



# Modellalapú szoftverfejlesztés

IX. előadás

Modelltranszformáció

Dr. Semeráth Oszkár

# Modelltranszformáció

**Alapfogalmak**

**Transzformációk láncolása**

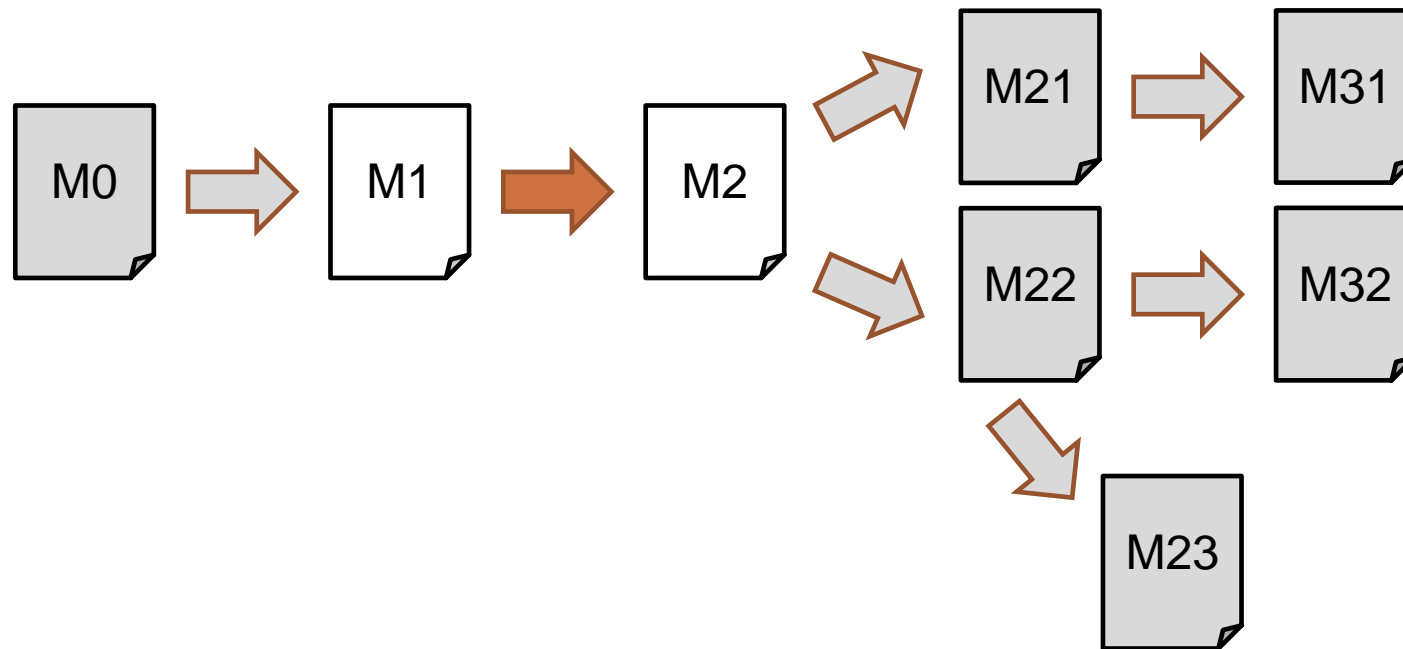
**Szabályalapú transzformációk**

**Technológiák**



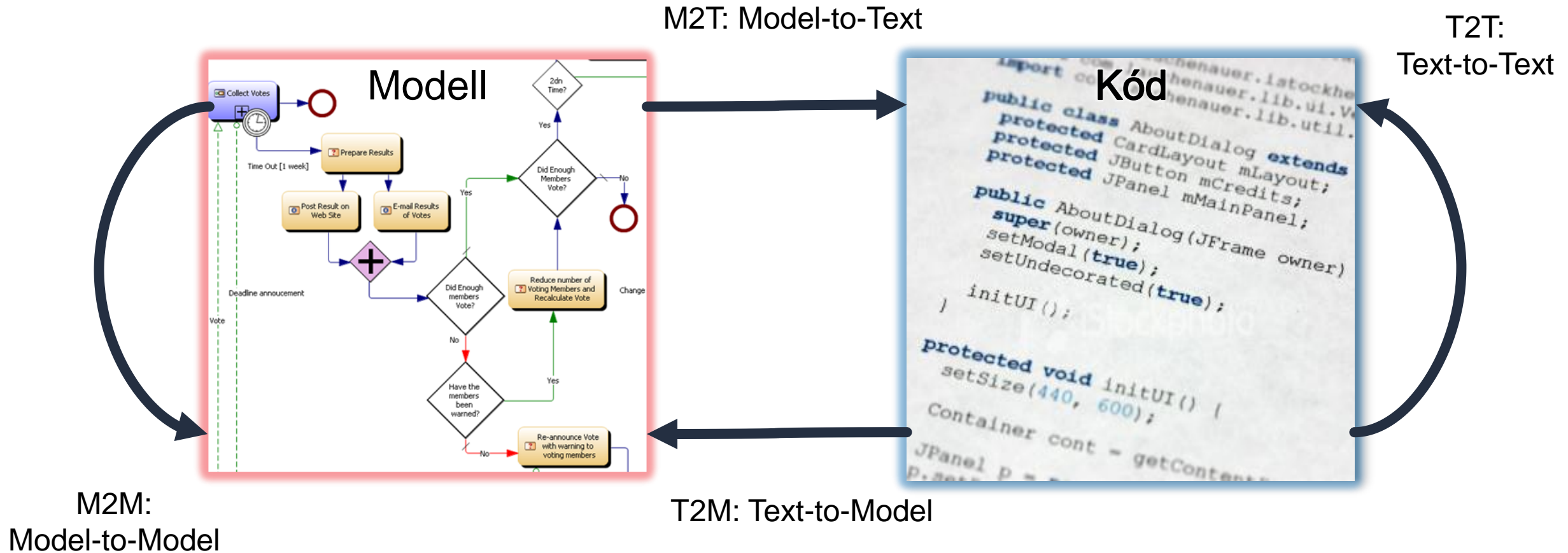
# Motiváció: Modellek transzformációja

- **Modellalapú fejlesztés:** Modellek az elsődleges dokumentumok
- Modelleket fejlesztünk, automatizáljuk a modellfeldolgozást



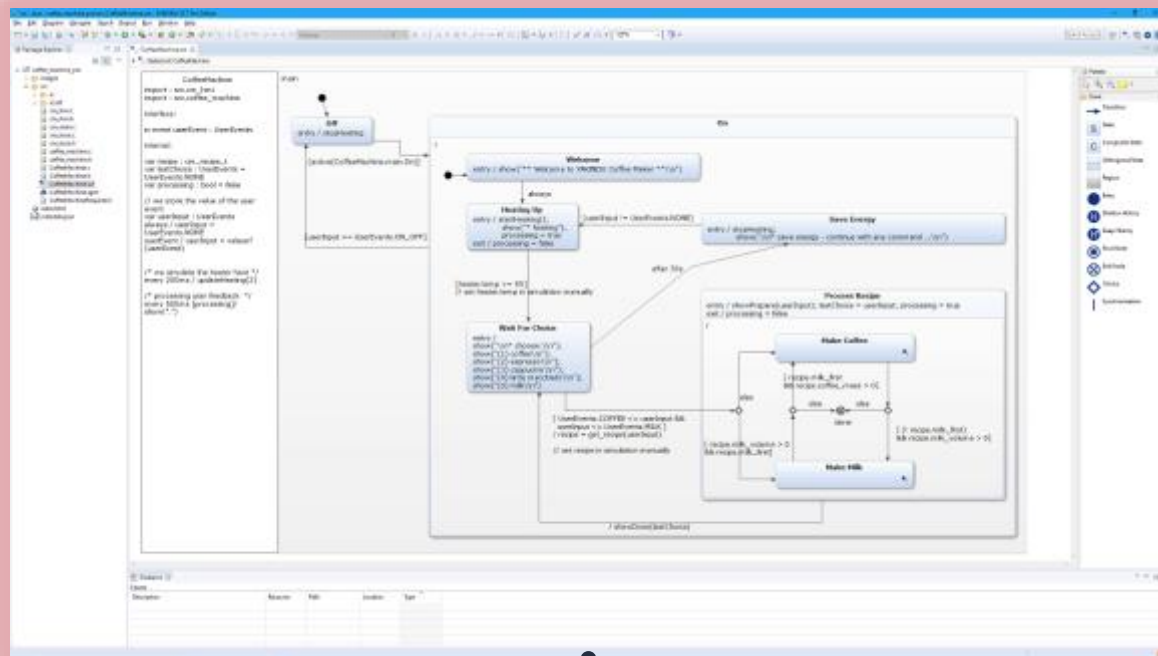
- **Cél:** modelltranszformációk hatékony megfogalmazása és végrehajtása

# Transzformációk fajtái



# M2T Példa

## ■ Kód generálás



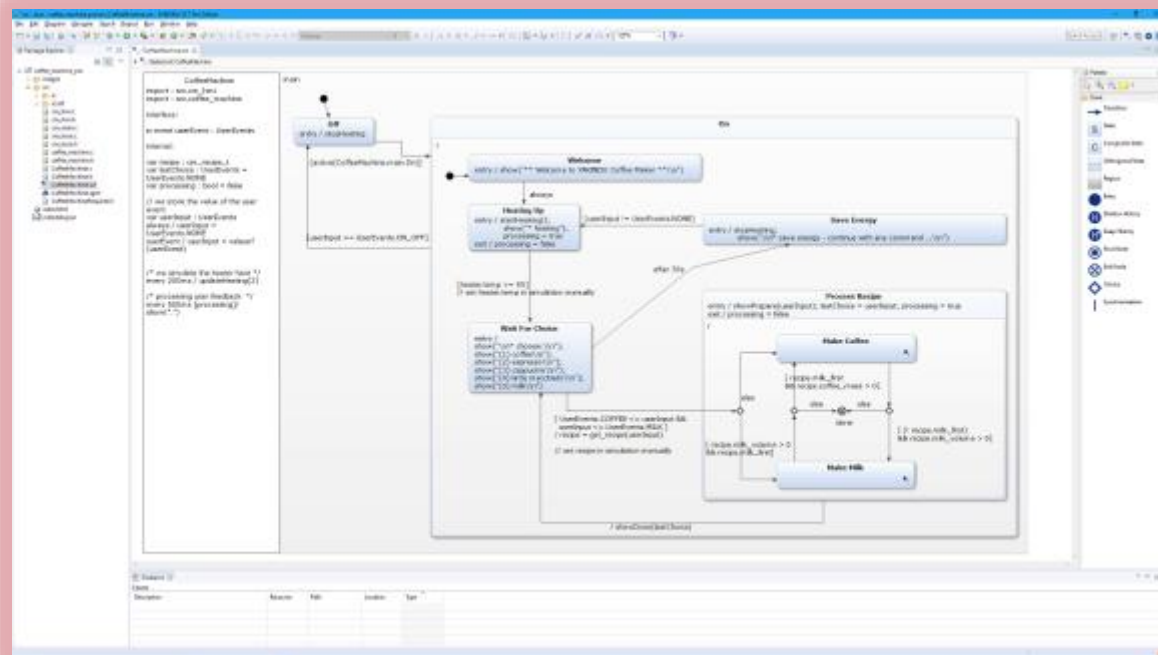
```
#ifndef DEFAULTSM_H_
#define DEFAULTSM_H_
#include "sc_types.h"
#include "StatechartInterface.h"

class DefaultSM : public StatechartInterface
{
public:
    DefaultSM();
    ~DefaultSM();
    /*! Enumeration of all states */
    typedef enum
    {
        main_region.MyState,
        DefaultSM_last_state
    } DefaultSMStates;
    /*! Inner class for Sample interface scope.
    class SCI_Sample
    {
    public:
        /*! Gets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
        sc_boolean get_a();
        /*! Sets the value of the variable 'a' that is defined in the interface scope 'Sample'. */
        void set_a(sc_boolean value);
        /*! Raises the in event 'evA' that is defined in the interface scope 'Sample'. */
        void raise_evA(sc_boolean value);
        /*! Checks if the out event 'evB' that is defined in the interface scope 'Sample' has been raised. */
        sc_boolean isRaised_evB();
        /*! Gets the value of the out event 'evB' that is defined in the interface scope 'Sample'. */
        sc_integer get_evB_value();
    private:
        friend class DefaultSM;
        sc_boolean a;
        sc_boolean evA_raised;
        sc_boolean evA_value;
        sc_boolean evB_raised;
        sc_integer evB_value;
    };
    /*! Returns an instance of the interface class 'SCI_Sample'. */
    SCI_Sample* getSCI_Sample();
    void init();
    void enter();
    void exit();
    void runCycle();
    sc_boolean isActive();
    sc_boolean isFinal();
    sc_boolean isStateActive(DefaultSMStates state);
private:
    static const sc_integer maxOrthogonalStates = 1;
    DefaultSMStates stateConfVector[maxOrthogonalStates];
    sc_ushort stateConfVectorPosition;
};
```



