Kódgenerálás: OpenAmeos, JET

Simon Balázs

BME IIT, 2011.

Tartalom

- Kódgenerálás
- CodeDOM
- OpenAmeos &
  Transformation Description Language (TDL)
- Java Emitter Templates (JET)

Szoftverfejlesztés

- Magas szintű, sokadik generációs programnyelvek, mégis:
  - sok glue kódra van szükség, pl. adatbázis-elérés
  - a programkód váza mindig ugyanolyan
  - még mindig követünk el hibákat és mindig is fogunk
  - alapvetően lusták vagyunk
- Megoldás: kódgenerálás

Kódgenerálás előnyei

- Sok területen alkalmazható: adatbázis-kezelés, UI, web szolgáltatások, stb.
- A generátor minőségi, konzisztens kódot állít elő
- Az unalmas, manuális részeket rábízhatjuk
- A generált kódot könnyebb karbantartani (feltéve, hogy megvan a generátor)
- Flexibilis architektúrát biztosít
- A jó kódgenerátor kényelmesen programozható
- Fontos, hogy megérje a befektetett munkát

Kódgenerálás hátrányai

- A generált kódot kézzel (a generátor hiányában) nehéz karbantartani
- A modellt és a kódot folyamatosan szinkronban kell tartani
- A generált kód sokszor nem teljes: manuális beavatkozást igényel
- A fejlesztők sokszor félnek az ismeretlen kódtól, nem bíznak meg benne

Kérdések

- Hogyan lehet a generált kódot manuálisan módosítani és ezt a változást megőrizni újragenerálás során?
- Lehet-e a kód 100%-át generálni?
  - Ha nem, miért nem?
  - Ha igen, hogyan?
- Milyen a jó kódgenerátor?

Esettanulmány

- Projekt: 4 hónap
- Programnyelv: C++
- Adatbázis:
  - speciális tömörített fájlok
  - ID és offset referenciák a fájlokban lévő rekordok között
- Problémák:
  - Egy normál C++ kódgenerátor nem elegendő:
    - az implementáció hiányzik
  - Az UML önmagában nem elegendő:
    - ID koncepció hiányzik
    - ID és offset alapú referenciák nem különböztethetők meg
- Megoldás:
  - Saját UML profile: <<identifier>>, <<offset>>, stb.
  - Fájlhozzáférés: speciális API manuálisan megírva (8 osztály)
  - Fájlok leképzése objektumokra: a manuális API-ra építve az implementáció 100%-a generált (kb. 70 osztály, 1.4 MB programkód)

Kódgenerálási technikák

- CodeDOM alapú:
  - a célnyelven előállítandó program fájának felépítése a memóriában
  - a programkód kinyomtatása automatikus
  - akár közvetlenül le is fordítható gépi kódra
  - pl. .NET CodeDOM
- Template alapú:
  - WYSIWYG stílusú sablonok megadása
  - vezérlés speciális nyitó és záró karaktersorozatok között
  - vezérlésben: modell bejárása, ciklusok, stb.
  - pl. OpenAmeos TDL, EMF JET

