



MODEĻU KARTĒŠANA UN TRANSFORMĀCIJA M2M

4. lekcija

Ērika Asņina, DITF LDI LDK

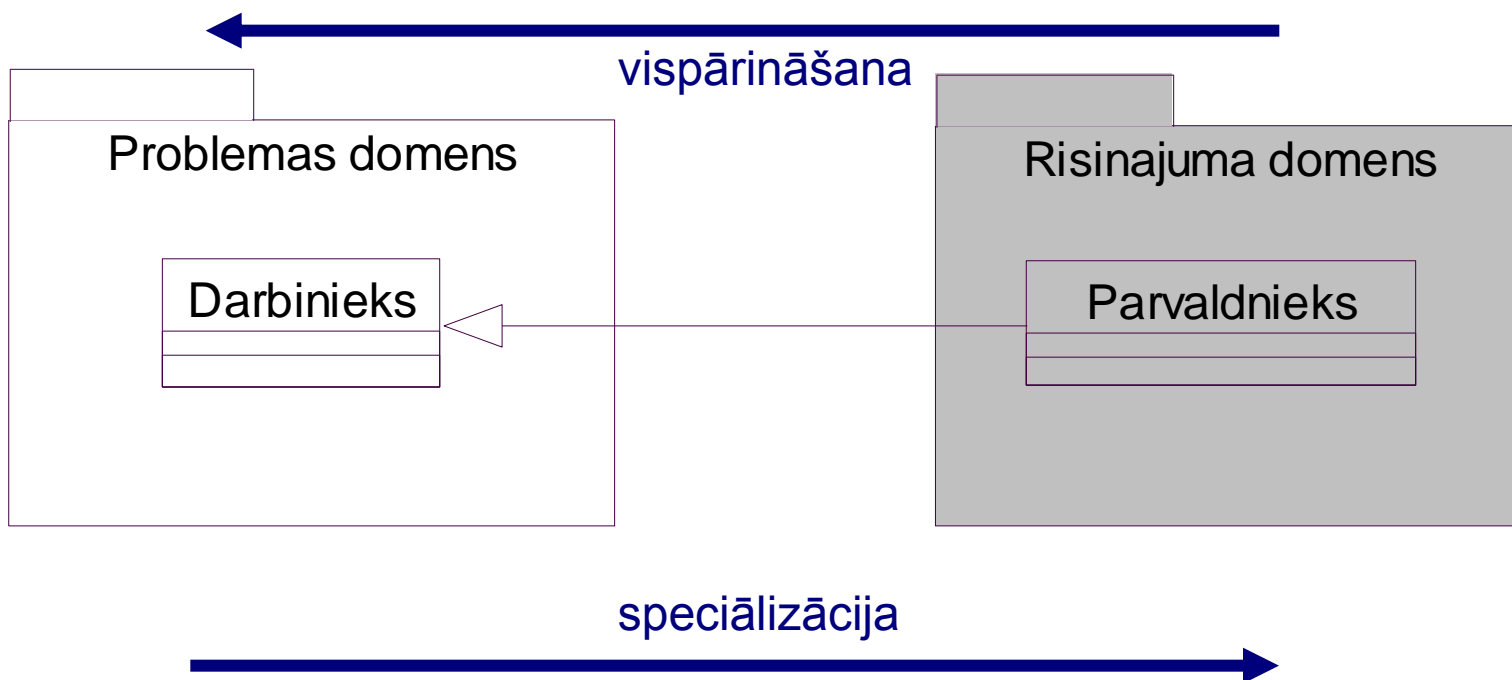


Transformācijas taksonomija

- Transformācija VS kartēšana (mapping)
- Transformācijas veidi
 - Vertikālā transformācija
 - Precizēšana (refinement)
 - Abstrahēšana (abstraction)
 - Horizontālā transformācija
 - Netieša (oblique) transformācija

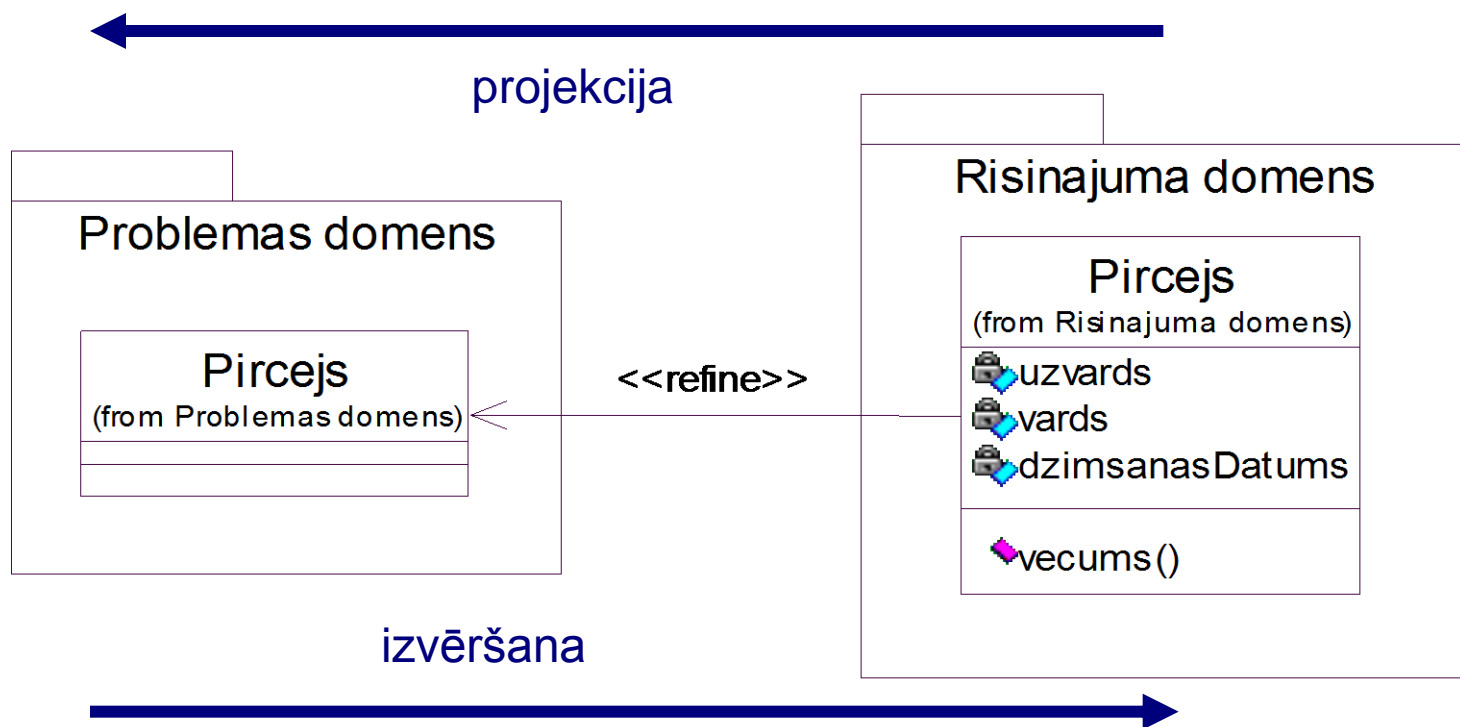
Vertikāla transformācija – detalizēšana (refinement)

- Specializācija (specialization) un vispārināšana



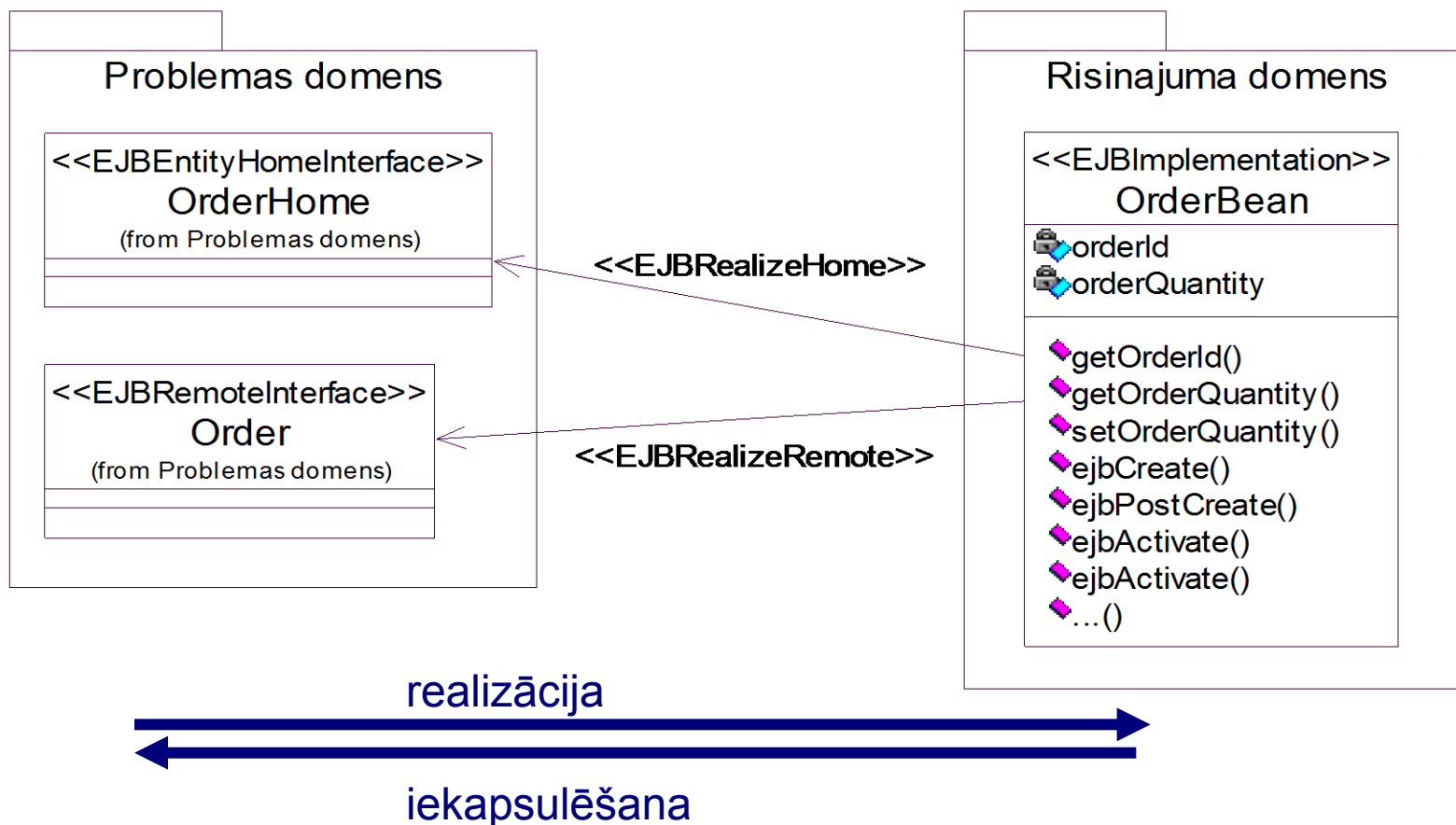
Vertikāla transformācija – detalizēšana (refinement)