# T2M Példa

- Forráskóddal megfogalmazott kódrészletek importálása



```
class Point
{
public:
    int32_t get_x();
    void set_x(int32_t x);
    int32_t get_y();
    void set_y(int32_t y);
private:
    int32_t x;
    int32_t y;
};
```

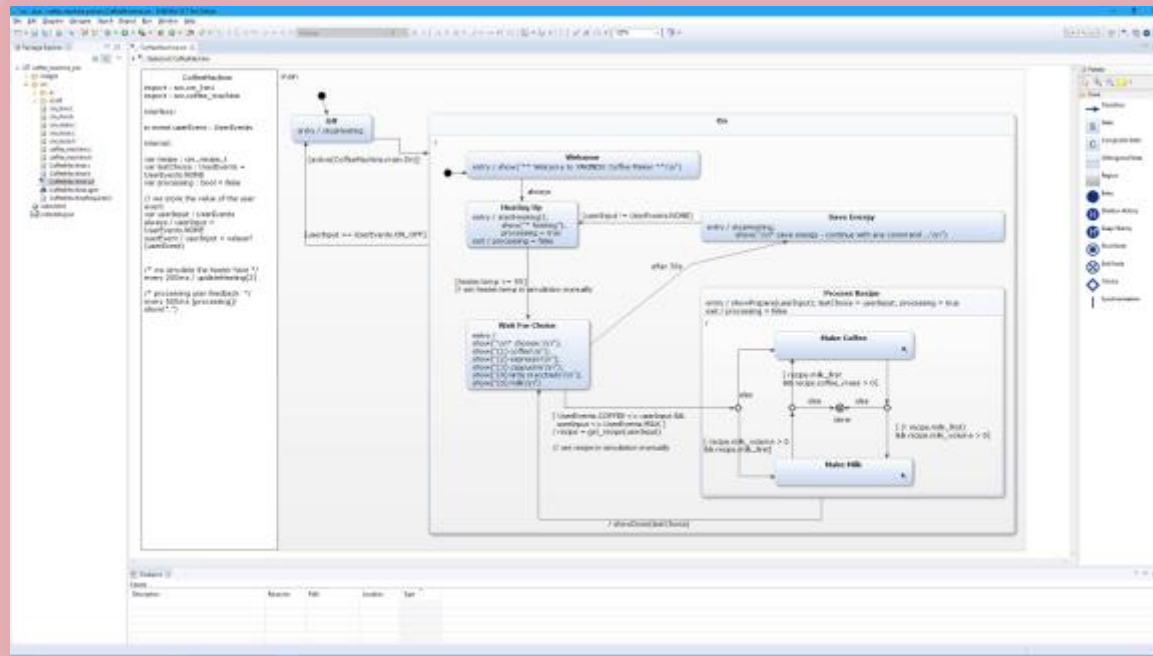


YAKINDU STATECHART TOOLS



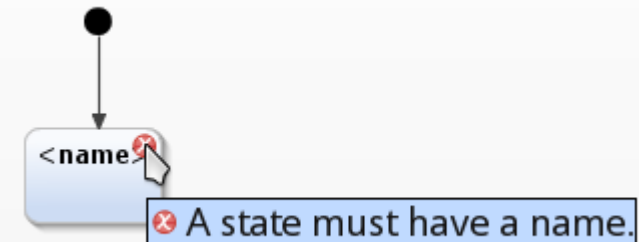
# 1. M2M Példa

- M2M: Modell validálás: hibaminta → hibaüzenet



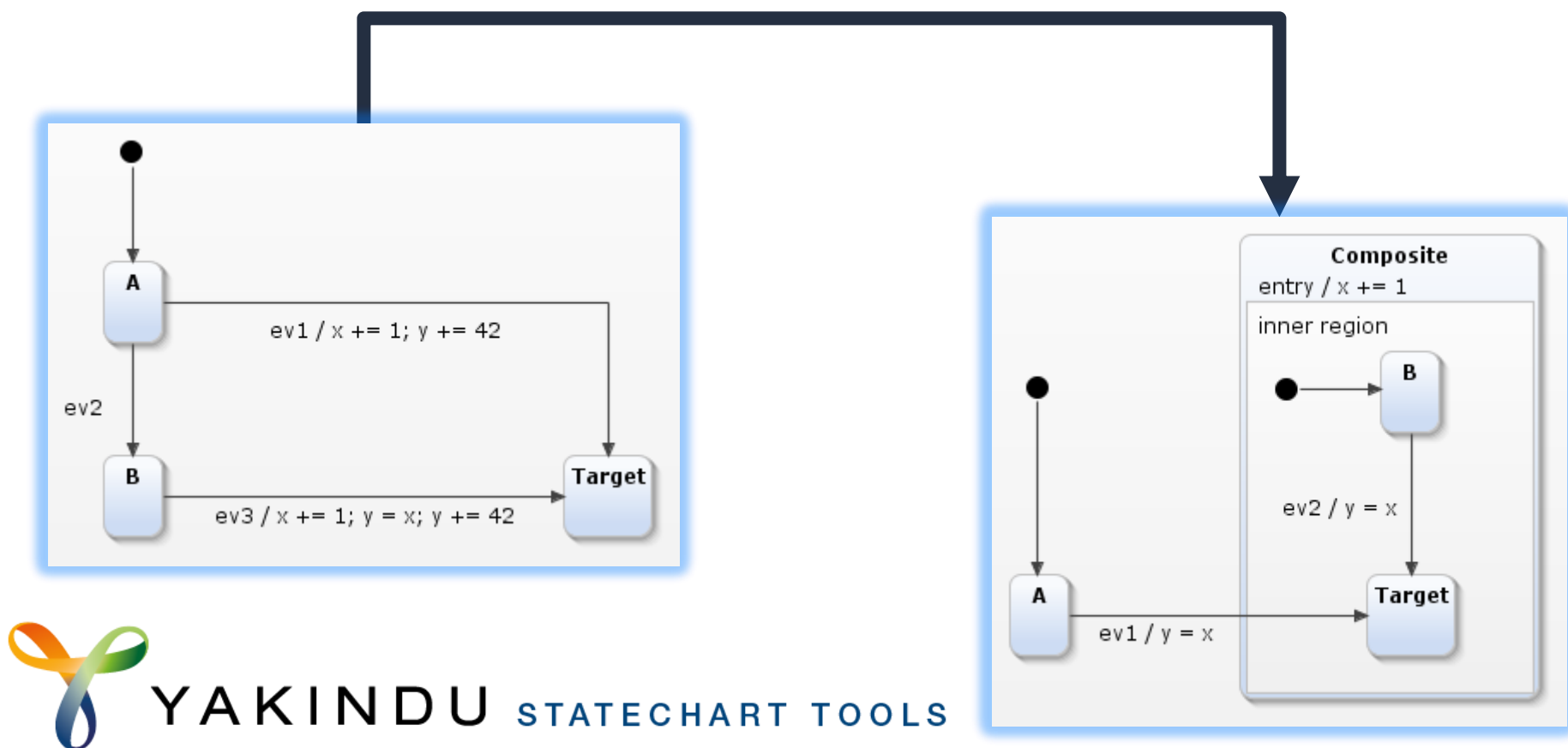
Vessük össze az OCL-lel:

- Nincs hibaüzenet
- Egyetlen objektumhoz van kötve
- Pozitívan van megfogalmazva (mikor helyes vs mikor hibás)
- Mikor kell futtatni
- Precíz szemantika
- Teljesítmény



