VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

Valentas Vasiliauskas

VERSLO TAISYKLIŲ VAIZDAVIMAS INFORMACINĖS SISTEMOS LYGMENYJE
REPRESENTATION OF BUSINESS RULES AT INFORMATION SYSTEM LEVEL

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P111

Informacinių sistemų specializacija

Informatikos mokslo kryptis

Vilnius, 2009

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas

_____
(Parašas)
prof. habil. dr. P. G. Adomėnas
(Vardas,Pavardė)

_____
(Data)

Valentas Vasiliauskas

VERSLO TAISYKLIŲ VAIZDAVIMAS INFORMACINĖS SISTEMOS LYGMENYJE
REPRESENTATION OF BUSINESS RULES AT INFORMATION SYSTEM LEVEL

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P111

Informacinių sistemų specializacija

Informatikos mokslo kryptis

Vadovas	Prof. Dr. Olegas Vasilecas (Moksl. Laipsnis, vardas, pavardė)	_____ (Parašas)	_____ (Data)
Konsultantas	Dr. Sergejus Sosunovas (Moksl. Laipsnis, vardas, pavardė)	_____ (Parašas)	_____ (Data)
Konsultantas	Doc. Dr. Angelė Kaulakienė (Moksl. Laipsnis, vardas, pavardė)	_____ (Parašas)	_____ (Data)

Vilnius, 2009

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

TVIRTINU
KATEDROS VEDĖJAS
_____
(Parašas)
_____
(Vardas, pavardė)
_____
(Data)

Technologinių	mokslo sritis
Informatikos	mokslo kryptis
Informatikos	studijų kryptis
Inžinerinės informatikos	studijų programa, valstybinis kodas 62409P111
Informacinių sistemų	specializacija

BAIGIAMOJO MAGISTRO DARBO UŽDUOTIS

…….....Nr. .....

Vilnius

Studentui **VALENTUI VASILIAUSKUI**

Baigiamojo darbo tema: **VERSLO TAISYKLIŲ VAIZDAVIMAS INFORMACINĖS SISTEMOS LYGMENYJE**

Patvirtinta 200...m. …….mėn. …..d. dekano įsakymu Nr. ……

Baigiamojo darbo užbaigimo terminas 200……..

BAIGIAMOJO DARBO UŽDUOTIS:

PROBLEMA:

Įmonės verslo taisyklės nėra pakankamai formalios ir detalios, kad iš jų modelio būtų galima iš karto generuoti programų sistemos kodą.

TIKSLAS:

Patobulinti verslo taisyklių atvaizdavimo informacinės sistemos lygmenyje metodą.

UŽDAVINIAI:

1. Išanalizuoti, kaip formalizuojamos verslo lygmens taisyklės. Nustatyti, kurie verslo taisyklių atvaizdavimo metodai dažniausiai naudojami, jų privalumus ir trūkumus.
2. Palyginus metodus, nustatyti patį tinkamiausią metodą pasirinktai verslo taisyklių klasei užrašyti.
3. Išnagrinėti, kokios problemos iškyla verslo taisykles atvaizduojant informacinės sistemos lygmenyje, naudojant pasirinktą metodą.
4. Sukurti priemonę (kompiuterinę programą), kuri pašalins šias problemas.

Baigiamojo darbo rengimo konsultantas: ……dr. Sergejus Sosunovas……..
(vardas, pavardė, mokslinis laipsnis ir vardas)

Vadovas …… .....prof. dr. Olegas Vasilecas.....
(parašas) (vardas, pavardė, mokslinis laipsnis ir vardas)

Užduotį gavau

……..
(Parašas)
Valentas Vasiliauskas
(Vardas, pavardė)
2008.02.07
(Data)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

Technologinių mokslo sritis

Informatikos inžinerijos mokslo kryptis

Inžinerinės informatikos studijų kryptis

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P111

PAŽYMA
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ

…….....Nr. .....

Vilnius

Studentas **Valentas Vasiliauskas** .....
(Vardas, pavardė)

Studento studijų svertinis įvertinimų vidurkis..... balo.

Baigiamojo darbo tema:

VERSLO TAISYKLIŲ VAIZDAVIMAS INFORMACINĖS SISTEMOS LYGMENYJE

Baigiamasis darbas peržiūrėtas ir studentui Valentui Vasiliauskui leidžiama ginti šį baigiamąjį darbą magistro laipsnio suteikimo komisijoje.

Katedros vedėjas ..... prof. habil. dr. P. G. Adomėnas
(Parašas) (Moksl. laipsnis, vardas, pavardė)

VADOVO ATSILIEPIMAS
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ
.....

Vadovas ..... prof. dr. Olegas Vasilecas
(Parašas) (Moksl. laipsnis, vardas, pavardė)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

RECENZIJA
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ
..... Nr. .....

Studentas **Valentas Vasiliauskas** parengė baigiamąjį magistro darbą tema:
VERSLO TAISYKLIŲ VAIZDAVIMAS INFORMACINĖS SISTEMOS LYGMENYJE

RECENZIJA

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Magistro laipsnio suteikimo komisijos narys

....
(Parašas) (Moksl. Laipsnis, vardas, pavardė)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

RECENZIJA
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ
..... Nr. .....

Studentas **Valentas Vasiliauskas** parengė baigiamąjį magistro darbą tema:
VERSLO TAISYKLIŲ VAIZDAVIMAS INFORMACINĖS SISTEMOS LYGMENYJE

RECENZIJA

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Magistro laipsnio suteikimo komisijos narys

....
(Parašas) (Moksl. Laipsnis, vardas, pavardė)

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių sistemų katedra

ISBN ISSN
Egz. sk. ……..
Data ….-….-….

Inžinerinės informatikos studijų programos baigiamasis magistro darbas
Pavadinimas **Verslo taisyklių vaizdavimas informacinės sistemos lygmenyje**
Autorius **Valentas Vasiliauskas** Vadovas prof. dr. **Olegas Vasilecas**

Kalba
[X] lietuvių
[ ] užsienio

Anotacija

Baigiamajame magistro darbe nagrinėjamas verslo taisyklių vaizdavimas informacinės sistemos lygmenyje. Pateikiamas bendras verslo taisyklių supratimas, jų svarba bei tipai. Nagrinėjama, kokie metodai naudojami verslo taisyklėms atvaizduoti informacinės sistemos lygmenyje. Aprašomi šių metodų privalumai ir trūkumai. Nustatoma, kad tinkamiausias verslo taisyklių atvaizdavimo metodas yra UML klasių diagrama naudojama kartu su OCL kalba. Nagrinėjamos problemos, su kuriomis susiduria naudotojas užrašydamas OCL išraiškas. Sukuriamas įrankis, išsprendžiantis šias problemas. Šis įrankis leidžia naudotojui generuoti keturias dažniausiai naudojamas OCL išraiškas. Baigiamojo darbo pabaigoje pateikiamos išvados.

Darbas sudarytas iš tokių pagrindinių dalių: įvado, verslo taisyklių užrašymo IS lygmenyje analizės, OCL išraiškų užrašymo metodo patobulinimo, išvadų, literatūros sąrašo.

Darbo apimtis – 75 puslapiai teksto be priedų, 40 paveikslėliai, 7 lentelės, 27 bibliografiniai šaltiniai.

Atskirai pridedami darbo priedai.

Prasminiai žodžiai: informacinė sistema, įrankis, klasių diagrama, koncepcinis modelis, metodas, papildinys, OCL išraiška, verslo taisyklė.

Vilnius Gediminas Technical University
Fundamental Sciences faculty
Information Systems department

ISBN ISSN
Copies No. ……
Date ….-….-….

Engineering Informatics study programme master thesis.

Title: **Representation of Business Rules at Information System Level**

Author **Valentas Vasiliauskas** Academic supervisor **prof. dr. Olegas Vasilecas**

Thesis language

[X] Lithuanian

[ ] Foreign (English)

Annotation

In final master thesis representation of business rules at information system level is analyzed. General understanding about business rules, its importance and types is described. It is analyzed, which methods are used for business rules representation at information system level. Is described the advantages and disadvantages of this methods. It is identified that the best method of business rules representation is UML class diagram using with OCL language. Problems, which occurs when user is trying to write OCL expression, is analyzed. Tool, which let to avoid this problems, is created. This tool helps user to generate four most used OCL expressions. Conclusions are represented in the end of master thesis.

Structure: introduction, analysis of business rules representation at information system level, improvement of OCL expression representation method, conclusions and references.

Thesis consists of: 75 pages text without appendixes, 40 pictures, 7 tables, 27 bibliographical entries.

Appendixes included.

Keywords: Information System, Tool, Class Diagram, Conceptual Model, Method, Plug-in, OCL Expression, Business Rule.

Turinys

Paveikslų sąrašas

Lentelių sąrašas

Santrumpos

AOR (angl. *Agent-Object-Relationship*) – agento-objekto-ryšio modelis

BIER (angl. *Behavior Integrated Entity Relationship*) modelis – integruotos elgsenos esybių ryšių modelis

BRIM (angl. *Business Rules Implementation Model*) – verslo taisyklių įgyvendinimo modelis

CPM (angl. *Conceptual Processing Model*) – koncepcinis vykdymo modelis

DBVS – duomenų bazių valdymo sistema

DFD (angl. *Data Flow Diagram*) – duomenų srautų diagramos

ECA (angl. *Event-Condition-Action*) – verslo taisyklės, vaizduojančios schemą įvykis – sąlyga – veiksmas

ELH (angl. *Entity Life History*) modelis – esybės gyvavimo istorijos modelis

ERM (angl. *Entity Relationship Model*) – esybės ryšių modelis

ER-RM (angl. *Entity Relationship-Rule Model*) – esybės ryšių-taisyklių modelis

IS (angl. *Information system*) – informacinė sistema

MBRM (angl. *Manchester Business Rules Management*) – Mančesterio verslo taisyklių valdymas

MCD (angl. *Conceptual Model of Data*) – koncepcinis duomenų modelis

MCT (angl. *Conceptual Model of Processing*) – koncepcinis apdorojimo modelis

MLD (angl. *Logical Model of Data*) – loginis duomenų modelis

MLT (angl. *Logical Model of Processing*) – loginis apdorojimo modelis

MOD (angl. *Organizational Model of Data*) – organizacinis duomenų modelis

MOT (angl. *Organizational Model of Processing*) – organizacinis apdorojimo modelis

MPD (angl. *Physical Model of Data*) – fizinis duomenų modelis

MPT (angl. *Physical Model of Processing*) – fizinis apdorojimo modelis

NIAM (angl. *Natural language Information Analysis Method*) – natūralios kalbos informacinės analizės metodas

OCL (angl. *Object Constraint Language*) – objektų ribojimų kalba

OMT (angl. *Object Modeling Technique*) – objekto modelio technika

OOA&D (angl. *Object – Oriented Analysis and Design*) – objektiškai orientuota analizė ir projektavimas

OOM (angl. *Object-Oriented Methodologies*) – objektiškai orientuoti metodai

ORM (angl. *Object – Role Modelling*) – objekto vaidmenų modeliavimas

PN (angl. *Petri Nets*) – Petri tinklai

SQL (angl. *Structured Query Language*) – struktūrinių užklausų kalba

STD (angl. *State Transition Diagram*) – būsenos parėjimo diagrama

UML (angl. *Unified Modeling Language*) – vieninga modeliavimo kalba

VT – verslo taisyklė

XER (angl. *Extensible Entity Relationship*) – pratęstas esybių ryšių modelis

XML (angl. *eXtensible Markup Language*) – praplečiamoji dokumentų aprašų kalba

ĮVADAS

Verslo taisyklė – direktyva, kuri nurodo poveikį verslui arba vadovauja verslo elgsenai. Verslo taisyklės – svarbus informacinės sistemos elementas. Jos paremtos etiniais, kultūriniais, teisiniais ir organizaciniais įsipareigojimais. Manoma, kad jos gali būti sudarytos remiantis ECA (angl. *event-condition-action*) mechanizmu. Verslo taisyklės informacinėje sistemoje atvaizduojamos koncepciniu modeliu. Plačiąja prasme koncepcinis modelis – tai teiginių, aprašančių dalykinę sritį ir pateikiamų kokia nors kalba, rinkinys (vaizdavimo formalizmas).

Dažniausiai naudojami į funkcijas, duomenis ir objektus orientuoti verslo sistemos analizės metodai ir pagal juos suprojektuoti koncepciniai modeliai.

Pagrindinės problemos, su kuriomis susiduriama nagrinėjant verslo taisyklių vaizdavimą informacinės sistemos lygmenyje:

- įmonės verslo taisyklės nėra pakankamai formalios ir detalios, kad iš jų modelio būtų galima iš karto generuoti programų sistemos kodą. Todėl, norint tinkamai atvaizduoti verslo taisykles, reikia jas užrašyti pakankamai formalia kalba;
- žinoma daug būdų ir metodų, kaip sumodeliuoti verslo taisyklių modelį. Tačiau universalus (iki galo išbaigtas) metodas, aprašantis kaip sumodeliuoti visas verslo taisyklių klases, iki šiol nėra sukurtas.

UML siūlo įvairias programų sistemų reikalavimų atvaizdavimo diagramas. Turbūt pati svarbiausia diagrama yra klasių diagrama, kuri atvaizduoja visus pagrindinius taikomosios programos struktūros aspektus. Tačiau UML klasių diagrama, tipiškai nėra tiek tobula, kad galėtų atvaizduoti visus tiesioginius sistemos specifikacijos aspektus. Be kitų dalykų, egzistuoja poreikis aprašyti papildomus ribojimus susijusius su objektais esančiais modelyje. Šie ribojimai dažniausiai aprašomi natūralia kalba. Praktika rodo, kad tai visuomet sukuria dviprasmybes. Siekiant parašyti nedviprasmiškus ribojimus, buvo sukurta taip vadinama formali kalba. Tradicinių formalių kalbų trūkumas yra tas, kad jos dažniausiai yra naudingos tik asmenims su stipriomis matematinėmis žiniomis, bet sudėtingos vidutiniam verslo ar sistemos modeliuotojui.