■ Izvēršana (elaboration) un projekcija



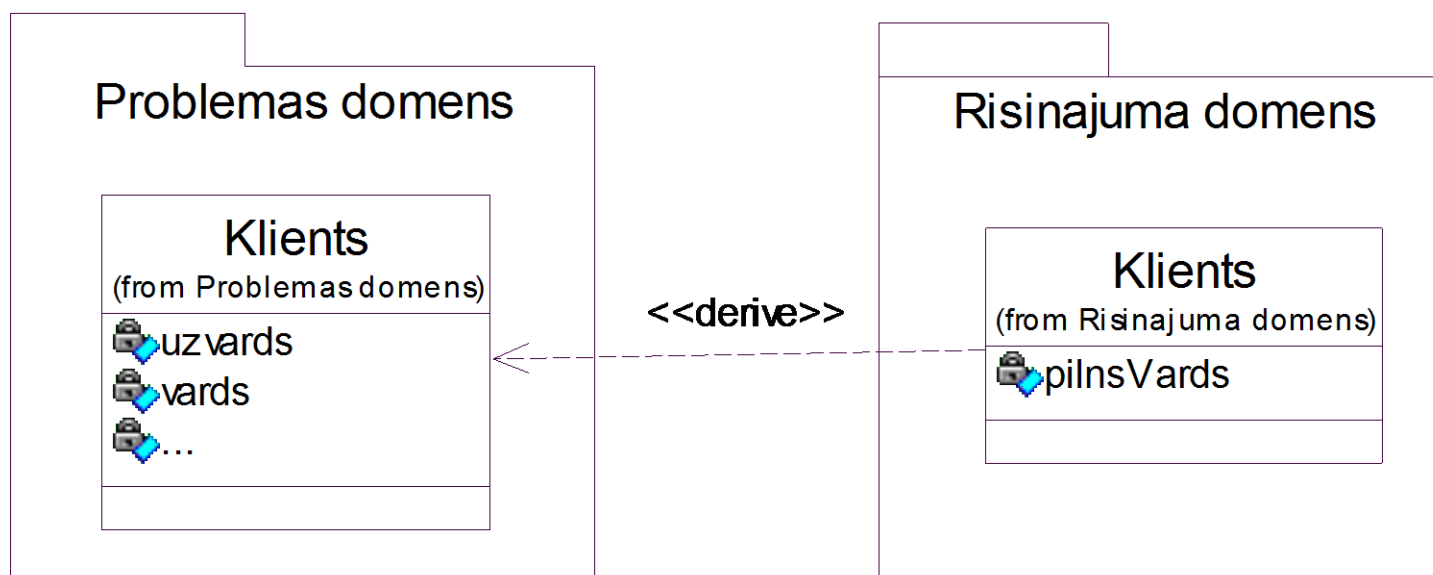
Vertikālā transformācija – detalizēšana (refinement)

- Realizācija (realization) un iekapsulēšana (encapsulation)



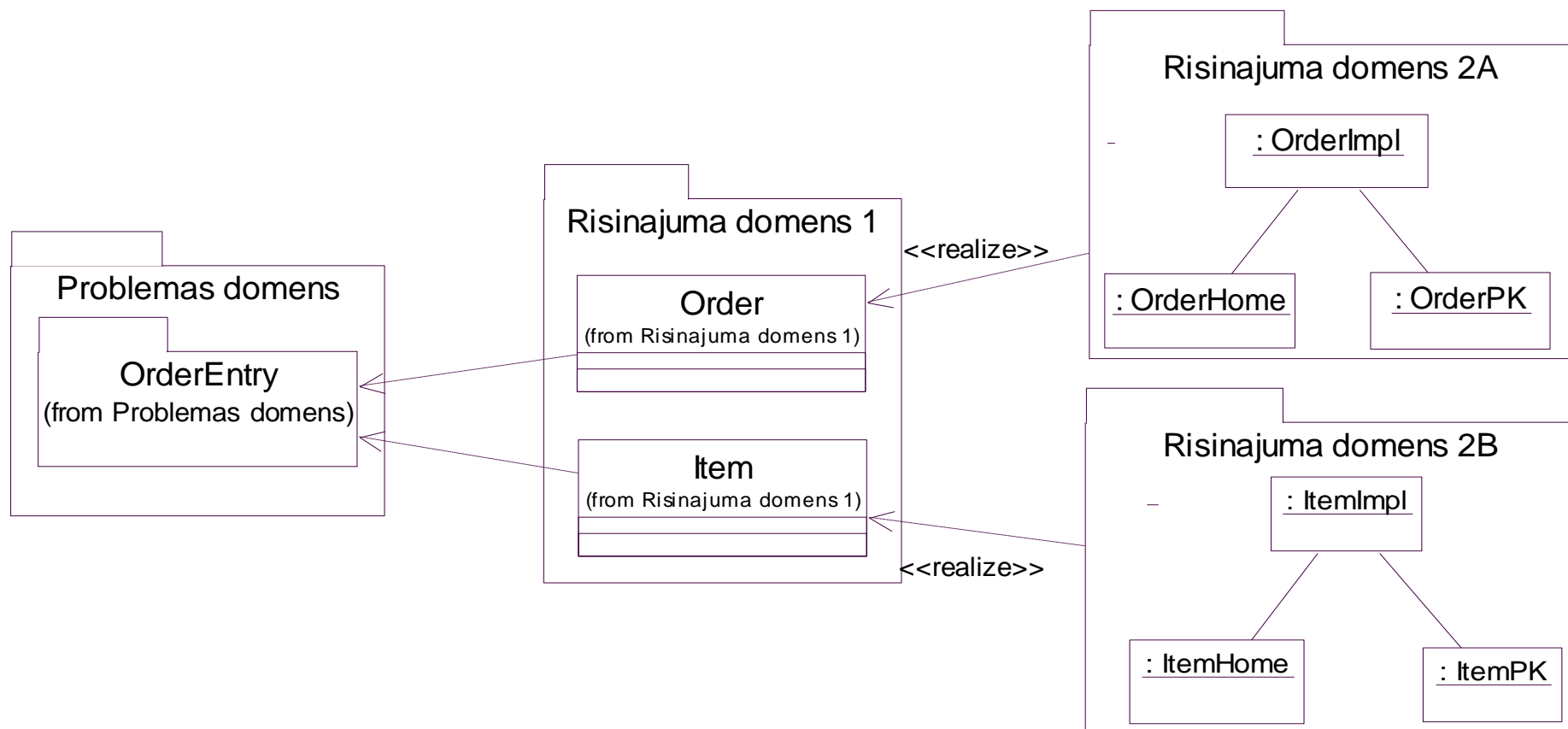
Vertikāla transformācija – detalizēšana (refinement)

■ Atvasināšana (derivation)



Vertikālā transformācija – detalizēšana (refinement)