## 2. M2M Példa

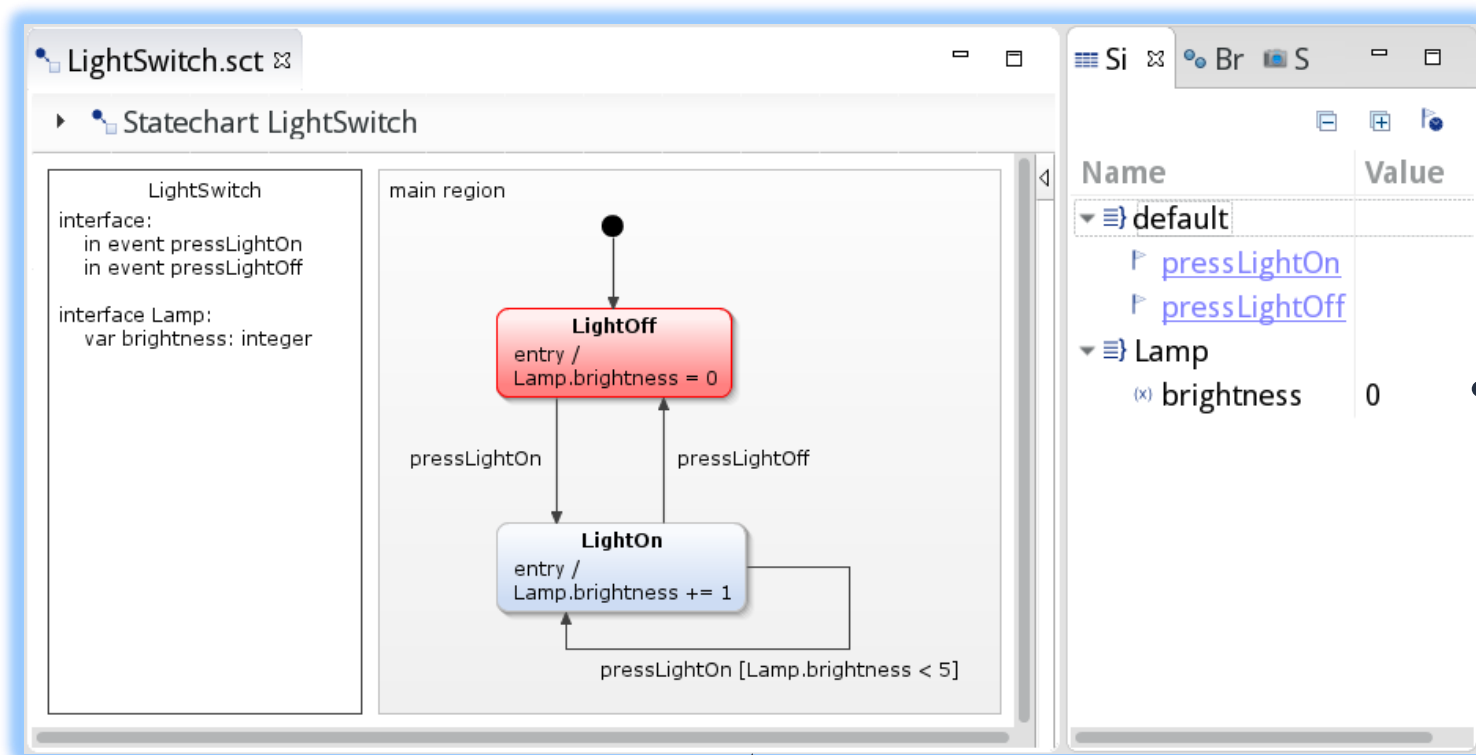
- Modellek refaktorálása





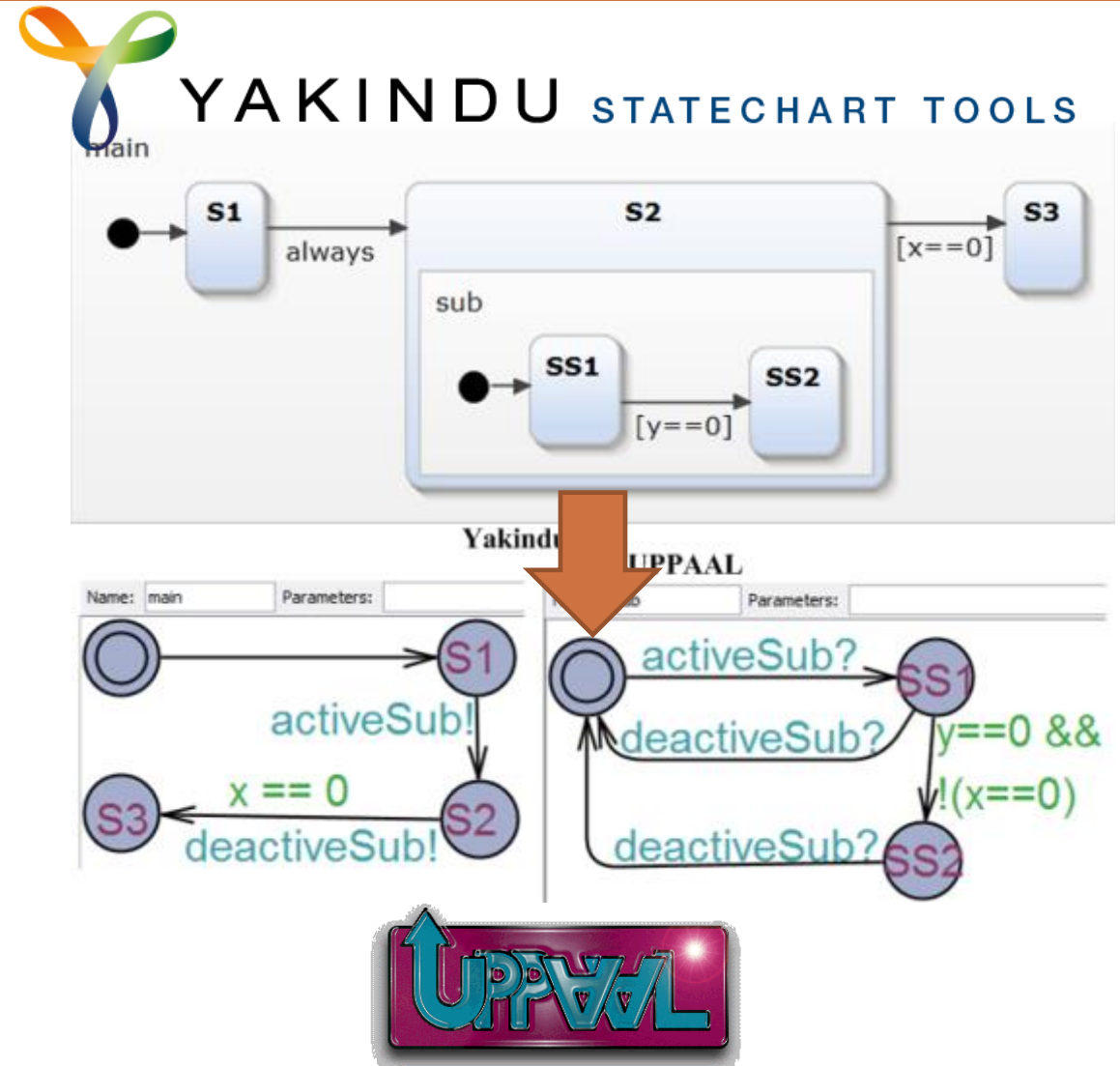
## 3. M2M Példa

- Szimuláció
- Szemantika

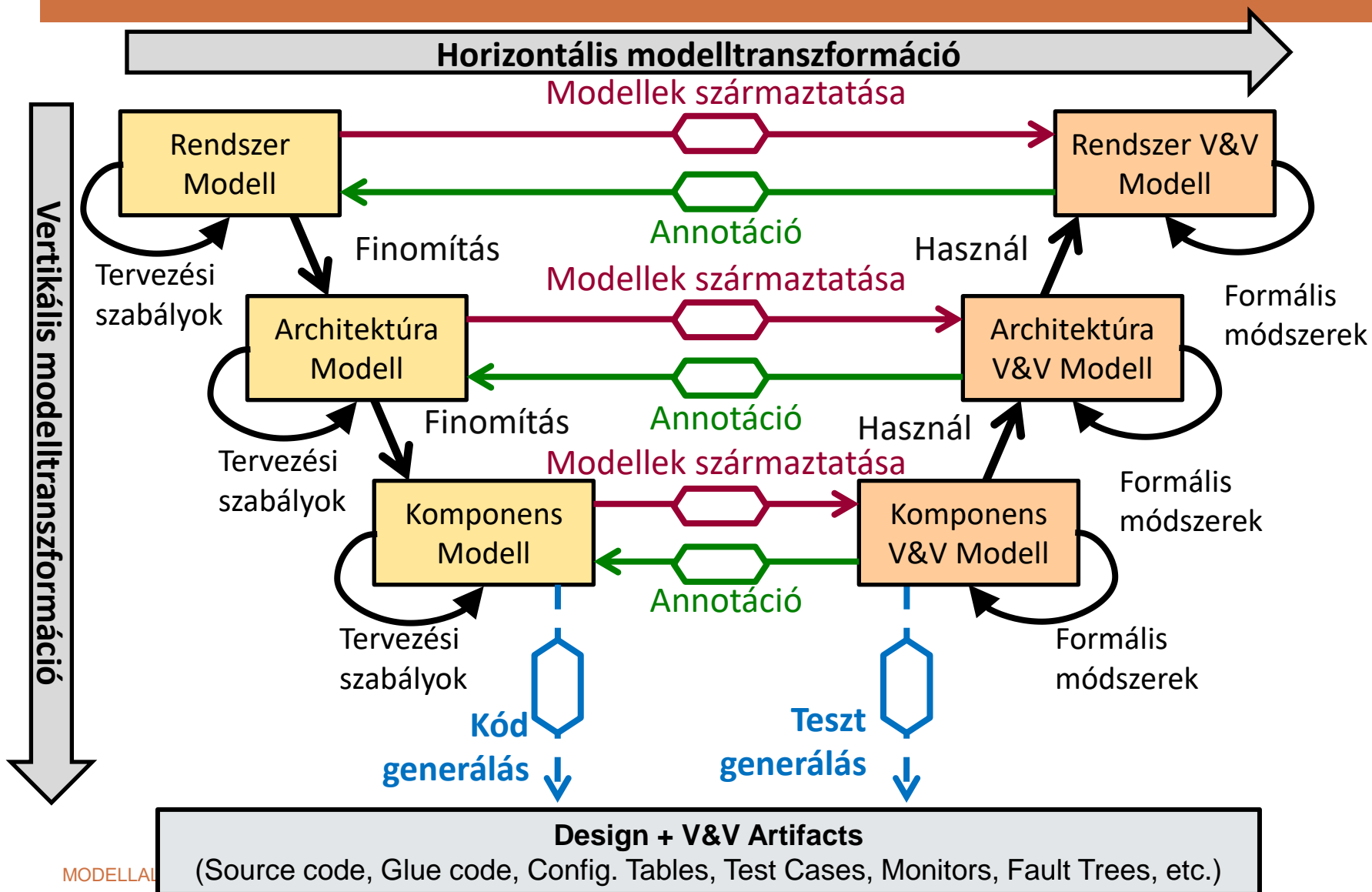


## 4. M2M Példa

- Formális módszerek alkalmazása
- Rejtett formális módszerek: Speciális szakértelem nélkül alkalmazható algoritmusok
  - > Eszköztámogatás
  - > Eredmények visszavetítése



# Modellek és Transzformációk kritikus rendszerek fejlesztésében



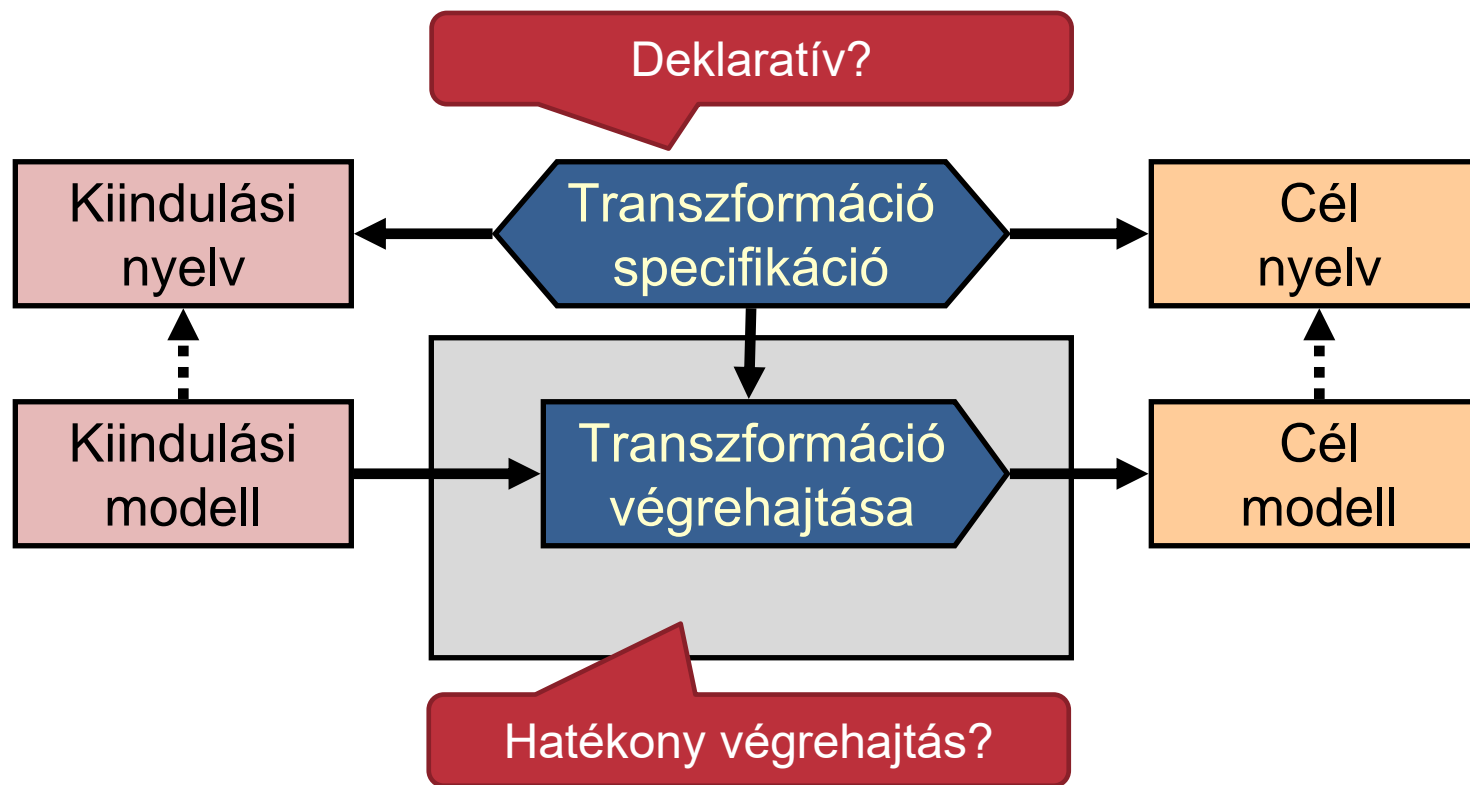
## Modelltranszformációk:

- Szakértelem reprezentációjának alapja:

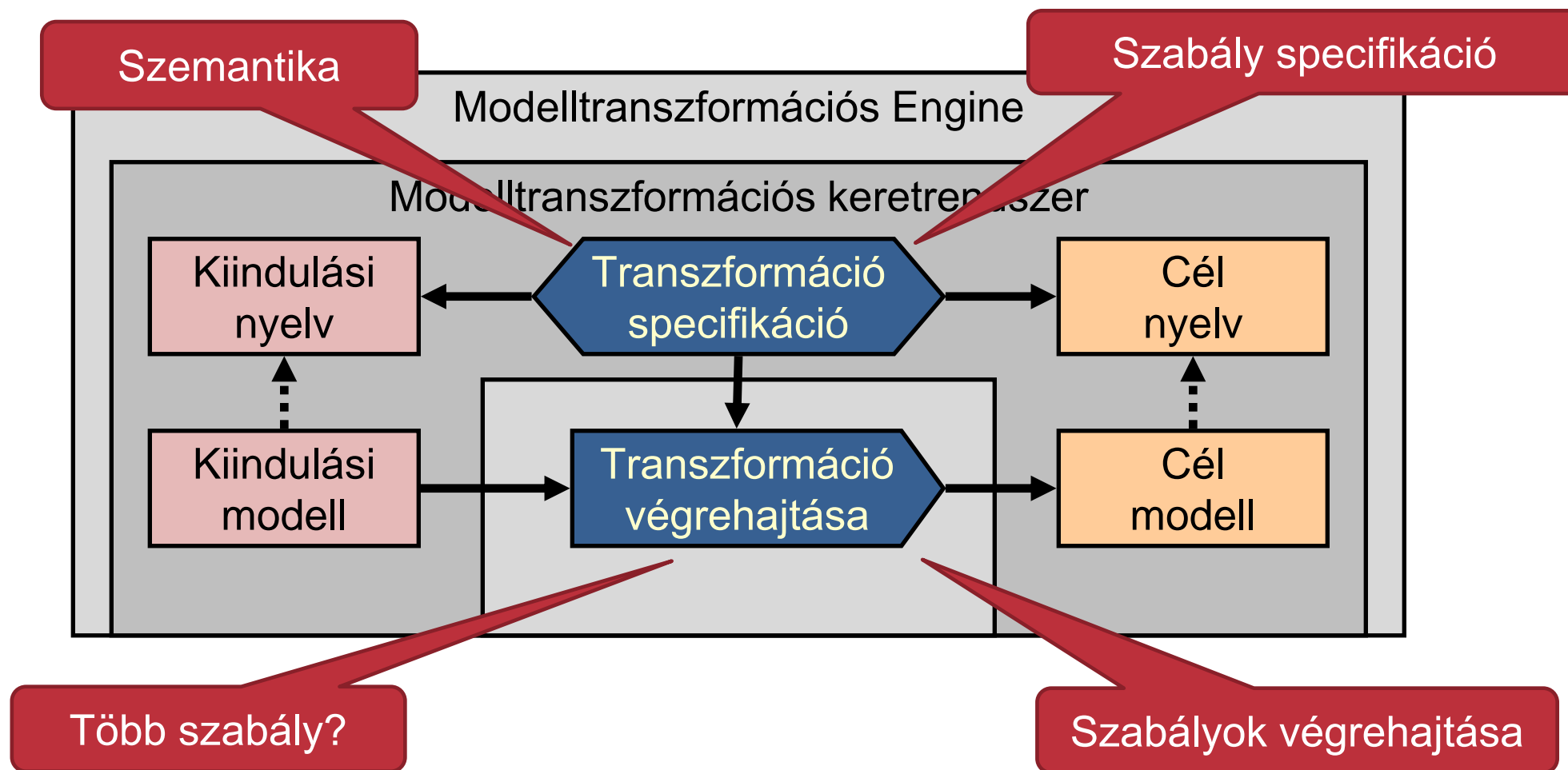
*Elméleti eredmények → Eszközök*

- Modellezési nyelvek és eszközök összekötése

# Definíciók



# Definíciók és Kérdések

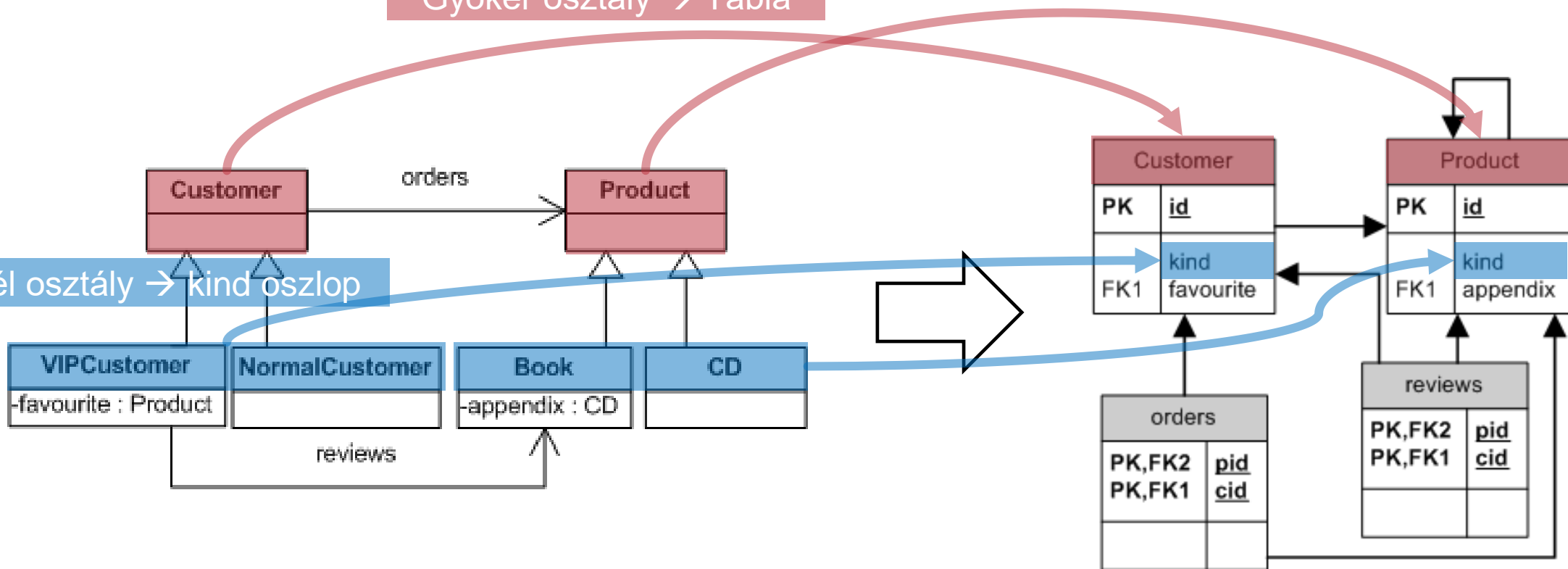


# Példa leképezés: ORM

- Tipikus példa: képezzünk le egy osztálydiagramot adatbázis táblákra!

Gyökér osztály → Tábla

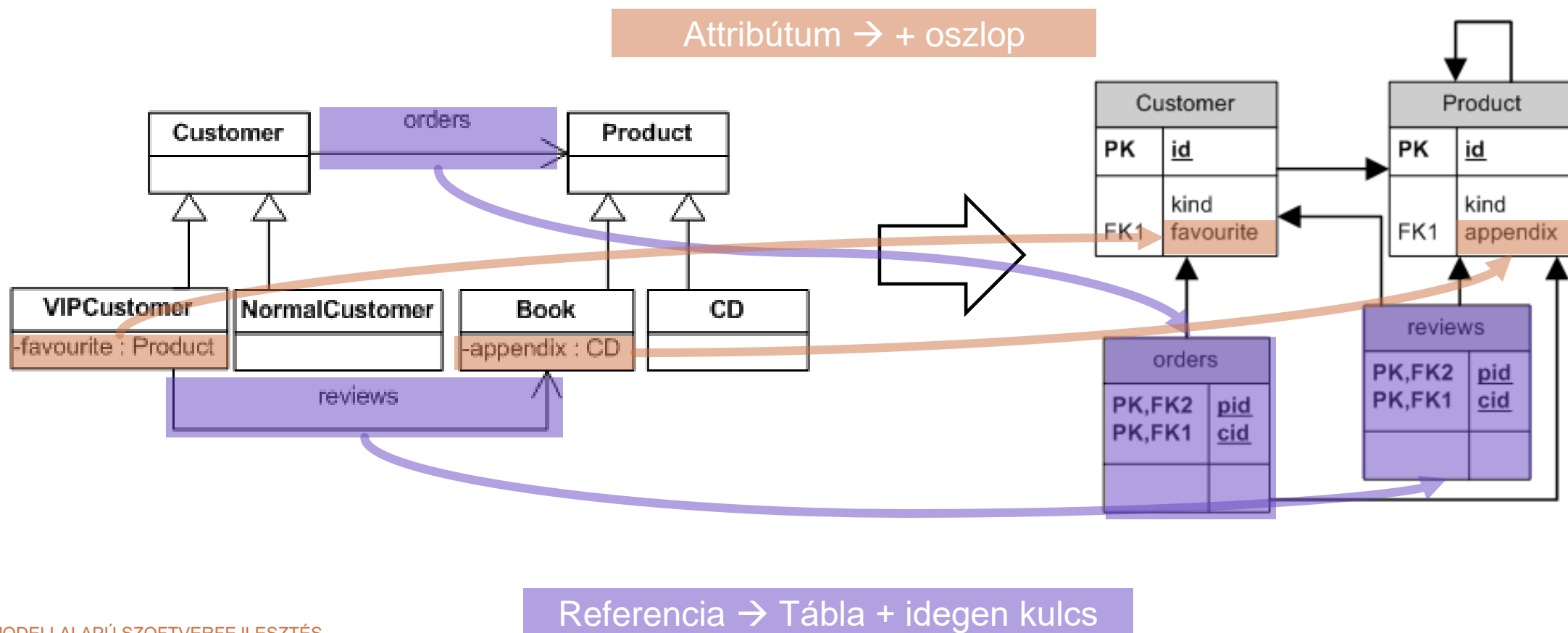
Levél osztály → kind oszlop





# Példa leképezés: ORM

- Tipikus példa: képezzünk le egy osztálydiagramot adatbázis táblákra!

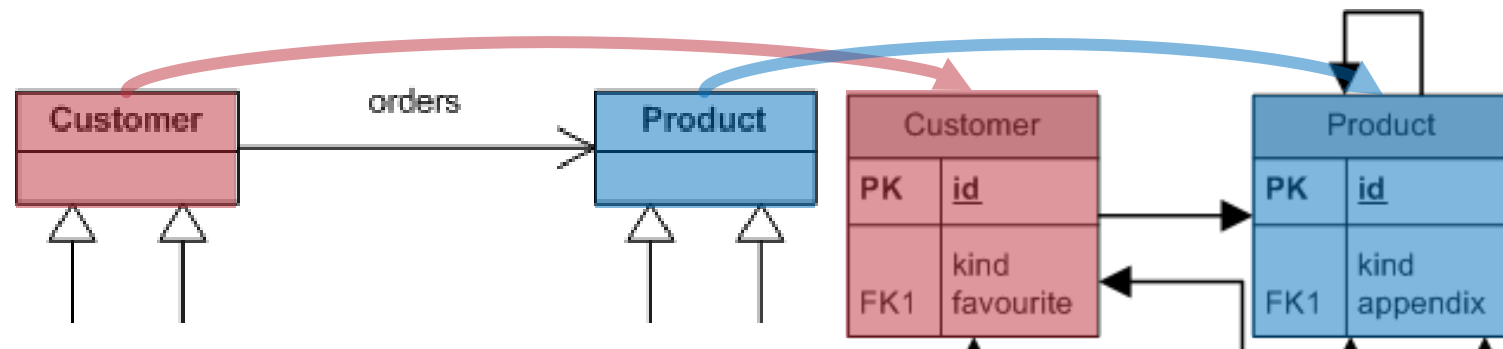
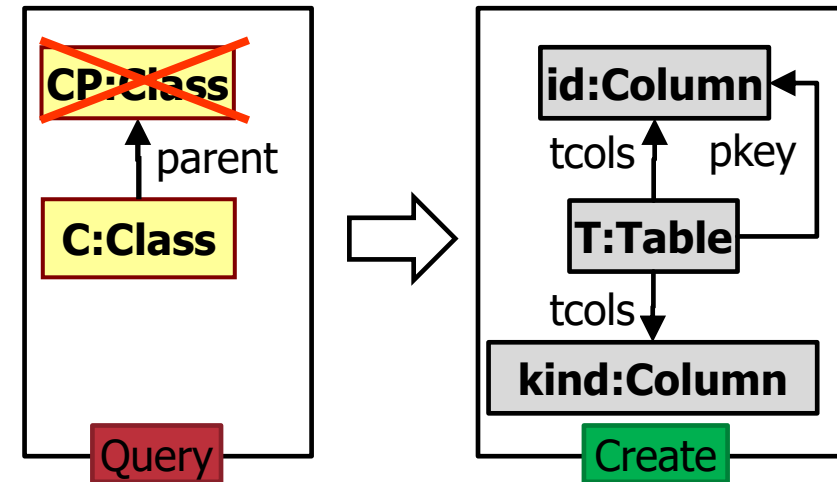


# Példa Transzformáció

- Hogyan oldanánk a gyöker osztályokat reprezentáló táblák létrehozását?