.NET CodeDOM

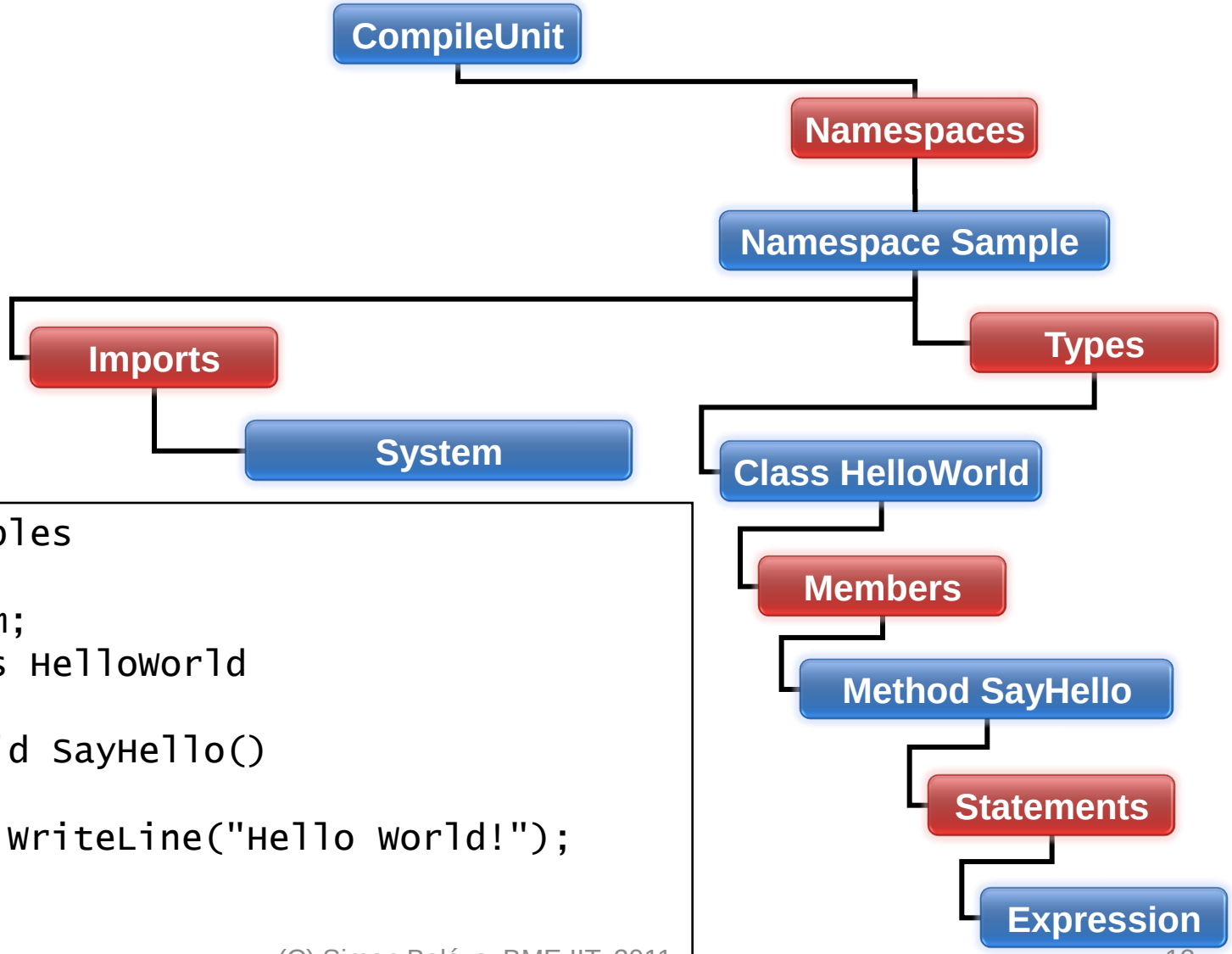
.NET CodeDOM

- CodeDOM
  - Code Document Object Model
  - a programkód szerkezete faként felépítve a memóriában
  - a programkód nyomtatása automatikus (több különböző nyelv is támogatható)
  - akár közvetlenül le is fordítható gépi kódra
- Névterek
  - System.CodeDom
    - Tartalma: CodeDOM gráf elemei
  - System.CodeDom.Compiler
    - CodeDomProvider interfészek és ősosztályok

CodeDOM providers

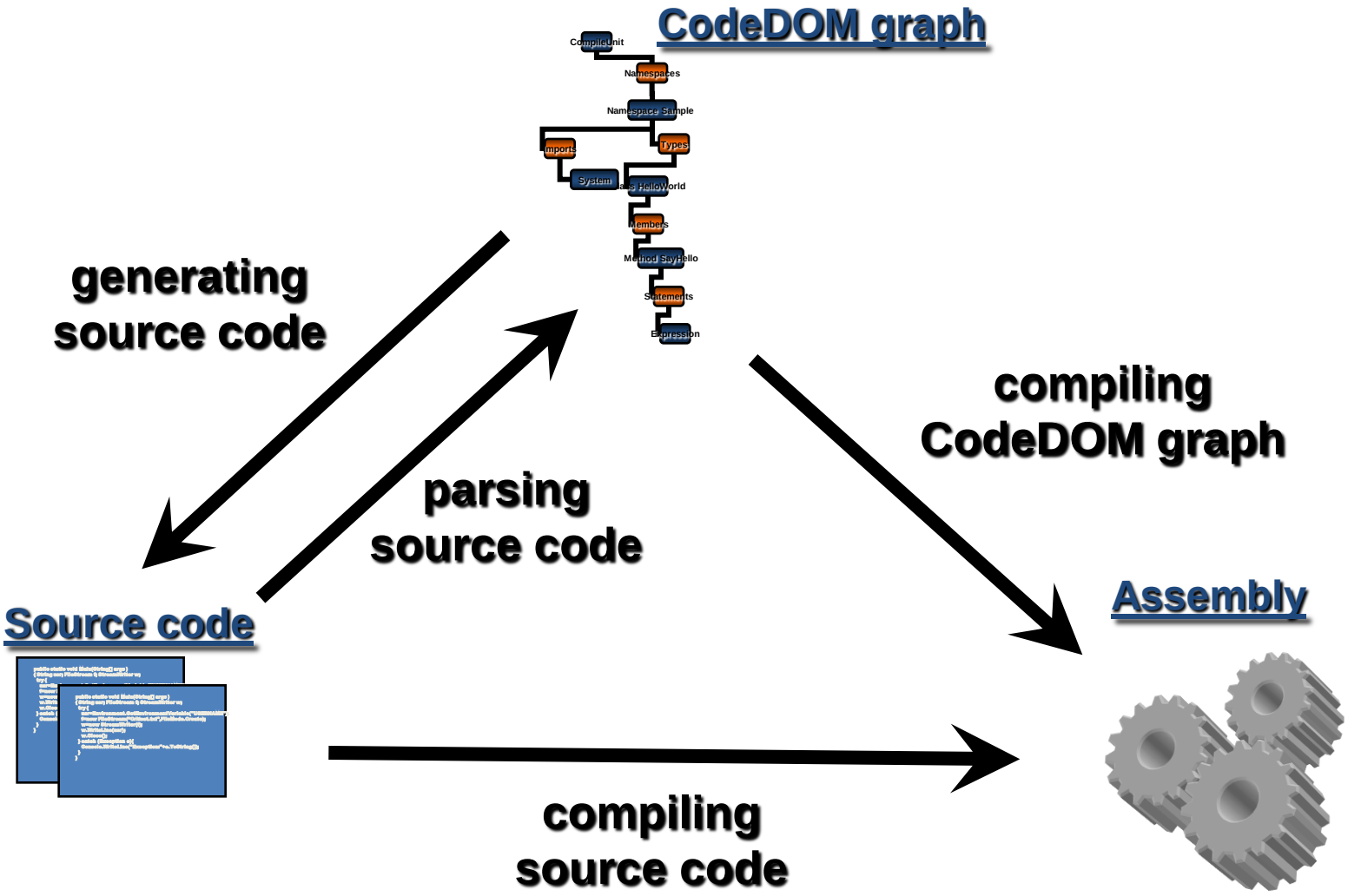
- Microsoft providers
  - C#, VB.NET, Managed C++, Jscript, J#
- Microsoft partners
  - NetCOBOL (Fujitsu)
  - ActivePERL (ActiveState)
  - Eiffel (Interactive Software Engineering)
  - …