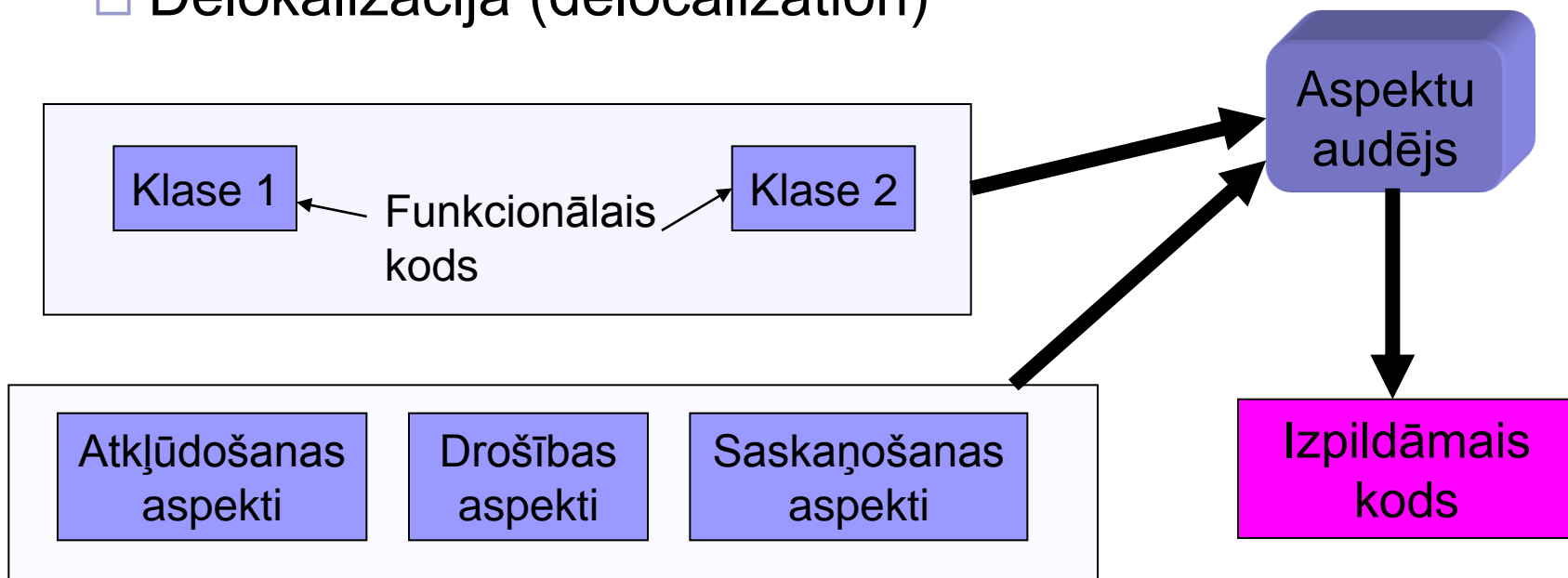
■ Dekompozīcija (decomposition) un kompozīcija



Horizontālās transformācijas

■ Horizontālo transformāciju veidi

- ☐ Refaktoring (refactoring)
- ☐ Optimizēšana (optimization)
- ☐ Delokalizācija (delocalization)



Ar transformācijām saistītās pamatgrūtības

- Modeļu sinhronizācija sistēmās, kuras pamatojas uz notikumu apstrādes
 - Ideāls gadījums ir abu sinhronas un asinhronas saskaņošanas atbalsts.
 - Vājā vieta: vienlaicīgi izveidotas izmaiņu kopas; sarežģītas prasības transformācijai
- Apgrūtinājumi
 - Grūti noteikt operāciju, kura izsauca izmaiņas.
 - Grūti noteikt kādā secībā notikumi izpildīsies.
 - Grūti adekvāti reaģēt uz vienu notikumu, ja tas ir notikumu kopas daļa.
 - Utt.

Transformāciju pamatgrūtību risinājums

- Predefinēt kartēšanas modeļus un ģenerēt modeļu sinhronizācijas kodu atbilstoši šiem modeļiem
- Risinājuma vājā vieta ir nesankcionēta izejas modeļa (vai uzģenerēta koda) rediģēšana
 - ⚠ Viena rīka robežās tas nav traģiski, jo visur paredz reversa inženierijas iespējas
 - ⚠ Ja modeļi tiek nodoti vairākiem rīkiem un notiek daudzkāršas transformācijas, tad šī problēma ir diezgan liela un sarežģīta



MDA kartēšanas paņēmieni

- Modeļu tipu kartēšana
 - Metamodeļu kartēšana
 - Citu tipu kartēšana
- Modeļu eksemplāru kartēšana
 - Iezīmes
- Kombinēta tipu un eksemplāru kartēšana

MDA kartēšanas paņēmieni

■ Iezīmējošie modeļi

□ Iezīmes avoti ir sekojošie:

- Tipi no modeļa, aprakstītie ar klasēm, asociācijām, vai citiem modeļa elementiem
- Lomas no modeļa, piemēram, no šabloniem (patterns)
- Stereotipi no UML profiliem
- Elementi no MOF modeļa
- Modeļu elementi, aprakstītie citā metamodelī

■ Šabloni (templates)

■ Kartēšanas valodas

Kartēšanas likumi (Mapping Rules)

- Kartēšanas likums var būt izteikts vispārīgā veidā kā IF/THEN likums
 - IF daļa sastāv no parametrizēta vaicājuma avota modeļiem
 - THEN daļa sastāv no parametrizētiem darbību teikumiem, kuriem jābūt izpildītiem mērķa modeļos

IF

{parametrizēts vaicājums objektiem avotā 1} and

{parametrizēts vaicājums objektiem avotā 2} and

...

