KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Tomas Simonaitis

UML klasių ir sekų diagramų transformavimas į programos kodą

Magistro darbas

Darbo vadovas
doc. dr. V. Pilkauskas

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAKTINĖS INFORMATIKOS KATEDRA

Tomas Simonaitis

UML klasių ir sekų diagramų transformavimas į programos kodą

Magistro darbas

Kalbos konsultantė
Lietuvių k. katedros lekt.
I. Mickienė
2006-05-22

Vadovas
doc. dr. V. Pilkauskas
2006-05-25

Recenzentė
doc. dr. L. Nemuraitė
2006-05-25

Atliko
IFM-0/1 gr. stud.
Tomas Simonaitis
2006-05-25

Kaunas, 2006

Turinys

Lentelės

Paveikslai

SUMMARY

Presented work analyzes possible scenarios of program code generation from UML class and sequence diagrams. Two different approaches to development using MDA architecture are discussed: elaboration and translation. Both, static code structure and execution aspects are reviewed.

New semantics of UML version 2.0 and their implications on model description are discussed in detail. Special attention is granted to *opaqueExpression* construct, analyzing its possible misuses.

Some of currently available tools are tried and analyzed. Found drawbacks are important when custom prototype code generator is developed.

Prototype code generator is used to check if current UML specification is elaborate enough for complete code generation. Author takes bottom-up approach, starting with generation of common code expressions, static structure, until complete program skeleton can be created.

1. ĮVADAS

OMG sukurtas UML standartas plačiai naudojamas programinės įrangos kūrimo procese, tačiau dažniausiai tik reikalavimams apibrėžti ir planuojamai programos struktūrai aprašyti. Realizuojant programą vadovaujamasi sudarytais UML modeliais, bet kodas kuriamas tradiciniais metodais.

Programų inžinerija (PI) iš kitų inžinerijos sričių išsiskiria tuo, jog joje nėra tiksliai specifikuoto kūrimo proceso. Įvairios kompanijos praktikuoja skirtingus programų kūrimo būdus: tradicinį krioklio modelį, XP (angl. *extreme programming*), kūrimą išskiriant galutinės sistemos ypatybes (angl. *feature oriented programming*), aspektais paremtą programavimą (angl. *aspect oriented programming*). Sunku pasakyti ar tokia situacija yra ženklas, jog PI visiškai nepanaši į kitas inžinerijos sritis, ar tiesiog parodo, jog ji dar nėra pakankamai subrendusi.

Šio darbo tikslas yra apžvelgti galutinio kodo generavimo iš UML modelių galimybes. Manoma, kad efektyvus galutinio kodo sugeneravimas, būtų paskatinimas kurti išsamius, tiksliai sistemą atitinkančius modelius. Tokiu būdu tampriai susiejant modelį su realizuota sistema.

Darbe nagrinėjami svarbiausi UML specifikacijos elementai, akcentuojant galimybę iš jų generuoti konkrečias kodo struktūras.

Kadangi kodo generavimą nuspręsta atlikti iš XMI formatu saugomų modelių, naudojama XSLT transformacijų kalba.

2. MDA ARCHITEKTŪRA

Didžiausias tradicinio programinės įrangos projektavimo ir kūrimo trūkumas yra silpnas ryšys tarp projekto modelio (verslo taisyklės, reikalavimai) ir realizacijos.

Tradicinis programų kūrimo procesas susideda iš šių pagrindinių stadijų:

- ✔ Reikalavimų surinkimas.
- ✔ Analizė.
- ✔ Projektavimas.
- ✔ Kodo rašymas.
- ✔ Testavimas.
- ✔ Diegimas.

Tradicinio kūrimo proceso trūkumai pradeda ryškėti pradėjus rašyti kodą. Kadangi paruošti dokumentai (reikalavimų specifikacijos, funkcinės specifikacijos, schemos ir pan.) nėra susiejami su rašomu kodu, dėl bet kokio programos struktūros ar funkcionalumo pasikeitimo tenka keisti tiek programos kodą tiek specifikacijas. Šis dvigubas darbas reikalauja daug žmogiškųjų resursų, neretai specifikacijos ir reikalavimų dokumentai nėra atnaujinami.

Nuolat tobulinant sistemą pirmų trijų etapų darbas nuvertėja: specifikacijos aprašo projektuotą ją sistemą, o ne galutinį variantą.

Dar vienas svarbus tradicinio programų kūrimo proceso trūkumas – tamprus ryšys su konkrečia platforma ir/ar operacine sistema.

OMG standartas MDA apibrėžia programinės įrangos kūrimo procesą kuriame svarbiausią vietą užima sistemos projektavimas, o kodo rašymas, idealiu atveju, turėtų būti automatinis.

Sistemos kūrimo projektinė dalis turėti būti grindžiama išsamaus, nuo platformos nepriklausomo, modelio (angl. *Platform Independent Model*) sukūrimu.

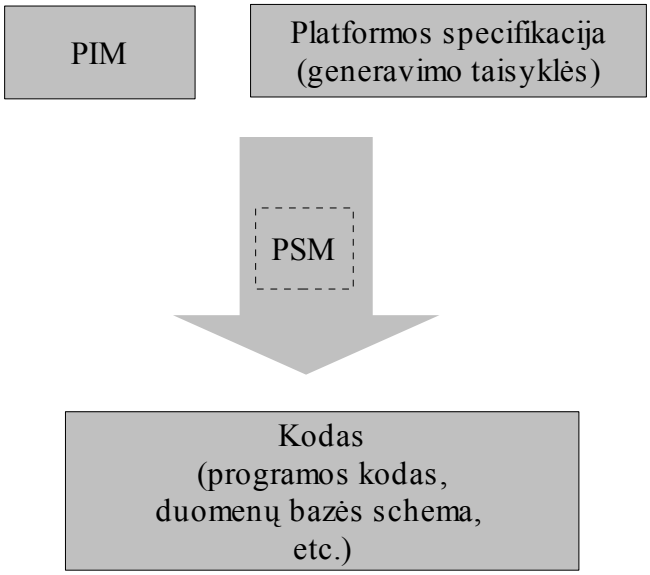
PIM modelis turi pateikti sistemos aprašymą, kiek galima mažiau detalizuojant realizacijai specifines detales. Kuo abstraktesnis PIM modelis, tuo daugiau platformų bus tinkamos galutiniam produktui naudoti. Vienas iš būdų kurti PIM modelį yra naudoti tik nuo technologijos nepriklausomos virtualios mašinos funkcionalumą.

3. MDA STRATEGIJOS

Pagrindinis MDA tikslas yra atskirti sistemos aprašymą nuo platformos, kurioje ji bus naudojama. Taip būtų užtikrintas verslo logikos išsaugojimas, net jeigu technologinė sistemos platforma pasikeistų.

MDA rėmėjus galima grubiai skirti į dvi grupes: remiančius visapusišką sistemos generavimą pasitelkiant nuo platformos nepriklausančius modelius (PIM), ir agituojančius naudoti modelių ir kodo patikslinimą, kol sukuriamas galutinis produktas.

Generuojant galutinę sistemą iš PIM modelių, turėtų būti naudojamas įrankis, kuris turėdamas platformos specifikaciją bei logines PIM struktūras gali visiškai atlikti PIM->kodas transformaciją (Pav. 1)



Pav. 1. Visapusiška generavimo strategija

Kadangi PIM modelyje neturi būti saugoma jokia platformos specifika ar realizacijos detalės, transformuojantis įrankis vieną PIM elementą gali pakeisti sudėtinga kodo struktūra.

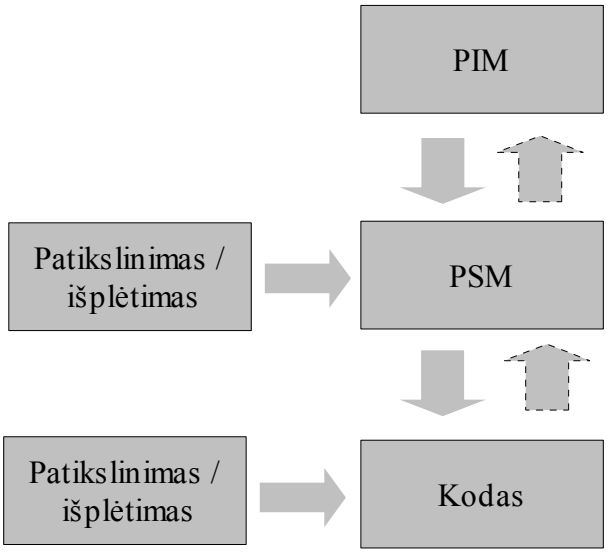
Laikant, kad transformuojantis įrankis veikia korektiškai, iš teisingo PIM modelio bus sugeneruojamas teisingas tarpinis PSM modelis, o vėliau ir galutinė sistema. Be to, garantuojama, kad kodas visuomet tiksliai atspindi PIM modelį.

Sugeneruotas kodas nėra keičiamas, todėl nebūtinai turi būti aiškios struktūros, vietoje to generatorius kodo struktūrą gali optimizuoti (spartai/maksimaliam veiksmų lygiagretumui ar pan.). Taip pat nebūtina ir galimybė sinchronizuoti PIM pakeitus kodą ir PSM modelį.

Visapusiško generavimo metodo kritikai teigia, jog tokia idealizuota schema praktikoje tinka tik keletui specializuotų sričių, o visos sistemos veiklos aprašymas PIM modeliu pernelyg sudėtingas. Vietoje to siūloma naudoti modelių patikslinamą (Pav. 2): paruošus PIM,

automatiškai sugeneruojamas pirminis PSM modelis, kuris išplečiamas platformai specifiniais elementais, o tada automatiškai generuojamas pirminis kodo skeletas, kurį vėlgi reikia papildyti, kol gaunama galutinė sistema.

Dirbant tokiu principu būtina išlaikyti atgalinį ryšį tarp PSM->PIM ir Kodo->PSM: patikslinant žemesnio lygio modelį ar kodą jis gali visiškai nebeatitikti tėvinio modelio, todėl įrankis turi sugebėti aptikti pasikeitimus ir atlikti sinchronizavimą.



Pav. 2. Patikslinimo strategija

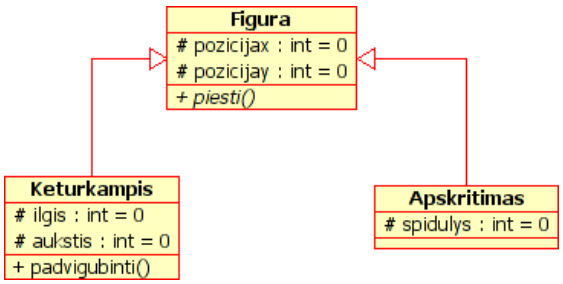
Kadangi sugeneruotas kodas bus papildomas programuotojo, jis turi būti lengvai skaitomas ir suprantamas.

4. MDA ĮRANKIŲ ANALIZĖ

4.1 Statinės struktūros generatoriai

Šiuo metu populiariausi ir plačiausiai praktikoje naudojami yra statinės programos struktūros generatoriai. Tokie įrankiai iš UML klasių diagramos sukuria klasių skeletus: aprašo atributus, metodus, automatiškai kuria savybių metodų aprašus (*get/set*).

Kadangi UML klasių diagramos pakankamai išsamiai aprašo elementus naudojamus OO programavimo kalbose, transformacija yra paprasta ir patikima.



Pav. 3. Primityvi klasių diagrama

Naudodami *Umbrello UML Modeler* programą, iš paprastos klasių diagramos (Pav. 3) galime gauti sistemos karkasą pageidaujama programavimo kalba, pvz. *Java*.

```
import Figura;
/**
 * Class Keturkampis
 */
public class Keturkampis extends Figura {
  protected int ilgis = 0;
  protected int aukstis = 0;
  // Methods
  // Accessor Methods
  /**
   * Get the value of ilgis
   * @return the value of ilgis
   */
  protected int getIlgis (  ) {
    return ilgis;
  }
  /**
   * Set the value of ilgis
   */
  protected void setIlgis ( int value  ) {
    ilgis = value;
  }
  /**
   * Get the value of aukstis
   * @return the value of aukstis
   */
  protected int getAukstis (  ) {
    return aukstis;
  }
  /**
   * Set the value of aukstis
   */
  protected void setAukstis ( int value  ) {
    aukstis = value;
  }
  // Operations
  /**
   * @return void
   */
  public void padvigubinti () {
  }
}
```

Toks karkasas patogus startinis taškas, o tolesnis sistemos realizavimas atliekamas programuojant pasirinktame redaktoriuje bei nuolat sinchronizuojant UML klasių modelį ir programos kodą.