1. Lekérdeznénk a gyöker osztályokat (osztály, aminek nincs őse)
2. Létrehoznánk a táblákat, és velük a szükséges oszlopokat
3. Ismételtnénk amíg tudjuk

- Cél: Hasonló szabályokkal megfogalmazni az egész transzformációt



# Modelltranszformáció

**Alapfogalmak**

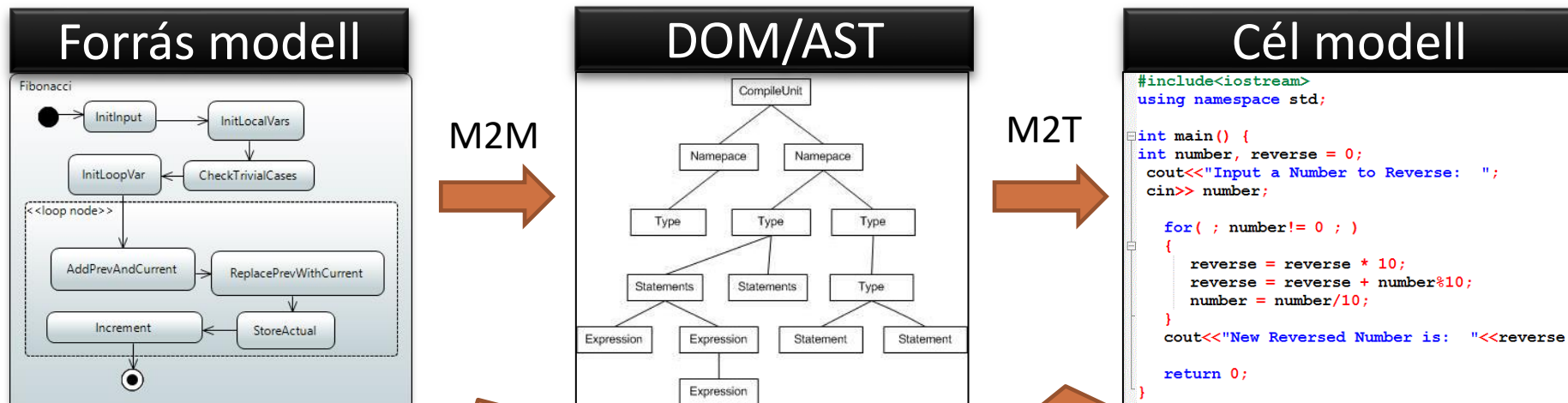
**Transzformációk láncolása**

**Szabályalapú transzformációk**

**Technológiák**



# Kódgenerálás modelltranszformációkkal



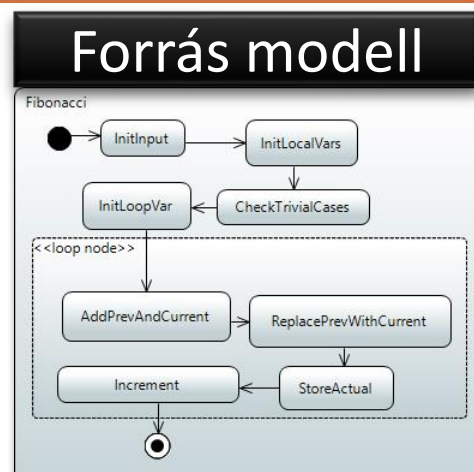
## Model-to-Model (M2M)

- SRC: In-memory modell (objektumok)
- TRG: In-memory modell (objektumok)

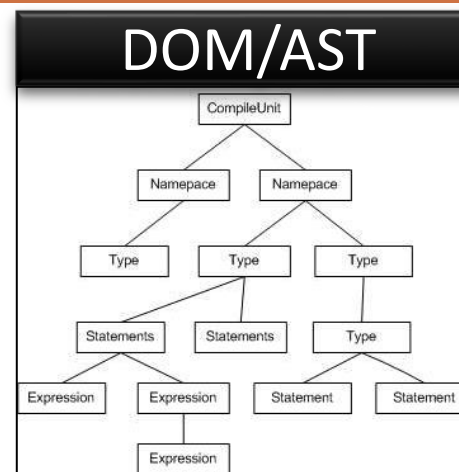
## Model-to-Text (M2T)

- SRC: In-memory modell (objektumok)
- TRG: szöveges kimenet (string)

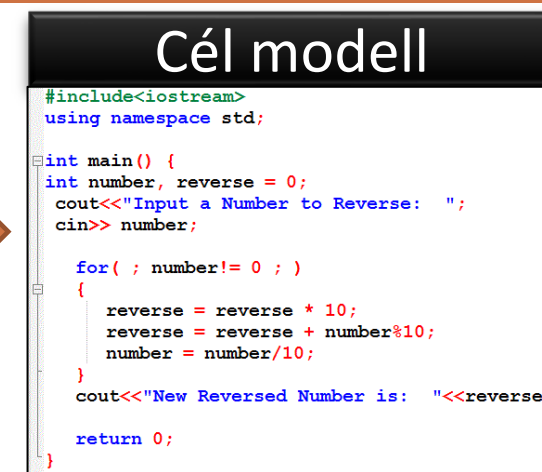
# Modell transzformációk láncolása



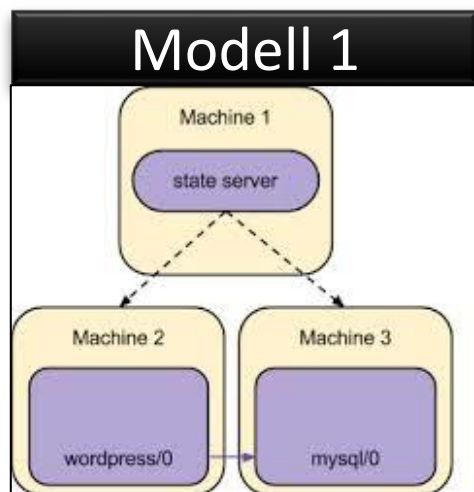
M2M



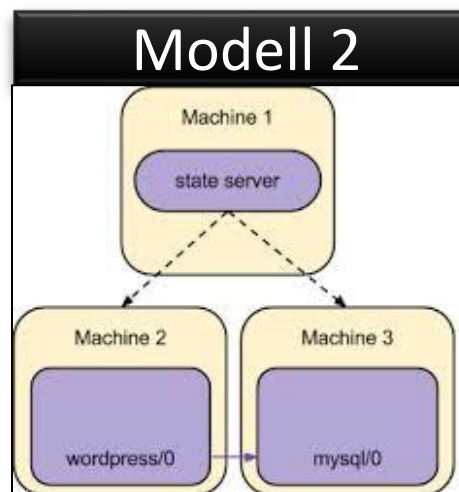
M2T



M2M



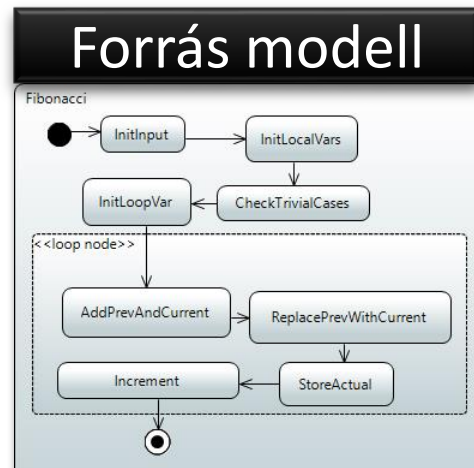
M2M



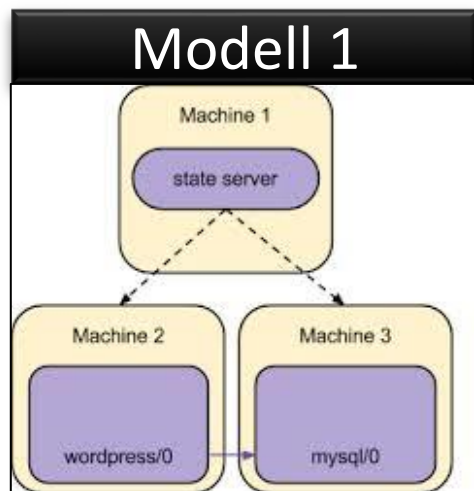
**Cél:**

- Absztrakciós szakadék csökkentése
- „Oszd meg és uralkodj”
- Közbenső modellek újrahasználása

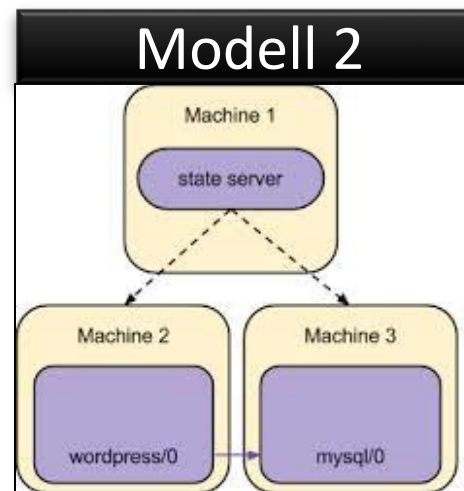
# Modell transzformációk láncolása



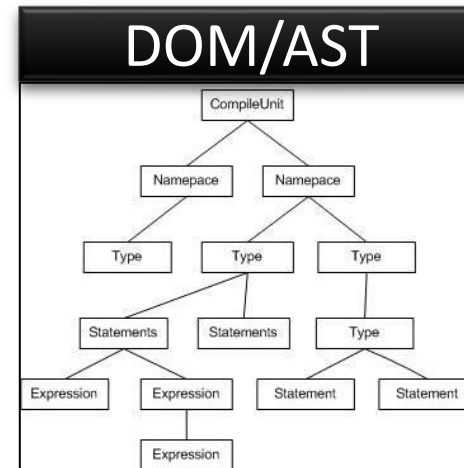
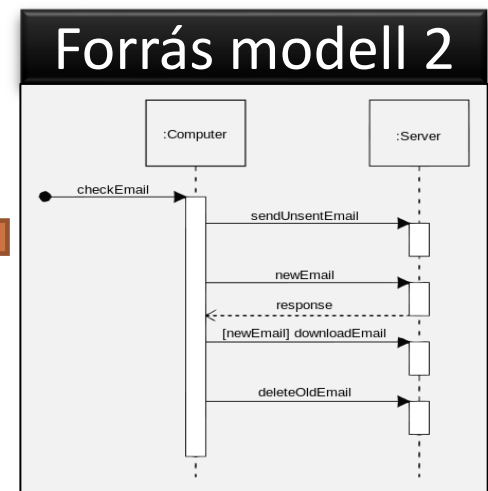
M2M