Objektų ribojimų kalba (OCL) – formali kalba, naudojama aprašyti išraiškas (angl. *expressions*) UML modeliuose. Šios išraiškos tipiškai apibrėžia invariantines (angl. *invariant*) sąlygas, kurios turi būti taikomos projektuojamoje sistemoje arba užklausas objektams aprašytiems modelyje. OCL išraiškos gali būti naudojamos apibrėžti operacijas arba veiksmus, kuriuos įvykdžius, pasikeis sistemos būsena. UML modeliuotojai gali naudoti OCL specifinių programos ribojimų apibrėžimui savo modelyje, bei OCL užklausų, kurios yra visiškai programiškai nepriklausomos, apibrėžimui UML modelyje.

TYRIMO OBJEKTAS

Verslo taisyklių vaizdavimas informacinės sistemos lygmenyje.

TYRIMO TIKSLAS IR UŽDAVINIAI

Darbo tikslas – patobulinti verslo taisyklių vaizdavimo informacinės sistemos lygmenyje metodą.

Tikslui pasiekti keliami tokie uždaviniai:

1. Išanalizuoti, kaip formalizuojamos verslo lygmens taisyklės. Nustatyti, kurie verslo taisyklių atvaizdavimo metodai yra dažniausiai naudojami, jų privalumus ir trūkumus.
2. Palyginus metodus, nustatyti patį tinkamiausią metodą pasirinktai verslo taisyklių klasei užrašyti.
3. Išnagrinėti, kokios problemos iškyla verslo taisykles atvaizduojant informacinės sistemos lygmenyje, naudojant pasirinktą metodą.
4. Sukurti priemonę (kompiuterinę programą), kuri pašalins šias problemas.

TYRIMO NAUJUMAS

Kuriant informacines sistemas vienas iš pagrindinių etapų yra verslo taisyklių modelio sukūrimas. Žinoma daug būdų ir metodų, kaip modeliuoti verslo taisyklių modelį. Tačiau universalaus metodo, kuriuo būtų galima sumodeliuoti visas verslo taisykles, iki šiol nėra sukurta.

TYRIMO AKTUALUMAS

Sprendžiama problema yra aktuali, nes šiuo metu vykdomame projekte „Veiklos taisyklių sprendimai informacinių sistemų kūrimui (VeTIS)“ reikia sukurti verslo taisyklių atvaizdavimo į OCL specifikacijas metodą.

TYRIMO METODIKA

Tyrimas apims literatūros apžvalgą ir modeliavimą.

Išanalizavus įvairius verslo taisyklių vaizdavimo metodus, išsiaiškinus jų privalumus ir trūkumus, palyginus visus metodus buvo nustatyta, kad tinkamiausiai verslo taisykles atvaizduoja UML klasių diagrama, naudojama kartu su OCL kalba.

MOKSLINĖ DARBO VERTĖ

Darbe analizuojami verslo taisyklių atvaizdavimo metodai. Šių metodų palyginimas parodė jų stipriąsias ir silpnąsias vietas, bei leido nustatyti, kad šiuo metu verslo taisyklių atvaizdavimui tinkamiausia yra UML klasių diagrama naudojama kartu su OCL kalba. Remiantis problemomis, su kuriomis susiduria naudotojas, buvo sukurtas papildinys (angl. *plug-in*), kuris supaprastina dažniausiai naudojamų OCL išraiškų rašymą

DARBO REZULTATAI

Galima išskirti tokius pagrindinius darbo rezultatus:

1) išnagrinėjus, kokie verslo taisyklių atvaizdavimo metodai yra dažniausiai naudojami, buvo nustatyti šių metodų privalumai ir trūkumai;
2) palyginus visus metodus, buvo nustatyta, kad verslo taisyklių atvaizdavimui tinkamiausia UML klasių diagrama naudojama kartu su OCL kalba;
3) išnagrinėta, su kokiomis problemomis susiduria naudotojai, OCL kalba užrašydami verslo taisykles UML klasių diagramoje;
4) sukurtas papildinys, pašalinantis šias problemas.

DARBO STRUKTŪRA

Darbą sudaro įvadas, analitinė ir projektinė dalis, išvados, literatūros sąrašas.

Analitinėje dalyje pateikiama verslo taisyklės samprata. Nagrinėjami dažniausiai verslo taisyklių atvaizdavimui naudojami metodai. Aprašomi jų privalumai ir trūkumai. Atliekamas šių metodų palyginimas ir, atsižvelgiant į palyginimo rezultatus, išrenkamas pats geriausias verslo taisyklių atvaizdavimo metodas.

Praktinėje dalyje pateikiamas įrankio, suteikiančio naudotojui galimybę, nežinant OCL kalbos sintaksės, generuoti OCL išraiškas, aprašymas.

DARBO APROBACIJA

Šio darbo medžiaga naudojama projekte “Veiklos taisyklių sprendimai informacinių sistemų kūrimui (VeTIS)”.

1. VERSLO TAISYKLIŲ UŽRAŠYMO IS LYGMENYJE METODŲ ANALIZĖ

1.1. Verslo taisyklės samprata

Pastaraisiais metais pabrėžiama, kad verslo taisyklės yra svarbus daugelio informacinių sistemų elementas. Daugelis verslo procesų yra vykdoma atsižvelgiant į taisykles, kurios aprašo tam tikrą veiksmą arba susieja sąrašą galimų veiksmų. Verslo taisyklės paremtos etiniais, kultūriniais, teisiniais ir organizaciniais įsipareigojimais. Manoma, kad jos gali būti sudarytos remiantis įvykis-sąlyga-veiksmas (angl. *event-condition-action*) mechanizmu, pasiūlytu nagrinėjant aktyvias duomenų bazes, t.y. nagrinėjant ryšius tarp įvykių, sąlygų ir veiksmų. Įvykis – momentinis atsitikimas įmonėje, kuriam priskiriamas laiko momentas. Sąlyga aprašo, kas turi būti patikrinta įvykus įvykiui. Veiksmai apibrėžia, kas turi būti atlikta įvykus konkrečiam įvykiui.

Tiktai maža dalis iš milžiniško skaičiaus verslo taisyklių yra realizuojama informacinėje sistemoje. Verslo atžvilgiu šie vientisumo ribojimai dažnai neatrodo trivialūs ir netgi ne visada galima aiškiai suformuluoti juos reikalavimų lygmenyje. Netrivialių verslo taisyklių turinys gali priklausyti nuo erdvės (skirtingos teisinės sistemos skirtingose valstybėse) ir laiko (teisinė sistema gali keistis laikui bėgant). Todėl taisyklės turi būti palaikomos visą taisyklėmis pagrįstų taikomųjų programų gyvavimo laiką.

Verslo taisyklės, naudojamos informacinėje sistemoje, formuoja pagrindinę taikomosios programos dalį ir tokiu būdu yra atskiros visoje sistemoje. Todėl šios taisyklės gali būti neužbaigtos ir perteklinės, jos gali būti apibrėžtos skirtingais būdais skirtingose taikomose programose ir/arba gali būti įgyvendintos skirtingai. Netgi jei taisyklės įgyvendinamos nuosekliai, tai neužtikrina, kad jos nepasimes nesuderintai palaikant perteklinę kodo dalį. Todėl yra daug argumentų palaikančių neperteklinį, centralizuotą, informacinei sistemai svarbių verslo taisyklių įgyvendinimą duomenų bazėje. Toks požiūris turi papildomų privalumų – taisyklės negali būti neatvaizduotos specialiose DBVS operacijose.

Keletas komercinių DBVS produktų ir tyrimų prototipų pateikia priemones, tokias kaip: trigerių mechanizmai ir saugojimo procedūros, skirtos taisyklių integracijai. Kai kurie vientisumo ribojimų tipai yra apibrėžti SQL2 pakete; nuo SQL3 standarto įdiegta loginė struktūra, apibrėžianti trigerius. Aktyvių DBVS sukūrimas pasiūlė aukštesnį taisyklių gavimo potencialą.

Būsimų ir egzistuojančių komercinių DBVS produktų galios savarankiškumas suteikia keletą privalumų aprašant verslo taisykles koncepciniame lygmenyje. Koncepcinis modelis turi būti vaizduojamas sistemiškai ir realizaciškai nepriklausomas, orientuotas į naudotoją ir pastovus.

Keletas koncepcinio modelio tipų yra naudojami sistemų inžinerijoje. Šie modeliai neužtikrina sistemingos verslo taisyklių elgsenos. Nepaisant to, egzistuojančios sistemos siūlo galimybę specifikuoti tam tikrą verslo taisyklių semantiką [1].

1.2. Verslo taisyklių tipai

1.2.1. Verslo taisyklių klasifikavimas

Rašant apie skirtingų verslo taisyklių modeliavimo metodų potencialą reikia išskirti keletą skirtumų. Realaus pasaulio įvykiai, procesai ir objektai turi būti išskirti jų informacinėje sistemoje (IS įvykiai, IS procesai, IS esybės). Taip pat turėtų skirtis nuoseklumo (angl. *consistency*) ir veiklos (angl. *activity*) taisyklės. Nuoseklumo taisyklės apibrėžia vientisumo ribojimus galiojančioje duomenų bazės būsenoje, o veiklos taisyklės aprašo vykdomus įvykius arba operacijų sekas. Neprieštaringumo taisyklės gali būti klasifikuojamos į aktyvius ir pasyvius ribojimus: pasyvūs vientisumo ribojimai apsaugo veiksmus nuo vietos užėmimo, siekiant užtikrinti duomenų vientisumą, o aktyvūs vientisumo ribojimai, naudodami duomenų manipuliavimo operacijas, palaiko nuoseklumą [18].

Kita dažnai pasitaikanti vientisumo ribojimų klasifikacija – statinių ir dinaminių ribojimų atskyrimas:

- Statiniai ribojimai apriboja duomenų bazės būseną iki leidžiamo sąrašo, be nuorodos į ankstesnę duomenų bazės būseną.
- Tarpiniai ribojimai padaro nuorodas į dvi iš eilės einančias duomenų bazės būsenas.
- Dinaminiai ribojimai leidžia pasirinkti, kurią duomenų bazės būseną laikyti pagrindine [16].

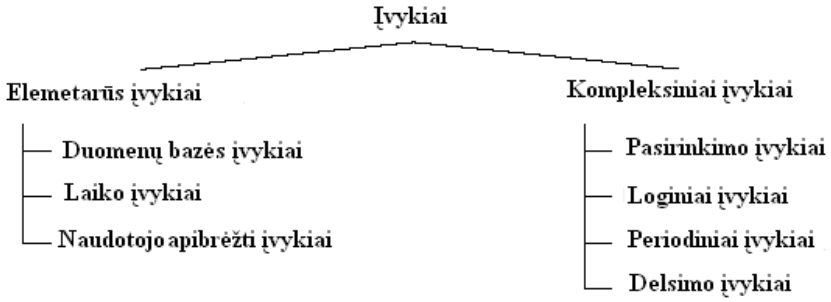
1.2.2. Taisyklių komponentų klasifikavimas

ECA aprašyme skirtumas tarp trijų taisyklės komponentų suteikia modalumo savybę ir geresnį naudojimo tinkamumą. Bet atskyrimas nėra visada paprastas ir gali priklausyti nuo modeliuotojo, priimančio sprendimą požiūrio (pavyzdžiui, kaip apibrėžti būtinus verslo taisyklių komponentus, kurie kartais būna dviprasmiški). Įvykis „kliento skambutis tarp 8^{00} ir 17^{00} valandų“ gali būti suformuluotas, kaip įvykis „kliento telefono skambutis“ ir sąlyga „jeigu laikas tarp 8^{00} ir 17^{00} valandų“. Be to, taisyklės gali neturėti sąlygos, t.y. priklausyti tik nuo įvykio ir veiksmo komponentų.

Išskiriami du įvykių tipai (1 pav.):

- Elementarūs įvykiai – primityvai, kurių negalima smulkiau išskaidyti. Duomenų bazės įvykiai, kaip elementarių įvykių potipis, yra sužadinimas duomenų bazės objektų operacijų (įterpti, atnaujinti, išrinkti ar ištrinti). Laiko įvykiais laikomi laiko taškai, apibrėžiantys priverstinį verslo taisyklių vykdymą. Naudotojo apibrėžti įvykiai apima situacijas, kurios nepriklauso duomenų bazės ar laiko įvykiams ir yra aiškiai apibrėžtos taikomosios programos arba naudotojo.

- Kompleksiniai įvykiai – elementarių įvykių arba kitų kompleksinių įvykių kombinacija su papildomomis savybėmis. Išskiriami tokie įvykiai:
    1) pasirinkimo įvykiai (pvz., „bet kurie trys iš [įvykis$_1$; įvykis$_2$; ...; įvykis$_n$]“),
    2) loginiai įvykiai (pvz., „[produktas X užsakytas] IR [kliento produkto x užsakymas gautas]“),
    3) periodiniai įvykiai (pvz., „kiekvieno mėnesio dešimtą dieną“),
    4) delsimo įvykiai (pvz., „dešimt dienų po kliento užsakymo priėmimo“).



1 pav. Įvykių klasifikavimas [1]

Analogiškai įvykiams išskiriamos elementarios ir kompleksinės sąlygos. Elementarios sąlygos yra arba predikatas arba duomenų užklausa duomenų bazėje. Pavyzdžiui:

1) objekto atributo ir konstantos palyginimas (pvz., „užsakymo būsena = 2”),
2) palyginimas dviejų objektų atributų reikšmių (pvz., „užsakymo suma ≤ kredito limitas“),
3) sąrašas duomenų operacijų (pvz., „kiekvienas klientas susietas su užsakymu“).

Kompleksinės sąlygos susideda iš keleto (elementarių arba kompleksinių) sąlygų, kurios sujungtos loginiais operatoriais.

Veiksmai gali būti klasifikuojami pagal jų vykdymo vietą:

- vidinis IS veiksmas (pvz., ištrynimas senos užsakymo informacijos),
- sąveika tarp IS ir aplinkos (pvz., išsiuntimas pranešimo naudotojui apie pavėluotą apmokėti sąskaitą),
- išorinis IS veiksmas (pvz., pardavimo atstovo telefoninis skambutis klientui).

Veiksmas atliekamas vykdant verslo taisyklę gali atrodyti kaip įvykis, logiškai priklausantis taisyklei [17].

1.2.3. Pavyzdžiai

Siekiant iliustruoti anksčiau aptartą informaciją, pateikiamas mažas verslo taisyklių sąrašas. Šie pavyzdžiai neapima visų taisyklių ir jų galimos tarpusavio priklausomybės. Tačiau tai padeda suprasti specifikavimo metodo esmę.