Umbrello privalumai:

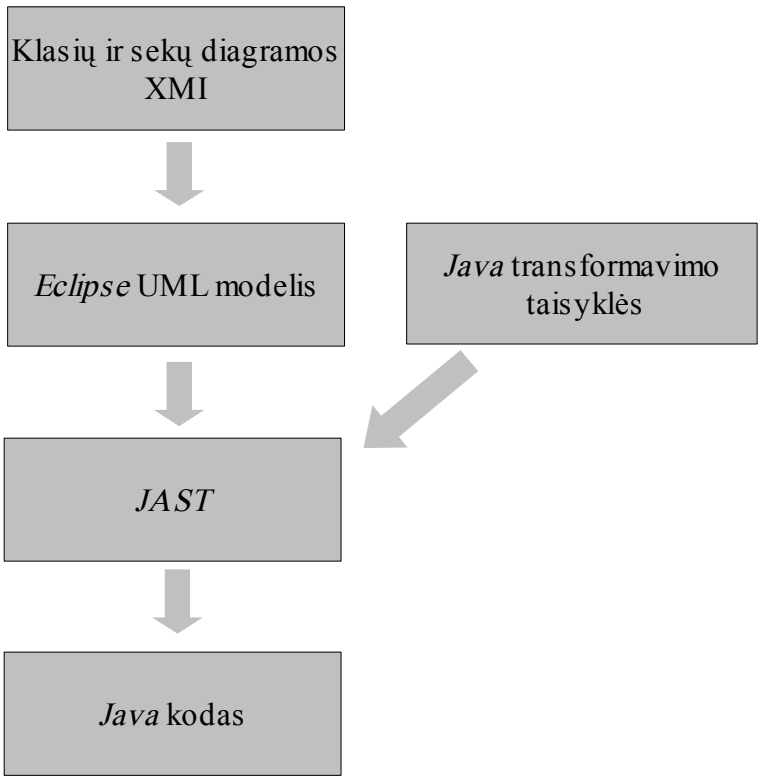
- ✔ Sugeneruojamas kompiliavimui paruoštas programos karkasas
- ✔ Sugeneruojama metodų, atributų, argumentų dokumentacija
- ✔ Sugeneruojami *get/set* metodai

Umbrello trūkumai:

- ✔ Kodas generuojamas tik iš klasių diagramų
- ✔ Menkai palaikomi stereotipai

4.2 Visapusiški generatoriai

Generatoriai, iš UML modelio sugeneruojantys visą galutinio produkto kodą, dažniausiai skirti specializuotai sričiai arba yra kūrimo stadijoje. Plačiau nagrinėjamas eksperimentinis generatorius *Matilda.*



Pav. 4. Matilda sistemos darbo principas

Matilda skirtas pilnų (angl. *computationally complete*) modelių transformavimui į galutinį *Java* kodą. Kadangi sugeneruojamas visas produkto kodas, išvengiama kodo susiejimo su modeliu problema – keičiamas tik modelis ir transformavimo taisyklės, tuomet vėl sugeneruojamas galutinis kodas. Šiuo metu *Matilda* transformuoja UML 2.0 klasių ir sekų diagramas, kitos diagramas (*state, activity, deployment*) nenagrinėjamos. Modeliai pateikiami XMI dokumentais.*Matilda* naudoja nuoseklaus nagrinėjimo (angl. *pipelines*) architektūrą: atskiras įskiepis nagrinėja jam perduotą modelį, jį patikrina ar transformuoja, ir perduoda sekančiam įskiepiui. Validavimo įskiepiai, aptikę modelio problemą, gali transformacijų seką nutraukti. Be to, kiekvienas įskiepis gali skaityti ir modifikuoti bendrus būsenos kintamuosius.

Matilda apibrėžia kelis stereotipus, būtinus visapusiškam kodo generavimui:

- ✔ *<<UMLVMexecutable>>* pažymi startinę sistemos klasę (*Java* atveju, klasė turinti viešą, statinį *main* metodą);
- ✔ *<<UMLVMarrayelement>>* stereotipas kartu su *index* žyme (angl. *tag*) skirtas masyvo elemento susiejimui sekų diagramos žinutėje;
- ✔ *<<JavaInterface>>* nurodo, kad sąsaja (angl. *interface*) gali turėti konstantų.

Matilda vykdymo specifikos aprašymui naudoja UML neapibrėžtas išraiškas (5.3, *17* psl.) *Java* kalba. Ištrauka iš XMI modelio:

```
...
<specification xmi:type="uml:Expression" name="Expr" body="else" language="Java"/>
...
        <argument xmi:type="uml:OpaqueExpression" body="Calculator calc = new Calculator()"
language="Java"/>
...
xmi:type="uml:OpaqueExpression" body="calculate(args)" language="Java"/>
...

<argument xmi:type="uml:OpaqueExpression" body="java.lang.Double result = calc.calculate(args)"
language="Java"/>
...
<specification xmi:type="uml:Expression" name="Expr" body="result == null" language="Java"/>
...
<argument xmi:type="uml:OpaqueExpression" body="out.println(&quot;Error: Malformed input
expression!&quot;)" language="Java"/>
...
```

Generuodama programos kodą, *Matilda* sistema neapibrėžtų išraiškų kūnus naudoja nuosekliai nagrinėdama sekų diagramą. Iš aukščiau pateikto XMI fragmento gaunamas *Java* kodas:

```
...
    public static void main(String[] args)
    {
...
            Calculator calc = new Calculator();
            Double result = calc.calculate(args);
            if (result == null)
                System.out.println ("Error: malformed input expression !");
....
    }
...
```

Matilda sistemos privalumai:

- ✔ Sugeneruojama 100% galutinio programos kodo.
- ✔ Pasirinkta architektūra leidžia atlikti paskirstytą modelių transformavimą.
- ✔ Naudojami tik UML 2.0 specifikacijoje numatyti modelių elementai.

Matilda sistemos trūkumai:

- ✔ Modelio sekų diagrama privalo būti visapusiškai aprašyta.
- ✔ Kiekvienai programavimo kalbai reikia pakeisti didelę dalį modelio.
- ✔ Generatorius turi mažai galimybių optimizuoti galutinį kodą.

5. SEKŲ DIAGRAMŲ ELEMENTAI

UML 2.0 specifikacijoje žymiai išplėstos sekų diagramų galimybės, ypač sąveikų (*Interactions*) paketas.

Sąveikos naudojamos tiksliam bendravimo tarp sistemos objektų aprašymui ir tiksliam vykdymo sekos apibūdinimui. Įprastai tik pačios svarbiausios sąveikos yra apibrėžiamos, naudojant sekų diagramas, kodo generavimui visi galimi sistemos veikimo scenarijai turi būti visapusiškai aprašyti.

5.1 CombinedFragment elementas

CombinedFragment elementas pateikia sąveikų naudojimo taisykles, pvz. galima aprašyti, kurias sąveikas reikia vykdyti priklausomai nuo išraiškos interpretavimo rezultato. Elemento semantika apsprendžiama pagal *InteractionOperator* atributo reikšmę (*seq, alt, opt, break, par, strict, loop, critical, neg, assert, ignore, consider*), interpretuojama išraiška – *InteractionOperand* atributų. Jeigu *InteractionOperator* reikšmė yra *opt, loop, break* arba *neg*, fragmentas gali turėti tik vieną *InteractionOperand* operandą.

5.1.1 Alternatyvos parinkimas

Nustačius *InteractionOperator* reikšmę *„alt“ CombinedFragment* elementas naudojamas alternatyviam vykdymo keliui išrinkti. Renkant vykdymo kelią, iš eilės analizuojamos visų operandų saugos išraiškos (*guard expressions*), jeigu jos interpretavimo rezultatas *true* – operandas parenkamas, o likusieji nėra nagrinėjami. Jeigu saugos išraiška nėra nurodyta laikoma, jog jos rezultatas visuomet *true* – taip patogu aprašyti situaciją „visais kitais atvejais...“. Jeigu nė vienos saugos išraiškos rezultatas nėra *true,* nė vienas operandas nėra išrenkamas.

Supaprastintam alternatyvos atvejui, kai tiesiog norima išrinkti operandą arba nevykdyti jokios operacijos, galima naudoti *InteractionOperator* tipą *„opt“*.

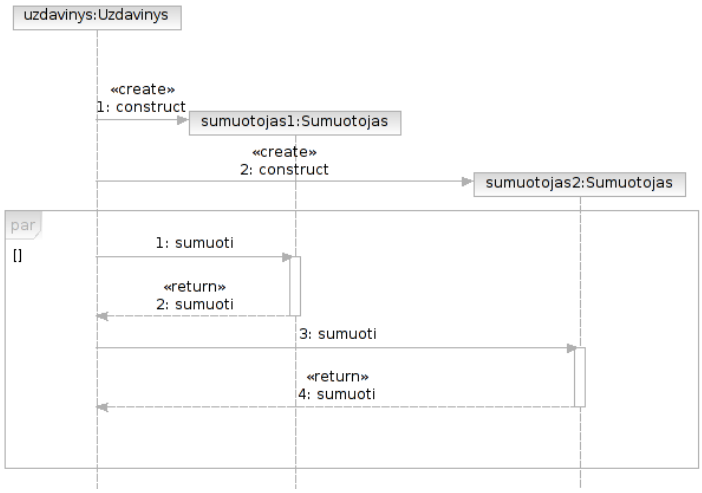
5.1.2 Vykdymo nutraukimas

InteractionOperator tipas *„break“,* skirtas aprašyti situacijai kai norima nutraukti tėvinės sąveikos vykdymą ir išrinkti *„break“* operandą. Operandas renkamas, kai saugos išraiškos rezultatas *true*, kitu atveju jis ignoruojamas, ir baigiama vykdyti tėvinė sąveika.

5.1.3 Vykdymo lygiagretumas

Svarbi galimybė sistemos darbo eigos apraše – galimybė nurodyti, kurios operacijos gali būti vykdomos lygiagrečiai. Tam naudojamas *InteractionOperator* tipas *„par“*.

„par“ tipo bloke esantys elementai vykdomi neapibrėžta tvarka (generatorius gali sukurti lygiagrečiai vykdomą kodą arba pakeisti elementų išdėstymo tvarką).

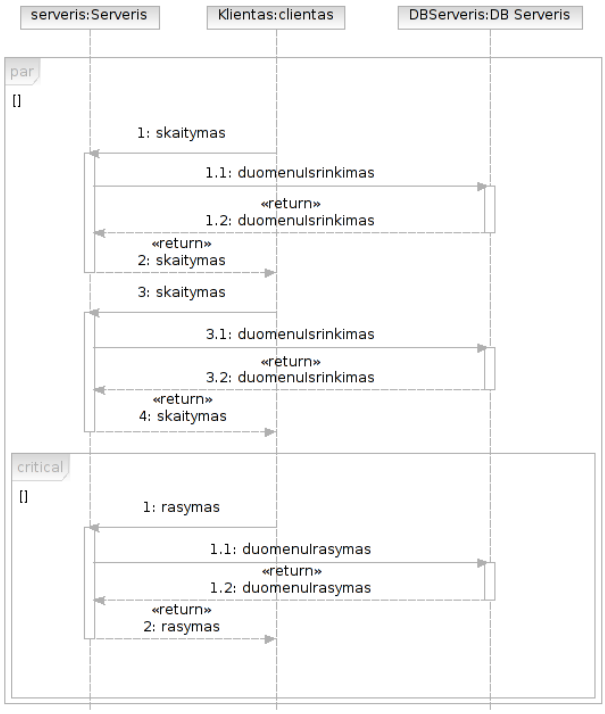


Pav. 5. Lygiagrečiai vykdoma seka

Pav. 5 parodyta primityvi lygiagrečių operacijų *sumuoti* sekų diagrama. Tokia struktūra garantuoja *Sumuotojas* tipo objektų kūrimo tvarką, tačiau neapibrėžia, kokia tvarka reikia išsiųsti sinchronines žinutes *sumuoti*.

5.1.4 Kritiniai regionai

Didelį fragmentą paskelbus lygiagrečiu, patogu dalį sąveikos apibrėžti kaip atominę. Toks regionas aprašomas *InteractionOperator* tipu *„critical“*. Vykdant kritinį regioną turi būti garantuojama, jog visos likę tėvinio fragmento operacijos nebus vykdomos.



Pav. 6. Kritinio regiono išskyrimas

Pav. 6 pateikiamas kritinio regiono išskyrimo pavyzdys, *Klientas* gali lygiagrečiai atlikti kelias skaitymo operacijas, rašymo operacija visuomet atliekama atomiškai.