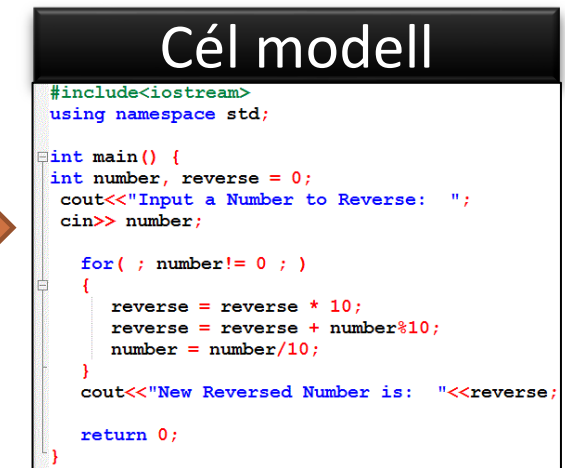
M2M



M2M



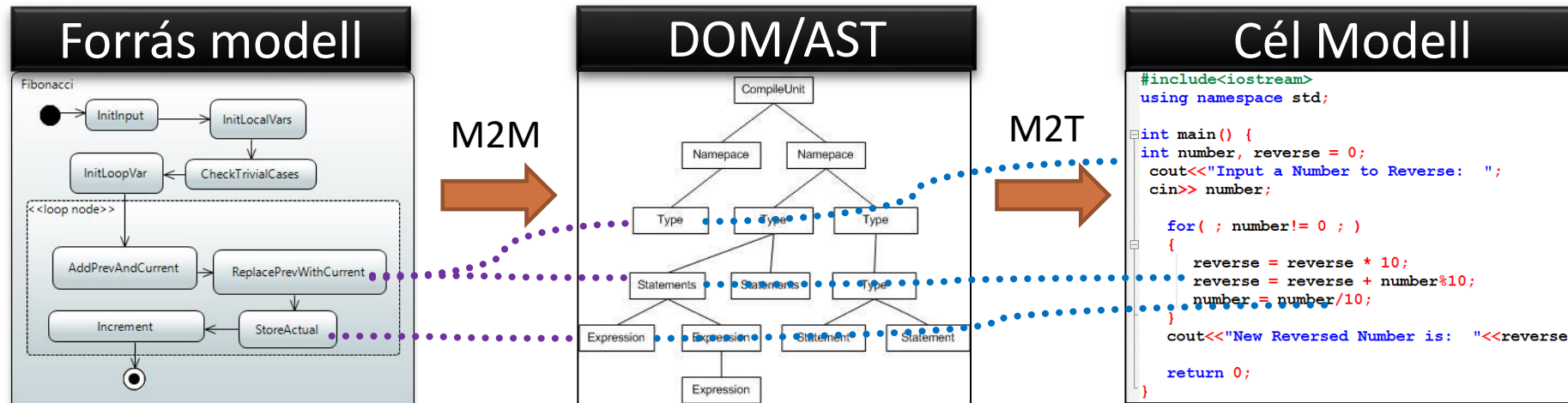
M2T



Közös optimalizálás

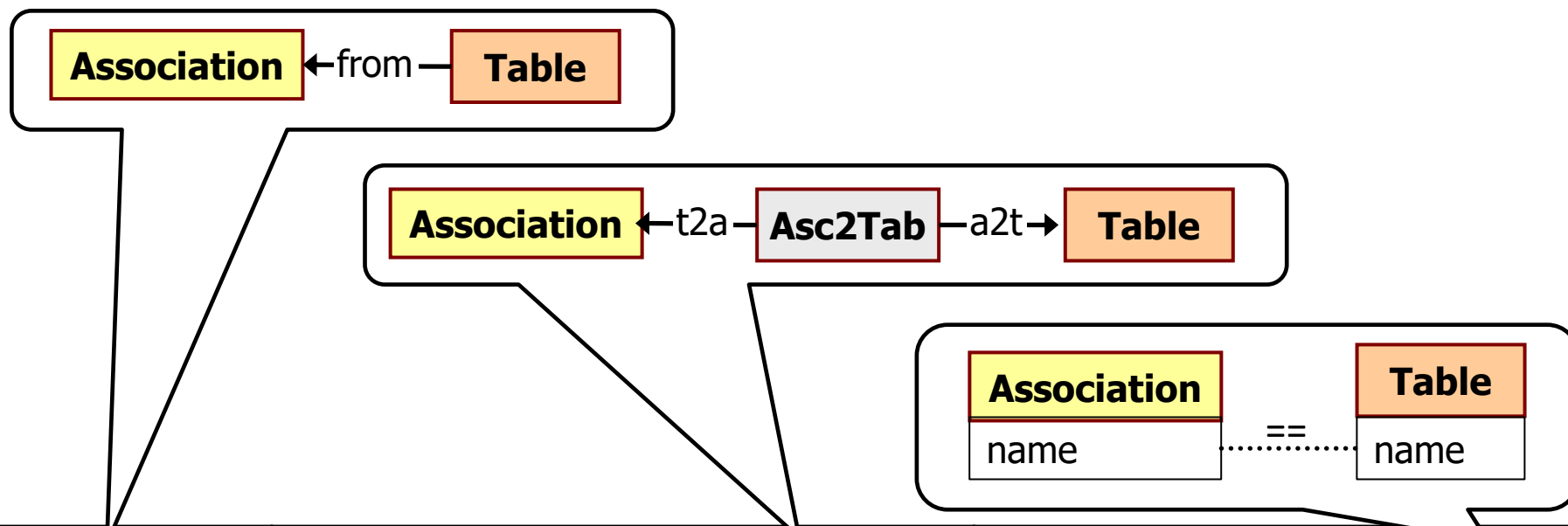


# Nyomonkövethetőség modelltranszformációk során



- **Nyomonkövethetőségi összeköttetések (traceability links):** összeköti a forrás és cél modell elemeit
- Egyszerűbbé teszi a transzformációk specifikációját
- Összeköti az összes modellt
- Lehetővé teszi az inkrementális végrehajtást

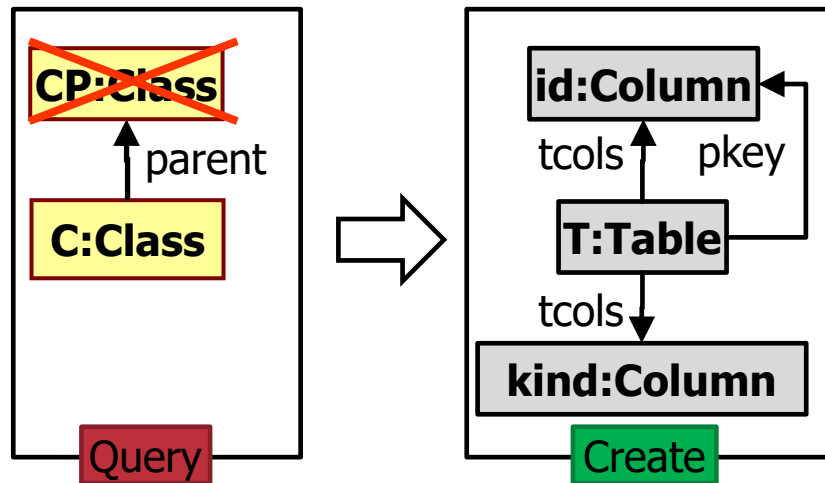
# Nyomonkövethetőség fajtái



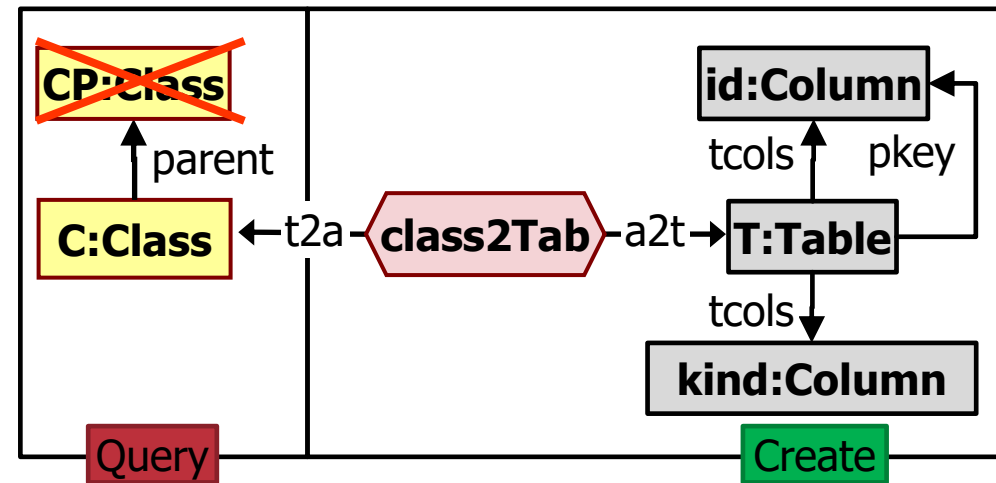
| Direkt referenciák                                      | Nyomonkövethetőségi modell | Soft referenciák   |
|---|----------------------------|--|
| SRC↔TRG<br>kereszthivatkozások                          | Új metamodel és modellek   | Logikai kapcsolat: id / név /<br>lekérdezés / modell index alapján                 |
| Intrúzív: metamodelt<br>és példánymodell<br>kiegészítse | Összetett, nagy overhead   | Azonosítás szükséges;<br>limitált kifejezőerő<br>(nem mondja el, ha nem található) |

# Példa: ORM

- Hogyan kössük össze a generált táblákat az osztályokkal nyomonkövethetőségi modellel?



Honnan tudjuk, hogy egy osztály kész van?



# Modelltranszformáció

Alapfogalmak

Transzformációk láncolása

Szabályalapú transzformációk

Technológiák



# Modelltranszformációk specifikációja

## ■ Imperatív: direkt modell manipulálás

- > Gyors és egyszerűen elkezdhető
- > De mi van akkor, ha valami bonyolult kell?
  - Bonyolultabb szabályok, újabb esetek?
  - Inkrementalitás?
  - Kétirányú modelltranszformáció?

## ■ Deklaratív, szabály alapú

- > *Gráftranszformációk*
- > Hybrid: lekérdezés + imperatív végrehajtás (pl. VIATRA)
- > „Relációs” (pl. QVT-R, TGG, ATL)

# Szabály alapú modelltranszformációk

## ■ Egység: MT szabály

| Minden előfordulásra...                                      | Alakítsa át...                                       |
|--|--|
| Öröklési hierarchia gyökéreleme                              | Készíts egy táblát                                   |
| Osztály attribútuma  | Adj hozzá egy oszlopot                               |
| Asszociáció osztályok között                                 | Készíts táblát idegen kulcsokkal                     |
| <b>I. Prekondíció</b><br><b>Deklaratív modell lekérdezés</b> | <b>II. Akció</b><br><b>Deklaratív vagy imperatív</b> |



# Szabály alapú rendszerek

Hol láthatunk hasonlót?