Šiame pavyzdyje visi asmenys ar organizacijos, kuriems (-oms) parduodami produktai, apibrėžiami (-os) kaip klientas; kliento duomenys identifikuojami unikaliu numeriu ir saugomi

duomenų bazėje. Ribojimas, apsaugantis nuo to, kad pardavimo transakcijos gali būti saugomos tik jeigu egzistuoja vienas ir tik vienas klientas susijęs su pardavimu, yra elementarios verslo taisyklės pavyzdys. Susijus taisyklė yra ta, kad kiekvienas klientas turi būti pateikęs vieną arba daugiau užsakymų, tačiau leidžiama egzistuoti klientui, kuris laikinai neturi aktualių užsakymų. Tokiu būdu objektų klasė UŽSAKYMAS priklauso nuo objektų klasės KLIENTAS. Užsakymo įterpimo prieš sąlyga yra naudotojo egzistavimas duomenų bazėje. Atitinkama verslo taisyklė gali būti apibrėžta kaip:

[VT1] *ON* pageidavimo įterpti užsakymą;

IF neegzistuoja klientas, susijęs su užsakymu;

DO užsakymo įterpimas atmestas; išvesti pranešimą „Klientas neegzistuoja“.

Norint įsitikinti, ar tikrai egzistuoja priklausomybė tarp kliento ir užsakymo, turi būti apribota ištrynimo operacija objektų klasėje UŽSAKYMAS, pvz.:

[VT2A] *ON* pageidavimo ištrinti klientą;

IF egzistuoja užsakymas, susijęs su klientu;

DO kliento ištrynimas atmestas; išvesti pranešimą „Ištrynimo operacija negalima dėl egzistuojančių užsakymų“.

Jeigu apibrėžiamas galimas veiksmas, pavyzdžiui, ištrynimas visų su klientu susijusių užsakymų:

[VT2B] *ON* pageidavimo ištrinti klientą;

IF egzistuoja užsakymas, susijęs su klientu;

DO ištrinti visus užsakymus susijusius su klientu, ištrinti klientą.

Šalia priklausomybės užtikrinimo operacijos statinės taisyklės nurodo tikras atributų reikšmes. Viena taisyklė gali būti, kad nepriimamas joks užsakymas, jei klientas neturi reikiamo kredito reitingo:

[VT3] *ON* pageidavimo įterpti užsakymą;

IF kliento kredito reitingas, susijęs su užsakymu, yra nepakankamas;

DO užsakymo įterpimas atmestas, išvesti pranešimą „Užsakymas nepriimtas dėl per mažo kredito reitingo“.

Tokia taisyklė turi prasmę, tiktai jeigu egzistuoja klientas, dėl to ji priklauso nuo [VT1]. Svarbu taikyti [VT1] prieš [VT3] taikomosios programos logiškumui užtikrinti, dėl to ši seka turi būti apibrėžta reikalavimuose. Kitu atveju taisyklių rūšiavimas gali būti paliktas projektavimo, realizavimo ar netgi taisyklių aktyvios DBVS vykdytojui.

Kita taisyklė apriboja trynimo operaciją atsižvelgiant į laiką:

[VT4] *ON* pageidavimo ištrinti užsakymą;

IF užsakymo įvedimo data > šiandienos data – 5 metai;

DO užsakymo trynimas atmestas.

Gali būti pageidaujama ištrinti visus klientus, kurie per pastaruosius 5 metus nepateikė nei vieno užsakymo. Ši verslo taisyklė yra veiklos taisyklė, nes aprašo veiksmą, kuris turi būti vykdomas, bet nekeičia duomenų bazės vientisumo. Verta pabrėžti, kad trina paketas, kuris yra sužadinamas laiko įvykio:

[VT5] *ON* kiekvieną mėnesio pirmą dieną;

IF neegzistuoja užsakymas, susijęs su klientu, kurio įvedimo data > šiandienos data – 5 metai;

DO ištrinti klientą.

Ši veiklos taisyklės remiasi viena iš [VT2A] arba [VT2B].

Verslo politika gali apibrėžti, kad klientui turi būti pateiktas patvirtinimas, jeigu jo užsakymas priimtas. Dėl šios priežasties išskiriamas realaus pasaulio įvykis užsakymo priėmimas, atitinkantis IS įvykį ir apibrėžiantis besąlyginę veiklos taisyklę:

[VT6] *ON* įvykdžius užsakymo įterpimą;

DO rašyti patvirtinimo laišką; pakeisti užsakymo būseną į „patvirtintas“.

Paskutiniame pavyzdyje pateikiama tarpinė vientisumo taisyklė, kuri reikalauja, kad užsakymas būtų uždarytas, jei užsakyti produktai pristatyti užsakovui:

[VT7] *ON* pageidavimo pakeisti užsakymo būseną į „uždarytas“;

IF užsakymo būsena nėra „pristatytas“;

DO užsakymo būsenos pakeitimas atmestas [1].

1.3. Verslo taisyklių užrašymo metodai

Egzistuoja gausybė aprašymo kalbų, nusakančių ribojimus ir/arba taisykles, bet tai turi labai mažą praktinę reikšmę dėl ribojimų abstraktumo ir formalumo. Galinio naudotojo įtraukimas į IS analizę reikalauja aiškaus ir lengvai suprantamo požiūrio į visa tai. Metodai, kurie leidžia grafinę taisyklių specifikaciją, yra ypač vertinami. Šiame darbe dėmesys telkiamas į praktikoje plačiai naudojamus metodus, analizuojant atidžiau tuos, kurie atrodo perspektyviausi.

1.3.1. Į funkcijas orientuoti metodai

1.3.1.1 VT specifikacija duomenų srautų diagramomis

Duomenų srautų diagramos (toliau – DFD) specifikuoja duomenų srautus tarp kelių procesų arba tarp procesų ir bet kurios išorinės dalies ar duomenų saugyklos. Jos neaprašo veiksmų sekos, t.y. duomenų srauto gavimas nebūtinai gali būti proceso vykdymas. Verslo taisyklių ECA struktūros veiksmo komponentas atitinka DFD procesą. Duomenų srautai kartais gali sutapti su įvykiais (yra duomenų srautai, kurie netiesiogiai priklauso nuo įvykio, ir yra laiko, ir kontrolės įvykiai, kurie negali būti pavaizduoti duomenų srautais).

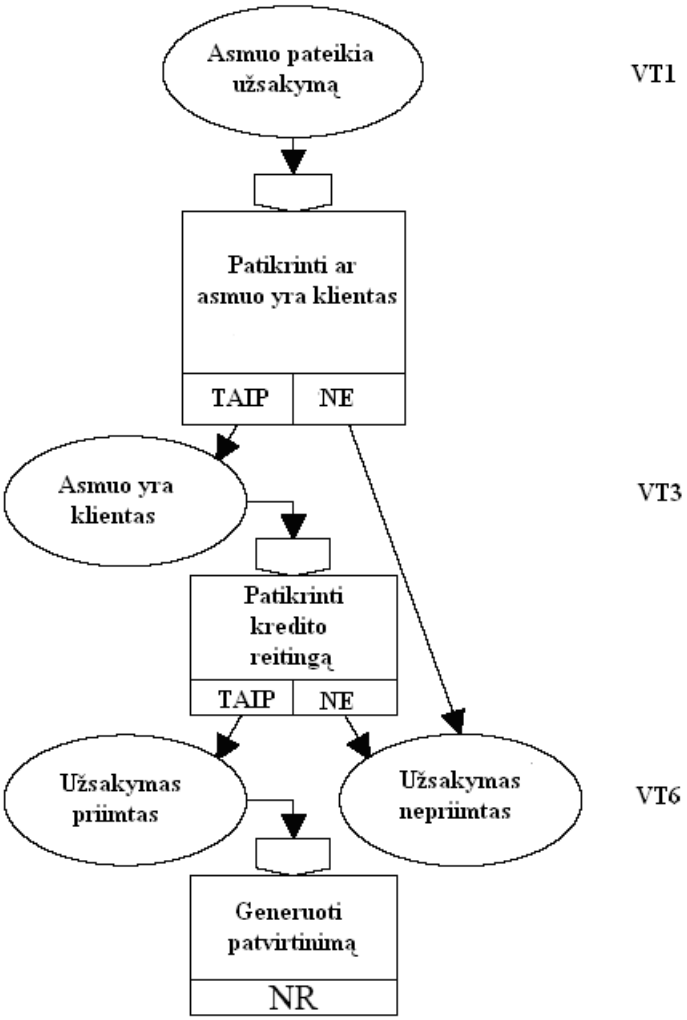
Dažnos DFD neleidžia išreikšti duomenų srautų sinchronizacijos sąlygų atsižvelgiant į proceso vykdymą. Dėl to iš anksčiau paminėtų pavyzdžių tiktai [VT6] gali būti sumodeliuota, nes ji yra aktyvi taisyklė ir neturi sąlygos, netgi tokiu atveju prarandamas numanomas laikinas ryšys tarp nagrinėjamo įvykio ir veiksmo. Siekiant apeiti kai kuriuos iš šių apribojimų, pateikiama keletas modeliavimo sinchronizacijos ir laiko DFD papildymų.

Elementarūs DFD procesai dažnai yra papildomai aprašomi mažomis specifikacijomis, kuriose gali būti įtrauktos verslo taisyklės. Vienas iš tinkamiausių būdų yra jų specifikavimas sprendimų lentelėse ir medžiuose: šie apibrėžia sąlygas ir veiksmus, bet formaliai nepasiūlo įvykio komponento. Tačiau toks verslo taisyklių vaizdavimas turi trūkumų, aptartų anksčiau [2].

1.3.1.2 Specifikacija Merise struktūra

Merise modelis (2 pav.) pateikia koncepcinį apdorojimo modelį (angl. *conceptual processing model - CPM*), kuris nagrinėja vykdomus įvykius, operacijas ir lemia įvykius. Sinchronizacija atitinka sąlygos komponentą ECA mechanizme. Operacija susideda iš vienos ar daugiau užduočių, kurios pagrįstos taisyklių valdymu ir vykdomos nuosekliai. Kiekviena operacija gali suteikti skirtingą įvykį, priklausomai nuo iškviestos taisyklės, kuri gali būti, pavyzdžiui, „operacija sėkmingai įvykdyta“ arba „operacija neįvykdyta“. Sutrumpinimas NR žymi, kad iš jo negalimas joks atsakymas. CPM modelyje įvykis matomas, kaip savybių perteikėjas: tai atitinka atributus Merise duomenų modelyje. Merise/2 pateikia objekto gyvavimo ciklą, kuriame modeliuojami įvykiai, keičiantys tam tikro objekto būseną [3].

Kadangi CPM turi beveik visus būtinus ECA taisyklių modeliavimo komponentus, iš principo beveik visos anksčiau minėtos taisyklės gali būti pavaizduotos. Tačiau CPM susitelkia ties dinaminių ir tarpusavyje nepriklausomų atskirų taisyklių modeliavimu: jis nenumato statinių taisyklių arba taisyklių poveikio modeliavimo objektams. Antrame paveiksle aprašoma [VT1], [VT3] ir [VT6] verslo taisyklės ir seka, pagal kurią jos turi būti vykdomos [1].



2 pav. Taisyklių aprašymas koncepciniame vykdymo modelyje [1]

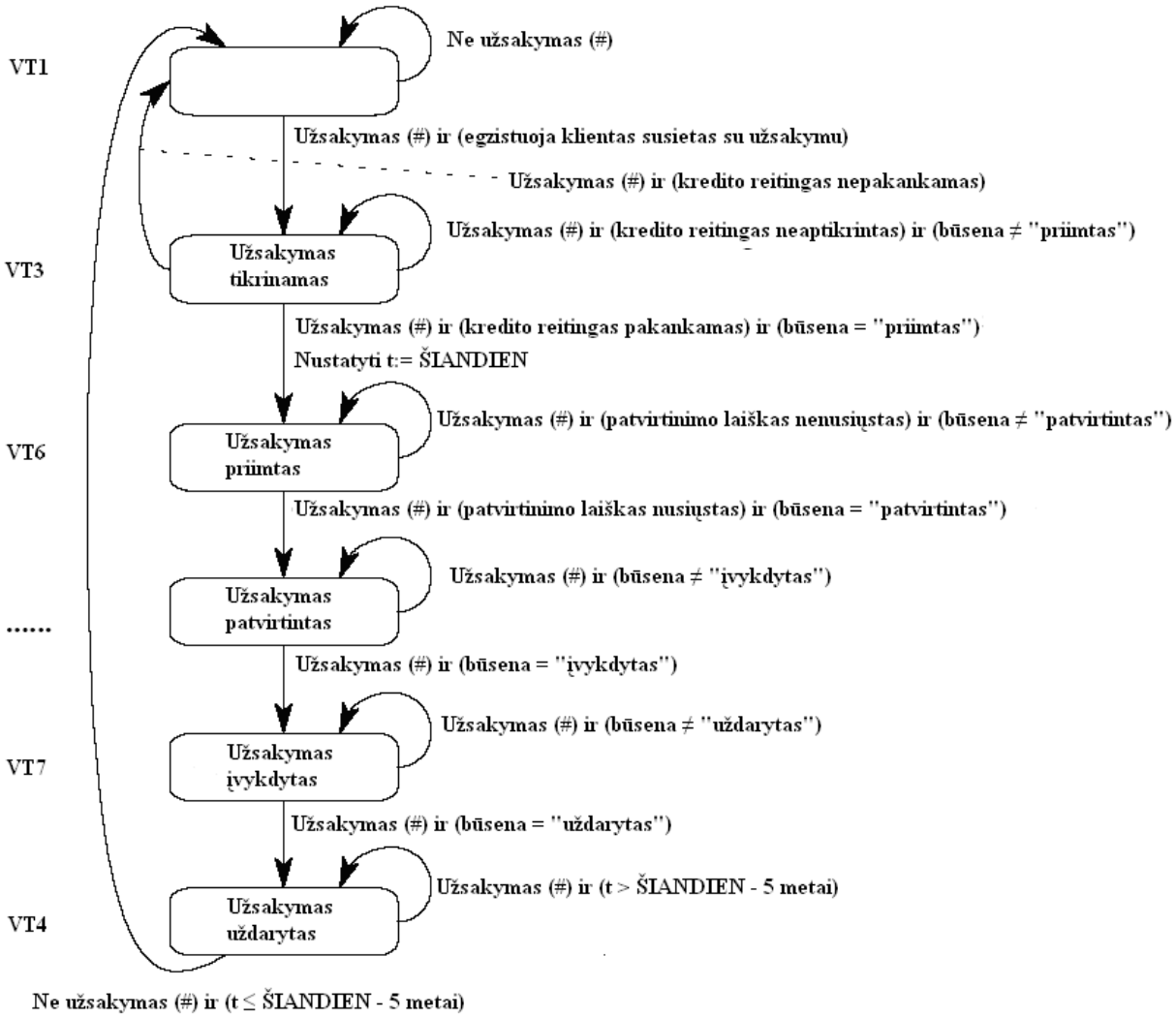
1.3.1.3 Aprašymas būsenos perėjimo modeliais, perėjimo grafais ir schemomis

Dažna sistemos elgsenos aprašymo technologija – būsenos perėjimo diagrama (angl. *state transition diagram - STD*). Šiuose grafuose (3 pav.) mazgai vaizduoja būsenas (pavaizduoti, kaip stačiakampiai apvalintais kampais), o lankai vaizduoja būsenos perėjimus. Sistema, pateikiama STD diagrama, gali būti tiktai vienos būsenos bet kuriuo pateikiamu laiko momentu. Jeigu keletas lankų išeina iš mazgo, tiktai vienas iš šių perėjimų gali būti vykdomas priklausomai nuo jo žymės. Todėl galimas išrinkimo modeliavimas.

