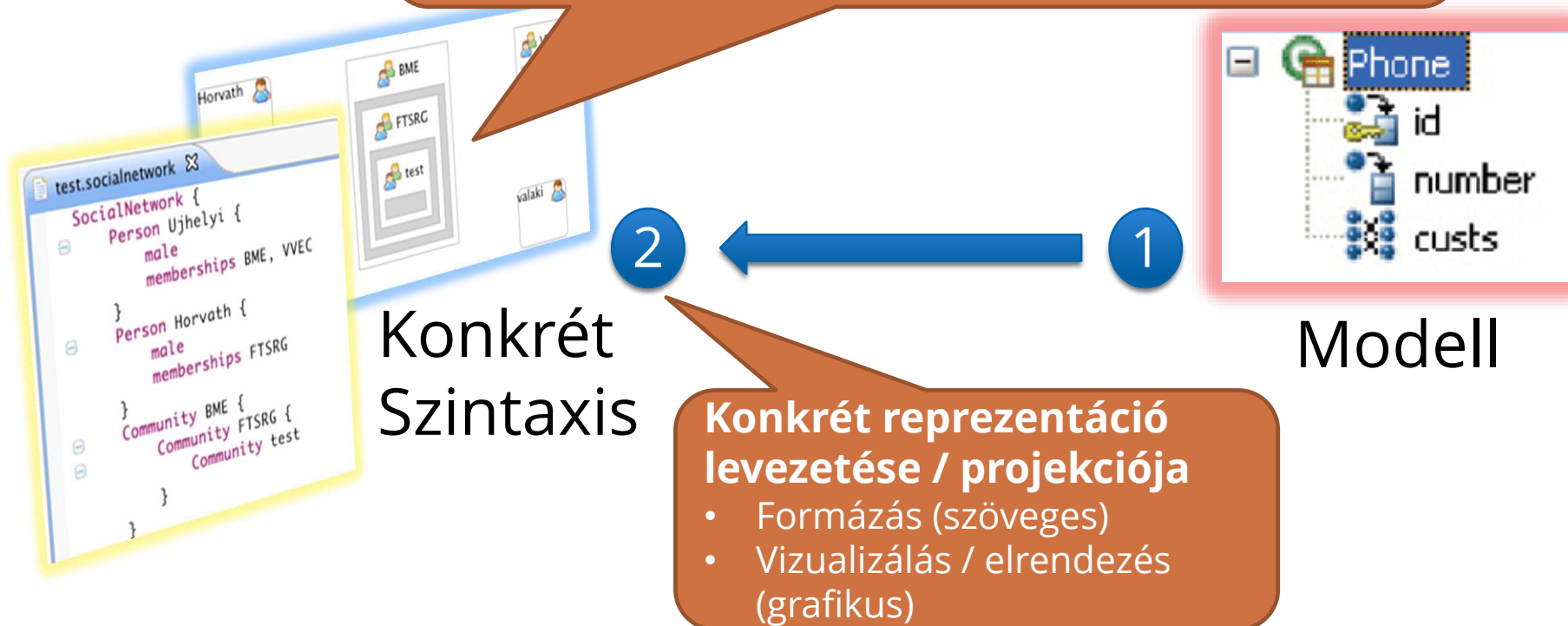
# Felesleges jelölési paraméterek

## ■ 1. Munkafolyamat: **projekciós szerkesztés**

### **Kötelező *jelölési paraméterek*:**

- Szóközök és megjegyzések stb. (szöveges)
- Elrendezés, elirányítás, méret, forma stb. (grafikus)

**...annak ellenére, hogy nem lényeges információ**



# Jelölési paraméterek levezetése

- Jelölési paraméterek lehetnek...