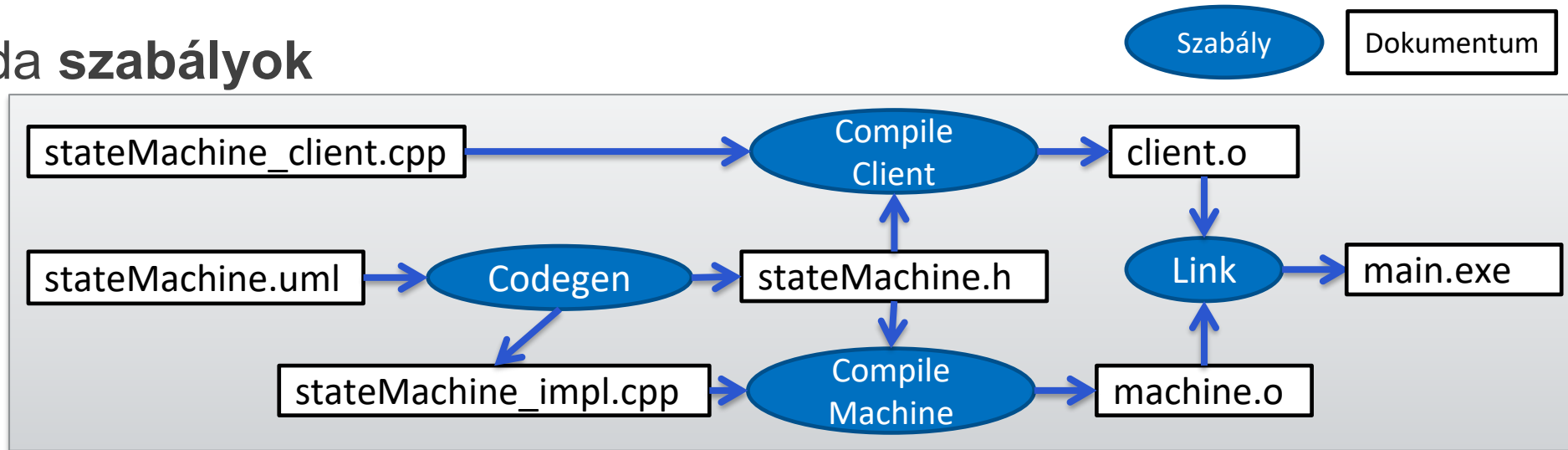
- **Modelltranszformációk**
- Build szkriptek (MAKEFILE, Maven, gradle etc.)
  - > Szabály: építsd meg a specifikáció alapján ezt az állományt (*akció*),
  - > Amikor az összes szükséges állomány készen áll (*prekondíció*)
- Üzleti szabályok és szakértői rendszerek (Jboss Drools, etc.)
- Nyelvtani szabályok (*lásd szöveges szintaxis előadások*)
- CSS

# Inversion of Control (IoC)

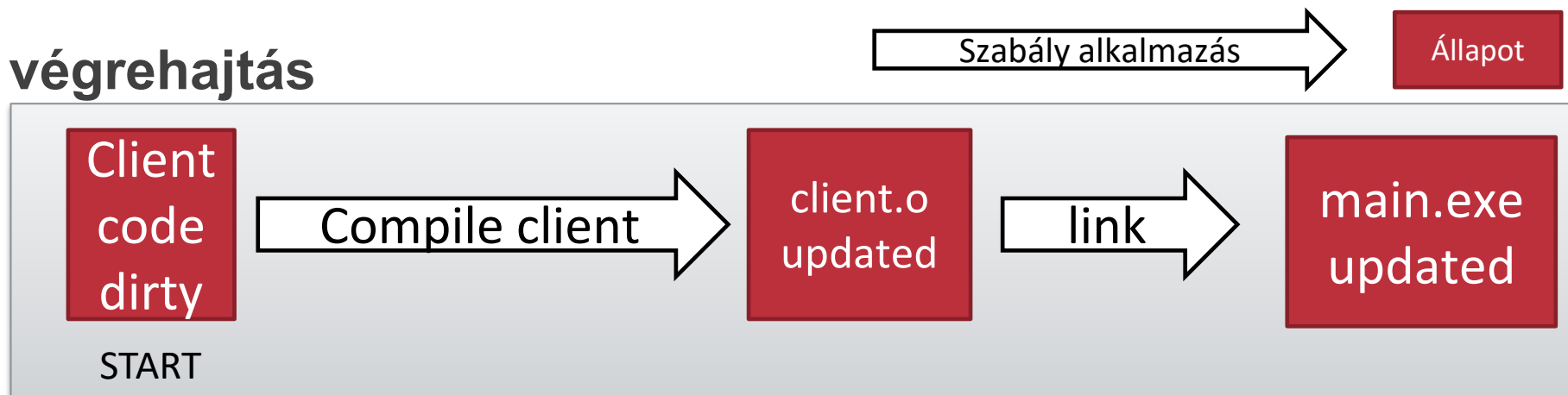
- Deklaratív szabályok végrehajtása
  - > **Transzformációs motor** interpretálja az előfeltételt
  - > Szabályokat **eltűzeli** amikor és ahol engedélyezettek
- Számos végrehajtási szemantika
  - > „As long as possible” (amíg lehetséges) / „fire when possible” (amikor lehetséges)
    - Iteráljunk amíg van szabálynak **aktivációja**
    - Válasszunk ki egy aktivációt (**conflict resolution**)
    - Tűzeljük el
  - > „Fire all current”
    - Keressük meg az összes *jelenlegi* aktivációt,
    - Tűzeljük el
  - > Tetszőleges vezérlés

# Build Szkript példa

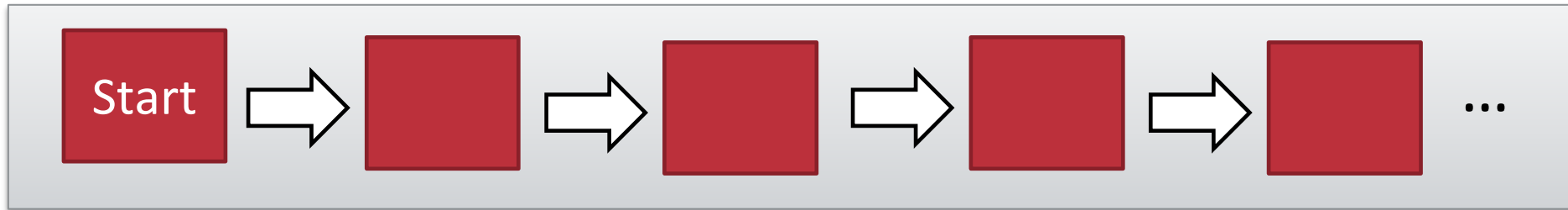
## ■ Példa szabályok



## ■ Példa végrehajtás



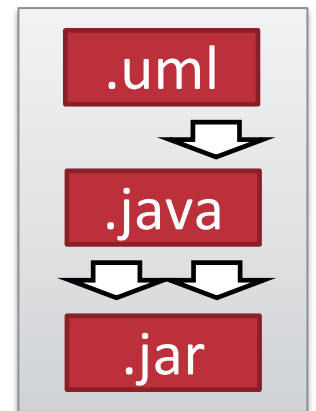
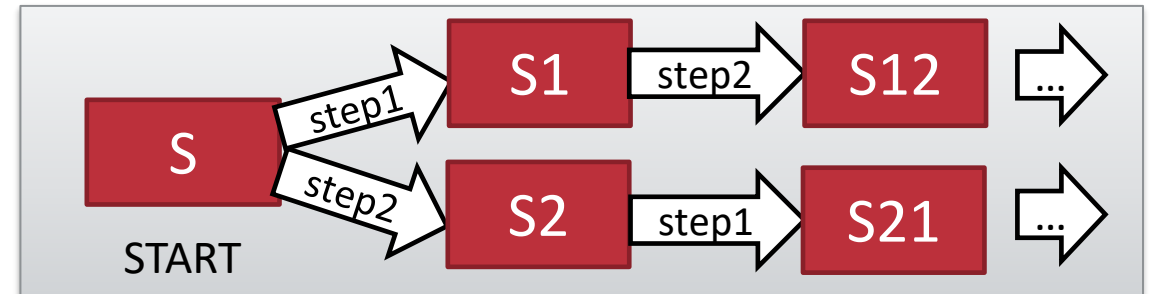
## Tipikus problémák: terminálódás



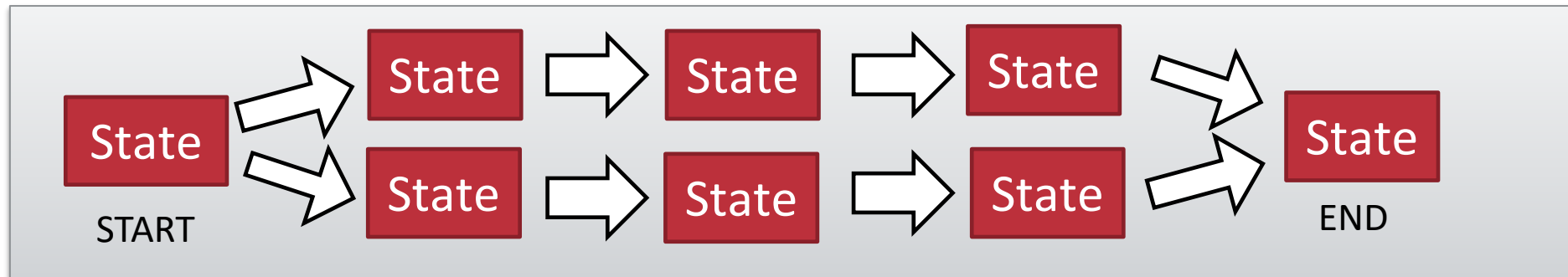
- Szükséges belátni, hogy a szabályok terminálódnak!
- Példák nem-termináló szabályokra:
  - > Makefile: egy build lépés felülírja az egyik bemenetét (*re-dirties*)
  - > MT szabály újabb és újabb objektumokat készít
  - > MT szabály 1 készít egy elemet, MT szabály 2 kiszedi, MIT szabály 1 újra beteszi...
- Általános esetben nem létezik szisztematikus módszer a terminálódás eldöntésére

# Tipikus problémák: Sorrendezés

- Szükséges lehet a helyességhez
  - > osztály leképezése →
  - > attribútum leképezése
- Máskülönben a teljesítményre lehet hatással
  - > Makefile: dirty .java forráskódot fordítunk,
  - > majd lefordítjuk a dirty .uml állományt, ami újra dirty-vé teszi a binárist
  - > a .java állományt újra le kell fordítani
- Hogy oldjuk meg?
  - > Okos végrehajtás (nem mindig megoldható, de pl: Makefile)
  - > Sorrendezést adjuk hozzá az előfeltételhez
  - > **Prioritások:** szabályokhoz prioritást rendelünk: alacsony prioritás → magas prioritás



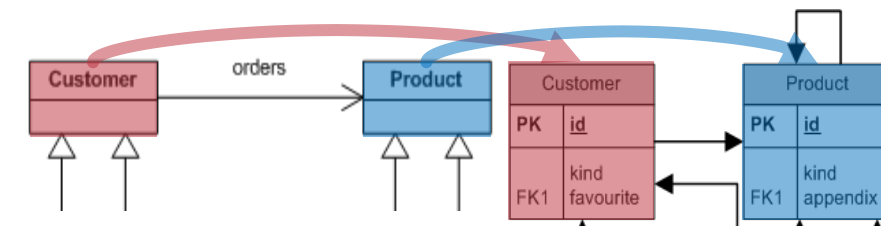
# Tipikus problémák: Konfluencia



- A kiindulóállapot határozza meg a végső állapotot
  - > A kimenet független legyen a transzformációs lépések választásától
  - > Konfluencia fontos, teljes determinisztikuság nem mindig szükséges.

## ■ Példák

- > Makefile: melyik fájlokat fordítsam le előbb? Nem számít...
- > Melyik osztályt képezzem le előbb? Nem számít...