STD modelyje aiškiai neapibrėžiami verslo taisyklių komponentų simboliai. Tačiau perėjimai tipiškai pažymimi atitinkama sąlyga ir veiksmu. Juos aprašant išskiriami skirtumai tarp įvykių, sąlygų ir veiksmų. Kadangi duomenų schema negali būti pateikiama yra būtina papildyti šią techniką tinkamu duomenų modeliu [4].

Speciali STD forma pavadinta tranzityviuoju grafu (angl. *transition graph*) naudojama siekiant apibrėžti perėjimo ir dinaminius vientisumo ribojimus. Tranzityvusis grafas konstruojamas iš laikinų formulių ir naudojamas specifinių objektų gyvavimo ciklui aprašyti atsižvelgiant į

ribojimus, mazgai vaizduoja tam tikrą objekto būseną, o lankai išreiškia perėjimą iš vienos būsenos į kitą.



3 pav. Verslo taisyklių aprašymas tranzityviaisiais grafais [1]

Kadangi perėjimo grafas sutelkia dėmesį į būseną ir galimus būsenos pokyčius, įvykiai ir veiksmai diagramose nėra aiškiai apibrėžiami. Manoma, kad formaliai turi būti pavaizduotas kiekvienas perėjimo trukmės vienetas. Laiko pokyčiai nebūtinai reiškia ir būsenos pokyčius: tai modeliuojama sąlyginiais ryšiais į save, kurie taip pat gali išreikšti ir delsimą [5].

Nors grafiniai ECA taisyklių modeliavimo simboliai yra negalimi, iš principo visos taisyklės gali būti modeliuojamos STD modelyje. Šios diagramos tinkamos taisyklių priklausomybei simbolizuoti. Kadangi nėra objektų vaizdavimo, taisyklių poveikis objektams turi būti išreiškiamas žodžiu, dėl kurio gali būti neaiškiai atvaizduotos taisyklės.

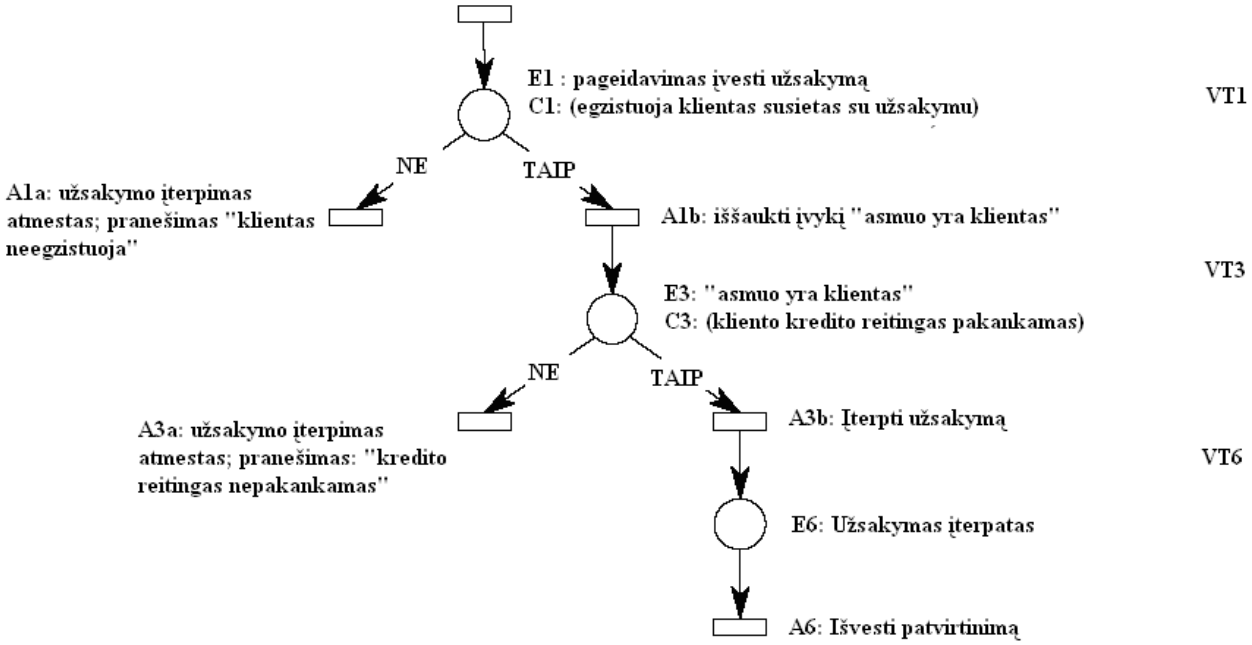
Šio trūkumo iš dalies išvengiama apribojimų tranzityvumo grafuose, vaizduojant tik vieno objekto būsenos pokyčius. Tačiau šiuo atveju taisyklės, kurios lemia keletą objektų, negali būti atitinkamai atvaizduotos viename grafe. Todėl pateiktame pavyzdyje nepavaizduotas [VT4] taisyklės kreipimasis į [VT2B] [1].

Susiję metodai yra būsenos schemos (angl. *statecharts*), kurios naudojamos aprašant sistemų elgseną ir reakciją, sistemos būsenos terminais ir atitinkamos būsenos keitimo įvykiais, jautriais sistemai, loginiai operatoriai įtraukti siekiant aprašyti ECA taisykles [6].

1.3.1.4 Aprašymas Petri tinklais

Petri tinklai (angl. *Petri Nets – PN*) naudoja du mazgų tipus, vaizduojančius vietas (angl. *places*) ir perėjimus, tarpusavyje sujungtus lankais. Šie grafai naudojami aprašyti ir imituoti sistemos elgseną žymėmis, kurios juda iš vienos vietos į kitą priklausomai nuo vykdomo perėjimo. Perėjimas sužadinamas, jeigu jo prieš sąlyga tenkinama, t.y. ankstesnė vieta talpina apibrėžtą žymių skaičių [7].

Modeliuojant verslo taisykles vieta naudojama vaizduoti įvykius ir/arba sąlygas, o perėjimai – žymės veiksmus. PN taip pat sudaro sąlygas elementarių ir kompleksinių įvykių aprašymui. Sumodeliuoti alternatyviems rezultatams, priklausantiems nuo sąlygos sakinių, PN turi būti išplėsta pažymėtais lankais. Ketvirtame paveiksle pateikiama [VT1], [VT3] ir [VT6] semantika, kaip atitinkami išplėsti PN (angl. *extended PN*). Su šiuo išplėtimu įmanoma sumodeliuoti visų aukščiau paminėtų taisyklių struktūrą, tačiau skirtingos semantikos sujungtos vietos simboliais, padaro šią tinklo interpretaciją daug sudėtingesnę [1].



4 pav. Verslo taisyklių vaizdavimas išplėstu Petri tinklu [1]

Dažno PN trūkumas yra, kad jis negali pavaizduoti tikros laikinos būsenos, pavyzdžiui, galimas užlaikymas tarp įvykusio įvykio, patikrinimo sąlygos ir atitinkamo veiksmo (tokie užlaikymai gali būti vaizduojami laiko PN). MODYN duomenų modeliavimo formalizme, Petri tinklas papildomas žymės reikšmėmis, siekiant apibrėžti objekto būsenos sekos, sugeneruotos įvykių ir objektų tipų dinamikos duomenų bazėje, ribojimus.

Dar vienas PN trūkumas yra tas, kad jo konstrukcija vaizduoja skirtingą semantiką, pvz.: vietos gali simbolizuoti objekto būseną, įvykį arba sąlygą. Taikant skirtingus išplėtimus gaunami gana skirtingi verslo sistemos atvaizdai. Be to, PN stokoja priemonių, vaizduojančių duomenų objektus. Kai kurie autoriai jungia PN tinklus su esybių-ryšių modeliu [7].

1.3.2. Į duomenis orientuoti metodai

1.3.2.1 Aprašymas esybių-ryšių modeliu ir NIAM

Daugelis literatūroje pateiktų verslo taisyklių yra statinės ir išreikštos skirtingais esybių-ryšių modelio (angl. *Entity-Relationship-Mode - ERMl*) variantais. Tačiau verslo taisyklės turi būti modeliuojamos netiesiogiai, naudojant ERM semantikas tokias kaip: kardinalumas ir gyvavimo priklausomybė [8].

[VT1], [VT2A] ir [VT2B] gali iš dalies būti išreikštos kaip parodyta penktame paveiksle: esybės tipas UŽSAKYMAS priklauso nuo KLIENTO, todėl sąlyga manipuliuoti užsakymu yra priklausoma nuo susijusio kliento.

5 pav. ER diagrama skirta užsakymo įvedimo sistemai

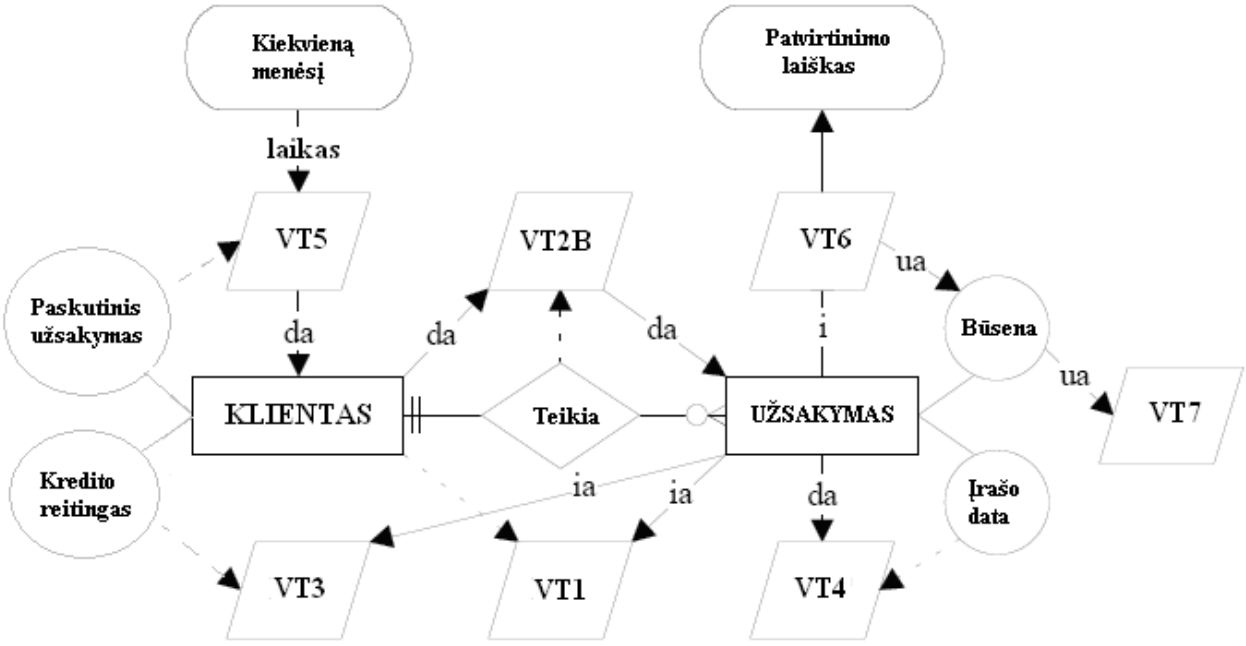
ERM nesuteikia aiškesnės įvykio, sąlygos ar veiksmo formuluotės. Kadangi kiekviena taisyklė, išreiškiama ERM modeliu, turi vientisumo ribojimo pobūdį, tai sakoma, kad kiekvienas pažeidimas nurodo pageidaujamos operacijos atrinkimą (kaip [VT1] taisyklėje). Tuo tarpu bet kuri įterpimo operacija, kuri pažeidžia duomenų bazės vientisumą, natūraliai nurodo į atmetimą, reakcija į trynimo veiksmą yra dviprasmiška, kadangi, pavyzdžiui, abi, tiek [VT2A], tiek [VT2B], gali būti taikomos. Visa tai negali būti išreiškiama ir suprantama ERM modelyje be papildomo natūralios kalbos aprašo. Kai kurios duomenų modifikacijos gali būti apribotos domeno apibrėžimais, tačiau statinės taisyklės, tokios kaip [VT3], kurios susijusios su atributų reikšmėmis, negalės būti atvaizduotos. Be to, perėjimo taisyklės [VT4] ir [VT7] taip pat kaip veiklos taisyklės [VT5] ir [VT6] negali būti modeliuojamos ERM modelyje [1].