- > ...”beégetve” a projekció kódjába

- p.l. minden vonal fekete, minden betűméret 10pt (grafikus)
    - p.l. alkalmazza ezt a kód formázási sablont (szöveges)

- > ...levezetve domain információból

- p.l. forma a típus alapján, szín a láthatóság alapján meghatározva

## 1. Probléma:

Szerkeszthető paraméterek  
nem lehetnek a domain  
modell függvényei, tárolni kell

## 2. Probléma:

Néhány paraméternél nehéz  
lehet értelmes értéket adni  
p.l. pozíció a diagrammon

- > ...modellben tárolva



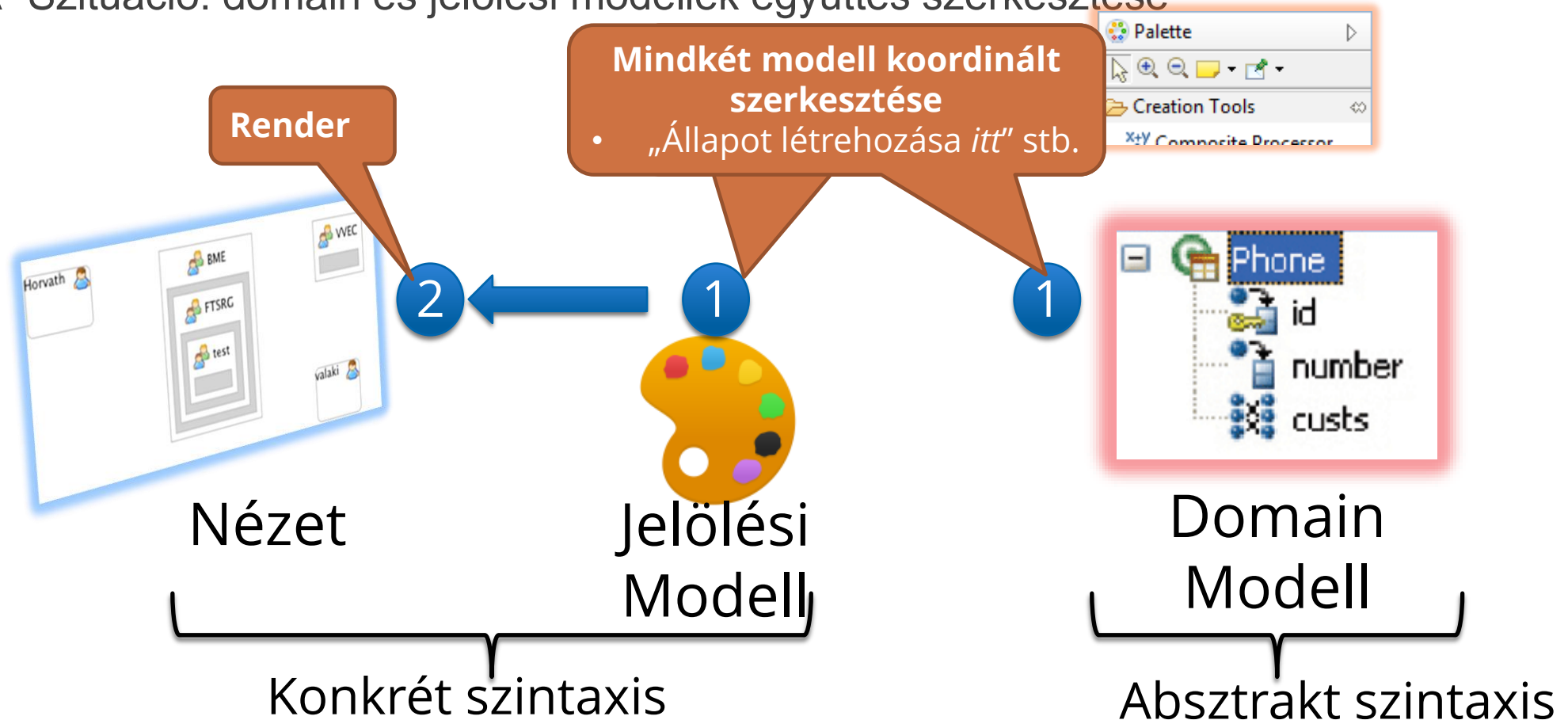
# Jelölés/nézet modellek

- Modell szétválasztása:
  - > Szakterületi / Szemantikai modell (absztrakt szintaxis)
  - > **Jelölés modell** (nézet modell): prezentációs állapot
    - Felhasználó lehet, hogy szerkesztheti
    - még mindig szükség van levezethető alapértékekre → lásd elrendezés
- Általános implementáció GMF-ben és Graphiti-ben
  - > Valójában EMF-en alapul
- Gyakran külső fájlokban tárolva
  - > Felelősségi körök szétválasztása
  - > P.I. kódgenerátort nem érdekli a nézeti információ