- Általános esetben nem létezik szisztematikus módszer konfluencia eldöntésére



# Modelltranszformáció

**Alapfogalmak**

**Transzformációk láncolása**

**Szabályalapú transzformációk**

**Technológiák**



# Milyen technológiák állnak rendelkezésre?

- **Imperatív megoldás:** bejárjuk a modellt, módosítjuk, elmentjük.
- **Sablon alapú:** tipikusan forráskód vagy más szöveges output generálására
  - > Adott a szöveges kimenet váza, a hiányzó részeket kiegészítjük
  - > Ebben az előadásban
- **Gráftranszformáció alapú**
  - > Következő előadás, Gyakorlat

# XSLT

- EXtensible Stylesheet Language Transformations
- Feldolgozás, sablonok (template) illesztésével
- XML dokumentumok transzformációja
  - > Deklaratív szemantika (XML)
  - > XML vagy más, tetszőleges szöveges kimenet
  - > Navigáció XPath-szal

# Példakód

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl"
href="pelda.xsl"?>
<Stílusok>
  <Piros tipus="szin">
    <Vörös tipus="szin">Ez vöröske</Vörös>
    <Bordó tipus="szin">Ez bordó hordó</Bordó>
  </Piros>
  <Kék tipus="szin">Ez bizony kék</Kék>
  <Dőlt tipus="font">Ez pedig dőlt</Dőlt>
</Stílusok>
```

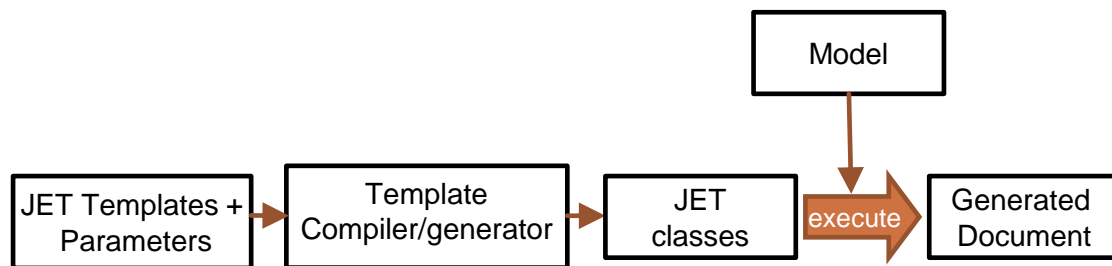
```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="Vörös">
    <font Color="#FFAA00"><xsl:value-of select="."/></font>
  </xsl:template>
  <xsl:template match="Bordó">
    <font Color="#900000"><xsl:value-of select="."/></font>
  </xsl:template>
  <xsl:template match="Kék">
    <font Color="#0000A0"><xsl:value-of select="."/></font>
  </xsl:template>
  <xsl:template match="Dőlt">
    <i><xsl:value-of select="."/></i>
  </xsl:template>
</xsl:stylesheet>
```

```
<font Color="#FFAA00">Ez vöröske</font>
<font Color="#900000">Ez bordó hordó</font>
<font Color="#0000A0">Ez bizony kék</font>
<i>Ez pedig dőlt</i>
```

# Még két XML szintaxisú

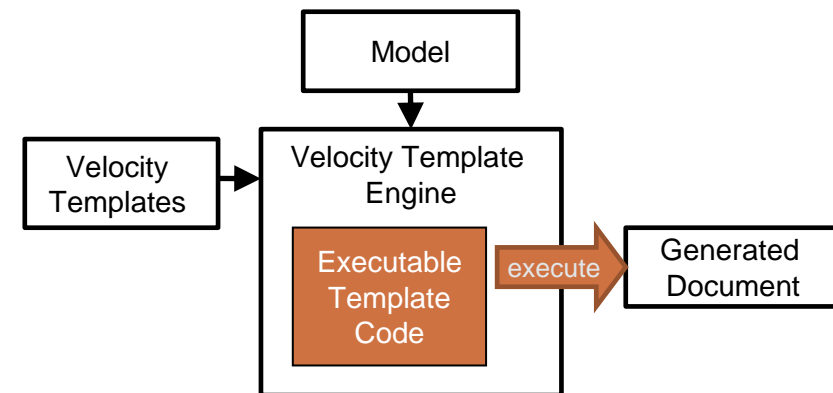
## Java Emitter Templates

- JSP-re hasonlító nyelv
- Java-ra fordul
- Input: java objektumok
- Output: szöveg
- EMF része, EMF kódgenerátor



## Apache Velocity

- JSP-re hasonlító nyelv
- Interpretált
- Input: Map
- Output: szöveg



# Példa kódok, avagy <% %> vs. #( )

## Java Emitter Templates

```
<%@ jet package="hello"
imports="java.util.*" class="XMLDemoTemplate" %>
<% List elementList = (List) argument; %>
<?xml version="1.0" encoding="UTF-8"?>
<demo>
<% for
    (Iterator i = elementList.iterator();
     i.hasNext(); ) { %>
<element><%=i.next().toString()%></element>
<% } %>
</demo>
```

## Apache Velocity

```
<?xml version="1.0" encoding="UTF-8"?>
<demo>

#set( $tempString = "Element")
#foreach( $element in $elementList)
    <element> ${element.toString()} <element>
#end
</demo>
```

### Példa kimenet output

```
<demo>
    <element>A</element>
    <element>B</element>
</demo>
```

Template

Output

# Acceleo



- Kódgenerátor EMF modellekhez
- OMG Model to Text Language (MOFM2T) implementáció
- Eclipse alapú, stabil fejlesztőeszköz
- Modulokból áll
  - > Import lehetséges
- Nyelvi elemek
  - > Sablonok
  - > Lekérdezések
  - > Ciklus, elágazás, értékadás, ...

A screenshot of the Acceleo IDE interface. The main editor window shows a template file named 'generate.mtl'. The code is written in a MOFM2T template language. It includes a comment, file encoding settings, package declaration, imports, and a public class definition. A loop is used to iterate over properties or owned attributes. A right-hand pane shows a list of available model elements for the current context, including 'before', 'separator', 'after', and various model types like 'Property', 'Class', 'AggregationKind', 'Association', and 'self'.

```
[comment @main /]  
[file (c.fullFilePath(), false, 'UTF-8')]  
package [packageName()];  
  
import java.util.List;  
  
public class [javaName()] {  
  
    [for (att : Property | ownedAttribute) ]  
    private [javaType()] [javaName()];  
  
    public [javaType()] get[javaName().toUpperFirst]  
        return [javaName()];  
    }  
  
    public void set[javaName().toUpperFirst]  
        this.[javaName()] = [javaName()];  
    }  
  
    [for]  
}  
  
[/file]
```

# Pár Acceleo sajátosság

- Sablonok

```
[template public generate(c : Class)]  
  
    [comment @main /]  
    [file (c.name, false, 'UTF-8')]  
    [c.name/]  
    [/file]  
  
[/template]
```

- Előfeltételek (mikor fusson le a sablon? → Funkcionális nyelvek?)

```
[comment Generates the java code for a class property that belongs to an association and is ordered /]  
[template public genAssociation(p : Property) ? (owningAssociation <> null and isOrdered)]  
  
[/template]
```

- OCL Lekérdezések:

```
[query public getPublicAttributes(c : Class) : Set(Property) =  
    c.attribute->select(visibility = VisibilityKind::public)  
/]
```



# Acceleo összefoglalás

## ■ Pro

- > Hatékony EMF modellekhez? → Lásd következő gyakorlat!
- > Van debug lehetőség!

## ■ Kontra

- > Csak Java, csak EMF
- > Új nyelvet kell megtanulni
  - Nem nehéz megtanulni
  - OCL-t viszont nehéz volt használni

# Xtend

- Általános célú programozási nyelv (Kotlin előtti modern java)
- Objektum orientált
- Transzparens együttműködés Java-val
  - > Statikus típusellenőrzés
  - > Java típusrendszer
  - > Java kódra fordul
  - > Oda-vissza hivatkozás



# Xtend nyelvi elemek

```
import com.google.inject.Inject
```

Java interop

```
class DomainmodelGenerator implements IGenerator {
```

```
@Inject extension IQualifiedNameProvider nameProvider
```

Nincs ;

```
    override void doGenerate(Resource resource, IFileSystemAccess fsa) {  
        for(e: resource.allContentsIterable.filter(typeof(Entity))) {  
            fsa.generateFile(  
                e.fullyQualifiedName.toString.replace(".", "/") + ".java",  
                e.compile)
```

Típuskövetkeztetés

Első paraméter kihagyható

```
        }  
    }  
    def compile(Entity e) '''
```

''' = template

```
        «IF e.eContainer != null»
```

```
        package «e.eContainer.fullyQualifiedName»;
```

```
        «ENDIF»
```

```
        public class «e.name»
```

```
        «IF e.superType != null»extends «e.superType.shortName» «ENDIF»
```

String interpolation

```
    {
```

```
        «FOR f:e.features»
```

```
        «f.compile»
```

```
        «ENDFOR»
```

Vezérlési struktúrák template-ekben

```
    }
```

```
...
```

Grey Space:

sablon szóköz vs kód szóköz

# Xtend összefoglalás

## ■ Pro

- > Könnyen tanulható, produktív kódolás
- > Nagy kifejezőerő (bonyolult kód röviden írható)
- > Java kompatibilis

## ■ Kontra

- > Csak Java támogatott
- > Automatikus build?
- > Eclipse-alapú (de ma már külön is futtat)

# Microsoft T4

- **Text Templating Transformation Toolkit**
- Szöveg blokkok és vezérlési logika egy fájlban
  - > Szöveg blokk kimásolódik a kimenetre
  - > C# vagy VB
    - Tud írni a kimenetre
  - > Hasonló, mint az ASP.NET, PHP, ...
- Ahol használják: DSL Tools, Entity Framework, VMTS...

# Vezérlési blokkok

- Kód blokk: `<# ... #>`
- Kifejezés blokk: `<#= ... #>`
  - > Kiértékelhető
- Számok négyzete:

```
<#@ template language="C#" #>
<#int top = 10;
    for (int i = 0; i<=top; i++) { #>
        The square of <#= i #> is <#= i*i #>
    } #>
```

```
The square of 0 is 0
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
...
```

# Hogy is működik?

- Generálódik (és lefut):

```
<#@ template language="C#" #>
<#int top = 10;
    for (int i = 0; i<=top; i++) { #>
        The square of <#= i #> is <#= i*i #>
    } #>
```

```
public partial class MyTemplate : ... {
    public string TransformText() {
        int top = 10;
        for (int i = 0; i<=top; i++) {
            this.Write("The square of ");
            this.Write(this.ToStringHelper.ToStringWithCulture(i));
            this.Write(" is ");
            this.Write(this.ToStringHelper.ToStringWithCulture(i*i));
            this.Write("\r\n");
        }
        return this.GenerationEnvironment.ToString(); }}
}
```

# Osztály kiegészítése

- `<#+ ... #>`

- > Helper metódusok és propertyk generálása

```
<#+ // Class feature block
private void WriteSquareLine(int i) { #>
    The square of <#= i #> is <#= i*i #>.
<# } #>
```

- > Tartalmazhat szöveg blokkot is

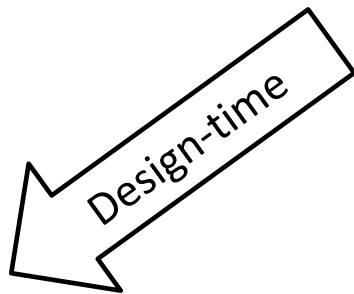
- > Hozzáadódik a generált osztályhoz, meghívható:

```
<#int top = 10;
    for (int i = 0; i<=top; i++)
        WriteSquareLine(i);
#>
```

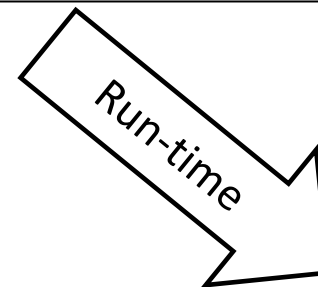


# Design-time vs run-time

```
<#int top = 5;  
    for (int i = 0; i<=top; i++) { #>  
        The square of <#= i #> is <#= i*i #>  
<# } #>
```



The square of 0 is 0  
The square of 1 is 1  
The square of 2 is 4  
The square of 3 is 9  
The square of 4 is 16  
The square of 5 is 25



```
public virtual string TransformText() {  
    int top = 5;  
    for (int i = 0; i <= top; i++) {  
        this.Write(" \r\n      The square of ");  
        this.Write(this.ToStringHelper.ToStringWithCulture(i));  
        this.Write(" is ");  
        this.Write(this.ToStringHelper.ToStringWithCulture(i * i));  
        this.Write(" \r\n");    }  
}
```

# T4 összefoglalás

## ■ Pro

- > Jól használható, rugalmas
- > Feladatautomatizáláshoz kiváló (100+ ezer sornyi kód)
- > Gyors, bináris kód generálható belőle

## ■ Kontra

- > T4 scriptek karbantarthatósága
- > Debug lehetőségek
- > Furcsa formázási elvárások
- > Automatikus build? Függőségek?



Köszönöm a figyelmet!