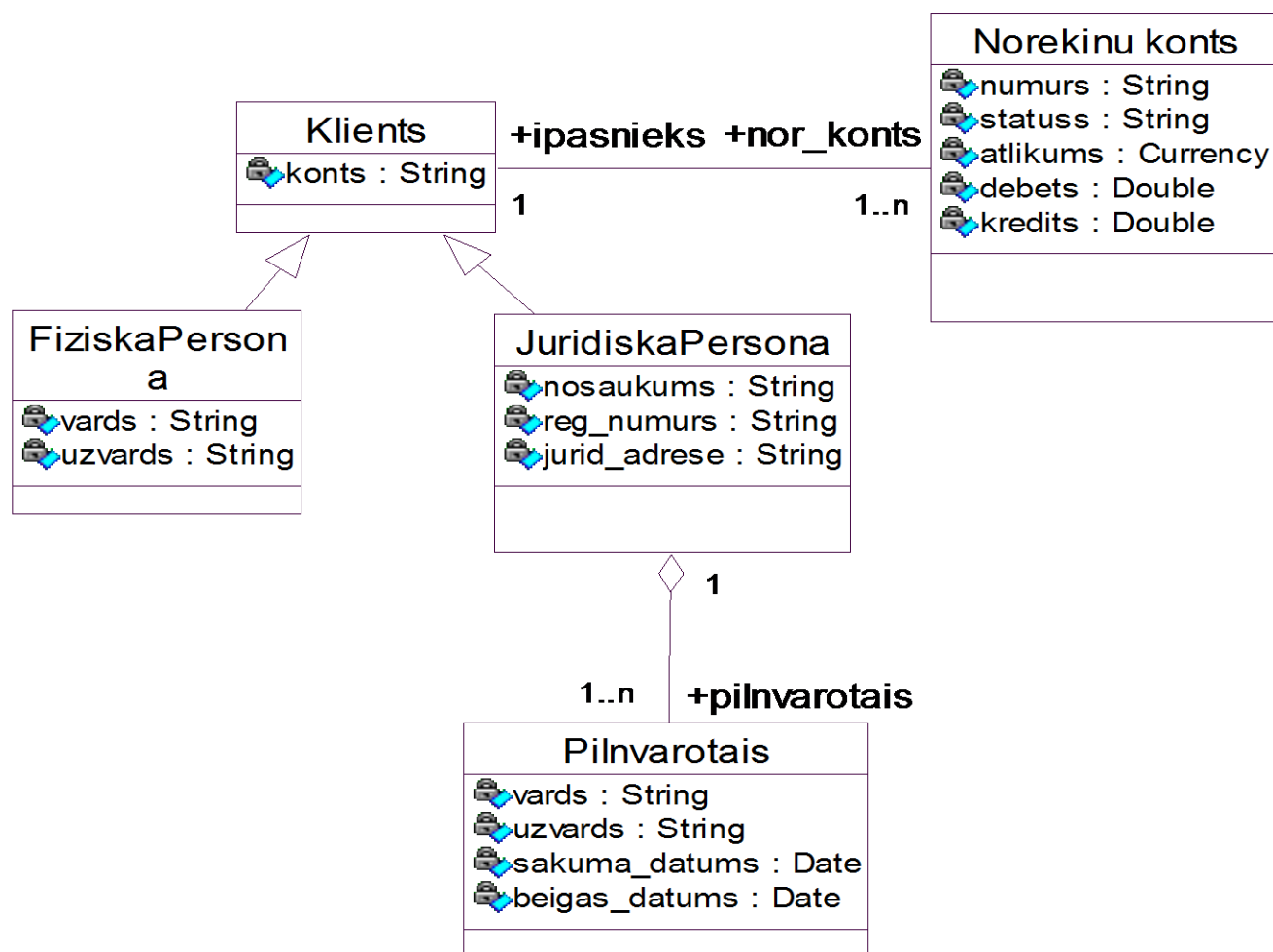
THEN

{darbības, kurām jābūt izpildītām mērķī 1}

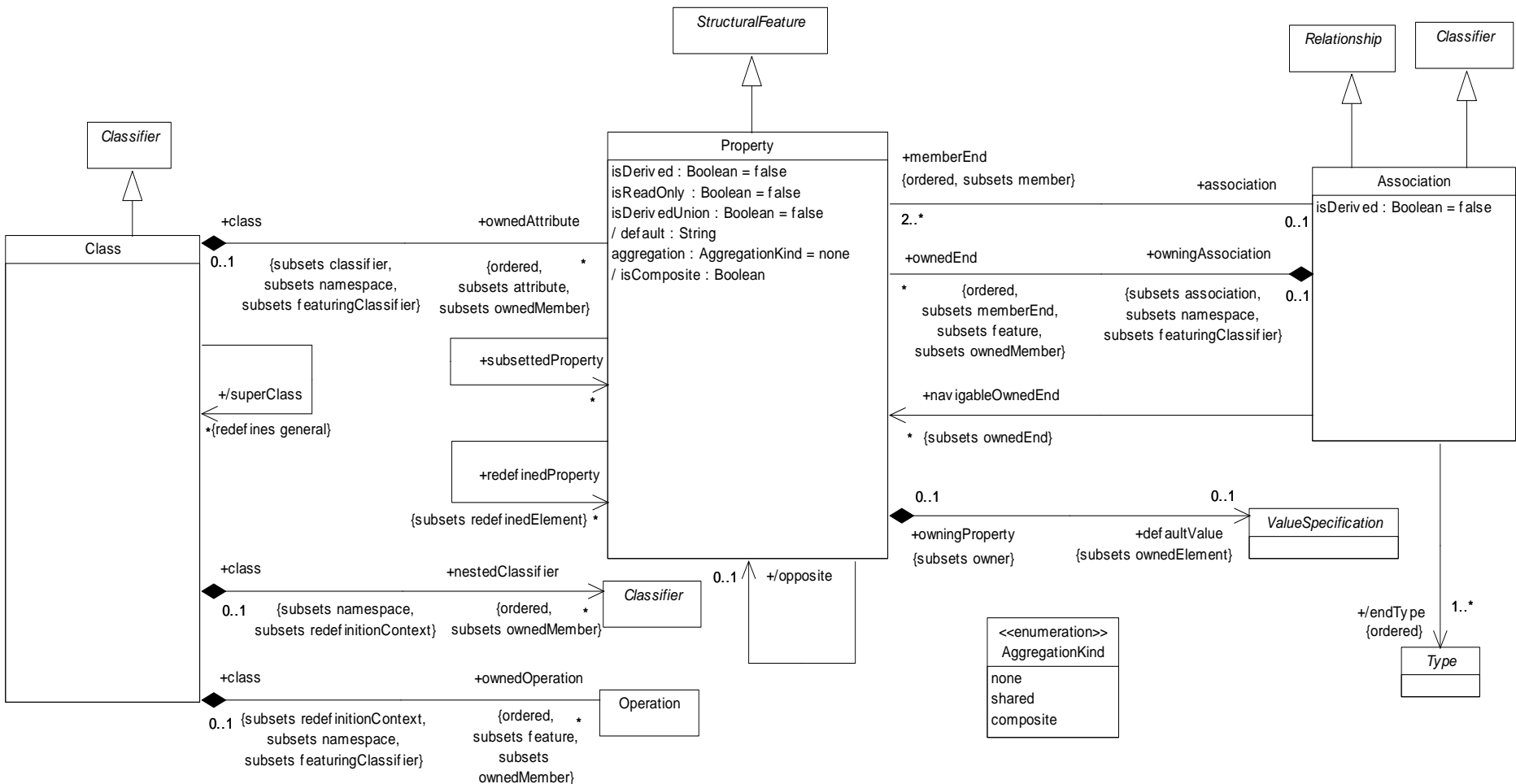
{darbības, kurām jābūt izpildītām mērķī 2}

...

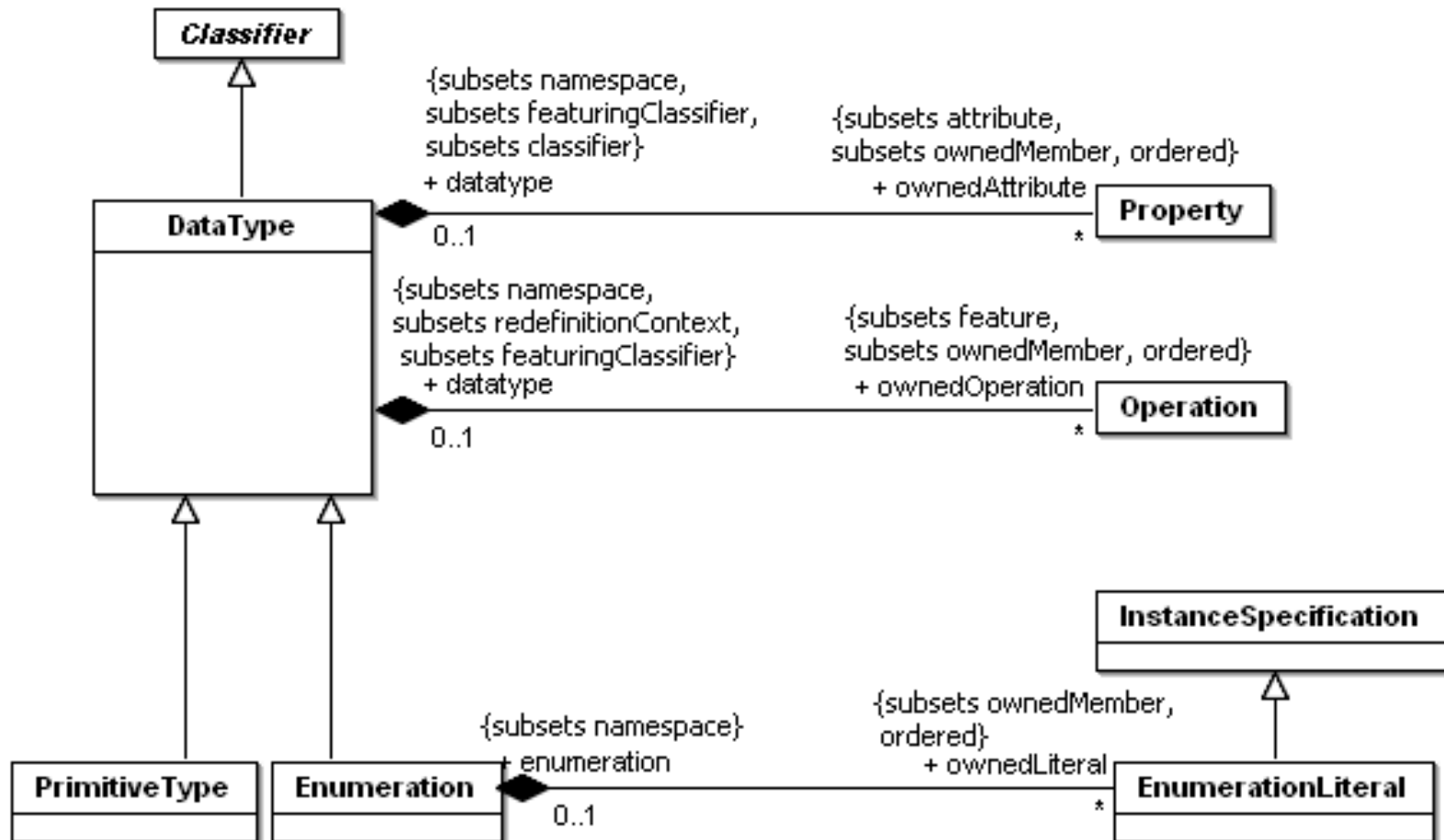
Kartēšanas likuma veidošanas piemērs



UML metamodel fragments



UML::DataType metamodelis



Kartēšanas likumi 1. piemēram

Ar dabisko valodu:

Ja klasifikators ir klase ar nosaukumu “Norekinu konts” un tam ir atribūts ar nosaukumu “debets”, kura tips nav “Currency”, **tad** piešķirt šim atribūtam tipu “Currency”

IF

{S: ?x īpašība datatype ir “UML::Class”} and

{S: ?x īpašība name ir “Norekinu konts”} and

{S: ?x īpašības ownedAttribute īpašība name = “debets” and
īpašība datatype nav “Currency”}

THEN

{T: set ?x īpašības ownedAttribute īpašība datatype = “Currency”}

Kartēšanas likums 2.piemērs

- Ja klasei “Fiziska persona” un klasei “Pilnvarotais” nav attiecību, tad izveidot jaunu agregācijas saiti starp tām.

IF

{S: ?x īpašība datatype ir “UML::Class”} and

{S: ?x īpašība name ir “FiziskaPersona”} and

{S: ?y īpašība datatype ir “UML::Class”} and

{S: ?y īpašība name ir “Pilnvarotais”} and

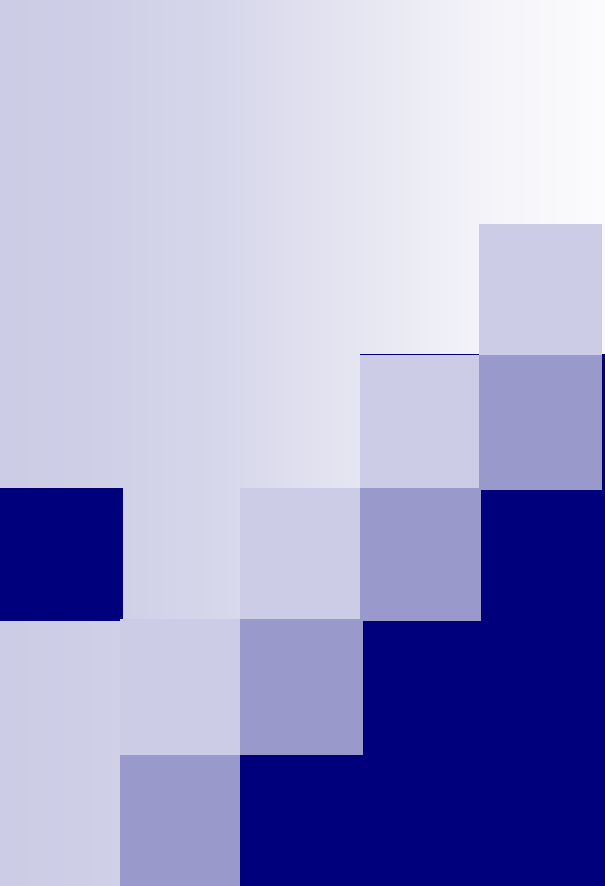
{S: neeksistē neviens ?y īpašības ownedAttribute elements ?z ar datatype vienādu ar “Pilnvarotais” un īpašību aggregation <> “shared”}

THEN

{T: izveidot jaunu atribūtu ?tz ar datatype = “Pilnvarotais” un name = “pilnv”} and

{T: set ?tz īpašībai aggregation vērtību “shared”}

{T: add ?x īpašībai ownedAttribute ?tz}



Transformācijas no modeļiem uz tekstu (M2C)

Kāpēc transformācijas no modeļa uz kodu ir nepieciešamas?