A HelloWorld CodeDom gráfja




```
namespace Samples
{
  using System;
  public class HelloWorld
  {
    public void SayHello()
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

CodeDOM concepts



OpenAmeos & Transformation Description Language (TDL)

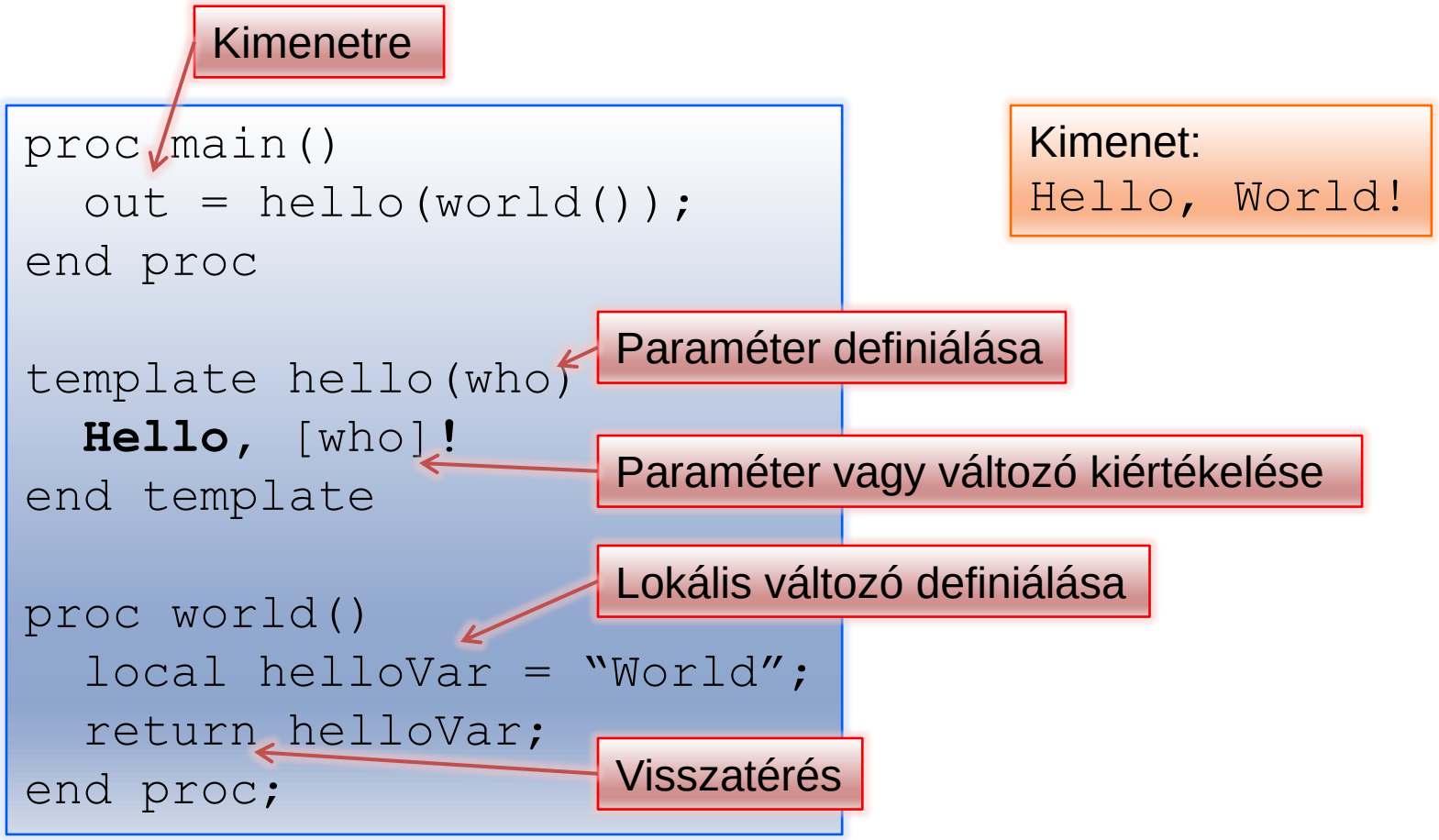
OpenAmeos

- http://www.openameos.org/
- UML 2.0-át támogató modellező eszköz
- Programozható kódgenerátor
- Kódgenerátor nyelve: TDL
  - Transformation Description Language
  - Template alapú scriptnyelv (vannak függvények is)
  - Leírás: ...\OpenAmeos\documentation\ACD\ACD.pdf
- A modell bővíthető: meta insert
- A régi kód átemelhető: merge
- Egyszerű modellbejárás: loop

Függvények és sablonok

- TDL-ben kétfajta meghívható kifejezés van: függvény és sablon
- Függvény (proc):
  - tartalma: végrehajtható TDL utasítások
- Sablon (template):
  - tartalma: tetszőleges szöveg, ami egyben a visszatérési érték is
  - vezérlés: szögletes zárójelek között megadott TDL utasításokkal
- A program belépési pontja: main() függvény

Példa: Hello world!



OpenAmeos metamodell: ACD

- Referencia: …\OpenAmeos\Documentation\ACD_METAMODEL\index.html
- Az UML metamodell megfelelője
- Az elnevezések eltérhetnek:

UML	ACD UML
Package	MPackage
Classifier	MClass
Interface	MInterface
Class	MNormalClass
Property	MAttribute
Operation	MOperation
Association	MAssociation
Stereotype	MStereotype
UML 1: tagged-value	MTaggedValue

Modell bejárása: loop

- Loop szintaxis:

```
loop (…navigation rules…)
  …
end loop
```

- Példák:
  - minden classifier: `loop (Instances->MClass)`
  - minden csomag: `loop (Instances->MPackage)`
  - minden osztály minden attribútuma:

```
loop (Instances->MNormalClass->MAttribute)
```

Loop navigáló lánc

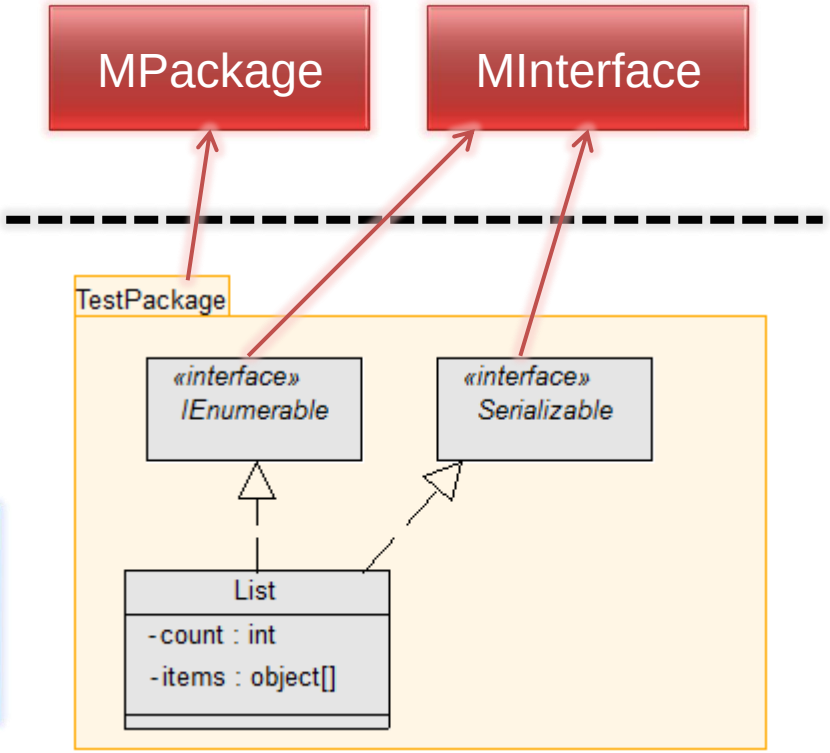
- Loop fejlécben tetszőlegesen hosszú lánc
- Szomszédos láncszemek:
  - az ACD metamodellben valamilyen összeköttetésben lévő elemek
- Összeköttetések típusai:
  - Instance navigator
  - Inheritance navigator
  - Relational navigator

Instance navigator

- Adott metatípusú elemek bejárása

```
loop (Instances->MPackage)
   ...
end loop
```

```
loop (Instances->MInterface)
   ...
end loop
```

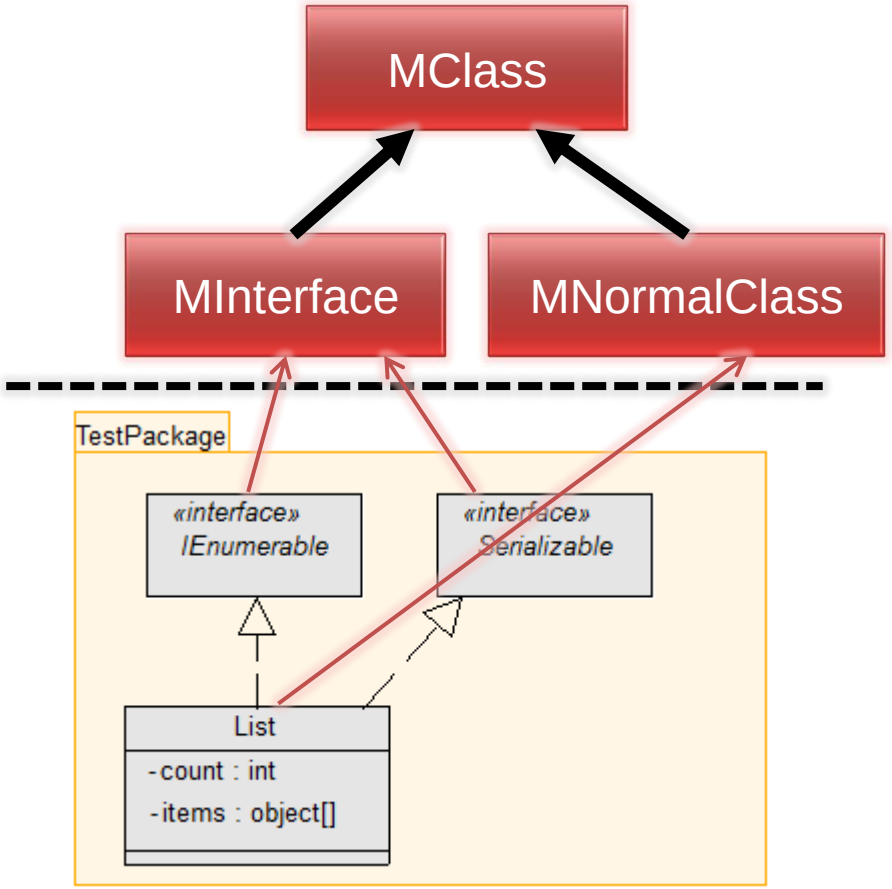


Inheritance navigator

- Öröklések bejárása: kasztolás speciálisabb metatípusra