5.1.5 Ciklų išskyrimas

Sudėtingoms vykdymo sekoms aprašyti būtina ciklų sąvoka, ji įvedama naudojant *InteractionOperator* tipą *„loop“*. Ciklo operandas (privalo būti tik vienas) vykdomas tam tikrą skaičių kartų. Ciklo kartojimas gali būti apibrėžtas:

- ✔ nurodžius *minint* ir *maxint* kaip skaitinius parametrus, ciklas vykdomas nuo *minint* iki *maxint* kartų;
- ✔ nurodžius tik *minint,* laikoma jog *maxint=minint;*
- ✔ nenurodžius nei *minint* nei *maxint* jie laikomi atitinkamai 0 ir * (begalybė)

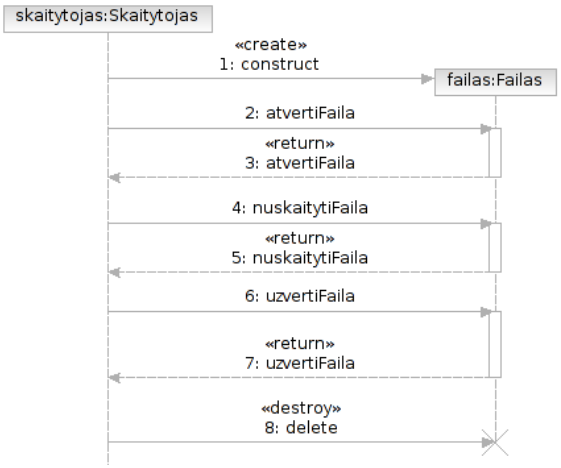
Nustačius minimalų ir maksimalų vykdymo skaičių cikle, turi būti tikrinama nurodyta saugos išraiška, kai ciklas atliktas daugiau nei *minint* kartų ir saugos išraiškos rezultatas *false* – ciklo vykdymas turi būti nutraukiamas.

5.2 Įvykiai

UML 2.0 specifikacijoje buvo įvesti du svarbūs įvykiai: objekto kūrimo ir objekto naikinimo.

CreationEvent naudojamas, siekiant nurodyti, kokioje sekos vietoje naujas klasės objektas turėtų būti sukurtas. Po įvykio laikoma, jog naujas objektas yra sukurtas ir jį galima pradėti

naudoti aprašant sąveikas. *DestructionEvent* įvykis sunaikina klasės objektą. Įvykis gali būti iššaukiamas tiek ir išorinės klasės, tiek ir pačios klasės, kuri yra naikinama. Po šio įvykio objektas negali būti naudojamas tolesnėse sąveikose.



Pav. 7. Objekto sukūrimas ir sunaikinimas

Pav. 7 pateikta paprasta sekų diagrama, kurioje yra sukuriamas objektas, nusiunčiamos trys sinchroninės žinutės ir objektas yra sunaikinamas.

5.3 Neapibrėžtos išraiškos

Neapibrėžtos išraiškos (angl. *opaque expressions*) yra tiksliai neapibrėžtas tekstinis fragmentas. Išraiškos aprašomos konkrečia programavimo kalba ar natūralia kalba.

Išraiškos interpretavimas priklauso nuo kalbos, modeliavimo įrankiai nesugebantys atpažinti išraiškos privalo ją palikti nepakeistą. Kokiu metu ir kaip neapibrėžtos išraiškos naudojamos priklauso nuo kiekvieno įrankio. Išraiškos sintaksė gali būti tikrinama tik jeigu įrankis atpažįsta naudojamą kalbą.

Kadangi UML 2.0 specifikacijoje nėra tiksliai nurodyta kas gali būti neapibrėžtos išraiškos kūne (angl. *body*), išraiškos ypač lanksčios, kita vertus, jų validavimas praktiškai negalimas.

5.4 Stereotipai

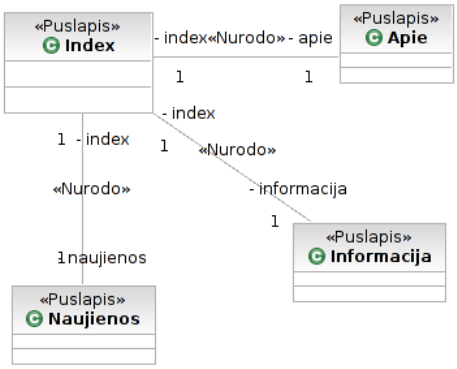
Stereotipai skirti išplėsti UML žodyną, įvedant naujus modelio elementus, skirtus tam tikros srities sistemai aprašyti.

Stereotipas gali turėti savybių atributus (angl. *properties*), klasė kuri yra išplečiama stereotipu šiuos atributus naudoja kaip žymes.

Įprastai stereotipai konkrečiai sričiai aprašomi atskirame profilyje (angl. *profile*), tada profilis prijungiamas prie UML modelio.

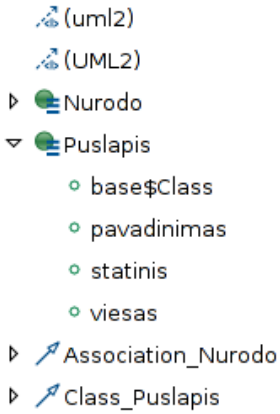
Stereotipo aprašyme informacija pakankama tik modelio kūrimui bei validavimui. Efektyvus kodo generatorius privalo transformuoti stereotipinius elementus į specifinę kodo struktūrą.

Pav. 8 pateikta klasių diagrama naudojanti tinklapio profilį. Stereotipu *Puslapis* aprašoma, jog klasė atstovauja atskirą tinklapio puslapį. Stereotipas *Nurodo* patikslina susiejimą (angl. *association*) – puslapiai privalo sietis nuorodomis.



Pav. 8. Tinklapio klasių diagrama

Stereotipas *Puslapis* (Pav. 9) turi tris papildomus atributus (*pavadinimas, statinis, viesas*), šie atributai tampa klasių (*Index,Apie...*) žymėmis.



Pav. 9. Tinklapio profilis

Kodo generatorius gali įvertinti klasės stereotipą ir žymių reikšmes.

6. OCL

UML klasių ir sekų diagramos nėra pakankamos, norint tiksliai aprašyti visos sistemos darbo aspektus. OCL kalba (angl. *Object Constraint Language*) skirta tiksliems apribojimams formaliai nusakyti, kurie įprastai pateikiami kaip komentarai. OCL yra grynai specifikacijų kalba, interpretuojant OCL išraišką visuomet gaunamas rezultatas, tačiau sistemos būsena interpretavimo metu nekinta.

Kadangi OCL specifikacija būtina išsamiam PIM modeliui aprašyti, galimybė OCL apribojimus naudoti galutinės sistemos darbo korektiškumui užtikrinti ypač naudinga. OCL būtų galima naudoti keliais aspektais:

- ✔ kaip transformuojamą kalbą, apribojimai būtų išreiškiami kodo fragmentais (metodui pradedant ir baigiant darbą ir pan.);
- ✔ korektiško sistemos darbo tikrinimui derinimo (angl. *debug*) režime;
- ✔ papildyti ir naudoti kaip veiksmų kalbą (angl. *actions language*) visapusiškam modelio veikimui aprašyti.

Transformuojant OCL išraiškas į galutinį kodą galima būtų išskirti tokius principus:

- ✔ Invariantus (angl. *Invariants*) (išraiškos kurios privalo būti teisingos visiems sistemos objektams bet kuriuo vykdymo metu) galima tikrinti kuomet keičiama susijusių atributų reikšmė. (Lentelė Nr. 1)
- ✔ Prieš (angl. *pre*-) sąlygas (išraiškos kurios privalo būti teisingos prieš vykdant susijusią operaciją), galima transformuoti į su OCL kontekstu susieto metodo startinių sąlygų patikrinimo kodą. (Lentelė Nr. 2)
- ✔ Po (angl. *post*-) sąlygas (išraiškos kurios privalo būti teisingos atlikus susijusią operaciją/veiksmą), galima transformuoti į su OCL kontekstu susieto metodo tikrinimus veiksmo pabaigoje. Be to, po sąlygos tinkamos sistemos būsenai keisti. (Lentelė Nr. 3)

Lentelė Nr. 1.OCL invarianto transformavimas

OCL išraiška	***Java kodas***

context Saskaita **inv:**
self.paskola < 100000
	public void setPaskola(int value) throws OCLException
	{
	paskola = value;
	//...
	bool oclres = (paskola < 100000);
	if (!oclres) throw new OCLException(“inv: self.paskola < 100000”);
	}

Lentelė Nr. 2.OCL prieš sąlygos transformavimas

OCL išraiška	*Java kodas*

context Saskaita::sumazPaskola(sum: Integer)
pre: sum > 0
	public int minPaskola() throws OCLException
	{
	bool oclres = (sum > 0);
	if (!oclres) throw new OCLException("pre: sum > 0");
	//..
	}

Lentelė Nr. 3.OCL po sąlygos transformavimas

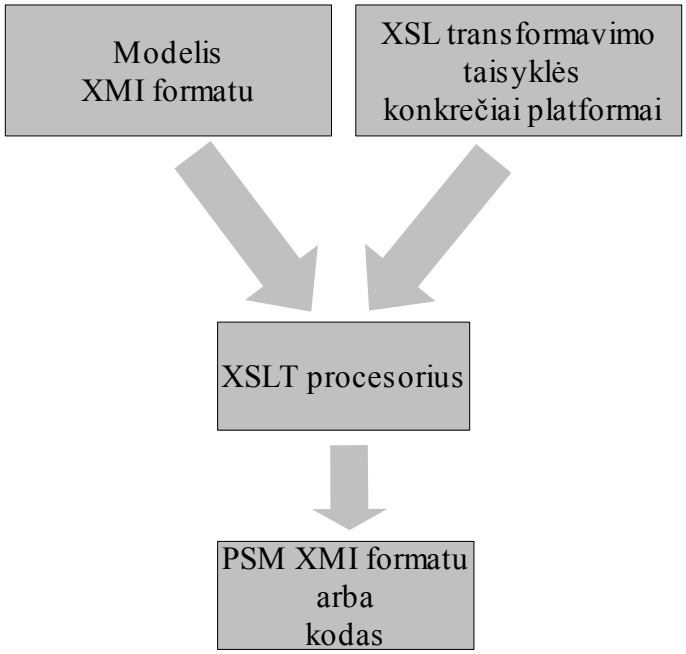
OCL išraiška	*Java kodas*

context Saskaita::minPaskola()
post: result=100
	public int minPaskola()
	{
	//...
	return 100;
	}

7. SEKŲ IR KLASIŲ DIAGRAMŲ TRANSFORMAVIMAS

Prototipiniam sekų ir klasių diagramų transformavimui iš XMI naudojama XSLT transformavimo kalba. XSLT pakankamai aiški, aprašant bendrus transformavimo principus ir pakankamai lanksti prototipiniam transformatoriui.

Atliekant transformaciją generuojamas vienos gijos *Java* kodas. Apžvelgiamos skirtingų loginių kodo struktūrų generavimo galimybės.

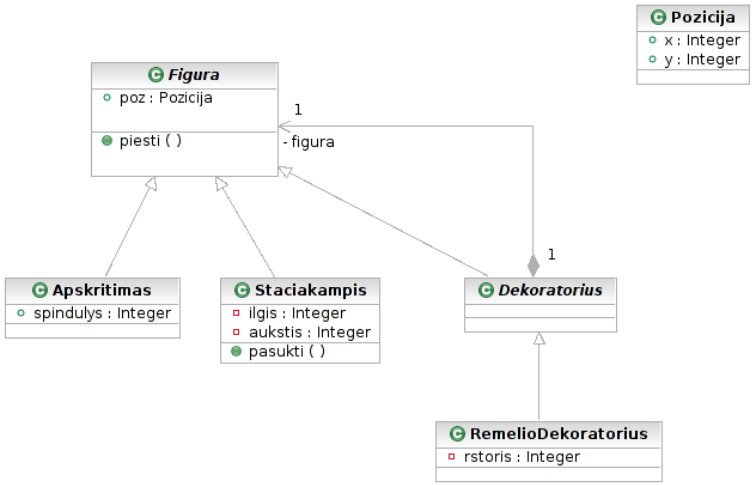


Pav. 10. Bendras XSLT transformavimo principas

7.1 Klasių aprašų generavimas

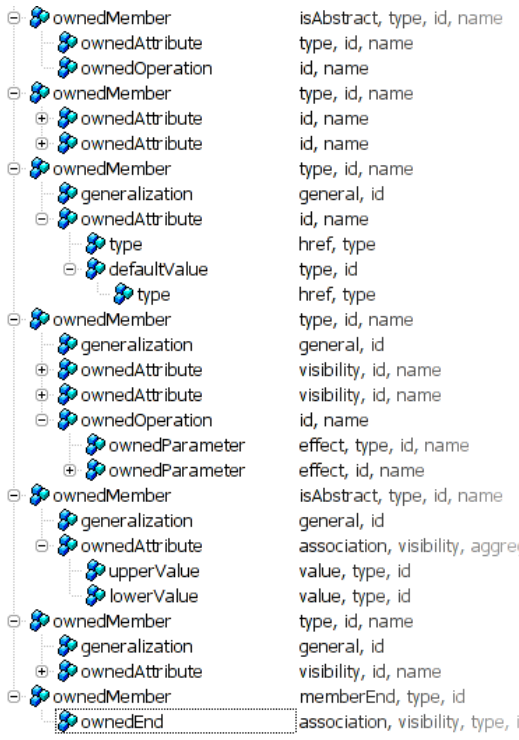
Dažniausiai praktikoje naudojamas statinės struktūros kodo generavimas iš klasių diagramų. OO programavimo kalbose, iš vienos UML klasės tiesiogiai sugeneruojamas vienas, programavimo kalbai specifinės, klasės aprašas.