Gerai žinomas ERM „giminaitis“ yra NIAM. Vienas esminių skirtumų tarp jų yra būdas, kuriuo modeliuojami atributai. Vientisumo ribojimų išreiškimui, NIAM naudoja unikalumo ir privalomumo konstrukcijas, kurios gali būti palygintos su kardinalumo ribojimais ERM modelyje. Be to, NIAM siūlo papildomus ribojimus tarp objekto tipų, tokius kaip: unikalumas tarp skirtingų vaidmenų, lygybių ir išskyrimų. Šie papildomi ribojimai daro NIAM šiek tiek galingesniu, atsižvelgiant į verslo taisyklių modeliavimą. Pagal vaizdavimą anksčiau apibrėžtų taisyklių aprašomoji galia tiek ERM, tiek NIAM yra vienoda [9].

1.3.2.2 Esybių-ryšių-taisyklių modelis

Įdomus ERM modelio išplėtimas, aprašant įvykių ir taisyklių sąvokas, yra esybių-ryšių-taisyklių (angl. *entity relationship-rules – ER-R*) modelis. Šiame modelyje yra įtraukta nauja konstrukcija, kuri vaizduoja situacija-veiksmas taisykles, siekdama kontroliuoti esybių būseną, ryšius ir jų atributus. Situacija apibrėžta kaip binarinė įvykių seka turinčia sąlygą: taisyklė bus teisinga ar bus vykdoma, kuomet įvyksta įvykis ir sąlyga yra tenkinama.

Taisyklės išreiškiamos ER-R diagramose lygiagretainių pavidalu. Šie žymimi su skirtingais vardais, tuo tarpu taisyklių turinys diagramoje nepateikiamas. Lygiagretainis gali būti sujungtas tiesioginiais lankais su trimis svarbiomis ERM konstrukcijomis, be to, egzistuoja lankai einantys iš taisyklių į sistemos aplinką. Lankai, einantys į lygiagretainį, simbolizuoja įvedimo arba aktyvacijos įvykius, iš lygiagretainio išeinantys lankai vaizduoja išvedimo arba rezultato įvykius. Lankais žymima, pavyzdžiui, duomenų bazės operacijos: įvesti (angl. *input*) i, atnaujinti (angl. *update*) u, ištrinti (angl. *delete*) d. Modeliai skiriasi tarp bandymo ir sėkmingo šių operacijų vykdymo, šabloniškai šios operacijos gali būti žymimos *ia*, *ua* ir *da*. Taškuoti lankai naudojami papildomam taisyklių duomenų poreikiui išreikšti [12].



6 pav. Verslo taisyklių aprašymas ER-RM modeliu [1]

Kai naudojamas šis žymėjimas, visos sužadintos duomenų bazės operacijų taisyklės gali būti išreikštos kaip pavaizduota šeštame paveiksle. Su šia verslo taisyklių konstrukcija įmanoma parodyti ryšius tarp taisyklių ir jas atitinkančių objektų, tuo tarpu taisyklių seka yra gana neaiški.

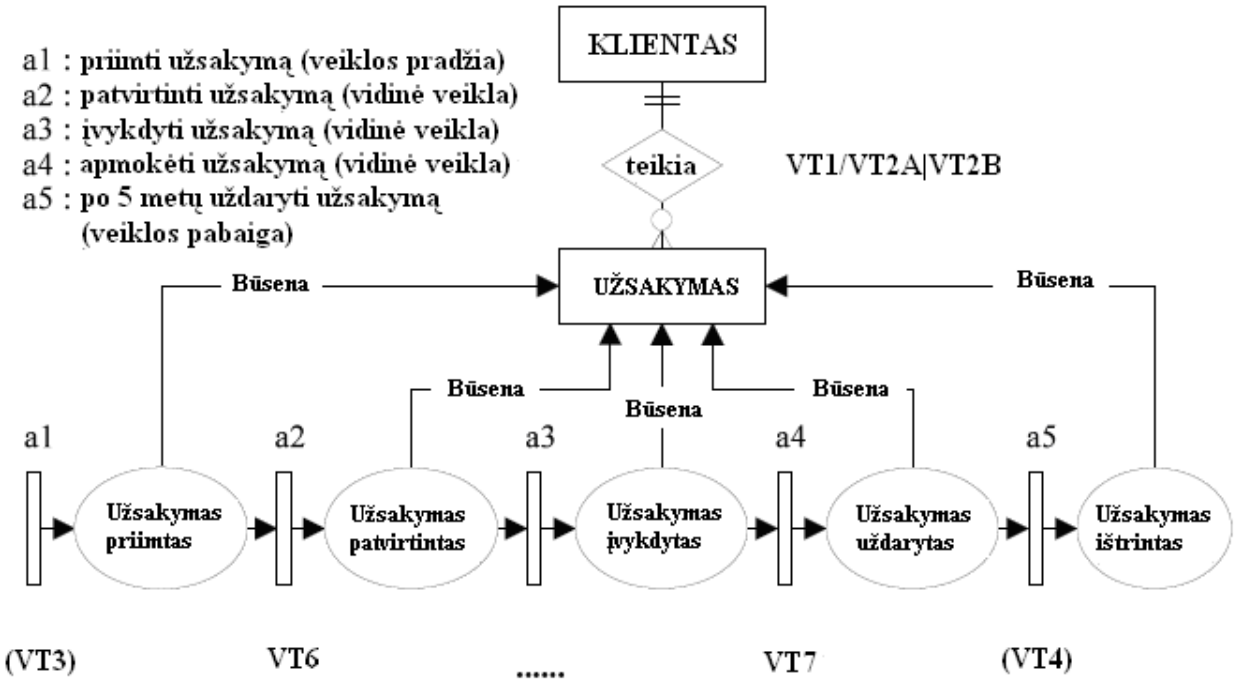
Šeštame paveikslėlyje [VT1] semantika jau yra išreikšta ER poaibio kardinalumu, tokiu būdu gaunamas pertekliškumas. Priklausomybė [VT3] nuo [VT1] nepavaizduota šioje diagramoje; jeigu būtų norima apibrėžti seką kaip antrame ir ketvirtame paveikslėlyje, turėtų būti pridėtas tiesioginis lankas iš [VT1] į [VT3]. Kiekviena vykdoma operacija gali būti priimta arba atmesta. Pirmu atveju

tokia operacija bus vykdoma, antru atveju vykdoma operacija bus atšaukta ir vietoj to vartotojui bus pateiktas klaidos pranešimas. Abi operacijos veikia kaip įvykiai, kurie yra trivialūs ir dėl to nebuvo įtraukti į diagramą [1].

1.3.2.3 Integruotos elgsenos esybės-ryšių modelis

Kitas labai svarbus modelis – integruotos elgsenos ERM (angl. *behavior integrated entity relationship – BIER*), kuris sujungia ERM su PN. Šiame modelyje realaus pasaulio objektų struktūra modeliuojama kaip ERM modelyje. Funkcinė BIER dalis leidžia modeliuoti realaus pasaulio įvykius ir jų laikiną tarpusavio priklausomybę, naudojant modifikuotą PN. Tai sujungiama su ER modelio dalimi per tiesioginius lankus, kurie jungia būsenos tipus su atitinkamais esybių tipais [15].

Esybės tipo UŽSAKYMAS pavyzdys pateikiamas septintame paveiksle. Ši diagrama dėmesį sutelkia į dinaminę elgseną. Vaizdžiausia vieta yra tarpinė tarp [VT6] ir [VT7], nes ji įtakoja būsenos pakeitimą; [VT3] ir [VT4] gali būti tiktai numanomos iš atvaizdavimo. Galimybės vaizduoti statines taisykles [VT1] ir [VT2A] arba [VT2B] BIER modelio ER dalyje yra lygiai tokios pačios, kaip aptartos kalbant apie esybių-ryšių modelį. [1]



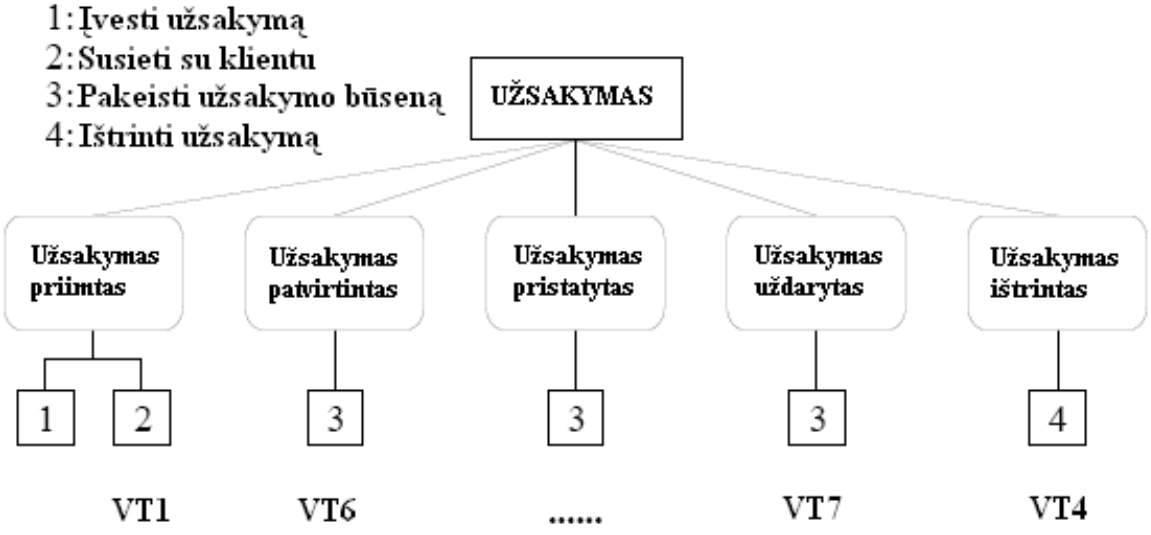
7 pav. Taisyklių aprašymas BIER modeliu [1]

Net jeigu BIER modelis leidžia modeliuot sistemos struktūrą ir sistemos elgseną, jis stokoja konstrukcijos, galinčios tiesiogiai išreikšti įvykius ir sąlygas. Nepatenkinamas taisyklių poveikio objektams vaizdavimas PN modelyje yra apeitas BIER modelyje, kadangi būsena susieta su esybės tipu. Dėl to funkcinė BIER dalis labai tinkama aprašant tarpines taisykles [15].

1.3.2.4 Esybės gyvavimo istorijos modelis

Sekantis modelis, kuris sutelkia dėmesį į esybių būsenos pokyčių išreiškimą – esybės gyvavimo istorijos (angl. *Entity Life History - ELH*) sąvoka. Ši technika naudojama duomenų rezultatų ir duomenų srautų modeliavimo sujungimui. Bazinė ELH idėja – lengvai aprašyti visus įvykius, susijusius su būsenos pokyčiais visame esybės gyvavimo cikle, t.y. nuo jos sukūrimo, galimų jos pakeitimų iki jos ištrynimo. Tai medžio struktūra su esybių tipais, vaizduojamais medžio šaknimis ir įvykiais, kurie įtakoja šiuos esybių tipus lapuose. Įvykiai simbolizuoja sekas, kurias reikia skaityti iš kairės į dešinę. Gali būti tarpinių mazgų tarp šaknų ir lapų, taip išreiškiant išrinkimą ir iteracijas [26].

Pateiktame pavyzdyje ELH modelis yra konstruojamas esybės tipams KLIENTAS ir UŽSAKYMAS. UŽSAKYMO gyvavimo istorija pateikiama aštuntame paveiksle. Užsakymas pradeda egzistuoti, kuomet priimamas užsakymo reikalavimas iš kliento. Užsakymo būsena yra „patvirtintas“, „pristatytas“ ir „uždarytas“. Taisyklės, išreiškiančios užsakymo būsenos pokyčius, yra [VT6] ir [VT7]. Tačiau šios taisyklės vykdomos vidinių įvykių, tokių kaip: „įterpti užsakymą“, tuo tarpu įvykiai diagramoje pradžioje yra išoriniai, tokie kaip: „užsakymas patvirtintas“. Vietoje vidinių įvykių diagramoje vaizduojamos akompanavimo operacijos. Vidinių įvykių perėjimas į išorinius veiksmus taip, kaip apibrėžta [VT6], gali būti specifikuotas tik tuo atveju, jeigu įvykis pakeičia objekto būseną. Kitos svarbios taisyklės, tokios kaip: [VT1] ir [VT3], skirtos užsakymo priėmimui, ir [VT5] skirta jo ištrynimui negali būti adekvačiai išreikštos [1].



8 pav. Verslo taisyklių aprašymas ELH modeliu [1]

EHL dėmesį sutelkia į objektus ir įvykius: sąlygos negali būti aprašytos, t.y. kiekvienas įvykis išreiškia besąlyginį prijungtų operacijų vykdymą. Iš principo, šis modelis gali išreikšti objekto būsenos perėjimo seką, kuri juda tiktai viena kryptimi, nuo įterpimo iki ištrynimo. Siekiant ELH modelyje aprašyti skirtingas sekas, padidinamas konstrukcijų skaičius ir dėl to modelis tampa mažiau suprantamas. Kitas ELH trūkumas yra apribojimas, leidžiantis modeliuoti tik vieną taisyklių

poveikį viename esybės tipe. Tačiau ELH laikomas lygiu ER diagramoms. Be to, atitinkamų technikų dėka ELH modelį galima papildyti, kad jis galėtų modeliuoti visus esybės tipus, paveiktus duoto įvykio.

Kadangi ELH remiasi medžio struktūra, o BIER naudoja tinklinį modelį, todėl BIER yra lankstesnis, išreiškiant kompleksinės būsenos sekas, pavyzdžiui, kilpas [26].

1.3.3. Objektiškai orientuoti metodai

Objektiškai orientuoti metodai (angl. *object-oriented methodologies – OOM*) siekia išspręsti skirtingas duomenų ir funkcijų modeliavimo silpnybes. OOM esmė yra objekto modeliai (objektai grupuojami į klases, jų valdymui įtraukiant atributus bei paslaugas). Objektai bendrauja pranešimais, kurie siunčiami vieno objekto, ir sužadina tam tikrų paslaugų vykdymą kitame objekte.

Semantinis ir grafinis objekto modelio žymėjimas panašus į išplėstą ERM modelį. Be to, verslo taisyklės gali būti suformuluotos pranešimų ir paslaugų terminais. Pranešimai gali būti interpretuoti kaip įvykiai, kurie sužadina paslaugas, šiuo atveju sąlygos ir veiksmai turi būti apibrėžti paslaugų detalizavimu.