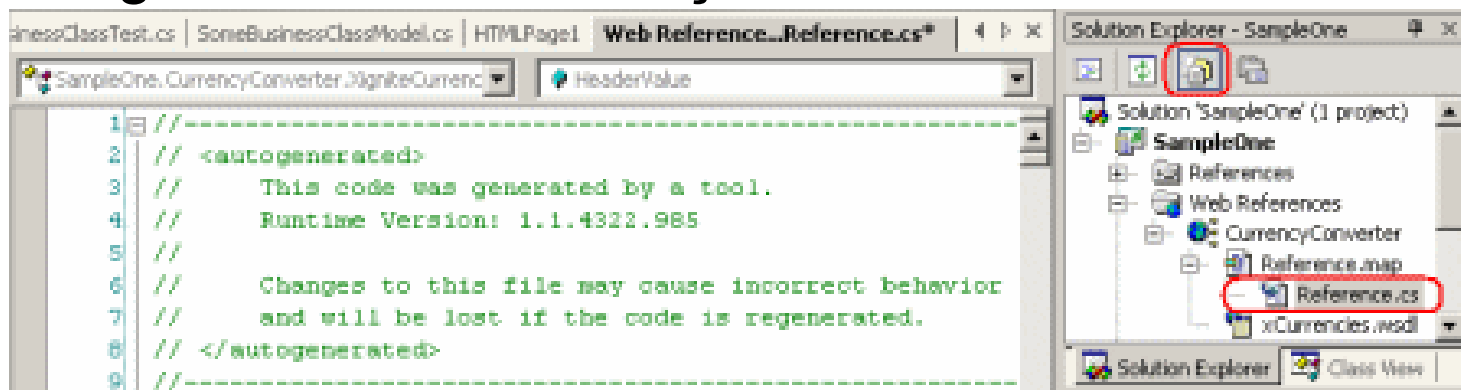
- Paaugstina abstrakcijas līmeni
 - Iegūtas sistēmas ir sarežģītākas
 - Abstrakcijas paaugstināšana ir pierādīti derīga
- Programmatūras izstrādes procesa automatizācijas
 - Samazina izstrādes laiku
 - Palielina programmatūras kvalitāti
 - Fokuss ir uz radošas daļas
- Jaunu artefaktu automatizēta ģenerēšana no modeļiem
 - Java, EJB, JSP, C#
 - SQL Scripts
 - HTML
 - Testa piemēri
 - Modeļu dokumentācija

Atpakaļejošas trasējamības problēma un risinājumi

- Neļaut nekontrolētu rediģēšanu, lai
 - Uzlabotu veiktspēju
 - Padarītu glītāku, lasāmāku, kompaktāku
 - Padarītu saskaņotu ar kodēšanas standartiem
 - Izlabotu kļūdas
- Aizsargāt uzģenerēto kodu
 - Izvietot marķierus uzģenerētā koda apzīmēšanai
 - Valodas direktīvas, kas paslēpj uzģenerēta koda apgabalu

.NET realizācija

1. Uzģenerēts kods tīmekļa atsaucei



2. Uzģenerēts kods saskarnes formai

#region Windows Form Designer generated code

/// <summary>

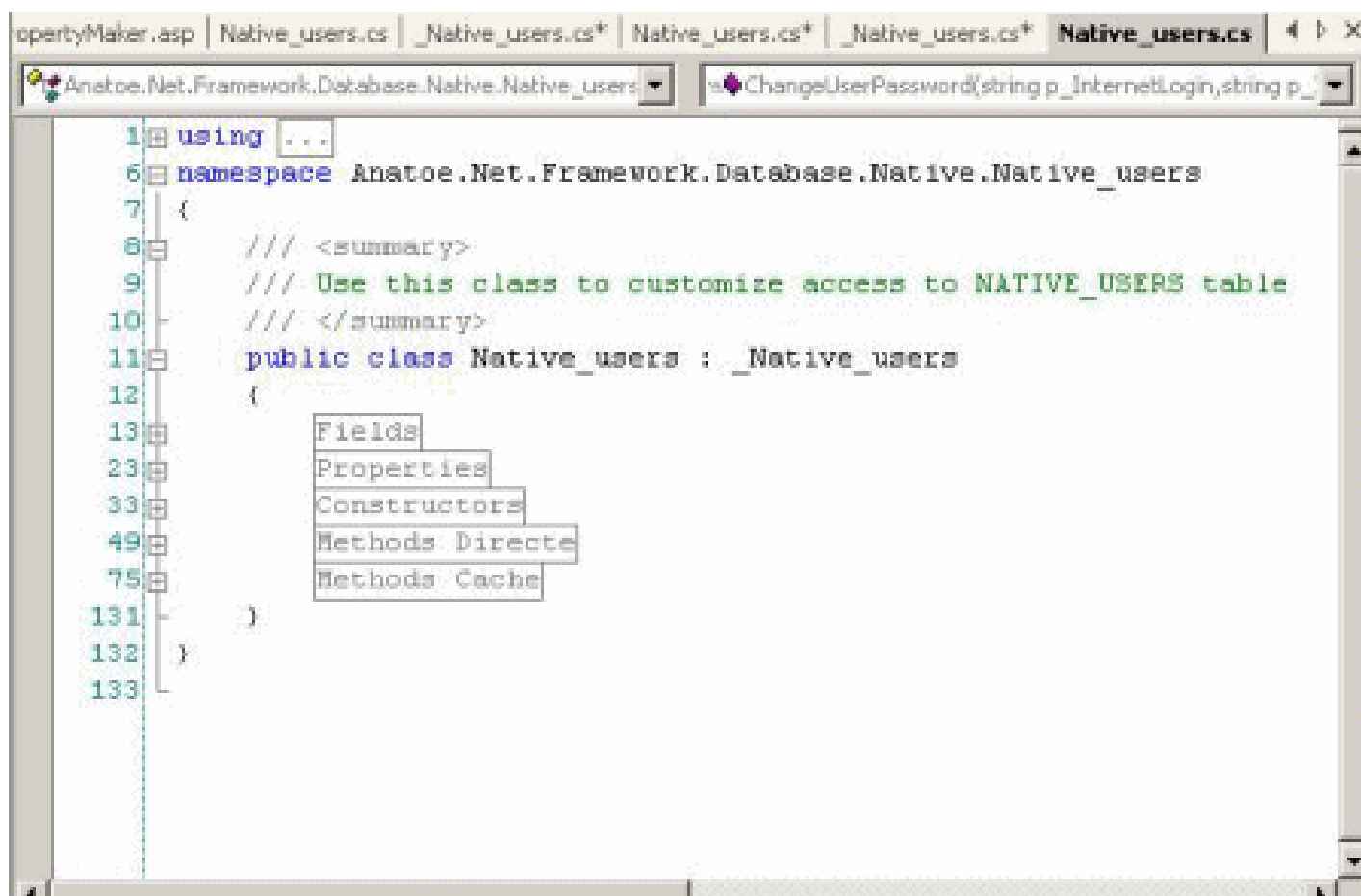
/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

```
private void InitializeComponent() {  
    this.button1 = new System.Windows.Forms.Button(); this.SuspendLayout();}
```