Klasių diagramos – kodo transformacija nėra izomorfinė, pavyzdžiui, klasių diagramą transformuojant į kodą prarandami skirtumai tarp stipraus ir silpno agregacijos ryšio.



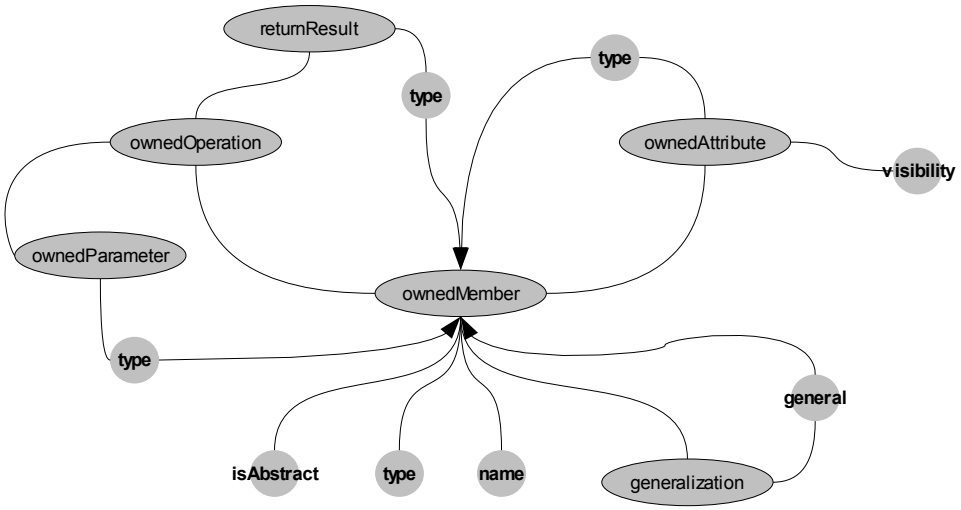
Pav. 11. Klasių diagrama

XSLT transformacijos taisyklės (priedas 11.1, 34 psl.) sudaromos pagal XMI struktūros medį (Pav. 12) ir ryšių tarp elementų grafą (Pav. 13).



Pav. 12: . XMI klasių diagramos struktūros medis

Iš kiekvieno *uml:Class* tipo nario kuriamas vienos *Java* klasės aprašas, tada išrišami ir aprašomi klasės atributai ir metodai.



Pav. 13. Ryšių tarp XMI elementų grafas

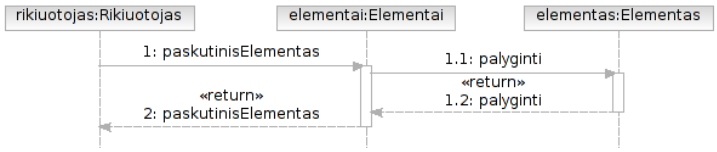
Iš klasių diagramos (Pav. 11) sugeneruotas *Java* kodas.

```
public class Pozicija
{
        public int x;
        public int y;
}

public abstract class Figura
{

        public Pozicija poz;
        public void piesti ()
```

```
        {
        }

}

public class Apskritimas extends Figura
{
        public int spindulys;
}

public class Staciakampis extends Figura
{
        private int aukstis;
        private int ilgis;

        public void pasukti (Pozicija centras, int kampas)
        {
        }
}

public abstract class Dekoratorius extends Figura
{
        private Figura figura;
}

public class RemelioDekoratorius extends Dekoratorius
{
        private int rstoris;
}
```

7.2 Elgsenos kodo generavimas

Sekų diagramos modelio elgseną aprašo objektų (angl. *instance*) lygyje. Bendru atveju, sekų diagrama gali pateikti vieną sistemos veiklos scenarijų, parodant kokie objektai joje dalyvauja ir kaip siejasi vienas su kitu. Tokiu atveju (Pav. 14) informacija kodo generavimui nepakankama, tačiau diagrama gali būti tinkama testinio kodo generavimui.



Pav. 14. Veiklos scenarijaus sekų diagrama

Norint sekų diagramų transformacijas į kodą padaryti efektyvesnes, įvedami papildomi apribojimai:

- ✔ bendravimo (angl. *iteraction*) schema privalo būti susieta su konkrečia operacija (naudojant *specification* atributą);
- ✔ objektų pavadinimai atitinka egzistavimo linijų (angl. *lifeline*) vardus;
- ✔ pranešimų pavadinime nurodomi perduodami parametrų vardai;
- ✔ neapibrėžtose išraiškose naudojamos tik bendros *boolean* rezultatą grąžinančios išraiškos, kurių operandai – objektų/metodų vardai, operatoriai - <>=!&| ir kt.

UML 2.0 pranešimo (angl. *message*) elementas gali turėti bet kokį, įrankio parenkamą, pavadinimą. Kadangi pranešimas su klasės operacija susiejamas *signature* atributu,

pavadinimą galima naudoti perduodamų parametrų vardams nusakyti. Lentelėje Nr. 4, pateikiama parinkta pranešimo pavadinimo sintaksė.

Lentelė Nr. 4.Parametrai pranešimo pavadinime

Operacija	***Pranešimo pavadinimas***	***Metodas***
Neturi parametrų	Bet koks	Metodas kviečiamas be parametrų
Turi parametrų	[*operacijos_pavadinimas*]<(>*parametras1*,[*parametras2*,...]<)>	Patikrinami atitinkamų parametrų tipai ir operacijos aprašas, metodas kviečiamas naudojant nurodytus parametrus

Laikoma, kad pirma egzistavimo linija sekų diagramoje atstovauja objektą, kurio operaciją (metodą) diagrama aprašo. Šios linijos vardas transformacijoje ignoruojamas (aiškumo dėlei gali būti naudojami *this* ar *self* vardai). Kitų egzistavimo linijų vardai naudojami ryšiams tarp objektų ir pranešimų parametrų nustatyti.

Neapibrėžtose išraiškose naudojama tokia sintaksė:

Išraiškos Elementas ::= '(' | ')' | '.' | Skaičius | Objekto vardas'.'Operacijos vardas | Operatorius

Objekto vardas :== Diagramos *Lifeline* vardas | Aktyvaus objekto atributas

Operacijos vardas :== Viena iš operacijų priklausančių objekto klasei

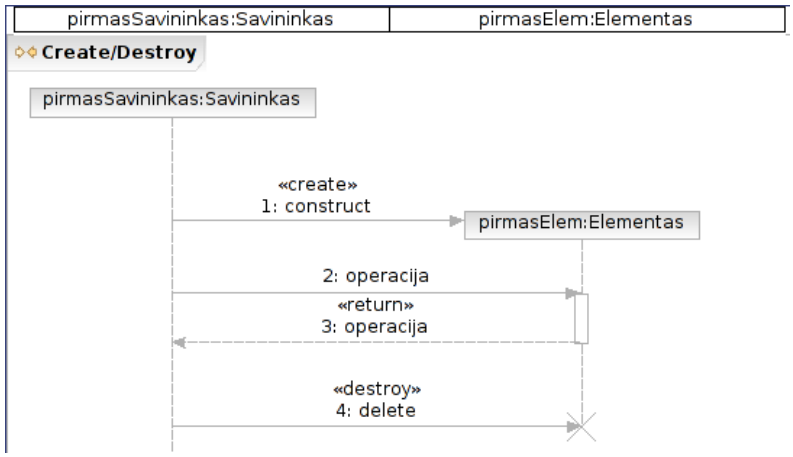
Skaičius :== [0-9]+

Operatorius ::= '==' | '!=' | '<' | '<=' | '>' | '>=' | '!' | '&&' | '||'

7.2.1 Objekto kūrimas ir naikinimas

Generuojant kodą paprasčiausiam objekto sukūrimo, metodo iškvietimo ir naikinimo scenarijui (Pav. 15) naudojamas ryšių tarp elementų grafas (Pav. 17).

Kuriamas objektas keičiamas lokaliu metodo kintamuoju arba naudojamas kaip inicijuojančios klasės atributas (jeigu randamas atributų sąraše).

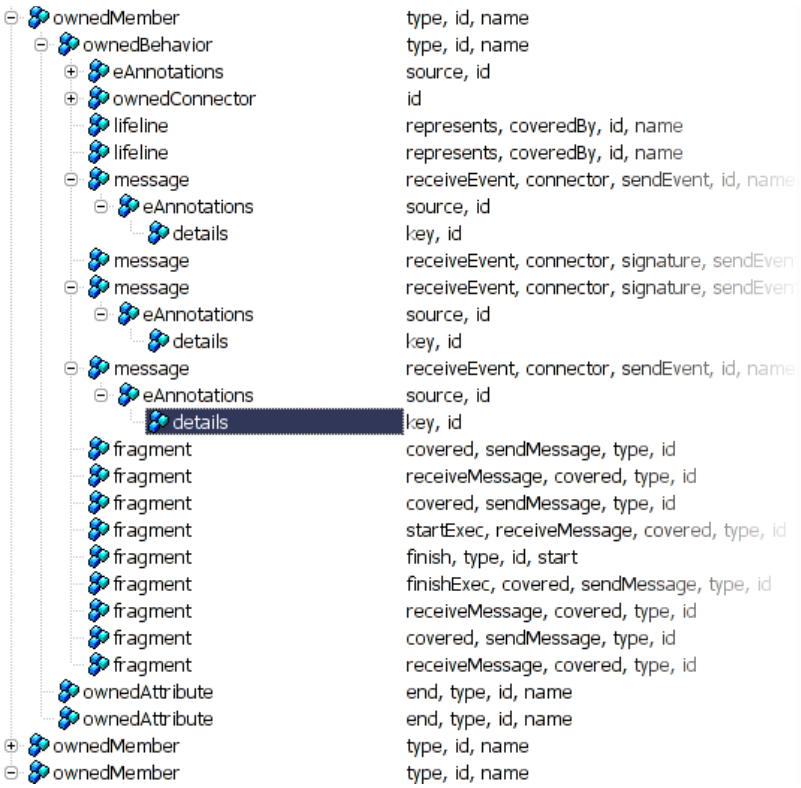


Pav. 15. Objekto kūrimo/naikinimo sekų diagrama

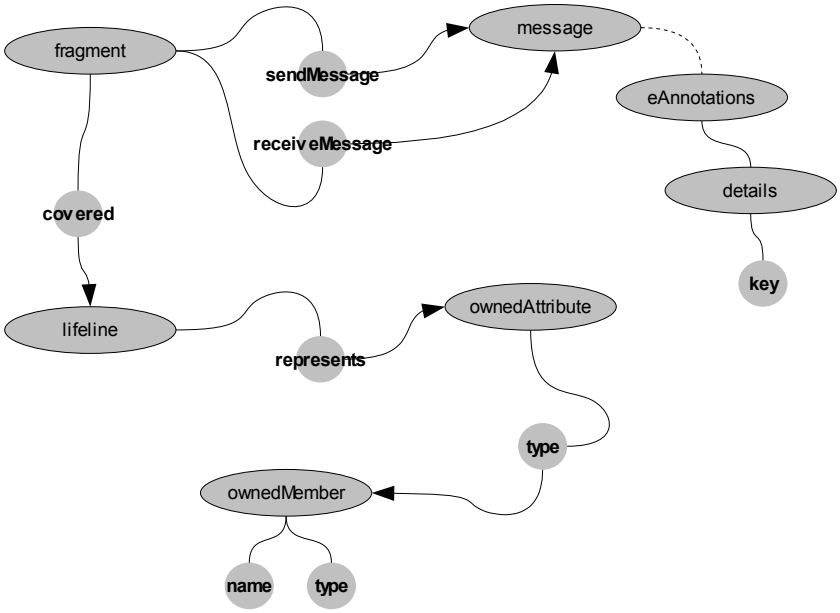
Sekų diagrama XMI formate (Pav. 16) apdorojama iš eilės nagrinėjant *fragment* elementus.