Dauguma objektiškai orientuotos analizės ir projektavimo metodų, pavyzdžiui, objekto modeliavimo technika (angl. *object modeling technique – OMT*), be objekto modelio taip pat naudoja ir funkcinius bei dinaminius modelius, kurie dažniausia išreiškiami STD modeliais. OMT, visos sistemos modeliavimui naudoja paprastą įvykių srautų diagramą ir STD, kiekvienai klasei su svarbia elgsena modeliuoti. OMT taip pat siūlo funkcinį modelį, kuris paremtas įprasta DFD technika.

Atsižvelgiant į verslo taisykles, specialus dėmesys skiriamas objektiškai orientuotos analizės ir dizaino (angl. *object-oriented analysis & design*) metodui. Šis metodas pasižymi objekto struktūros ir paties objekto analize. Objekto struktūros modeliuojamos ERM, objektų elgsena aprašoma STD, įvykio schemomis, objektų srautų diagramomis ir procesų priklausomybės diagramomis.Verslo taisyklių vaizdavimui naudojama įvykių schema, kuri apima elementų operacijas, įvykių tipus ir trigerių taisykles.

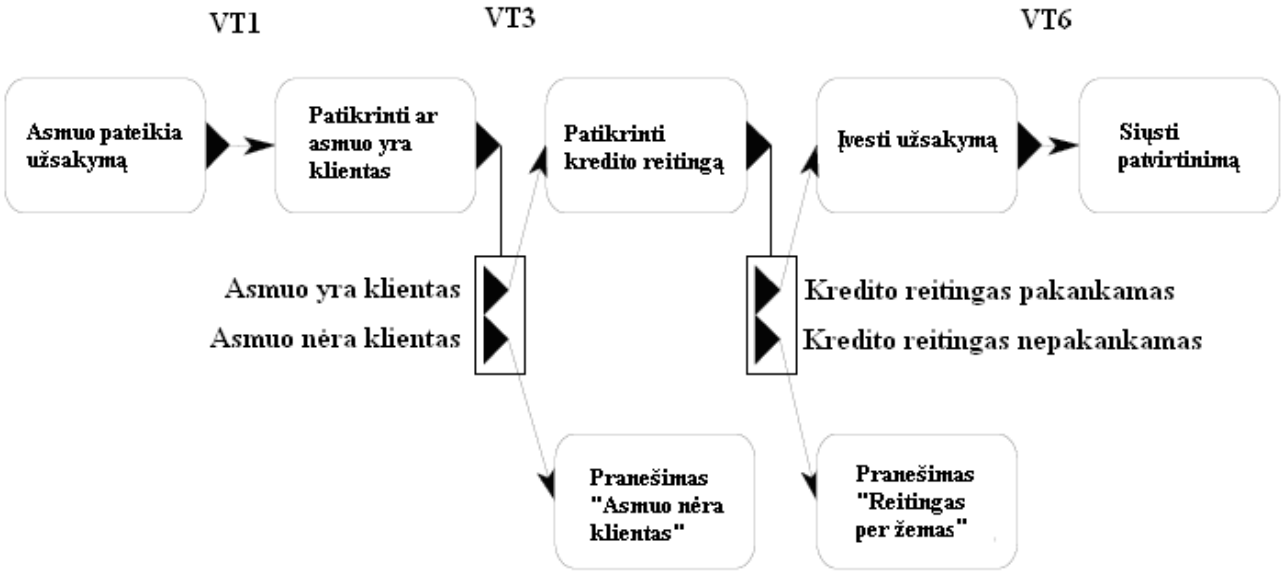
Operacija apibrėžiama kaip procesas, kuris gali būti aprašomas kaip elementas ir yra vaizduojamas kaip stačiakampis su užapvalintais kampais. Operacijos sėkminga baigtis apima vieną arba daugiau įvykių. Kiekvienas įvykis yra atskiras įvykio tipo atvejis: šie tipai vaizduojami trikampiais, kurie dažniausia prijungti prie operacijų stačiakampių. Vidiniai, išoriniai ir laikini įvykiai yra žymimi grafiškai, pavyzdžiui, laikinas įvykis išreiškiamas laikrodžio simboliu. Sąvoka „trigerio taisyklė“ labai panaši į sąvoką „verslo taisyklė“ ir išreiškia tam tikros operacijos, priklausančios specialiam įvykio tipui, reikalavimą. Tokiu būdu trigerio taisyklė įvykio schemoje susideda iš trijų bazinių komponentų:

- įvykio tipo (priežasties),

- operacijos (rezultato),
- žymėjimo tarp jų.

Priežasties ir rezultato ryšys vaizduojamas tiesioginiais lankais. Žymėjimas taip pat leidžia išreikšti nuoseklų ir lygiagretų kreipimąsi. Vykdymo sąlygos yra mechanizmas, kuris turi būti vykdomas taip, kad susijusios operacijos būtų vykdomos kartu; jos vaizduojamos rombo forma ir prijungtos prie operacijos stačiakampio. Valdymo sąlygos priklauso trigeriui, tokiu būdu gali būti keletas valdymo sąlygų, skirtų specifinių operacijų vykdymui. [10]

Naudojant įvykio schemas iš principo įmanoma išreikšti visas verslo taisykles. Devintas paveikslas vaizduoja [VT1], [VT3] ir [VT6] semantiką ir seką, pagal kurią taisyklės turi būti vykdomos. Šitas modelis panašus į CPM modelį pavaizduotą antrame paveiksle [1].



9 pav. Verslo taisyklių aprašymas įvykių schema [1]

1.3.4. Objekto vaidmens modeliavimas

Objekto vaidmens modeliavimas (angl. *Object-Role Modeling – ORM*) yra modeliavimo metodika, kuri pasaulį suvokia paprasčiausiai kaip objektus, atliekančius. ORM modelį sukurtą Europoje 1970-ųjų metų pradžioje ir kadaise žinomą NIAM vardu, laikui bėgant jis buvo žymiai išplėtotas Europos, Australijos ir Jungtinių Valstijų mokslininkų. Ši metodika yra svarbi verslo taisyklių pasaulyje, kadangi jos pagrindu galima aprašyti daugelį atributų ir esybių tarpusavio taisyklių [27].

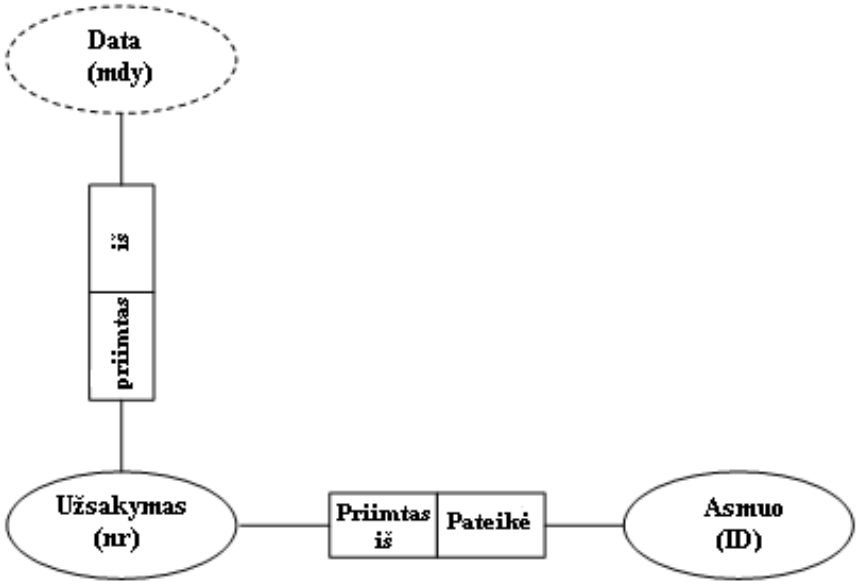
1.3.4.1 Esybių tipai, atributai ir ryšiai

ORM modelyje esybių tipai vaizduojami kaip paprastos elipsės ar apskritimai. Kaip ir esybių/ryšių (ER) diagramose, o esybių tipai yra susiję vieni su kitais per apibrėžtus ryšius. Kiekvienas ryšys susideda iš vieno vaidmens, skirto kiekvienam jo dalyviui (gali būti daugiau kaip dvi esybės, dalyvaujančios ryšyje).

ORM diagramos objektai yra esybių tipai ir reikšmių tipai. Esybių tipas yra organizacijai reikšmingas dalykas. Reikšmės tipas yra reikšmės klasifikavimas, panašus į domeną ER modeliavime. Pavyzdžiui, „piniginė suma“, „data“ ir t.t.

Esybių ir reikšmių tipai objekto vaidmens modeliavime yra vadinami objektais.

Iš esmės, faktas paprasčiausiai yra objekto atliekamas vaidmuo. Vaidmuo yra vienos krypties ryšio dalis. Du ar daugiau esybių tipų atlikdami vaidmenis, vienas kito atžvilgiu sudaro ryšį. 10 paveiksle užsakymas ir asmuo yra realaus pasaulio esybių tipai, o data yra reikšmės tipas (ištisinės elipsės nurodo esybių tipus, o punktyrinės elipsės nurodo reikšmių tipus). Kai reikšmės tipas atlieka identifikatoriaus funkciją (kaip asmens elipsėje „ID“ ir užsakyme „nr“), jis vaizduojamas elipsėje, kaip esybės tipo žymėjimas, nurodantis, kad jis yra unikalus identifikatorius. Reikšmių tipo atveju, po reikšmės pavadinimu nurodomas duomenų tipas (toks kaip „mdy“, reiškiantis „mėnuo-diena-metai“).



10 pav. Objekto vaidmens modelis [27]

Vaidmenų vardų sudarymui taisyklių nėra, išskyrus tai, kad juos būtų galima skaityti kaip sakinius. Šiame pavyzdyje (10 pav.) kiekvienas užsakymas yra priimtas iš asmens, ir kiekvienas asmuo pateikė užsakymą. Užsakymas taip pat atlieka vaidmenį su reikšmės tipu data [27].

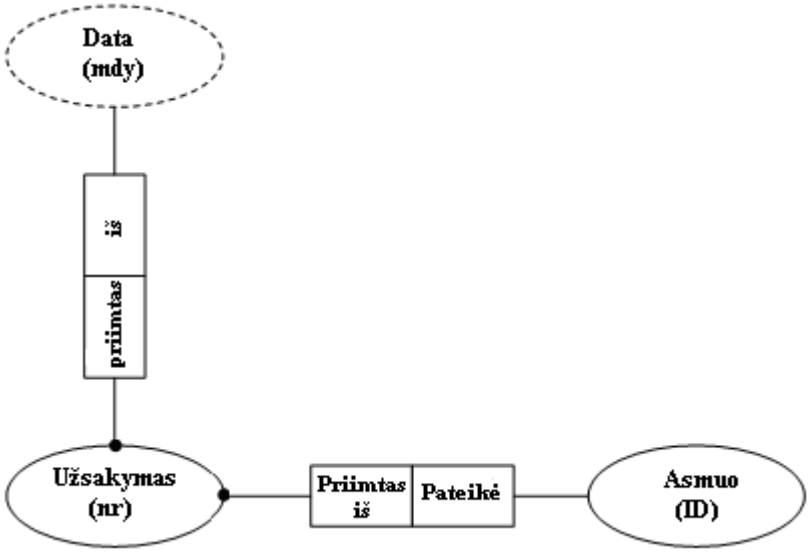
1.3.4.2 Ribojimai

ORM yra daug įvairiapusiškesnė nei kitos modeliavimo kalbos tuo, kad ji gali pavaizduoti verslo taisykles ir ribojimus. Tolimesni skyriai parodo, kaip tai vyksta. ORM, žinoma, nepavaizduoja visų įmanomų duomenų ribojimų, tačiau gali pavaizduoti didelę jų dalį [27].

1.3.4.3 Privalomi vaidmenys

Pirmiausiai, kaip ir ER modeliavimas, ORM turi galimybę pavaizduoti, kad vaidmuo yra privalomas. ORM modelyje tai vaizduoja užtušuotas taškas tarp objekto ir su juo susijusio

vaidmens. 11 paveiksle, vaidmuo iš užsakymo į asmenį yra privalomas. Kiekvienas užsakymas turi būti gautas iš vieno asmens. Tas pats simbolis tai pat naudojamas pažymėti, kad atributas yra privalomas. Paveiksle, kiekvienas užsakymas turi būti priimtas tam tikrą datą [27].

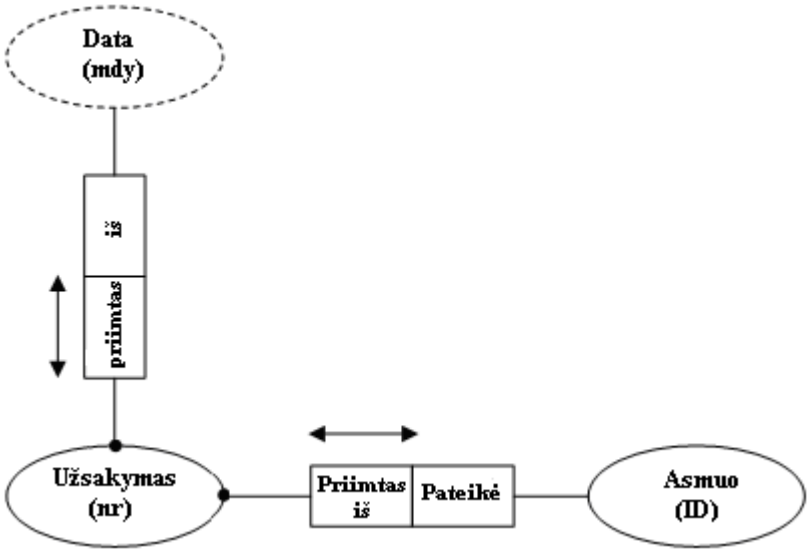


11 pav. Privalomos taisyklės [27]

1.3.4.4 Unikalumas

Sąvoka, nusakanti, ar vienas esybės tipo egzempliorius yra susiję su vienu esybės tipu ar daugybe jų, ER modeliavimo pasaulyje yra žinoma kaip "kardinalumas". Dažnai jis vaizduojamas „varnos kojos" arba žvaigždutės buvimu ar nebuvimu prie esybės tipo, kuris yra vienas arba daug.