.NET realizācija

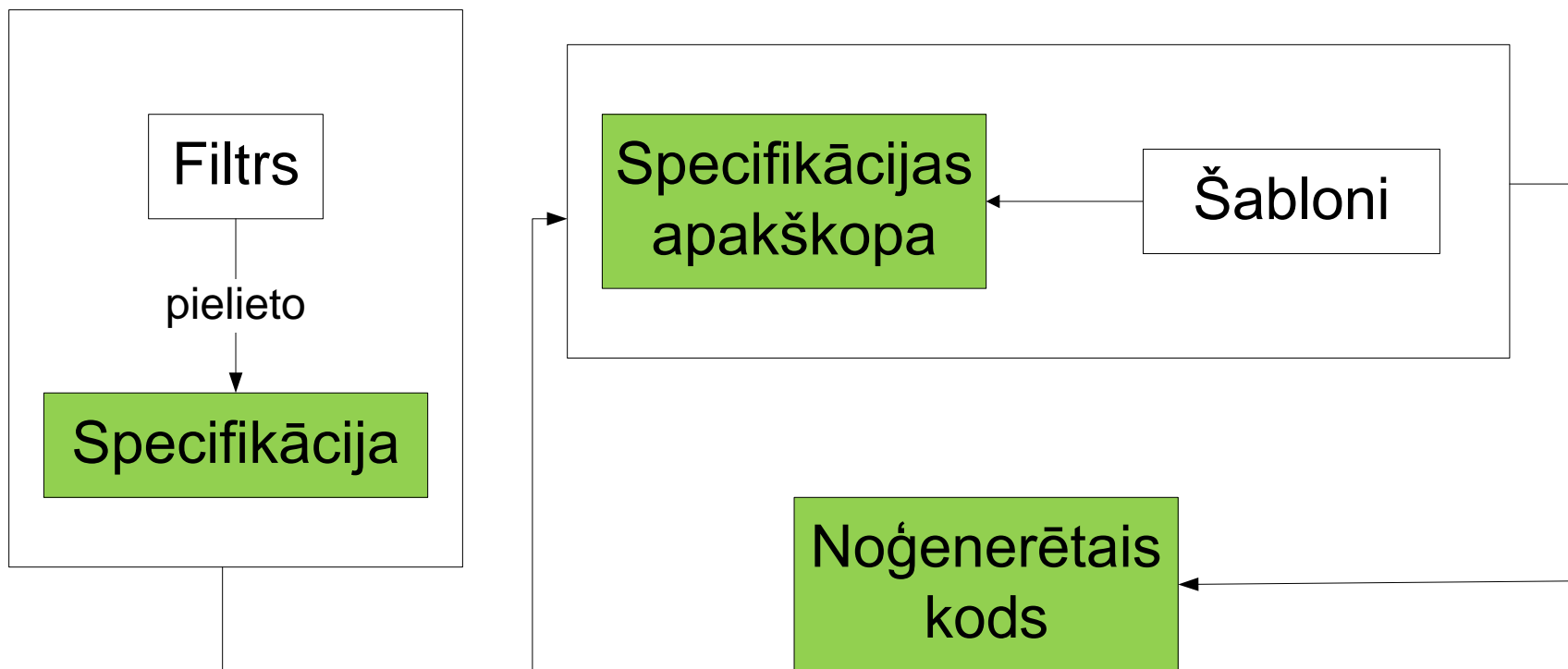


```
1 using ...
6 namespace Anatoe.Net.Framework.Database.Native.Native_users
7 {
8     /// <summary>
9     /// Use this class to customize access to NATIVE_USERS table
10    /// </summary>
11    public class Native_users : _Native_users
12    {
13        Fields
23        Properties
33        Constructors
49        Methods Directe
75        Methods Cache
131    }
132 }
133
```

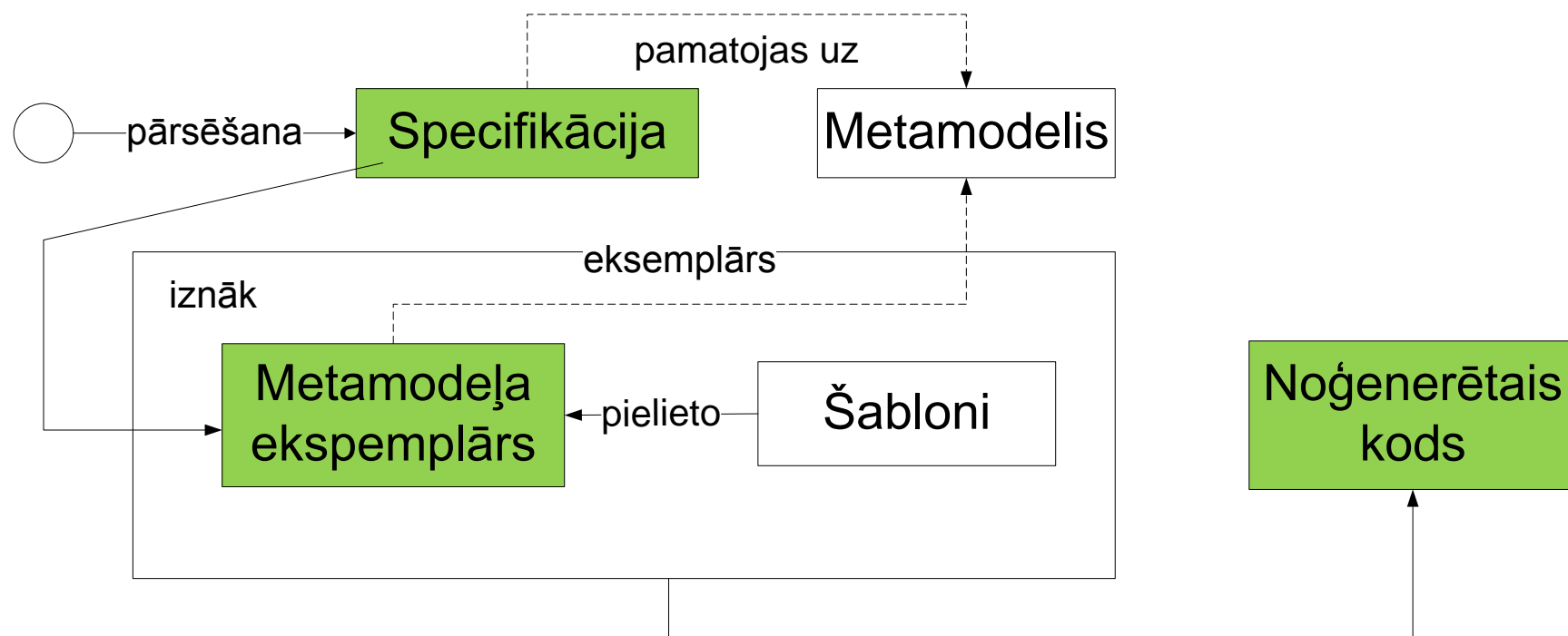
Ģenerēšanas metodes

- Šabloni + filtrēšana
- Šabloni + metamodelis
- Kadru procesori (frame processors)
- Uz API bāzētie ģeneratori
- Ierindota (in-line) ģenerēšana
- Koda atribūti
- Koda aušana (code weaving)
- Metožu kombinācijas