Fragmentas gali būti susietas su išsiunčiamu ar gaunamu pranešimu (*message*). Pranešimo detalėse aprašomi keli tipai:

- ✔ kūrimo pranešimas (*key=create*)
- ✔ naikinimo pranešimas (*key=destroy*)
- ✔ grįžtantis sinchroninis pranešimas (*key=return*)



Pav. 16. XMI sekų diagramos medis



Pav. 17. Ryšių tarp XMI elementų grafas

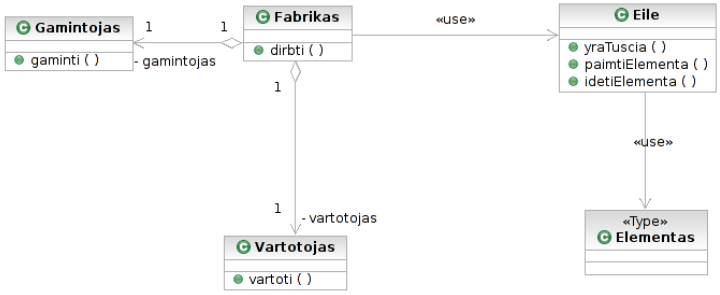
Remiantis sudarytu ryšių grafu paruošiamos XSLT transformacijos taisyklės (priedas 11.2, 36 psl.). Atlikus transformaciją gaunamas metodo kodo fragmentas *Java* kalba.

```
Elementas pirmasElem = new Elementas();
pirmasElem.operacija();
//delete pirmasElem
```

7.2.2 Ciklų struktūrų generavimas

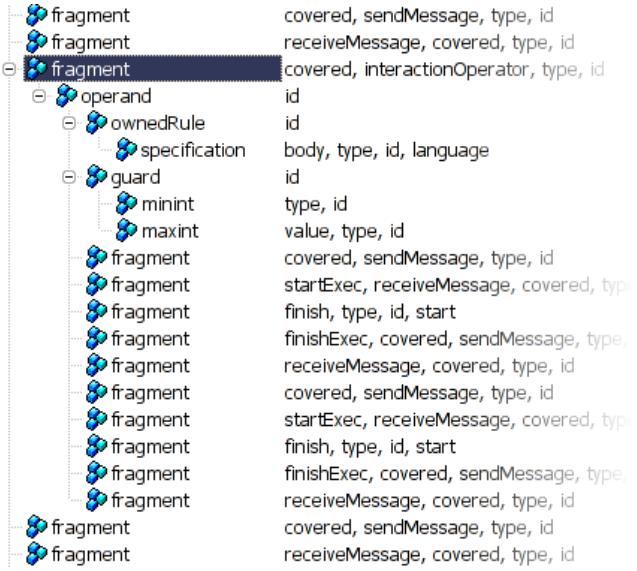
Ciklų išraiškoms aprašyti sekų diagramose naudojami *„loop“* tipo fragmentai (11.3, 37 psl.). Kadangi fragmentas tiesiog nurodo sąveikas ir kartojimo sąlygas (saugos ir susietos apribojimų (angl. *constraint*) išraiškos), kodo generatorius gali parinkti tinkamiausią ciklo operatorių (*for,while,repeat...*).

Kiekvienas apjungiantis fragmentas (angl. *combinedFragment*) gali turėti vieną ar daugiau jam priklausančių fragmentų (Pav. 19), tokiu būdu gali būti sudaromi reikiamo sudėtingumo fragmentų medžiai.

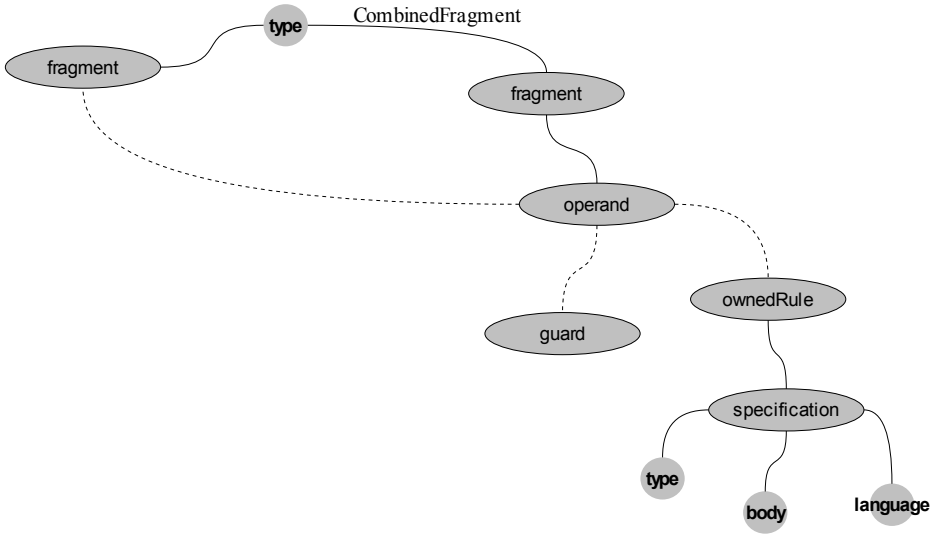


Pav. 18. Gamintojo-vartotojo klasių diagrama

Kodo generatorius kuriamas remiantis sudarytu rekursyviu, ryšių tarp fragmentų, grafu (Pav. 20).



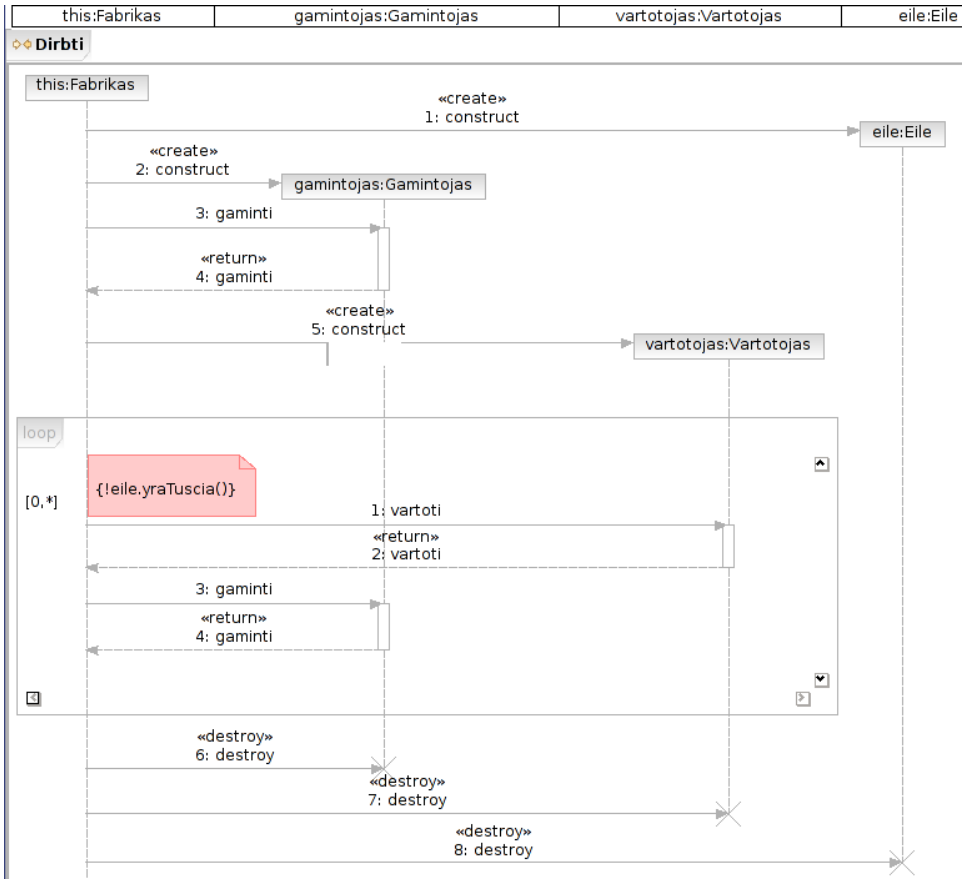
Pav. 19.Ciklo fragmentas XMI medyje



Pav. 20. Fragmentų ryšių grafas

Sukurta XSLT transformacija (11.3, 37 psl.) naudojama kodo generavimui iš Pav. 21 pavaizduotos sekų diagramos.

Statinės struktūros generavimui naudojama modelio Pav. 18 pavaizduota klasių diagrama.



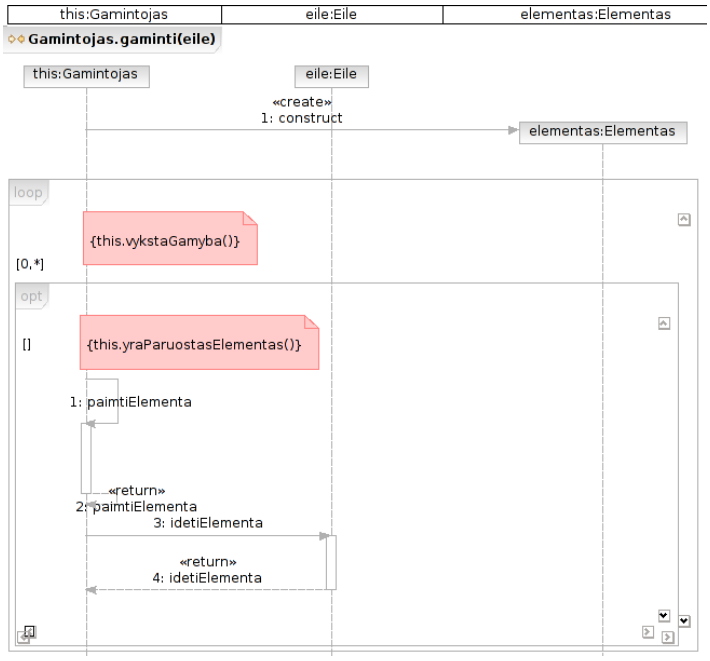
Pav. 21. Sekų diagrama turinti ciklo fragmentą

Atlikus transformaciją gauta *Fabrikas* klasė *Java* kalba:

```
// Failas Fabrikas.java
public class Fabrikas
{

        // Attributes
        private Gamintojas gamintojas;
        private Vartotojas vartotojas;

        // Methods
        public void dirbti ()
        {
                Eile eile = new Eile();
                Gamintojas gamintojas = new Gamintojas();
                gamintojas.gaminti(eile);
                Vartotojas vartotojas = new Vartotojas();
                while (!eile.yraTuscia()) {
                        vartotojas.vartoti(eile);
                        gamintojas.gaminti(eile);
                }
                gamintojas = null;
                vartotojas = null;
                eile = null;
        }
}
```

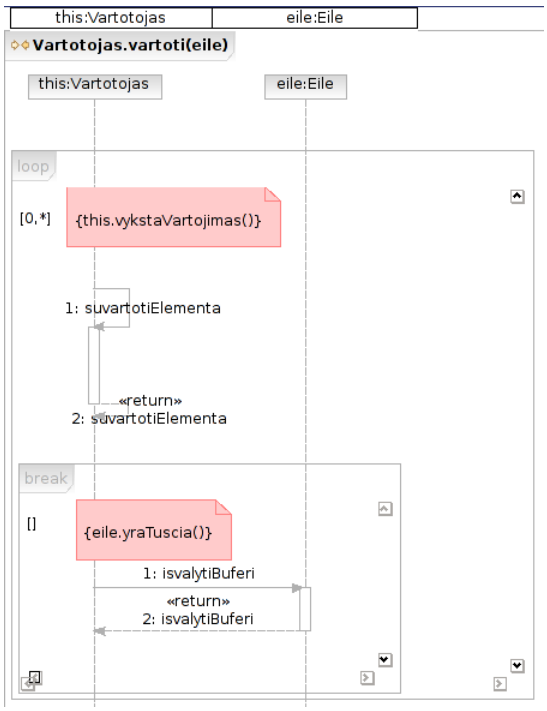
7.3 Pilno programos skeleto generavimas

Papildytoje XSLT transformacijoje transformuojami *opt/break/loop* apjungiantys fragmentai, sugeneruojama statinė programos struktūra. Galutinė prototipinio kodo generavimo XSLT transformacija pateikiama prieduose (11.5, 39psl.).



Pav. 22. Gaminti operacijos sekų diagrama

Testuojant kodo generavimą, Pav. 18 pavaizduotas modelis, papildomas sekų diagramomis,



Pav. 23. Vartotoji operacijos sekų diagrama

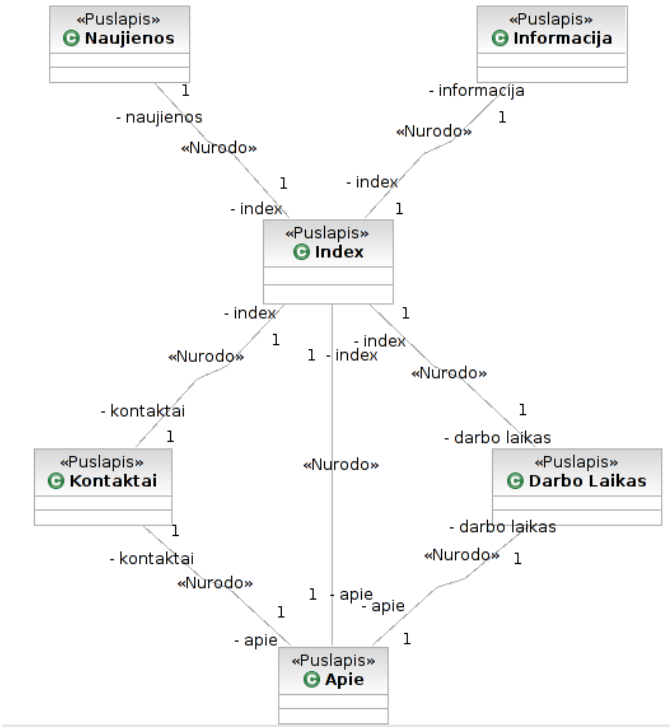
detalizuojamos *gaminti* (Pav. 22) ir *vartoti* (Pav. 23) operacijos .

Visas sugeneruotos programos skeletas pateikiamas prieduose (11.6, 43 psl.).

7.4 Srities kodo generavimas

Kodas sričiai generuojamas naudojant specifinius profilius.