ORM modelyje tai ne "Kiek kito esybės tipo egzempliorių yra šio vaidmens objektai?" Greičiau tai, "Kiek kartų ta pati esybė gali reikalauti atlikti šį vaidmenį (bet kokioje duomenų bazės būsenoje)?" Tai yra, "Ar gali objektas reikalauti atlikti šį *vaidmenį* tiek daug kartų?" 12 paveiksle, kiekvienas užsakymas turi vieną vaidmens priimtas iš asmens egzempliorių. Tai yra, kad kiekvienas užsakymas turi būti priimtas iš vieno ir tik vieno asmens. Tai vaizduojama horizontale rodykle virš "priimtas iš" vaidmens langelio. Vaidmuo „pateikė" asmuo – nėra unikalus. Čia nėra horizontalios rodyklės. Tai reiškia, kad asmuo daug kartų gali pateikti užsakymą [27].



12 pav. Unikalumas [27]

1.3.5. UML klasių diagrama

UML siūlo įvairias diagramas skirtas programų sistemų reikalavimų atvaizdavimui. Turbūt pati svarbiausia diagrama yra klasių diagrama, kuri atvaizduoja visus taikomosios programos pagrindinius struktūros aspektus: klases, ryšius, ISA hierarchijas tarp klasių, ryšių kardinalumo ribojimus [21].

Klasių diagramos tikslas yra pavaizduoti klases modelio viduje. Objektiškai orientuotoje programoje, klasės turi atributus (atstovo kintamuosius), operacijas (atstovo funkcijas) ir ryšius su kitomis klasėmis. UML klasių diagramoje visus šiuos tris dalykus galima atvaizduoti labai paprastai. Svarbiausias klasės diagramos elementas – stačiakampis atvaizduojantis klasę. Stačiakampis pavaizduotas 13 pav.

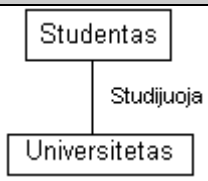
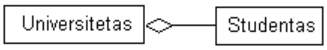
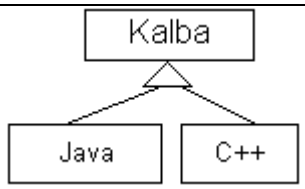
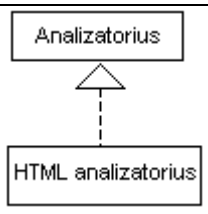
13 pav. Klasės stačiakampis

Klasių stačiakampis yra paprastas stačiakampis padalintas į tris sekcijas. Viršutinėje sekcijoje talpinamas klasės vardas. Vidurinėje sekcijoje – atributų sąrašas ir apatinėje sekcijoje – operacijų sąrašas. Daugelyje klasių diagramų vidurinė ir apatinė sekcijos neįtraukiamos. Net kuomet jos įtraukiamos, jos dažniausiai neparodo visų atributų ir operacijų. Tikslas yra parodyti tik tuos atributus ir operacijas, kurie yra naudingi konkrečiai diagramai [22].

1.3.5.1 Ryšiai tarp klasių

Akivaizdu, kad klasių diagramoje negali būti nesusietų klasių, t.y. visoms klasės reikia nustatyti ryšius su kitomis klasėmis. Žemiau pateiktoje lentelėje (1 lentelė) pateikiamos visos galimų ryšių rūšys, ryšių žymėjimai ir aprašoma ką jie reiškia [23].

1 lentelė. Ryšiai tarp klasių

Eil. Nr.	**Ryšys**	**Žymėjimas**	**Aprašymas**
1	Asociacija (*association*)		Kuomet dvi klasės yra sujungtos viena su kita bet kokiu būdu, nustatomas asociacijos ryšys. Pavyzdžiui, „studentas studijuoja universitete“.
1a.	Kardinalumas (*multiplicity*)		Šios rūšies asociacijos pavyzdys: „daug studentų priklauso vienam universitetui“. Tokiu atveju žvaigždės (*) simbolis pateikiamas prie studento klasės.
1b.	Nurodomoji asociacija (*directed association*)		Pagal nutylėjimą asociacija tarp klasių yra dvikryptė. Galima nustatyti asociacijos kryptį naudojant nurodomąją asociaciją.
1c.	Refleksyvi asociacija (*reflexive association*)	Nėra atskiro simbolio. Ryšys grįžta į tą pačią klasę.	Šios rūšies ryšio pavyzdys yra tuomet, kai klasė turi daugybę atsakomybių. Pavyzdžiui, universiteto tarnautojas gali būti profesorius, ūkvedys ir administracijos padėjėjas.
2	Agregacija (*aggregation*)		Kuomet klasė yra suformuluota kaip kolekcija kitos klasės, tai vadinama agregacijos ryšiu tarp klasių. Tai taip pat vadinama „**has a**“ ryšiu.
2a.	Kompozicija (*composition*)		Kompozicija yra agregacijos ryšio variacija. Kompozicija pabrėžia, kad stiprus gyvavimo ciklas yra tarp dviejų klasių.
3	Apibendrinimas (*generalization*)		Dar vadinamas „**is a**“ ryšys, kadangi vaiko klasė **yra** tėvo klasės tipas. Apibendrinimas yra ryšio pagrindinis tipas naudojamas apibrėžti naudojamus elementus klasių diagramoje. Vaiko klasės paveldi pagrindines funkcijas apibrėžtas tėvo klasėje.
4	Realizacija (*realization*)		Realizacijos ryšyje viena esybė apibrėžia funkcijų aibę kaip kontraktą, o kita esybė realizuoja kontraktą įgyvendindama funkcionalumą apibrėžtą kontrakte.

1.3.5.2 Kardinalumo tipai

Asociacijos ryšys nurodo, kad (bent) viena iš dviejų susijusių klasių daro nuorodą į kitą. Lyginant su apibendrinimo ryšiu, šis būdas yra lengviausiai suprantamas (studentas studijuoja universitete, universitete studijuoja studentas).

UML asociacijos atvaizdavimas žymimas linija su nebūtina strėlyte, nurodančia objekto vaidmenį ryšyje ir nebūtinu žymėjimu kiekvienos linijos pabaigoje, nurodančiu esybės egzempliorių kardinalumą (objektų kiekį, kurie dalyvauja asociacijoje). Kardinalumo tipai pateikiami 2 lentelėje [24].

2 lentelė. Kardinalumo tipai

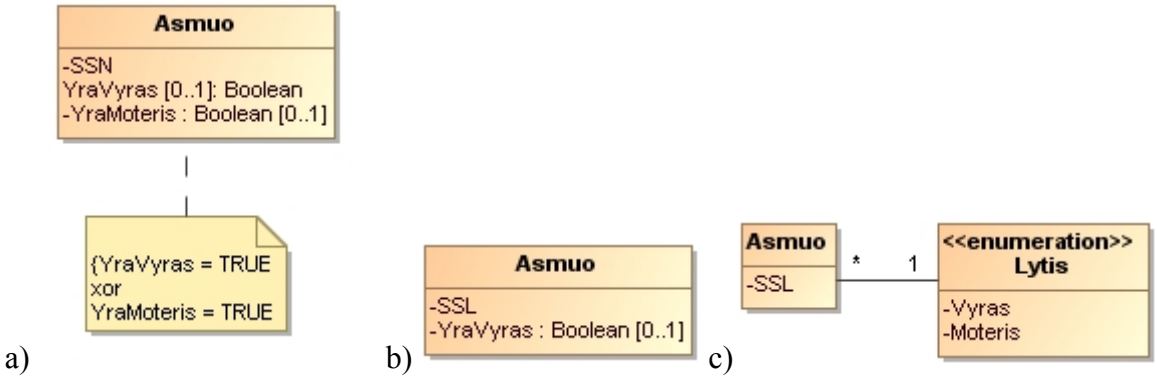
Žymėjimas	Aprašymas
0..1	Nei vieno egzemplioriaus arba vienas
1	Tiktai vienas egzempliorius
0..* arba *	Nulis arba daugiau egzempliorių
1..*	Vienas arba daugiau egzempliorių (bent vienas)

1.3.5.3 Verslo taisyklių atvaizdavimas

UML klasių diagramoje verslo taisyklės atvaizduojamos OCL (angl. *object constraint language*) kalbos pagalba. Detalesnis verslo taisyklių atvaizdavimas OCL kalba bus aptartas 1.3.6 skyrelyje. Šiame skyrelyje pateiksime tik vieną pavyzdį kaip verslo taisyklė gali būti realizuota UML klasių diagramoje (14 pav.). Turime tokį ribojimą:

Kiekvienas asmuo yra vyras **arba** moteris **bet ne abu**
Kiekvienam asmeniui yra **teisingas tik vienas iš sekančių teiginių**:
kad asmuo yra vyras
kad asmuo yra moteris.

UML klasių diagramoje šis ribojimas gali būti atvaizduotas trimis būdais (14 pav.) [25].



14 pav. Ribojimo modeliavimo būdai UML kalboje [25]

UML klasių diagramos nėra pakankamai detalios ir negali atspindėti visų verslo objekto aspektų. Siekiant išspręsti šią problemą, į UML yra įtraukta objektų ribojimo kalba OCL (žr. 1.3.6 skyrius).

1.3.6. Verslo taisyklių atvaizdavimas OCL kalba

Šiame skyriuje aprašoma OCL kalba, kuri leidžia formalizuoti tokias verslo taisykles, kurios negali būti aprašytos naudojant tik UML klasių diagramos žymėjimus.

Objektų ribojimų kalba OCL – formali kalba, naudojama aprašyti išraiškas (angl. *expressions*) UML modeliuose. Šios išraiškos tipiškai apibrėžia invariantines (angl. *invariant*) sąlygas, kurios turi būti taikomos projektuojamoje sistemoje arba užklausas objektams aprašytiems modelyje [13].

Reikia pažymėti, kad kuomet OCL išraiškos yra išreiškiamos, jos neturi šalutinio poveikio (t.y. jų išreiškimas negali pakeisti atitinkamos sistemos būsenos). OCL išraiškos gali būti naudojamos apibrėžti operacijas arba veiksmus, kurie, kai bus įvykdyti, pakeis sistemos būseną. UML modeliuotojai gali naudoti OCL, specifinių programos ribojimų apibrėžimui savo modelyje. UML modeliuotojai taip pat gali naudoti OCL, užklausų, kurios yra visiškai programiškai nepriklausomos, apibrėžimui UML modelyje [20].

1.3.6.1 Kodėl OCL?

UML diagrama, tokia kaip klasių diagrama, tipiškai nėra tiek tobula, kad galėtų atvaizduoti visus tiesioginius specifikacijos aspektus. Be kitų dalykų, egzistuoja poreikis aprašyti papildomus ribojimus susijusius su objektais esančiais modelyje. Šie ribojimai dažniausiai aprašomi natūralia kalba. Praktika rodo, kad tai visuomet sukuria dviprasmybes. Siekiant parašyti nedviprasmiškus ribojimus, buvo sukurta taip vadinama formali kalba. Tradicinių formalių kalbų trūkumas yra tas, kad jos dažniausiai yra naudingos tik asmenims su stipriomis matematinėmis žiniomis, bet sudėtingos vidutiniam verslo ar sistemos modeliuotojui.

OCL buvo sukurtas užpildyti šią spragą. Tai yra formali kalba, kurią paprasta skaityti ir rašyti. Ji buvo sukurta kaip verslo modeliavimo kalba IBM viduje ir paremta sintropijos metodu.

OCL yra gryna specifikacijos kalba, todėl OCL išraiškos užtikrina šalutinio poveikio nebuvimą. Kuomet OCL išraiškos išreiškiamos, jos lengvai grąžina reikšmę. Jos negali pakeisti modelio. Tai reiškia, kad sistemos būsena niekuomet nepasikeis dėl OCL išraiškos išreiškimo, net jeigu OCL išraiška naudojama būsenos pakeitimo apibrėžimui (pvz., po sąlygoje).

OCL nėra programavimo kalba, todėl neįmanoma parašyti programos loginei ar srautų OCL kontrolei. Negalima taikyti procesų ar aktyvų neužklausinių operacijų OCL viduje. Kadangi OCL pirmoje vietoje yra modeliavimo kalba, OCL išraiška pagal apibrėžimą nėra tiesiogiai vykdomas.

OCL yra tipų kalba, todėl kiekviena OCL išraiška turi tipą. Norint gerai suformuoti OCL išraišką, ją reikia pritaikyti prie tipo atitikimo kalbos taisyklių. Pavyzdžiui, negalima suderinti sveikųjų skaičių (angl. *Integer*) tipo su tekstiniu (angl. *String*). Kiekvienas klasifikatorius, apibrėžtas UML modelyje, vaizduoja skirtingą OCL tipą.

Kadangi OCL yra specifikacijos kalba, visos įgyvendinimo problemos nėra nagrinėjamos ir negali būti išreikštos OCL kalba. OCL išraiškos analizė yra momentinė. Tai reiškia, kad objekto būsena modelyje negali pasikeisti analizės metu [20].

1.3.6.2 Kur naudoti OCL?

OCL gali būti naudojama:

1. kaip užklausų kalba,
2. invariantų nustatymui klasėse ir kaip tipas klasių modelyje,
3. tipų invariantų nustatymui stereotipuose,
4. prieš ir po sąlygų aprašymui operacijose ir metoduose,
5. saugiklių aprašymui,
6. pranešimų ir veiksmų užduočių apibrėžimui,
7. operacijų ribojimų apibrėžimui,
8. atributų išvedimo taisyklių apibrėžimui kiekvienai išraiškai visame UML modelyje [20].