# Szerkesztési folyamat jelölési modellekkel

## ■ 1. Munkafolyamat: **projekciós szerkesztés**

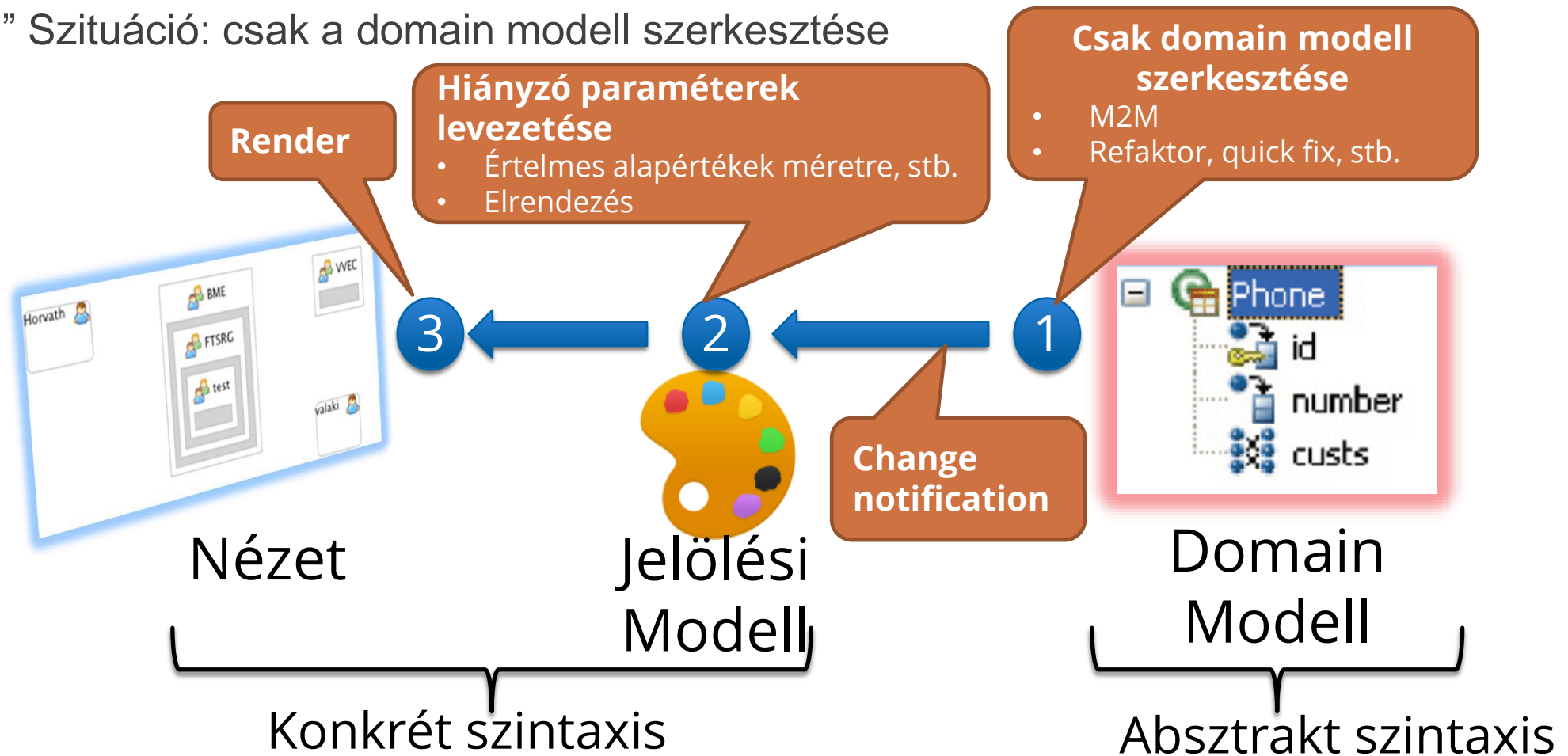
> „A” Szituáció: domain és jelölési modellek együttes szerkesztése