Nagrinėjamas pavyzdys – klasių diagrama apibūdinanti svetainės struktūrą: puslapius ir ryšius tarp jų (Pav. 24).



Pav. 24. Svetainės struktūros klasių diagrama

Sudarytos XSLT transformavimo taisyklės (11.4, 38 psl.) iš klasių diagramos sugeneruoja HMTL puslapių struktūrą (11.7, 45 psl.): iš kiekvienos klasės sugeneruojamas puslapio failas, ryšiai tarp klasių išreiškiami nuorodomis (angl. *href*).

Transformacijai naudojami Pav. 9 (18 psl.) parodyto profilio stereotipai.

8. IŠVADOS

- ✔ Nagrinėjant programos kodo generavimo iš UML klasių ir sekų diagramų galimybes, buvo sudaryti eksperimentiniai kodo generatoriai XSLT kalba. Sėkmingai atliktos transformacijos, generuojančios programos struktūros kodą iš UML klasių diagramų, bei elgsenos kodą iš sekų diagramų.

- ✔ Nustatyta, jog sekų ir klasių diagramų visiškai pakanka programos skeletui sugeneruoti (struktūra ir pagrindinių metodų realizacijos). Kartu pastebėta, jog kuriant modelius, skirtus transformacijai į kodą, sunku išvengti realizacijai specifinių detalių modelyje. Be to, kuriant modelį, iš kurio bus generuojamas kodas konkrečiai programavimo kalbai, dažnai išryškėja UML dialektas. Vengiama naudoti struktūras kurios sunkiai išreiškiamos realizacijos kalboje (pvz. daugialypis paveldėjimas).

- ✔ Galutinio programos kodo generavimas iš UML modelių vis dar sudėtinga problema. Vėlesni tyrimai galėtų nagrinėti kodo generavimo iš būsenų ir veiklos diagramų galimybes.

9. LITERATŪRA

1. OMG. XML Metadata Interchange (XMI) Specification [žiūrėta 2005-05-02]
   Prieiga per internetą: http://www.omg.org/docs/formal/03-05-02.pdf
2. OMG. MDA Guide Version 1.0.1 [žiūrėta 2005-05-03]
   Prieiga per internetą: http://www.omg.org/docs/omg/03-06-01.pdf
3. OMG. UML 2.0 Infrastructure Specification [žiūrėta 2005-05-03]
   Prieiga per internetą: http://www.omg.org/docs/ptc/03-09-15.pdf
4. Abouzahra Anas. Bridging UML profiles and Domain Specific Languages [žiūrėta 2006-01-19]
   Prieiga per internetą:
   http://www.eclipse.org/gmt/amw/examples/UMLProfiles/ExampleUMLProfiles_DSLs.pdf
5. Sten Loecher, Stefan Ocke. A Metamodel-Based OCL-Compiler for UML and MOF [žiūrėta 2006-03-03]
   Prieiga per internetą:
   http://i12www.ira.uka.de/~baar/oclworkshopUml03/papers/03_meta_model_based_ocl_compiler.pdf
6. Andreas Thuy, Manfred Glesner. On the Integration of UML Sequence Diagrams and Actor-Oriented Simulation Models [žiūrėta 2006-02-19]
   Prieiga per internetą: http://jerry.c-lab.de/uml-soc/uml-soc05/07indrusiak.pdf
7. Victoria Cengarle1, Alexander Knapp. Operational Semantics of UML 2.0 Interactions [žiūrėta 2006-03-15]
   Prieiga per internetą: http://www4.in.tum.de/publ/papers/CeKn05.pdf
8. Torger Kielland. Consistency checking of UML models on the XMI format [žiūrėta 2006-02-12]
   Prieiga per internetą: http://www.idi.ntnu.no/grupper/su/su-diploma-2005/kielland-diplom2005.pdf
9. Tiziana Allegrini. Code generation starting from statecharts specified in UML [žiūrėta 2006-01-12]
   Prieiga per internetą: http://argouml.tigris.org/docs/allegrini_dissertation/tesi_en.pdf
10. Ashley McNeile. MDA: The Vision with the Hole? [žiūrėta 2005-11-18]
    Prieiga per internetą: http://www.metamaxim.com/download/documents/MDAv1.pdf
11. Georg Beier, Markus Kern. Aspects in UML Models from a Code Generation Perspective [žiūrėta 2006-02-16]
    Prieiga per internetą: http://lgl.epfl.ch/workshops/uml2002/papers/beier.pdf
12. Dave Thomas. MDA: Revenge of the Modelers or UML Utopia? [žiūrėta 2005-12-09]
    Prieiga per internetą: http://www.martinfowler.com/ieeeSoftware/mda-thomas.pdf
13. Matilda: A Distributed UML Virtual Machine for Model-Driven Software Development [žiūrėta 2006-01-20]
    Prieiga per internetą: http://www.cs.umb.edu/~jxs/pub/sci05-umlvm.pdf

10.TERMINAI, SANTRUMPOS IR ŽYMĖJIMAI

Lentelė Nr. 5.Naudojamos santrumpos

Santrumpa	***Pilnas pavadinimas***	***Paaiškinimas/vertimas***
OCL	Object Constraint Language	OMG objektų apribojimų specifikavimo kalba
MDA	Model Driven Architecture	Modeliais pagrįsta architektūra
XSLT	Extensible Stylesheet Language Transformations	XSLT transformacijų kalba
XMI	XML Metadata Interchange	OMG standartas meta duomenų informacijai pateikti XML formatu

Lentelė Nr.6. Naudojami žymėjimai

Žymė	***Reikšmė***
	XMI elementas (angl. *node*)
	XMI elemento atributas
	XMI elementas turi [0,*] susietų šiuo ryšiu elementų
	XMI elementas turi [1,*] susietų šiuo ryšiu elementų
	Atributo reikšmė nurodo XMI elemento identifikatorių

11.PRIEDAI

11.1 Klasių struktūros generavimo XSLT kodas

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:notation="http://www.ibm.com/xtools/1.5.0/Notation"
  xmlns:uml="http://www.eclipse.org/uml2/1.0.0/UML"
  xmlns:umlnotation="http://www.ibm.com/xtools/1.5.0/Umlnotation"
  version="1.0">
<xsl:output method="text" indent="yes"/>

<xsl:key name="ownedMembers" match="ownedMember" use="@xmi:id"/>

<!-- -->
<xsl:template match="/">
 <xsl:apply-templates />
</xsl:template>

<!-- Klases -->
<xsl:template match="ownedMember">
 <xsl:if test="@xmi:type='uml:Class'">
// Class
<xsl:text>public </xsl:text>
 <xsl:choose>
  <xsl:when test="@isAbstract='true'">abstract </xsl:when>
 </xsl:choose>
 <xsl:text>class </xsl:text>
 <xsl:value-of select="@name" />
 <xsl:if test="generalization/@general">
  <xsl:text> extends </xsl:text><xsl:value-of select="key('ownedMembers', generalization/@general)/@name"
/>
 </xsl:if>
{

 // Attributes
 <xsl:apply-templates select="ownedAttribute" />

 // Methods
 <xsl:apply-templates select="ownedOperation" />
}
 </xsl:if>
</xsl:template>

<!-- Atributai -->
<xsl:template match="ownedAttribute">
 <xsl:text>&#10;
 </xsl:text>
 <xsl:choose>
  <xsl:when test="@visibility='private'">private </xsl:when>
  <xsl:when test="@visibility='protected'">protected </xsl:when>
  <xsl:otherwise>public </xsl:otherwise>
 </xsl:choose>
 <xsl:choose>
  <xsl:when test="key('ownedMembers', @type)/@name">
   <xsl:value-of select="key('ownedMembers', @type)/@name" />
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="primitivestempl">
    <xsl:with-param name="pnode" select="." />
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
 <xsl:text> </xsl:text>
 <xsl:value-of select="@name" /><xsl:text>;</xsl:text>
</xsl:template>


<!-- Metodai -->
<xsl:template match="ownedOperation">
 <xsl:text>&#10;
 </xsl:text>
 <xsl:text>public </xsl:text>
 <xsl:choose>
  <xsl:when test="returnResult/@type">
   <xsl:value-of select="key('ownedMembers', @type)/@name" />
```

```
  </xsl:when>
  <xsl:when test="returnResult/type">
   <xsl:choose>
    <xsl:when test="key('ownedMembers', returnResult/type/@type)/@name">
     <xsl:value-of select="key('ownedMembers', returnResult/type/@type)/@name" />
    </xsl:when>
    <xsl:otherwise>
     <xsl:call-template name="primitivestempl">
      <xsl:with-param name="pnode" select="returnResult" />
     </xsl:call-template>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
   <xsl:text>void</xsl:text>
  </xsl:otherwise>
 </xsl:choose>
 <xsl:text> </xsl:text>
 <xsl:value-of select="@name" /><xsl:text> (</xsl:text>
 <!-- Parametrai -->
 <xsl:for-each select="ownedParameter">
  <xsl:choose>
   <xsl:when test="key('ownedMembers', @type)/@name">
    <xsl:value-of select="key('ownedMembers', @type)/@name" /><xsl:text> </xsl:text>
   </xsl:when>
   <xsl:otherwise>
    <xsl:call-template name="primitivestempl">
     <xsl:with-param name="pnode" select="." />
    </xsl:call-template>
    <xsl:text> </xsl:text>
   </xsl:otherwise>

  </xsl:choose>
  <xsl:value-of select="@name" />
  <xsl:if test="position() != last()"><xsl:text>, </xsl:text></xsl:if>
 </xsl:for-each>
 <xsl:text>)</xsl:text>
 {
 }
</xsl:template>

<!-- UML2 Primityvai -->
<xsl:template name="primitivestempl">
 <xsl:param name="pnode" />
 <xsl:variable name="primfile">
  <xsl:call-template name="basename">
   <xsl:with-param name="filename"><xsl:value-of select="$pnode/type/@href" /></xsl:with-param>
  </xsl:call-template>
 </xsl:variable>
 <xsl:variable name="primid" select="substring-after($pnode/type/@href, '#')" />

 <xsl:for-each select="document($primfile)//ownedMember">
  <xsl:if test="@xmi:id=$primid">
   <xsl:call-template name="primitivemap">
    <xsl:with-param name="primname" select="@name" />
   </xsl:call-template>
  </xsl:if>
 </xsl:for-each>
</xsl:template>

<!-- UML2 => Java primityvu keitimas -->
<xsl:template name="primitivemap">
 <xsl:param name="primname" />
 <xsl:choose>
  <xsl:when test="$primname='Integer'">
   <xsl:text>int</xsl:text>
  </xsl:when>
  <xsl:when test="$primname='Boolean'">
   <xsl:text>boolean</xsl:text>
  </xsl:when>
  <xsl:when test="$primname='UnlimitedNatural'">
   <xsl:text>int</xsl:text>
  </xsl:when>
  <xsl:otherwise>
   <xsl:value-of select="$primname" />
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<!-- Pagalbiniai sablonai -->

<!-- failo vardo ispakavimas is URI -->
<xsl:template name="basename">
 <xsl:param name="filename" />
 <xsl:choose>
```

```
  <xsl:when test="contains($filename, '/')">
   <xsl:call-template name="basename">
    <xsl:with-param name="filename"><xsl:value-of select="substring-after($filename, '/')" /></xsl:with-
param>
   </xsl:call-template>
  </xsl:when>
  <xsl:when test="contains($filename, '#')">
   <xsl:call-template name="basename">
    <xsl:with-param name="filename"><xsl:value-of select="substring-before($filename, '#')" /></xsl:with-
param>
   </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
   <xsl:value-of select="$filename" />
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>