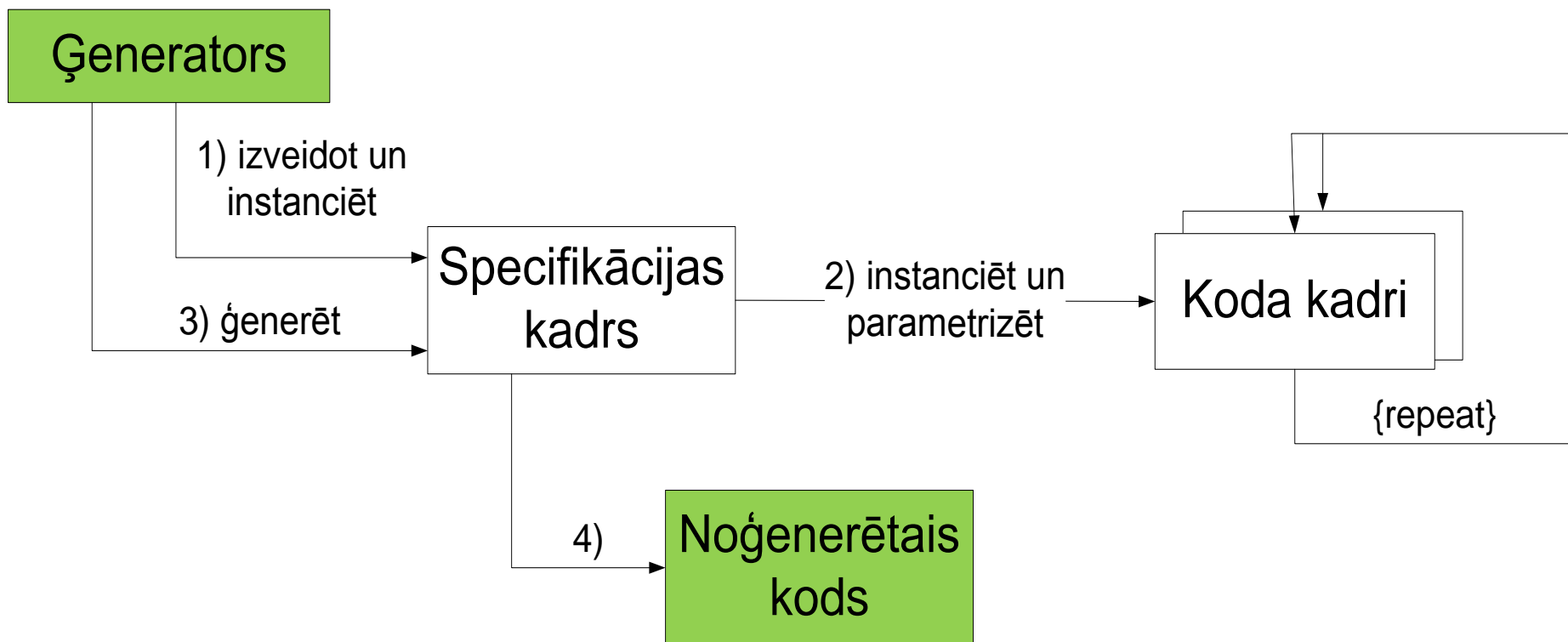
Šabloni + filtrēšana



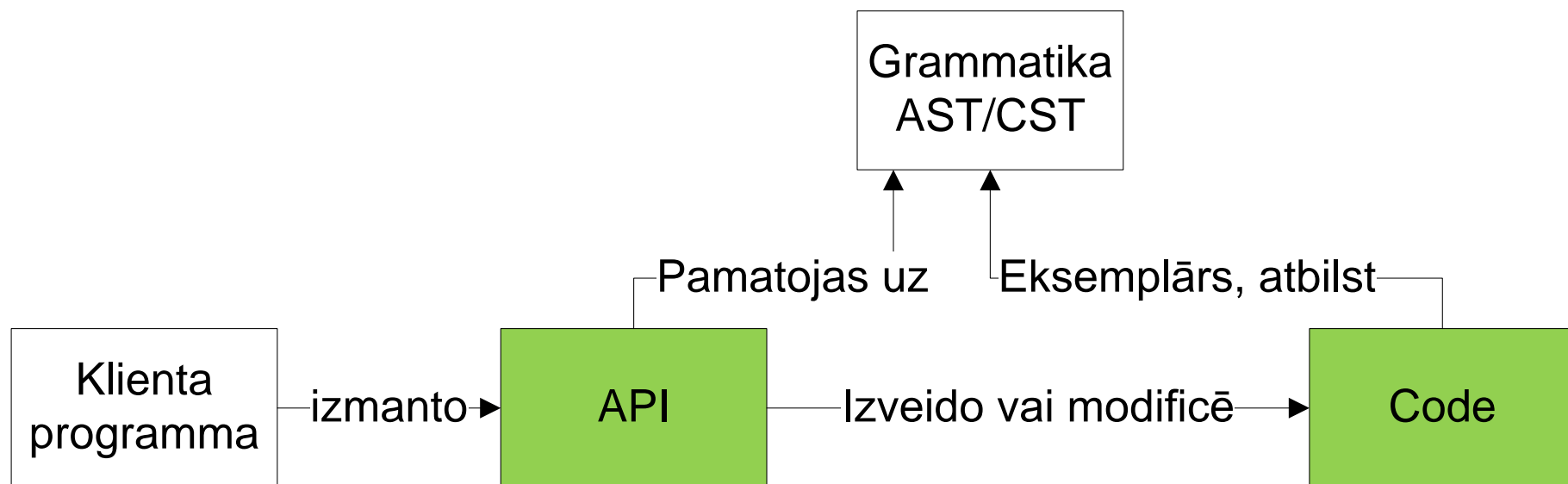
Šabloni + metamodelis



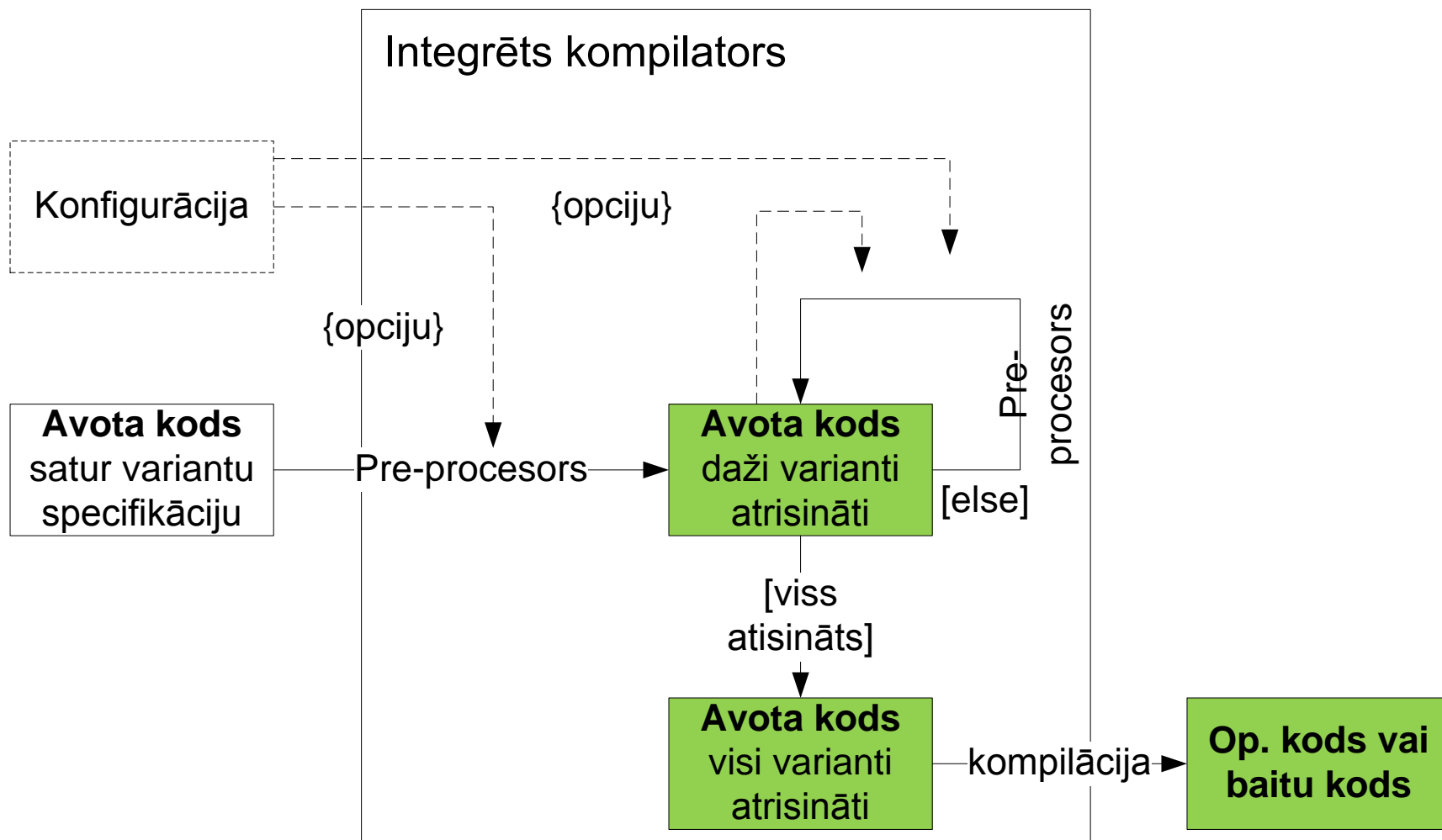
Kadru (frame) procesori



Uz API bāzētie ģeneratori



lerindota (in-line) ġenerēšana



Koda atribūti

.NET framework

[QoSServicePriority(Prio.HIGH)]

Class SomeService : ServiceBase {

 [QoSTimeLimit (100, Quantity.MS)]

 public void processRequest (Request r) {

 }

}

Koda aušana (code weaving)

```
aspect Logger {  
    public void log (String className, String methodName) {  
        System.out.println(className+"."+methodName);  
    }  
    pointcut accountCall():call(*Account.*(*));  
    before() calling: accountCall() {  
        log (thisClass.getName(), thisMethod.getName() );  
    }  
}
```

Koda aušana (code weaving)

```
public class SomeClass {  
    private Account someAccount = ...;  
    public someMethod (Account account2, int d) {  
        //aspect Logger  
        System.out.println("SomeClass.someMethod");  
        someAccount.add(d);  
        // aspect Logger  
        System.out.println("SomeClass.someMethod");  
        account2.subtract(d);  
    }  
    public void anotherMethod() {  
        //aspect Logger  
        System.out.println("SomeClass.anotherMethod");  
        int bal = someAccount.getBalance()  
    }  
}
```


Pieeju līdzības un atšķirības

	Kompilācijas laiks	Programma/ Metaprogramma	Ģenerēts/ Manuāli izveidots kods
Šabloni + filtrēšana	Pirms	Atsevišķi	Atsevišķi
Šabloni + metamodelis	Pirms	Atsevišķi	Atsevišķi
Kadru procesors	Pirms	Atsevišķi	Atsevišķi
Uz API bāzēts ģenerators	Pirms/laikā/pēc	Atsevišķi	Atsevišķi
Ierindota ģenerēšana	Pirms/laikā	Sajaukts	Integrēts
Koda atribūti	Pirms/laikā	Atsevišķs/sajaukts	Atsevišķs
Koda aušana	Pirms/laikā/pēc	Atsevišķs	Integrēts

UML klašu diagrammas iegūšana no koda (1)

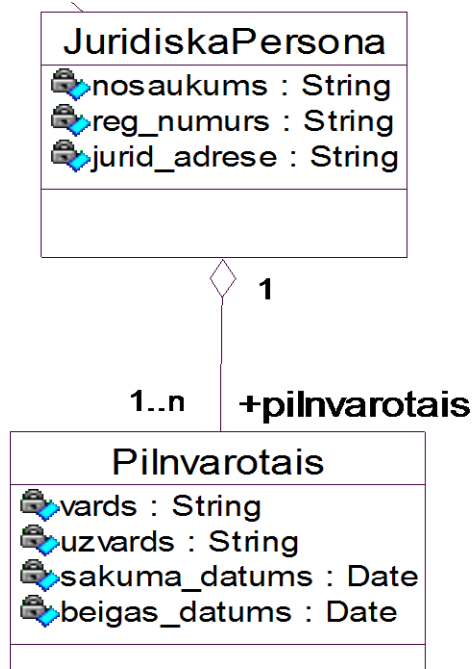
```
public class JuridiskaPersona {  
    private string nosaukums;  
    private string reg_numurs;  
    private string jurid_adrese;  
    public Pilnvarotais[] pilnvarotais;  
    // apjoms no 1 līdz n
```

```
    public JuridiskaPersona(){  
    }  
    ~JuridiskaPersona(){  
    }  
}
```

```
public class Pilnvarotais {  
    private string vards;  
    private string uzvards;  
    private DateTime sakuma_datums;  
    private DateTime beigas_datums;
```

```
    public Pilnvarotais(){  
    }  
    ~Pilnvarotais(){  
    }  
}
```