# Szerkesztési folyamat jelölési modellekkel

## ■ 1. Munkafolyamat: **projekciós szerkesztés**

> „B” Szituáció: csak a domain modell szerkesztése



# Szintaxis és Szemantika

## I. Kényszerek

## II. Konkrét és Absztrakt szintaxis

## III. Szerkesztők

## IV. Szemantika



# Szemantika

- Sokat beszéltünk a szintaxisról.
- Szemantika: a fogalmak jelentése egy nyelvben
  - > Statikus: mit jelent egy modell pillanatképe?
  - > Dinamikus: hogyan változik/fejlődik/viselkedik a modell?
- A modellező nyelv szemantikája a modelleket leképezi egy létező vagy matematikai szemantikai doménre.

# Axiomatikus szemantika

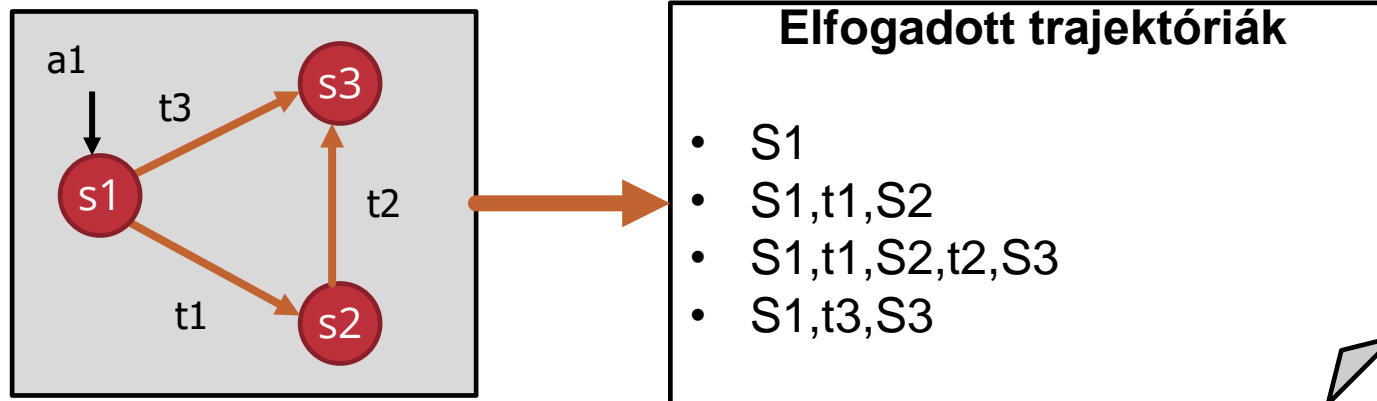
## ■ Statikus Szemantika

> A metamodellelemeneinek értelmezése: fogalmak jelentése az absztrakt szintaxisban

## ■ **Axiomatikus:** matematikai állítások az értelmezésről

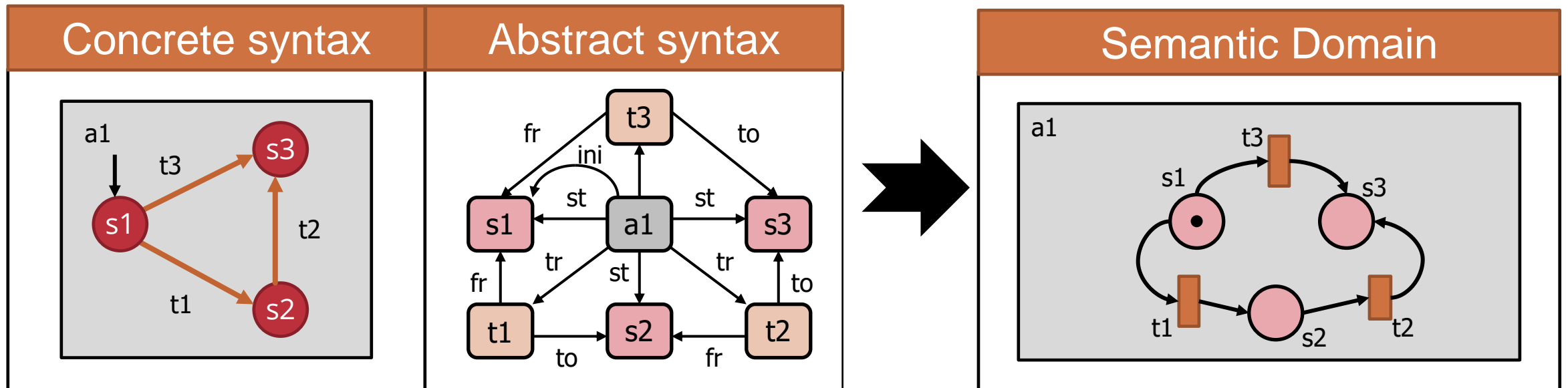
## ■ Matematikai módszerekkel fogalmazzuk meg a modell jelentését