</xsl:stylesheet>
```

11.2 Objekto kūrimo/naikinimo scenarijaus XSLT kodas

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:notation="http://www.ibm.com/xtools/1.5.0/Notation"
  xmlns:uml="http://www.eclipse.org/uml2/1.0.0/UML"
  xmlns:umlnotation="http://www.ibm.com/xtools/1.5.0/Umlnotation"
  version="1.0">
<xsl:output method="text" indent="yes"/>

<xsl:key name="ownedMembers" match="ownedMember" use="@xmi:id"/>
<xsl:key name="ownedAttributes" match="ownedAttribute" use="@xmi:id"/>
<xsl:key name="lifelines" match="lifeline" use="@xmi:id"/>
<xsl:key name="messages" match="message" use="@xmi:id"/>
<xsl:key name="fragments" match="fragment" use="@xmi:id"/>


<!-- Fragmentai -->
<xsl:template match="ownedMember">
 <xsl:if test="@xmi:type='uml:Collaboration'">
  <xsl:apply-templates select="ownedBehavior" />
 </xsl:if>
</xsl:template>

<!-- Elgsenos -->
<xsl:template match="ownedBehavior">
 <xsl:if test="@xmi:type='uml:Interaction'">
  <xsl:for-each select="fragment">
   <xsl:variable name="c_ownlife" select="key('lifelines', @covered)" />
   <xsl:variable name="c_ownattr" select="key('ownedAttributes', $c_ownlife/@represents)" />
   <xsl:variable name="c_ownmem" select="key('ownedMembers', $c_ownattr/@type)" />

   <xsl:if test="@sendMessage">
    <xsl:variable name="c_message" select="key('messages', @sendMessage)" />
    <xsl:variable name="c_endfrag" select="key('fragments', $c_message/@receiveEvent)" />

    <xsl:variable name="c_endlife" select="key('lifelines', $c_endfrag/@covered)" />
    <xsl:variable name="c_endattr" select="key('ownedAttributes', $c_endlife/@represents)" />
    <xsl:variable name="c_endmem" select="key('ownedMembers', $c_endattr/@type)" />
    <xsl:choose>
     <xsl:when test="$c_message/eAnnotations/details/@key='create'">
      <xsl:value-of select="$c_endmem/@name" />
      <xsl:text> </xsl:text>
      <xsl:value-of select="$c_endattr/@name" />
      <xsl:text> = new </xsl:text>
      <xsl:value-of select="$c_endmem/@name" />
      <xsl:text>();</xsl:text>
      <xsl:text>&#10;</xsl:text>
     </xsl:when>
     <xsl:when test="$c_message/eAnnotations/details/@key='destroy'">
      <xsl:text>//delete </xsl:text>
      <xsl:value-of select="$c_endattr/@name" />
      <xsl:text>&#10;</xsl:text>
     </xsl:when>
     <xsl:when test="$c_message/eAnnotations/details/@key='return'">
     </xsl:when>
     <xsl:otherwise>
      <xsl:value-of select="$c_endattr/@name" /><xsl:text>.</xsl:text>
      <xsl:value-of select="$c_message/@name" /><xsl:text>();</xsl:text>
      <xsl:text>&#10;</xsl:text>
     </xsl:otherwise>
```

```
    </xsl:choose>
   </xsl:if>
  </xsl:for-each>
 </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

11.3 Ciklų generavimo XSLT kodas

```
...
<!-- Metodai -->
<xsl:template match="ownedOperation">
 <xsl:param name="owner" />
 <xsl:text>&#10;
 </xsl:text>
 <xsl:text>public </xsl:text>
 <xsl:choose>
  <xsl:when test="returnResult/@type">
   <xsl:value-of select="key('ownedMembers', @type)/@name" />
  </xsl:when>
  <xsl:when test="returnResult/type">
   <xsl:choose>
    <xsl:when test="key('ownedMembers', returnResult/type/@type)/@name">
     <xsl:value-of select="key('ownedMembers', returnResult/type/@type)/@name" />
    </xsl:when>
    <xsl:otherwise>
     <xsl:call-template name="primitivestempl">
      <xsl:with-param name="pnode" select="returnResult" />
     </xsl:call-template>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
   <xsl:text>void</xsl:text>
  </xsl:otherwise>
 </xsl:choose>
 <xsl:text> </xsl:text>
 <xsl:value-of select="@name" /><xsl:text> (</xsl:text>
 <!-- Parametrai -->
 <xsl:for-each select="ownedParameter">
  <xsl:choose>
   <xsl:when test="key('ownedMembers', @type)/@name">
    <xsl:value-of select="key('ownedMembers', @type)/@name" /><xsl:text> </xsl:text>
   </xsl:when>
   <xsl:otherwise>
    <xsl:call-template name="primitivestempl">
     <xsl:with-param name="pnode" select="." />
    </xsl:call-template>
    <xsl:text> </xsl:text>
   </xsl:otherwise>

  </xsl:choose>
  <xsl:value-of select="@name" />
  <xsl:if test="position() != last()"><xsl:text>, </xsl:text></xsl:if>
 </xsl:for-each>
 <xsl:text>)</xsl:text>
 {
   <xsl:if test="key('ownedBehaviors', @xmi:id)">
    <xsl:call-template name="behaves">
     <xsl:with-param name="behavior" select="key('ownedBehaviors', @xmi:id)" />
    </xsl:call-template>
   </xsl:if>
 }
</xsl:template>


<!-- Metodu elgsenos -->
<xsl:template name="behaves">
 <xsl:param name="behavior" />
  <xsl:call-template name="fragments">
   <xsl:with-param name="fragment" select="$behavior/fragment" />
  </xsl:call-template>
</xsl:template>

<!-- Fragmentu interpretavimas -->
<xsl:template name="fragments">
<xsl:param name="fragment" />
<!--<xsl:value-of select="count($fragment)" />
<xsl:text>&#10;</xsl:text>-->
<xsl:for-each select="$fragment">
```

```
<!--<xsl:value-of select="position()" />
<xsl:text>&#10;</xsl:text>-->
<xsl:choose>
<xsl:when test="@xmi:type='uml:EventOccurrence'">
 <xsl:variable name="c_ownlife" select="key('lifelines', @covered)" />
 <xsl:variable name="c_ownattr" select="key('ownedAttributes', $c_ownlife/@represents)" />
 <xsl:variable name="c_ownmem" select="key('ownedMembers', $c_ownattr/@type)" />

 <xsl:if test="@sendMessage">
  <xsl:variable name="c_message" select="key('messages', @sendMessage)" />
  <xsl:variable name="c_endfrag" select="key('fragments', $c_message/@receiveEvent)" />

  <xsl:variable name="c_endlife" select="key('lifelines', $c_endfrag/@covered)" />
  <xsl:variable name="c_endattr" select="key('ownedAttributes', $c_endlife/@represents)" />
  <xsl:variable name="c_endmem" select="key('ownedMembers', $c_endattr/@type)" />

  <xsl:choose>
   <xsl:when test="$c_message/eAnnotations/details/@key='create'">
    <xsl:value-of select="$c_endmem/@name" />
    <xsl:text> </xsl:text>
    <xsl:value-of select="$c_endattr/@name" />
    <xsl:text> = new </xsl:text>
    <xsl:value-of select="$c_endmem/@name" />
    <xsl:text>();</xsl:text>
    <xsl:text>&#10;</xsl:text>
   </xsl:when>
   <xsl:when test="$c_message/eAnnotations/details/@key='destroy'">
    <xsl:value-of select="$c_endattr/@name" /><xsl:text> = null;</xsl:text>
    <xsl:text>&#10;</xsl:text>
    </xsl:when>
    <xsl:when test="$c_message/eAnnotations/details/@key='return'">
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="$c_endattr/@name" /><xsl:text>.</xsl:text>
     <xsl:value-of select="$c_message/@name" /><xsl:text>;</xsl:text>
     <xsl:text>&#10;</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
 </xsl:if>
</xsl:when>
<xsl:when test="@xmi:type='uml:CombinedFragment'">
 <xsl:choose>
 <xsl:when test="@interactionOperator='loop'">
  <xsl:text>while (</xsl:text>
   <xsl:call-template name="rules">
    <xsl:with-param name="operand" select="operand" />
   </xsl:call-template>
  <xsl:text> </xsl:text>
  <xsl:text>) {</xsl:text>
  <xsl:text>&#10;</xsl:text>
   <xsl:call-template name="fragments">
    <xsl:with-param name="fragment" select="operand/fragment" />
   </xsl:call-template>
  <xsl:text>&#10;}&#10;</xsl:text>
 </xsl:when>
 </xsl:choose>
</xsl:when>
</xsl:choose>
</xsl:for-each>
</xsl:template>