UML klašu diagrammas iegūšana no koda (1)

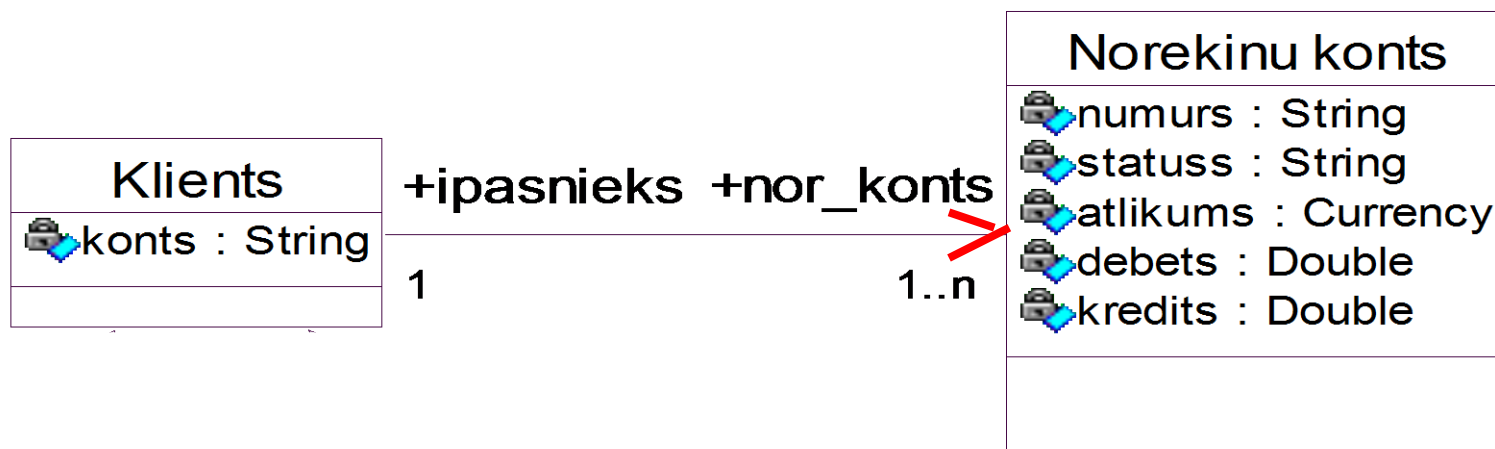


UML klašu diagrammas iegūšana no koda (2)

```
public class Klienti{  
    private string konts;  
    public Norekinukonts[] nor_konts;  
    // apjoms no 1 līdz n  
  
    public Klienti() {}  
    ~Klienti() {}  
}
```

```
public class Norekinukonts{  
    private string numurs;  
    private string statuss;  
    private Currency atlikums;  
    private double debets;  
    private double credits;  
  
    public Norekinukonts() {}  
    ~Norekinukonts() { }  
}
```

UML klašu diagrammas iegūšana no koda (2)



UML klašu diagrammas iegūšana no koda (3)

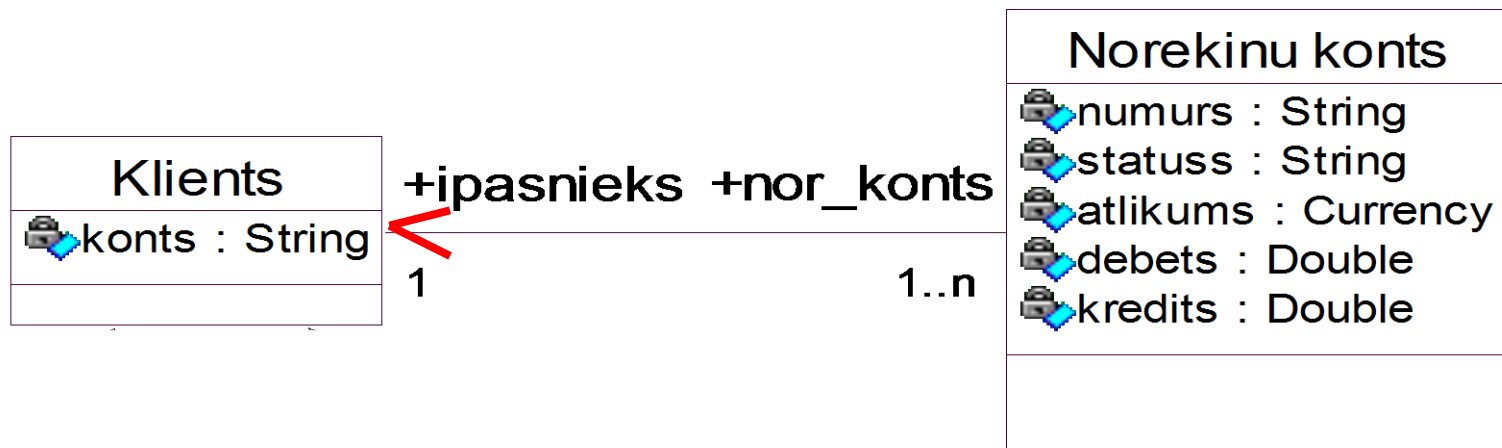
```
public class Klienti{  
    private string konts;
```

```
    public Klienti() {}  
    ~Klienti() {}  
}
```

```
public class Norekinukonts{  
    private string numurs;  
    private string statuss;  
    private Currency atlikums;  
    private double debets;  
    private double kredits;  
    public Klienti ipasnieks;
```

```
    public Norekinukonts() {}  
    ~Norekinukonts() { }  
}
```

UML klašu diagrammas iegūšana no koda (3)

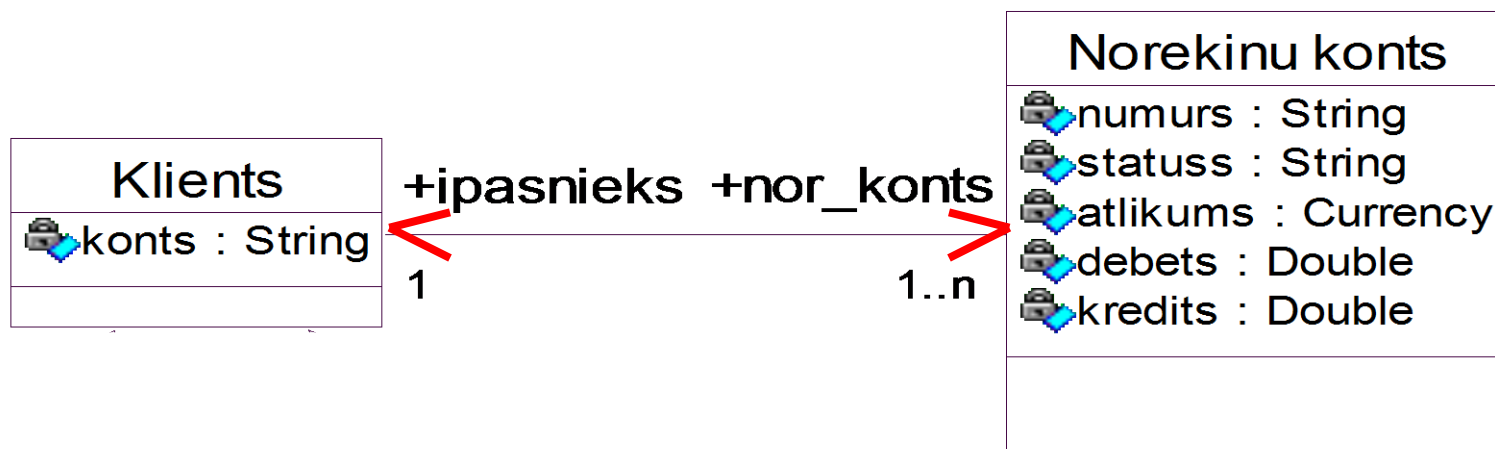


UML klašu diagrammas iegūšana no koda (4)

```
public class Klienti{  
    private string konts;  
    public Norekinukonts[] unor_konts;  
    // apjoms no 1 līdz n  
  
    public Klienti() {}  
    ~Klienti() {}  
}
```

```
public class Norekinukonts{  
    private string numurs;  
    private string statuss;  
    private Currency atlikums;  
    private double debets;  
    private double credits;  
    public Klienti ipasnieks;  
  
    public Norekinukonts() {}  
    ~Norekinukonts() { }  
}
```


UML klašu diagrammas iegūšana no koda (4)



UML klašu diagrammas iegūšana no koda (5)

```
public abstract class Klienti{  
    protected abstract string konts;  
  
    public Klienti() {}  
    ~Klienti() {}  
    public virtual void Dispose(){}  
}
```

```
public class JuridiskaPersona : Klienti {  
    private string nosaukums;  
    private string reg_numurs;  
    private string jurid_adrese;  
  
    public JuridiskaPersona(){  
    }  
    ~JuridiskaPersona(){  
    }  
    public override void Dispose(){}  
}
```

UML klašu diagrammas iegūšana no koda (5)