```
loop (MClass->MInterface)
   ...
end loop
```

```
loop (MClass->MNormalClass)
   ...
end loop
```



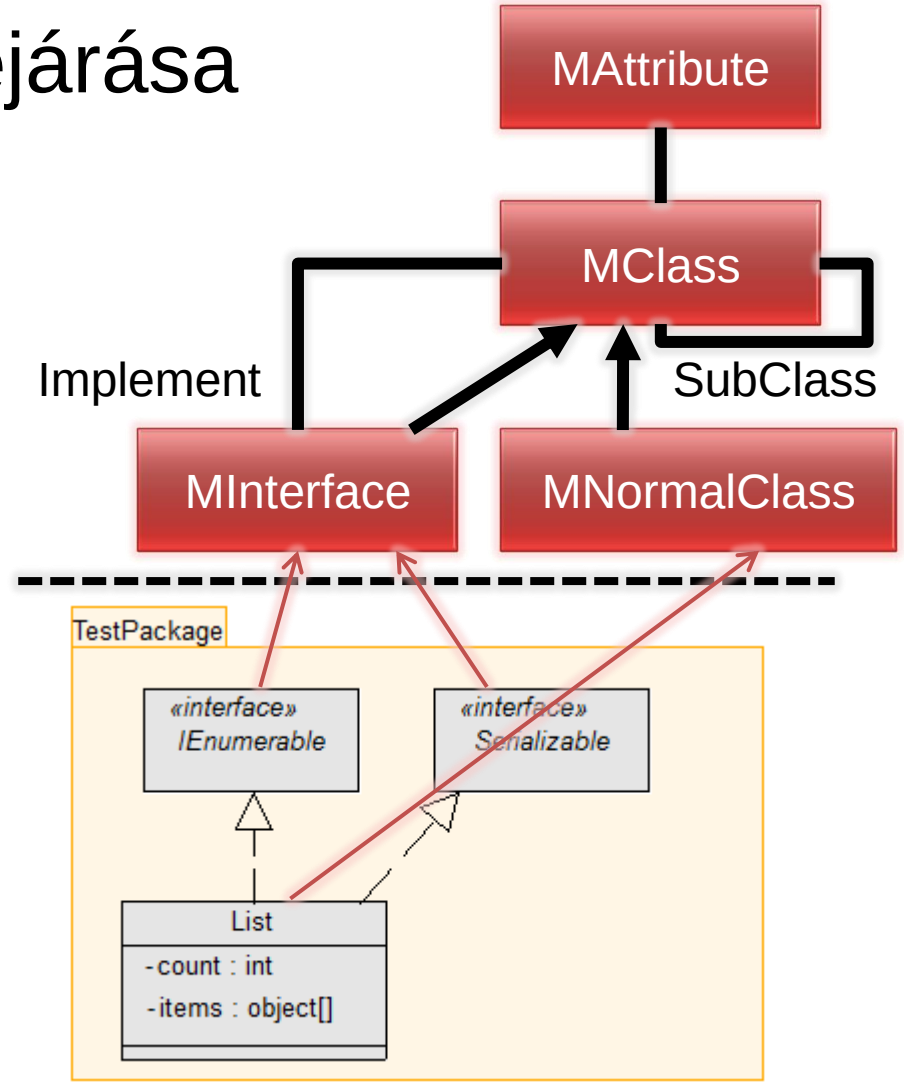
Relational navigator

- Metaasszociációk bejárása

```
loop (MClass->MAttribute)
   ...
end loop
```

```
loop (MClass->Implement)
   ...
end loop
```

```
loop (MClass->SubClass)
   ...
end loop
```



Loop kiértékelése

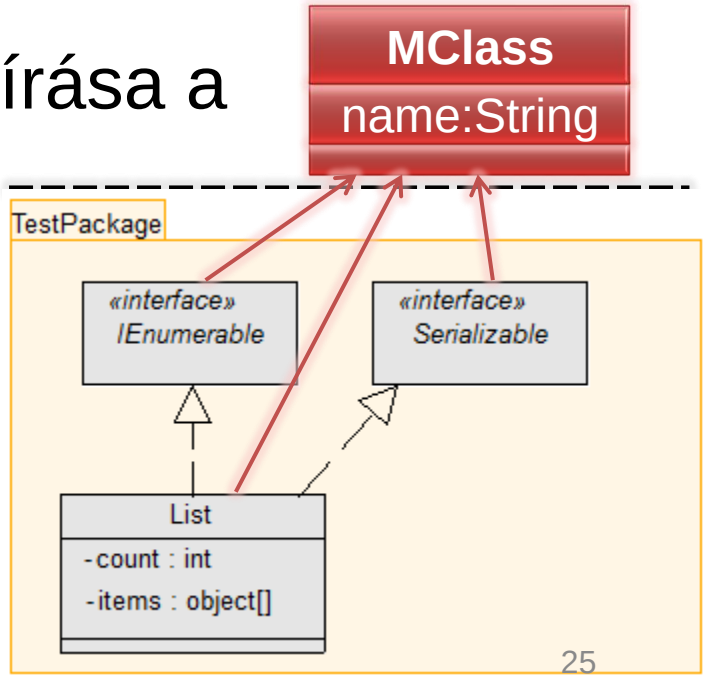
- Végrehajtás:
  - a lánc mintáját az összes lehetséges módon illeszti a modellre
  - minden mintailleszkedésre lefut a ciklusmag
  - az illesztett példányok elérhetők a láncszem nevére hivatkozva
  - a láncszemek átnevezhetők az **As** kulcsszóval (alias)
  - lényegében: egymásba ágyazott foreach-ciklusok
- Példa:

```
loop(Instances->MClass->Role As FromRole->
     MAssociation As MRel->MAssociationEnd As ToRole
     Where [FromRole.id] != [ToRole.id])
        out = [MRel.name];
end loop
```

Metaosztály attribútumai

- Az ACD metamodellben definiált osztályok attribútumainak kiértékelése:
  - szögletes zárójelek között
- Példa:
  - minden classifier nevének kiírása a kimenetre:

```
loop (Instances->MClass)
  out = [MClass.name];
end loop
```



Szűrő feltételek

- **Where** kulcsszó segítségével
- Csak a feltételt teljesítő mintaillesztésekre fut le a ciklusmag
- Példák:
  - interface sztereotípiával ellátott classifier-ek:

```
loop(Instances->MClass
     Where [MClass.stereotype]=="interface")
    …
end loop
```

  - Összes asszociáció kiindulva az adott MClass-ból:

```
loop(MClass->Role As FromRole->MAssociation->
     MAssociationEnd As ToRole->MClass As Partner
     Where [FromRole.id] != [ToRole.id])
    …
end loop
```

Egymásba ágyazás

- Loop-ok egymásba ágyazhatók
- Legkülső loop mindig **Instances**-zal kezdődik (lehet másik függvényben is)
- Bárhonnan folytatható a lánc, nem csak az utolsó láncszemtől
- Példa:

```
loop (Instances->MNormalClass)
    out = [MNormalClass.name];
    loop (MNormalClass->MAttribute)
        out = [MAttribute.name];
    end loop
end loop
```

Függvényhívás loop fejlécben

- A navigáció és Where feltétel után pontosvesszőkkel elválasztva
  - Első pontosvessző utáni függvény: első lefutáskor hívódik meg
  - Második pontosvessző utáni függvény: második lefutáskor hívódik meg
  - stb.
  - Ha több lefutás van, mint függvény, akkor a maradék esetekben a legutolsó függvény hívódik meg
- Példa:

```
loop(MOperation->OpPara; setDelim(""); setDelim(","))
    [para] = [para] delim() [OpPara.type] [OpPara.name]
end loop
```

Példa: osztályok és attribútumok

```
proc main()
  out = genClasses();
end proc

template genClasses()
  [loop(Instances->MNormalClass)]
public class [MNormalClass.name] {
    [loop(MNormalClass->MAttribute)]
    [genAttribute([MAttribute])]
    [end loop]
}
  [end loop]
end template

template genAttribute(MAttribute)
private [MAttribute.type] [MAttribute.name];
end template
```

Egy lehetséges kimenet:

```
public class A {
    private int x;
    private String str;
}
public class B {
    private double y;
}
```

Feltételes kifejezések

- if (feltétel) … else … end if
- Példák:

```
proc felt1(a)
  if ([a] == “valami”)
    return 5;
  else
    return 6;
  end if
end proc

proc felt2(a)
  if ([a] == “valami”)
    return 7;
  end if
end proc
```

```
template felt3(a)
  [if ([a] == “valami”)]
    8
  [else]
    9
  [end if]
end proc

template felt4(a)
  [if ([a] == “valami”)]
    10
  [end if]
end proc
```

Merge: összefésülés a régi kóddal

- Speciális formájú kommentsorok a generált célnyelvben
- A köztük lévő kód átemelése az újragenerált kódba
- Blokkok azonosítása: egyedi azonosító alapján
- Pl. Java esetén egy lehetséges generátor:

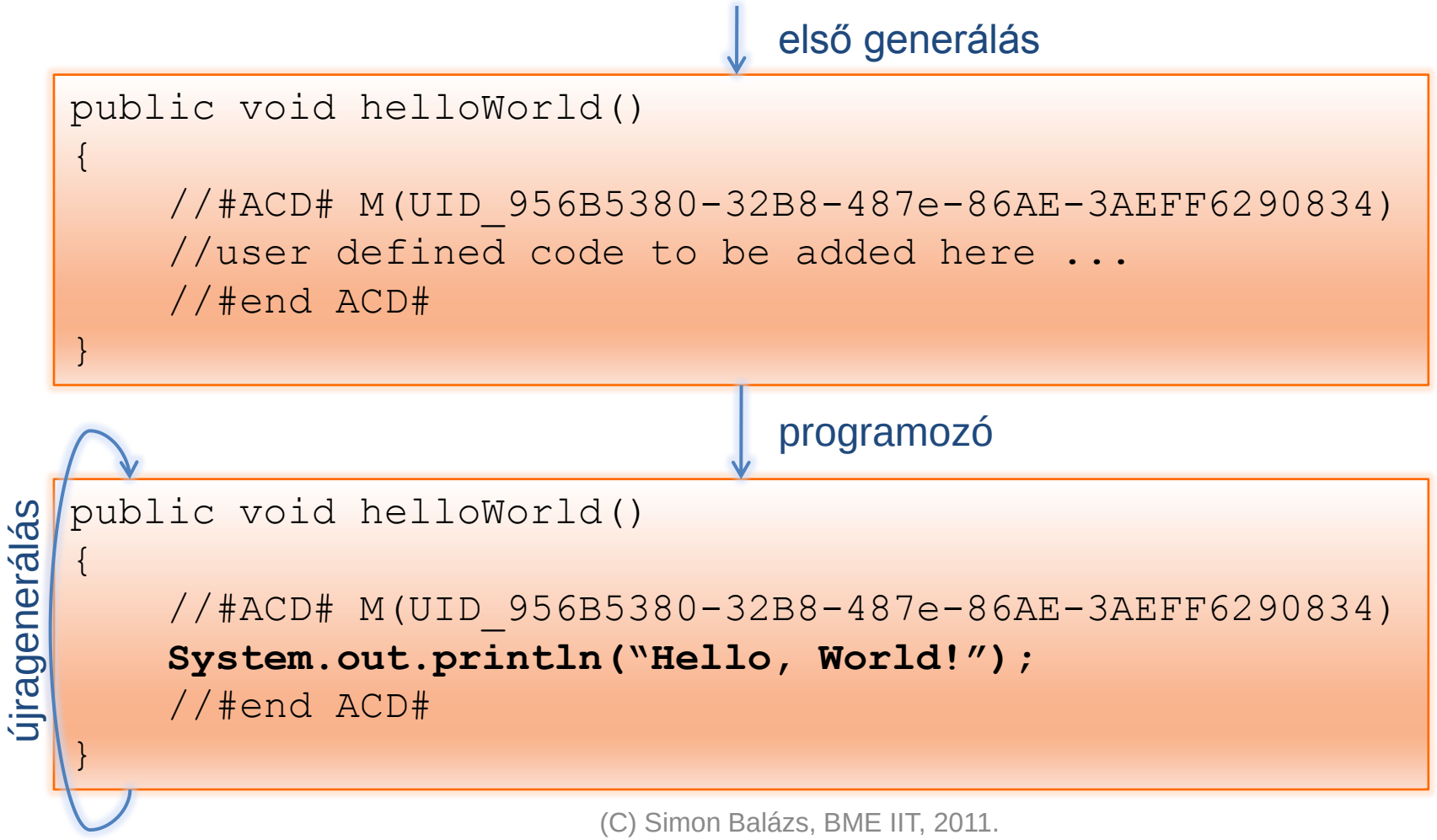
```
template mergeOut(uniqueId,desc)
//#ACD# M([uniqueId]) [desc]
//user defined code to be added here ...