<!-- Taisyklės -->
<xsl:template name="rules">
 <xsl:param name="operand" />
 <xsl:for-each select="$operand/ownedRule">
  <xsl:choose>
   <xsl:when test="specification/@language='Logic'">
    <xsl:value-of select="specification/@body" />
   </xsl:when>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>
..
```

11.4 Svetainės struktūros generavimo XSLT kodas

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:notation="http://www.ibm.com/xtools/1.5.0/Notation"
  xmlns:uml="http://www.eclipse.org/uml2/1.0.0/UML"
  xmlns:umlnotation="http://www.ibm.com/xtools/1.5.0/Umlnotation"
```

```
  version="1.0">

<xsl:output method="xml" indent="yes" omit-xml-declaration="yes" />
<xsl:key name="ownedMembers" match="ownedMember" use="@xmi:id"/>

<!-- -->
<xsl:template match="/">
 <xsl:apply-templates />
</xsl:template>

<xsl:template match="ownedMember">
<xsl:if test="@xmi:type='uml:Class'">
<html>
<xsl:comment><xsl:text>Failas </xsl:text><xsl:value-of select="@name" /></xsl:comment>
<head>
<title>
<xsl:call-template name="pageName">
 <xsl:with-param name="member" select="." />
</xsl:call-template>
</title>

<div id="menu">
 <xsl:for-each select="ownedAttribute">
  <xsl:if test="key('ownedMembers', @type)/eAnnotations/contents/@xmi:type">
   <div class="plink">
    <a href="{key('ownedMembers', @type)/@name}">
     <xsl:call-template name="pageName">
      <xsl:with-param name="member" select="key('ownedMembers', @type)" />
     </xsl:call-template>
    </a>
   </div>
  </xsl:if>
 </xsl:for-each>
</div>
</head>
</html>
</xsl:if>
</xsl:template>


<!-- Puslapio vardo sablonas -->
<xsl:template name="pageName">
<xsl:param name="member" />
 <xsl:choose>
  <xsl:when test="$member/eAnnotations/contents/@pavadinimas">
   <xsl:value-of select="$member/eAnnotations/contents/@pavadinimas" />
  </xsl:when>
  <xsl:otherwise>
   <xsl:text>Bevardis</xsl:text>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

11.5 Programos skeleto generavimo prototipinis XSLT kodas

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:notation="http://www.ibm.com/xtools/1.5.0/Notation"
  xmlns:uml="http://www.eclipse.org/uml2/1.0.0/UML"
  xmlns:umlnotation="http://www.ibm.com/xtools/1.5.0/Umlnotation"
  version="1.0">
<xsl:output method="text" indent="yes"/>

<xsl:key name="ownedMembers" match="ownedMember" use="@xmi:id" />
<xsl:key name="ownedBehaviors" match="//ownedBehavior" use="@specification" />
<xsl:key name="ownedAttributes" match="ownedAttribute" use="@xmi:id"/>
<xsl:key name="lifelines" match="lifeline" use="@xmi:id"/>
<xsl:key name="messages" match="message" use="@xmi:id"/>
<xsl:key name="fragments" match="fragment" use="@xmi:id"/>


<!-- -->
<xsl:template match="/">
 <xsl:variable name="root" select="." />
 <xsl:apply-templates />
</xsl:template>

<!-- Klases -->
<xsl:template match="ownedMember">
 <xsl:if test="@xmi:type='uml:Class'">
```

```
// Failas <xsl:value-of select="@name" />.java
<xsl:if test="@clientDependency">
 <xsl:variable name="depMember" select="key('ownedMembers', @clientDependency)" />
 <xsl:text>// naudojamas </xsl:text><xsl:value-of select="key('ownedMembers', $depMember/@supplier)/@name"
/><xsl:text>.java</xsl:text>
//
</xsl:if>
<xsl:text>public </xsl:text>
 <xsl:choose>
  <xsl:when test="@isAbstract='true'">abstract </xsl:when>
 </xsl:choose>
 <xsl:text>class </xsl:text>
 <xsl:value-of select="@name" />
 <xsl:if test="generalization/@general">
  <xsl:text> extends </xsl:text><xsl:value-of select="key('ownedMembers', generalization/@general)/@name"
/>
 </xsl:if>
{

 // Attributes
 <xsl:apply-templates select="ownedAttribute" />

 // Methods
 <xsl:apply-templates select="ownedOperation">
  <xsl:with-param name="owner" select="." />
 </xsl:apply-templates>
}
 </xsl:if>
</xsl:template>

<!-- Atributai -->
<xsl:template match="ownedAttribute">
 <xsl:text>&#10;
 </xsl:text>
 <xsl:choose>
  <xsl:when test="@visibility='private'">private </xsl:when>
  <xsl:when test="@visibility='protected'">protected </xsl:when>
  <xsl:otherwise>public </xsl:otherwise>
 </xsl:choose>
 <xsl:choose>
  <xsl:when test="key('ownedMembers', @type)/@name">
   <xsl:value-of select="key('ownedMembers', @type)/@name" />
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="primitivestempl">
    <xsl:with-param name="pnode" select="." />
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
 <xsl:text> </xsl:text>
 <xsl:value-of select="@name" /><xsl:text>;</xsl:text>
</xsl:template>


<!-- Metodai -->
<xsl:template match="ownedOperation">
 <xsl:param name="owner" />
 <xsl:text>&#10;
 </xsl:text>
 <xsl:text>public </xsl:text>
 <xsl:variable name="returnType">
 <xsl:choose>
  <xsl:when test="returnResult/@type">
   <xsl:value-of select="key('ownedMembers', @type)/@name" />
  </xsl:when>
  <xsl:when test="returnResult/type">
   <xsl:choose>
    <xsl:when test="key('ownedMembers', returnResult/type/@type)/@name">
     <xsl:value-of select="key('ownedMembers', returnResult/type/@type)/@name" />
    </xsl:when>
    <xsl:otherwise>
     <xsl:call-template name="primitivestempl">
      <xsl:with-param name="pnode" select="returnResult" />
     </xsl:call-template>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
   <xsl:text>void</xsl:text>
  </xsl:otherwise>
 </xsl:choose>
 </xsl:variable>

 <xsl:value-of select="$returnType" />
 <xsl:text> </xsl:text>
```

```
 <xsl:value-of select="@name" /><xsl:text> (</xsl:text>
 <!-- Parametrai -->
 <xsl:for-each select="ownedParameter">
  <xsl:choose>
   <xsl:when test="key('ownedMembers', @type)/@name">
    <xsl:value-of select="key('ownedMembers', @type)/@name" /><xsl:text> </xsl:text>
   </xsl:when>
   <xsl:otherwise>
    <xsl:call-template name="primitivestempl">
     <xsl:with-param name="pnode" select="." />
    </xsl:call-template>
    <xsl:text> </xsl:text>
   </xsl:otherwise>

  </xsl:choose>
  <xsl:value-of select="@name" />
  <xsl:if test="position() != last()"><xsl:text>, </xsl:text></xsl:if>
 </xsl:for-each>
 <xsl:text>)</xsl:text>
 {
  <xsl:if test="key('ownedBehaviors', @xmi:id)">
   <xsl:call-template name="behaves">
    <xsl:with-param name="behavior" select="key('ownedBehaviors', @xmi:id)" />
   </xsl:call-template>
  </xsl:if>

  <xsl:choose>
   <xsl:when test="$returnType='boolean'"><xsl:text>return false;</xsl:text></xsl:when>
   <xsl:when test="$returnType='int'"><xsl:text>return 0;</xsl:text></xsl:when>
   <xsl:when test="$returnType='void'"><xsl:text>return;</xsl:text></xsl:when>
   <xsl:otherwise><xsl:text>return null;</xsl:text></xsl:otherwise>
  </xsl:choose>
 }
</xsl:template>


<!-- Metodu elgsenos -->
<xsl:template name="behaves">
 <xsl:param name="behavior" />
  <xsl:call-template name="fragments">
   <xsl:with-param name="fragment" select="$behavior/fragment" />
  </xsl:call-template>
</xsl:template>

<!-- Fragmentu interpretavimas -->
<xsl:template name="fragments">
<xsl:param name="fragment" />
<xsl:for-each select="$fragment">
<xsl:choose>
<xsl:when test="@xmi:type='uml:EventOccurrence'">
 <xsl:variable name="c_ownlife" select="key('lifelines', @covered)" />
 <xsl:variable name="c_ownattr" select="key('ownedAttributes', $c_ownlife/@represents)" />
 <xsl:variable name="c_ownmem" select="key('ownedMembers', $c_ownattr/@type)" />

 <xsl:if test="@sendMessage">
  <xsl:variable name="c_message" select="key('messages', @sendMessage)" />
  <xsl:variable name="c_endfrag" select="key('fragments', $c_message/@receiveEvent)" />

  <xsl:variable name="c_endlife" select="key('lifelines', $c_endfrag/@covered)" />
  <xsl:variable name="c_endattr" select="key('ownedAttributes', $c_endlife/@represents)" />
  <xsl:variable name="c_endmem" select="key('ownedMembers', $c_endattr/@type)" />

  <xsl:choose>
   <xsl:when test="$c_message/eAnnotations/details/@key='create'">
    <xsl:value-of select="$c_endmem/@name" />
    <xsl:text> </xsl:text>
    <xsl:value-of select="$c_endattr/@name" />
    <xsl:text> = new </xsl:text>
    <xsl:value-of select="$c_endmem/@name" />
    <xsl:text>();</xsl:text>
    <xsl:text>&#10;</xsl:text>
   </xsl:when>
   <xsl:when test="$c_message/eAnnotations/details/@key='destroy'">
    <xsl:value-of select="$c_endattr/@name" /><xsl:text> = null;</xsl:text>
    <xsl:text>&#10;</xsl:text>
    </xsl:when>
    <xsl:when test="$c_message/eAnnotations/details/@key='return'">
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="$c_endattr/@name" /><xsl:text>.</xsl:text>
     <xsl:value-of select="$c_message/@name" /><xsl:text>;</xsl:text>
     <xsl:text>&#10;</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
```

```
 </xsl:if>
</xsl:when>
<xsl:when test="@xmi:type='uml:CombinedFragment'">
 <xsl:choose>
 <xsl:when test="@interactionOperator='loop'">
  <xsl:text>while (</xsl:text>
   <xsl:call-template name="rules">
    <xsl:with-param name="operand" select="operand" />
   </xsl:call-template>
  <xsl:text>) {</xsl:text>
  <xsl:text>&#10;</xsl:text>
   <xsl:call-template name="fragments">
    <xsl:with-param name="fragment" select="operand/fragment" />
   </xsl:call-template>
  <xsl:text>&#10;}&#10;</xsl:text>
 </xsl:when>
 <xsl:when test="@interactionOperator='opt'">
  <xsl:text>if (</xsl:text>
  <xsl:call-template name="rules">
    <xsl:with-param name="operand" select="operand" />
   </xsl:call-template>
  <xsl:text>) {</xsl:text>
  <xsl:text>&#10;</xsl:text>
  <xsl:call-template name="fragments">
    <xsl:with-param name="fragment" select="operand/fragment" />
   </xsl:call-template>
  <xsl:text>&#10;}&#10;</xsl:text>
 </xsl:when>
 <xsl:when test="@interactionOperator='break'">
  <xsl:text>if (</xsl:text>
  <xsl:call-template name="rules">
    <xsl:with-param name="operand" select="operand" />
   </xsl:call-template>
  <xsl:text>) {</xsl:text>
  <xsl:text>&#10;</xsl:text>
  <xsl:call-template name="fragments">
    <xsl:with-param name="fragment" select="operand/fragment" />
   </xsl:call-template>
  <xsl:text>&#10;break;&#10;}&#10;</xsl:text>
 </xsl:when>
 </xsl:choose>
</xsl:when>
</xsl:choose>
</xsl:for-each>
</xsl:template>


<!-- Taisykles -->
<xsl:template name="rules">
 <xsl:param name="operand" />
 <xsl:for-each select="$operand/ownedRule">
  <xsl:choose>
   <xsl:when test="specification/@language='Logic'">
    <xsl:value-of select="specification/@body" />
   </xsl:when>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>

<!-- Saugos israiskos -->
<xsl:template name="guard">
 <xsl:param name="operand" />
 <xsl:for-each select="$operand/guard">
  <xsl:value-of select="@xmi:id" />
 </xsl:for-each>
</xsl:template>

<!-- UML2 Primityvai -->
<xsl:template name="primitivestempl">
 <xsl:param name="pnode" />
 <xsl:variable name="primfile">
  <xsl:call-template name="basename">
   <xsl:with-param name="filename"><xsl:value-of select="$pnode/type/@href" /></xsl:with-param>
  </xsl:call-template>
 </xsl:variable>
 <xsl:variable name="primid" select="substring-after($pnode/type/@href, '#')" />

 <xsl:for-each select="document($primfile)//ownedMember">
  <xsl:if test="@xmi:id=$primid">
   <xsl:call-template name="primitivemap">
    <xsl:with-param name="primname" select="@name" />
   </xsl:call-template>
  </xsl:if>
 </xsl:for-each>
</xsl:template>
```

```
<!-- UML2 => Java primityvu keitimas -->
<xsl:template name="primitivemap">
 <xsl:param name="primname" />
 <xsl:choose>
  <xsl:when test="$primname='Integer'">
   <xsl:text>int</xsl:text>
  </xsl:when>
  <xsl:when test="$primname='Boolean'">
   <xsl:text>boolean</xsl:text>
  </xsl:when>
  <xsl:when test="$primname='UnlimitedNatural'">
   <xsl:text>int</xsl:text>
  </xsl:when>
  <xsl:otherwise>
   <xsl:value-of select="$primname" />
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<!-- Pagalbiniai sablonai -->

<!-- failo vardo ispakavimas is URI -->
<xsl:template name="basename">
 <xsl:param name="filename" />
 <xsl:choose>
  <xsl:when test="contains($filename, '/')">
   <xsl:call-template name="basename">
    <xsl:with-param name="filename"><xsl:value-of select="substring-after($filename, '/')" /></xsl:with-
param>
   </xsl:call-template>
  </xsl:when>
  <xsl:when test="contains($filename, '#')">
   <xsl:call-template name="basename">
    <xsl:with-param name="filename"><xsl:value-of select="substring-before($filename, '#')" /></xsl:with-
param>
   </xsl:call-template>
  </xsl:when>
  <xsl:otherwise>
   <xsl:value-of select="$filename" />
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>


</xsl:stylesheet>
```

11.6 Sugeneruotas programos skeletas

```
// Failas Fabrikas.java
// naudojamas Eile.java
//
public class Fabrikas
{

    // Attributes
    private Gamintojas gamintojas;
    private Vartotojas vartotojas;

    // Methods
    public void dirbti ()
    {
        Eile eile = new Eile();
        Gamintojas gamintojas = new Gamintojas();
        gamintojas.gaminti(eile);
        Vartotojas vartotojas = new Vartotojas();
        while (!eile.yraTuscia()) {
            vartotojas.vartototi(eile);
            gamintojas.gaminti(eile);
        }
        gamintojas = null;
        vartotojas = null;
        eile = null;
        return;
    }

}

// Failas Vartotojas.java
public class Vartotojas
{

    // Methods
    public void suvartotiElementa ()
```

```
    {
        return;
    }


    public boolean vykstaVartojimas ()
    {
        return false;
    }


    public void vartoti (Eile eile)
    {
        while (this.vykstaVartojimas()) {
            this.suvartotiElementa(eile);
            if (eile.yraTuscia()) {
                eile.isvalytiBuferi();
                break;
            }

        }
        return;
    }

}


// Failas Gamintojas.java
public class Gamintojas
{

    // Methods
    public boolean vykstaGamyba ()
    {
        return false;
    }


    public void paimtiElementa (Elementas elementas)
    {
        return;
    }


    public boolean yraParuostasElementas ()
    {
        return false;
    }


    public void gaminti (Eile eile)
    {
        Elementas elementas = new Elementas();
        while (this.vykstaGamyba()) {
            if (this.yraParuostasElementas()) {
                this.paimtiElementa(elementas);
                eile.idetiElementa(elementas);
            }
        }
        return;
    }

}

// Failas Eile.java
// naudojamas Elementas.java
//
public class Eile
{

    // Methods
    public void isvalytiBuferi ()
    {
        return;
    }


    public boolean yraTuscia ()
    {
        return false;
    }


    public Elementas paimtiElementa ()
    {
```

```
        return null;
    }


    public void idetiElementa (Elementas elementas)
    {
        return;
    }

}


// Failas Elementas.java
public class Elementas
{
}
```

11.7 XSLT transformacijos sugeneruoti svetainės puslapiai

```
<html xmlns="http://www.w3.org/1999/xhtml">
<!--Failas Index-->
  <head>
    <title>Pagrindinis Puslapis</title>
    <div id="menu">
      <div class="plink">
        <a href="Apie">Apie mus</a>
      </div>
      <div class="plink">
        <a href="Naujienos">Svetainės naujienos</a>
      </div>
      <div class="plink">
        <a href="Informacija">Naudinga informacija</a>
      </div>
    </div>
  </head>
</html>

<html xmlns="http://www.w3.org/1999/xhtml">
<!--Failas Apie-->
  <head>
    <title>Apie mus</title>
    <div id="menu">
      <div class="plink">
        <a href="Index">Pagrindinis Puslapis</a>
      </div>
      <div class="plink">
        <a href="Kontaktai">Kontaktinė informacija</a>
      </div>
      <div class="plink">
        <a href="Darbo Laikas">Darbo laiko informacija</a>
      </div>
    </div>
  </head>
</html>

<html xmlns="http://www.w3.org/1999/xhtml">
<!--Failas Naujienos-->
  <head>
    <title>Svetainės naujienos</title>
    <div id="menu">
      <div class="plink">
        <a href="Index">Pagrindinis Puslapis</a>
      </div>
    </div>
  </head>
</html>

<html xmlns="http://www.w3.org/1999/xhtml">
<!--Failas Informacija-->
  <head>
    <title>Naudinga informacija</title>
    <div id="menu">
      <div class="plink">
        <a href="Index">Pagrindinis Puslapis</a>
      </div>
    </div>
  </head>
</html>

<html xmlns="http://www.w3.org/1999/xhtml">
<!--Failas Kontaktai-->
  <head>
    <title>Kontaktinė informacija</title>
```

```
    <div id="menu">
      <div class="plink">
        <a href="Apie">Apie mus</a>
      </div>
    </div>
  </head>
</html>

<html xmlns="http://www.w3.org/1999/xhtml">
<!--Failas Darbo Laikas-->
  <head>
    <title>Darbo laiko informacija</title>
    <div id="menu">
      <div class="plink">
        <a href="Apie">Apie mus</a>
      </div>
    </div>
  </head>
</html>
```