## ■ PI: Egy modellhez egy halmazt rendelünk, két halmaz ugyanaz, a modellek egyeznek



# Denotációs szemantika

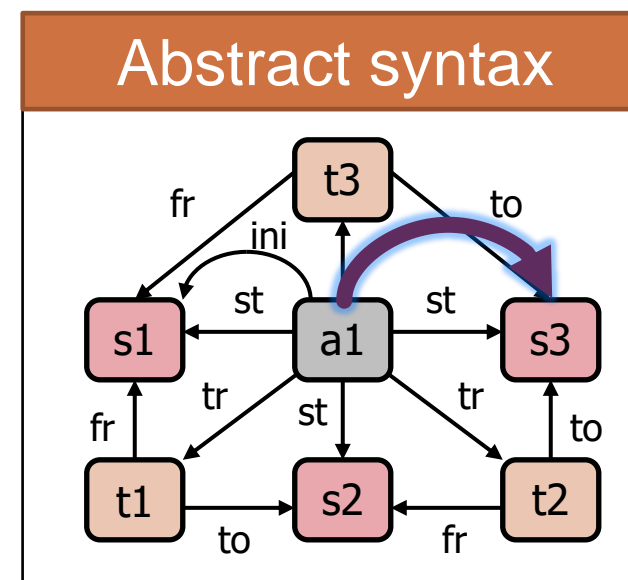
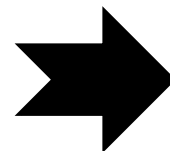
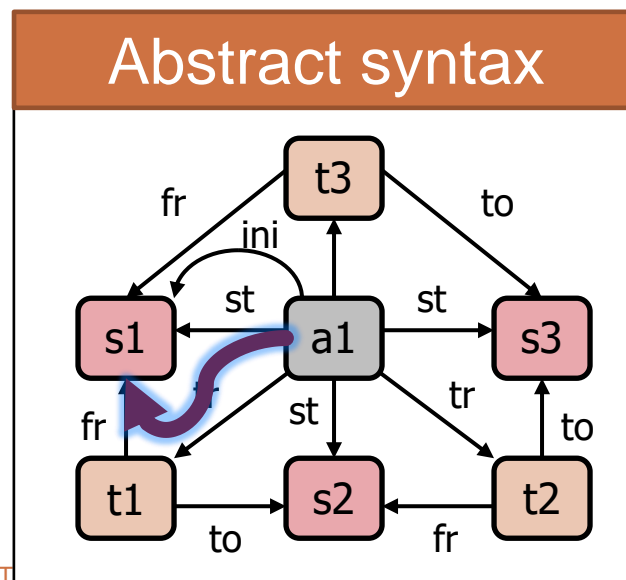
- fogalmak fordítása egyik nyelvből a másikba (**szemantikai domain-nek hívják**)
- „Fordítás”
- P.I. állapotgépek magyarázata Petri-hálóként



# Operációs szemantika

- A nyelvi fogalmak operációs viselkedésének modellezése
- „Interpretáció”
- P.I. a véges automata hogyan változhat állapotot futás közben
- Néha dinamikus funkciókat csak dinamikus szemantika formalizálására vezetnek be

Dinamikus  
él:  
current







Köszönöm a figyelmet!