//#end ACD#
end template

template genOperation(MOperation)
public void [MOperation.name]() {
  [mergeOut([MOperation.guid], "")]
}
end template
```

Merge

- Egy lehetséges kimenet:



Generátorok felépítése

- Könyvtár:
  - …\OpenAmeos\templates\uml\tdl\Languages**Nyelv**
  - ahol **Nyelv**-nek egy tetszőleges nevet meg lehet adni, pl. **Java**, **XMI**, stb.
- Fő fájl:
  - a fenti könyvtárban egy **main.tdl** nevű fájl, ebben egy **proc main() … end proc** függvény
  - (ld. következő slide-ok)
- Többi fájl:
  - valamilyen prefixszel ellátott **.tdl** kiterjesztésű fájl
  - pl. **java_MClass.tdl**, **java_MAttribute.tdl**, stb.
  - tartalma: template-ek és proc-ok gyűjteménye

A main.tdl szerkezete

```
/*{
{ Category
   { Label "Java" }
   { Variable
        { Name ProjectName }
        { Type text }
        { DefaultValue "${system}" }
        { EditorType text }
   }
}
}*/

USES std;
USES java_MClass;
USES java_MAttribute;
...

proc main()
    …
end proc
```

Opcionális rész:
ezek a paraméterek az OpenAmeos grafikus felületén állíthatók

A többi TDL fájl importálása

Főfüggvény

A main.tdl szerkezete (folytatás)

```
proc main()
    init();
    if([srcdir] == "")
        info = "There is no output directory specified !\n";
        return;
    end if

    mkdir([srcdir] "/" [ProjectName]);

    output([srcdir] "/" [ProjectName] "/" "build.xml");
    out = genAntBuild();

    …
end proc

proc init()
       delMergeFile();
       appendMergeFile("merge_config_Java.txt");
       setLanguage("Java");
end proc
```

merge inicializálása, ha használni kívánjuk

kimeneti könyvtár ellenőrzése

naplózás

hivatkozás a grafikus felületen megjelenő paraméterre

könyvtár létrehozása

kimenet előállítása

aktuális kimeneti fájl specifikálása

merge stratégia kiválasztása, ha használni kívánjuk

Java Emitter Templates (JET)

Kódgenerálás: JET

- Java Emitter Template
- Java nyelven vezérelt sablon alapú generátor
- JSP-hez hasonló: <% %>
  - (lecserélhetők más karakterláncokra)
- Gyengébb, mint a TDL:
  - az egész fájl egy nagy template
  - nincsenek külön definiálható függvények
- De:
  - tetszőleges Java kód hívható
  - így több template fájl összekombinálható

Követelmények

- Tervezéshez:
  - Eclipse SDK
  - vagy:
  - Eclipse Galileo (3.5) + “Modeling/Java Emitter Templates (JET) SDK” plugin telepítése
- Felhasználáshoz:
  - a template fájlból az SDK által automatikusan generált Java fájlok .jar-rá fordítva

JET projekt

- Az EMF projektet JET-essé kell változtatni
  - (Eclipse-ben egy egyszerű varázsló segítségével)
- Létrejön egy **templates** nevű könyvtár
- Ebben **jet** végződéssel rendelkező kiterjesztésű fájlokat kell létrehozni
  - pl. Sample.htmljet, Example.cppjet, stb.
- Mentéskor automatikusan Java kód képződik a template-ekből, ezek lesznek a generátorok

JET HelloWorld példa

HelloWorld.txtjet:

```
<%@ jet package="hellopkg" class="HelloWorldGenerator" %>
Hello World!
```

Program.java:

```
package main;

import hellopkg.HelloWorldGenerator;

public class Program {
  public static void main(String[] args) {
    HelloWorldGenerator generator = new HelloWorldGenerator();
    System.out.println(generator.generate(null));
  }
}
```

Eredmény:

```
Hello World!
```

JET template felépítése

- Fejléc példa (MyGenerator.jet):

```
<%@ jet package="mymodel.generator"
  class="MyGenerator"
  imports="java.util.Iterator org.eclipse.emf.common.util.EList
           mymodel.*;"
  startTag="[[" endTag="]]"
%>
[[ MyModel model = (MyModel)argument; ]]
```

Kimeneti csomag neve

Kimeneti osztály neve

Importált típusok

Kezdő és záró tag-ek lecserélése (alapértelmezett: <% és %>)

A bemenet kasztolása a megfelelő típusra (ld. később)

- További kód példa:

```
[[ for (int i = 1; i <= 10; ++i) { ]]
    [[= i ]]. Hello, World!
[[ } ]]
```

Vezérlés

Generált kód

Kiértékelés

- Várt kimenet:

```
1. Hello, World!
2. Hello, World!
...
10. Hello, World!
```

JET template-ből készülő Java kód

- A template elmentésekor automatikusan előáll
- A fejlécben megadott csomagba, a megadott néven
- Tartalma egy alábbi szignatúrájú függvény:

```
public String generate(Object argument)
```

  - A paramétert a JET sablon kódjában kell a megfelelő típusra kasztolni!
- Generátor lefuttatása: Java kódból (pl. MyGenerator nevű osztály esetén):

```
MyModel model = …;
MyGenerator generator = new MyGenerator();
String code = generator.generate(model);
```

M Code Generator

M Code Generator

- A tanszéken kifejlesztett kódgenerátor
- Erősen típusos
- Függvények és sablonok
- A C# és TDL kombinációja
- Teljes .NET API és C# vezérlés elérhető, támogatja a loop konstrukciót is
- Beépített Visual Studio támogatás syntax-highlighting-gal

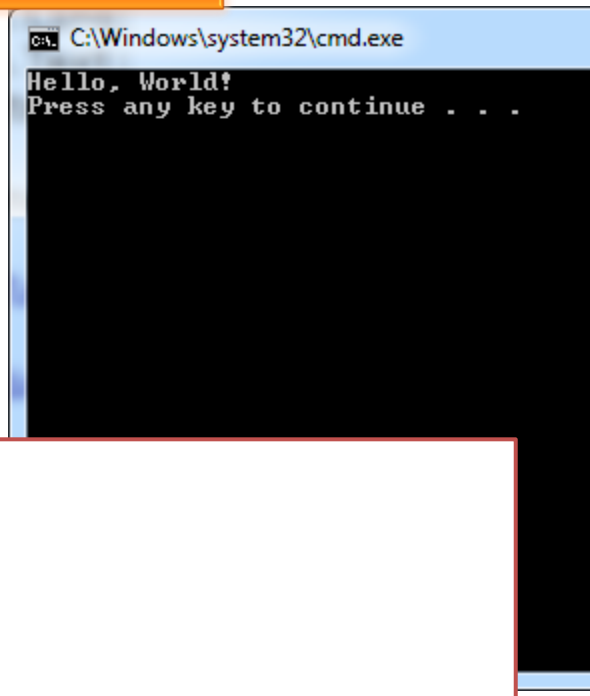
MCG HelloWorld példa

Generator.mcg:

```
function void Main()
Context.Output(SayHello("World"));
end function

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

Kimenet:



Program.cs:

```
using OsloExtensions;
namespace GeneratorProject1
{
  public class Program
  {
    public static void Main(string[] args)
    {
      GeneratorContext context = new GeneratorContext();
      Generator generator = new Generator(null, context);
      generator.Execute();
    }
  }
}
```

Összefoglalás

Kódgenerálás előnyei

- Sok területen alkalmazható: adatbázis-kezelés, UI, web szolgáltatások, stb.
- A generátor minőségi, konzisztens kódot állít elő
- Az unalmas, manuális részeket rábízhatjuk
- A generált kódot könnyebb karbantartani (feltéve, hogy megvan a generátor)
- Flexibilis architektúrát biztosít
- A jó kódgenerátor kényelmesen programozható
- Fontos, hogy megérje a befektetett munkát

Kódgenerálás hátrányai

- A generált kódot kézzel (a generátor hiányában) nehéz karbantartani
- A modellt és a kódot folyamatosan szinkronban kell tartani
- A generált kód sokszor nem teljes: manuális beavatkozást igényel
- A fejlesztők sokszor félnek az ismeretlen kódtól, nem bíznak meg benne

Összehasonlítás

	.NET CodeDOM	OpenAmeos TDL	JET	MCG
Hivatalosan támogatott programnyelvek	C#, VB.NET, C++, Jscript, J#	Ada, C, C++, C#, Ecore, Java, UML2, XMI	Java	SOA: C#, Java, XSD, WSDL, konfig fájlok
Módszer	Kódfa	Sablon	Sablon	Sablon
Generátor nyelve	C#, VB, stb.	TDL	Java	C# + TDL
Típusosság	erősen	gyengén	erősen	erősen
Fejlesztőeszköz	Visual Studio	notepad (-sy.hl.)	Eclipse (-sy.hl.)	Visual Studio
Karbantarthatóság	Nehéz	Könnyű	Viszl. könnyű	Könnyű
Kódformázás	Támogatott	Támogatott	Nehéz	Támogatott
Segédfüggvények	Támogatott	Támogatott	Nincs	Támogatott
Kimenet/generátor	Több fájl	Több fájl	1 fájl	Több fájl
Kézi módosítás megtartása	Nem támogatott	Támogatott: merge	Nem támogatott	Még nem támogatott
Modell bejárása	C#, VB, stb. loop-konstrukció		Java	C# + loop