1.3.6.3 OCL sintaksės aprašymas

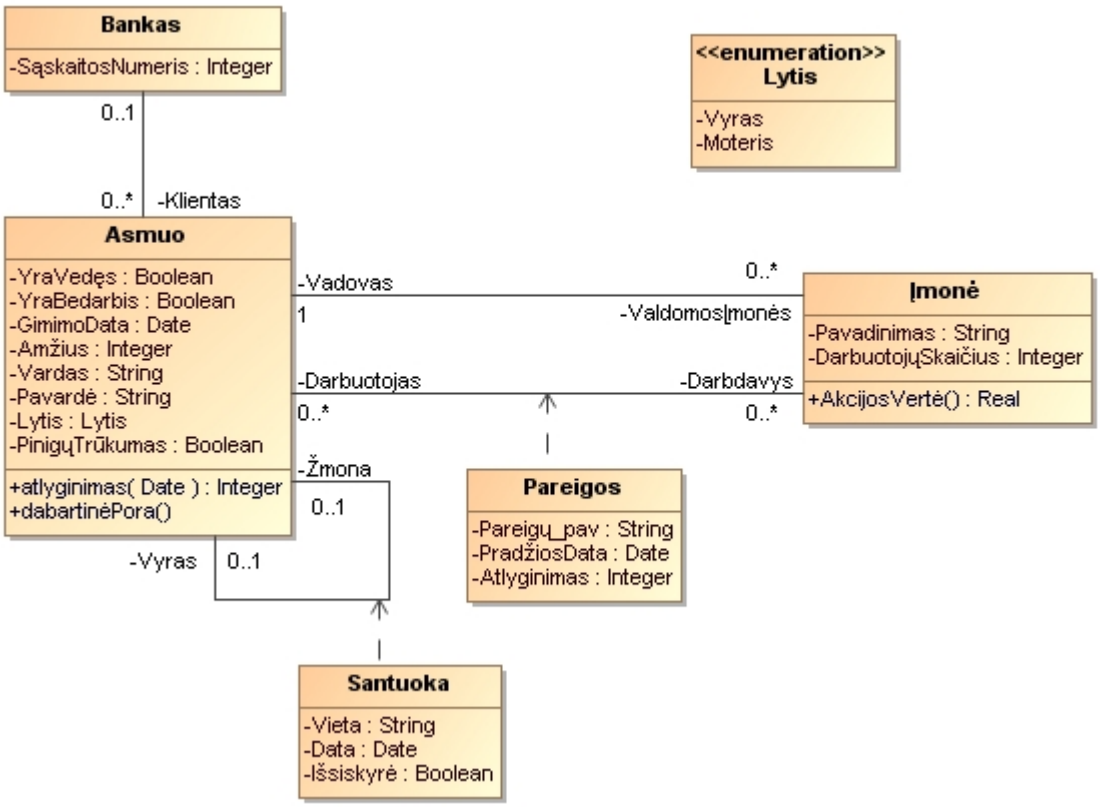
Turinio (angl. *context*) reikšminis žodis pateikia išraiškos turinį. Reikšminiai žodžiai inv, pre ir post nurodo ribojimų stereotipą, atitinkamai «invariant», «precondition» ir «postcondition». Tikras OCL išraiška eina po dvitaškio:

```
context TipoVardas inv:
    'ši OCL išraiška su stereotipu <<invariant>> TipoVardo turinyje' = 'Kitas tekstas'
```

Šio darbo pavyzdžiuose reikšminiai OCL žodžiai yra parašyti paryškintai. Paryškinimas neturi jokios formalios prasmės, bet jis naudojamas siekiant išraišką padaryti lengviau skaitomą. OCL išraiška rašoma tiktai ASCII simboliais.

Žodžiai pagrindiniame tekste pasvirusiu šriftu nurodo OCL išraiškos sudedamąsias dalis.

Žemiau (15 pav.) pateikiamas klasių diagramos pavyzdys, kuris bus naudojamas visos OCL kalbos analizės metu [20].



15 pav. Klasės diagramos pavyzdys [20]

1.3.6.4 OCL ryšys su UML metamodeliu

Kiekviena OCL išraiška aprašyta specifinio tipo egzemplioriaus kontekste. OCL išraiškoje rezervinis žodis pats (angl. *self*) naudojamas nurodyti konteksto egzemplioriui. Pavyzdžiui, jeigu kontekstas yra kompanija, tuomet pats nurodo įmonės egzempliorių.

UML modelyje OCL išraiškos kontekstas gali būti specifikuotas OCL išraiškos pradžioje per vadinamąją deklaraciją. Jeigu ribojimas yra parodytas diagramoje, su tinkamu stereotipu ir brūkšniuota linija jungiančia jį su konteksto elementu, tuomet, testuojant ribojimą nereikalinga detali konteksto deklaracija. Konteksto deklaracija nėra būtina.

OCL išraiška gali būti invarianto, kuris yra ribojimas, dalis apibrėžiama «invariant» stereotipu. Kuomet invariantas yra susietas su klasifikatoriumi, tai yra vadinama tipu. OCL išraiška yra tipo invariantas ir turi būti teisingas visiems tipo egzemplioriams, bet kuriuo laiko momentu. Visos OCL išraiškos, kurios išreiškia invariantus, turi būti loginio (angl. *boolean*) tipo.

Pavyzdžiui, įmonės tipo kontekste (15 pav.), išraiška gali apibrėžti invariantą, kad darbuotojų skaičius visada turi būti didesnis negu 50:

self.DarbuotojųSkaičius > 50,

kur pats yra įmonės egzempliorius. (Galima pats vaizduoti kaip objektą nuo kurio pradedama išreikšti išraišką). Šis invariantas taikomas kiekvienam įmonės tipo egzemplioriui.

OCL išraiškos, kuri yra invarianto dalis, konteksto egzemplioriaus tipas yra rašomas su turinio (angl. *context*) reikšminiu žodžiu ir su tipo vardu. Reikšmė inv: nurodo, kad ribojimas bus «invariant» tipo ribojimas.

```
context Įmonė inv:
    self.DarbuotojųSkaičius > 50
```

Daugeliu atvejų, reikšminis žodis pats gali būti nenaudojamas, kadangi kontekstas yra aiškus, kaip aukščiau pateiktame pavyzdyje. Kaip pats alternatyva, kitas vardas, vaidinantis pats vaidmenį, gali būti apibrėžtas. Pavyzdžiui:

```
context i : Įmonė inv:
    i.DarbuotojųSkaičius > 50
```

Šis invariantas yra ekvivalentus prieš tai buvusiam.

Ribojimo vardas gali būti užrašytas po inv reikšminio žodžio, tokiu būdu apibrėžiant ribojimui vardą. Sekančiame pavyzdyje ribojimo vardas yra PakankamaiDarbuotojų. UML 1.4 metamodelyje, šis vardas yra metaklasės ribojimas (angl. *constraint*):

```
context i : Įmonė inv PakankamaiDarbuotojų:
    i.DarbuotojųSkaičius > 50 [20]
```

1.3.6.5 Prieš ir po sąlygos

OCL išraiška gali būti prieš arba po sąlygų dalis, atitinkamai ribojimo «precondition» ir «postcondition» stereotipai susieti su operacijos (angl. *operation*) arba kitomis elgesio savybėmis. Konteksto egzempliorius pats tuomet yra tipo, kuris valdo operaciją ar metodą kaip savybę, egzempliorius. Turinio deklaracija OCL kalboje naudoja turinio (angl. *context*) reikšminį žodį su tipo ir operacijos deklaracija. Pridedama pre: ir post: prieš tam tikrą prieš arba po sąlygą. Pavyzdžiui:

```
context TipoVardas::OperacijosVardas(parametras1 : tipas1, ... ): GrąžinamasTipas
    pre : parametras1 > ...
    post: result = ...
```

Vardas pats gali būti naudojamas išraiškoje nurodant objektą, kuriam operacija yra kviečiama. Rezervinis žodis rezultatas (angl. *result*) nurodo operacijos rezultatą, jei toks yra. Parametrų vardai (pavyzdžiui, param1) gali tai pat būti naudojami OCL išraiškose. Nagrinėjamai diagramai, galima parašyti:

```
context Asmuo::atlyginimas(D : Data) : Integer
    post: result = 5000
```

Prieš arba po sąlygos vardas gali būti užrašytas po pre arba post reikšminių žodžių, tokiu būdu ribojimui suteikiant vardą. Sekančiame pavyzdyje prieš sąlygos vardas yra parametrasGerai (angl. *parameterOK*) ir po sąlygos vardas – rezultatasGerai (angl. *resultOK*). UML metamodelyje šie vardai yra metaklasės ribojimas (angl. *constraint*), kuris yra paveldėtas iš modelioElemento (angl. *modelElement*), atributo vardas (angl. *name*) reikšmės.

```
context TipoVardas::OperacijosVardas(parametras1 : Tipas1, ... ): GrąžinamasTipas
    pre parametrasGerai: parametras1 > ...
```

post rezultatasGerai : result = ... [20]

1.3.6.6 Operacijos užklausos išraiška

OCL išraiška gali būti naudojama užklausos operacijos rezultatų nustatymui. Tai gali būti atlikta naudojant tokią sintaksę:

context TipoVardas::OperacijosVardas(parametras1 : Tipas1, ...): GrąžinamasTipas
body: -- tam tikra išraiška

Kaip ir prieš ir po sąlygose, parametrai gali būti naudojami išraiškose. Prieš ir po sąlygos bei operacijos užklausos išraiškos gali būti naudojamos kartu po operacijos konteksto sakiniu. Pavyzdžiui:

context Asmuo::dabartinėPora() : Asmuo
pre: self.YraVedęs = true
body: self.Santuoka->select(s | s.Išsiskyrė = false) [20]

1.3.6.7 Pradinės ir gautos reikšmės

OCL išraiška gali būti naudojama atributų arba asociacijos (angl. *association*) pabaigos (angl. *end*), pradinių (angl. *initial*) arba gautų (angl. *derived*) reikšmių parodymui. Tai gali būti atlikta naudojant tokią sintaksę.

context TipoVardas::AtributoVardas: Tipas
init: -- tam tikra išraiška, nurodanti pradinę reikšmę

context TipoVardas::AsociacijosVardas: Tipas
derive: -- tam tikra išraiška, nurodanti išvedimo reikšmę

Išraiška turi atitikti atributo rezultato tipą. Tuo atveju, kuomet kardinalumas (angl. *multiplicity*) yra vienas, išraiška turi atitikti klasifikatorių pabaigoje, arba aibė (angl. *set*) ir išrūšiuota aibė (angl. *OrderedSet*), kuomet kardinalumas daugiau nei vienas. Pradinė ir išvesta išraiška gali būti naudojamos kartu viename kontekste. Pavyzdžiui:

context Asmuo::atlyginimas : Integer
init: Asmuo.tlyginimas->sum() * 1%
derive: if PinigųTrūkumas
then Asmuo.atlyginimas->sum() * 1%
else Pareigos.Atlyginimas – atlyginimas iš reguliatrių pareigų
endif [20]

1.3.6.8 Pagrindinės reikšmės ir tipai

OCL kalboje yra apibrėžti pagrindiniai tipai ir jie yra visada prieinami modeliuotojui. Šie apibrėžti reikšmių tipai yra nepriklausomi nuo bet kokio objekto modelio ir yra OCL apibrėžimo dalis.

Pagrindinės reikšmės OCL kalboje yra vieno iš pagrindinių tipų reikšmės. Pagrindiniai OCL tipai su atitinkamų reikšmių pavyzdžiais pateikti 3 lentelėje.

3 lentelė. Baziniai OCL tipai ir jų reikšmės [20]

Tipas	Reikšmė
Boolean	True, false
Integer	1, -5, 2, 34, 26524, ...
Real	1.5, 3.14, ...
String	‘to be or not to be‘, ...

OCL apibrėžia kokios operacijos gali būti vykdomos su apibrėžtais tipais. Lentelėje 4 pateikiama keletas operacijų pavyzdžių [20].

4 lentelė. Operacijos su duomenų tipais [20]

Tipas	Operacija
Boolean	and, or, xor, not, implies, if-then-else
Integer	*,+,-,/,abs()
Real	*,+,-,/,floor()
String	concat(), size(), substring()

1.3.6.9 Tipai iš UML modelio

Kiekviena OCL išraiška yra parašyta UML modelio, klasifikatorių, jų savybių ir asociacijų, bei jų apibendrinimų (angl. *generalizations*) kontekste. Visi klasifikatoriai iš UML modelio yra tipai OCL išraiškose, kurie yra prijungti prie modelio [20].

1.3.6.10 Išvardinimo tipai

Išvardinimai (angl. *enumeration*) UML kalboje yra duomenų tipai, jie turi vardą, kaip ir bet koks kitas klasifikatorius. Išvardinimas apibrėžia tam tikrą skaičių išvardinimo literalų, kurie yra tikėtinos išvardinimo reikšmės. OCL kalboje galima nurodyti į išvardinimo reikšmę. Kuomet turimas duomenų tipas pavadintas Lytis (15 pav.) su reikšmėmis „vyras” ir „moteris“, OCL kalboje jos gali būti naudojamos taip:

```
context Asmuo inv:
Lytis = Lytis::Vyras
```
[20]

1.3.6.11 Apibrėžimo išraiškos

Kartais subišraiška ribojime yra naudojama daugiau negu kartą. Apibrėžimo išraiškos leidžia vieną kartą apibrėžti kintamąjį, kuris gali būti naudojamas ribojime.

```
context Asmuo inv:
    let atlyginimas : Integer = self.Pareigos.Atlyginimas->sum() in
    if yraBedarbis then
    atlyginimas < 100
    else
    atlyginimas >= 100
    endif
```

Apibrėžimo išraiška gali būtų naudojamas bet kurioje OCL išraiškos rūšyje. Ji (apibrėžimo išraiška) yra žinoma tik specifinės išraiškos viduje [20].

1.3.6.12 Papildomos operacijos (atributai) «definition» išraiškose

Apibrėžimo išraiška suteikia galimybę naudoti kintamąjį vienoje OCL išraiškoje. Norint leisti naudoti kintamuosius (operacijas) daugelyje OCL išraiškų galima naudoti ribojimą su stereotipu «definition», kuriame apibrėžiami pagalbiniai kintamieji (operacijos). Šis «definition» ribojimas turi būti prijungtas prie klasifikatoriaus ir gali talpinti tik kintamųjų ir/arba operacijų apibrėžimus ir nieko daugiau. Visi kintamieji ir operacijos, apibrėžti «definition» ribojime, yra žinomi tokiame kontekste kaip: „bet kokia klasifikatoriaus savybė gali būti naudojama“. Tokie kintamieji ir operacijos yra atributai ir operacijos su klasifikatoriaus atributu «OclHelper». Jie yra naudojami OCL išraiškoje tiksliai tokiu pat būdu kaip yra naudojami įprasti atributai ar operacijos. Atributų arba operacijų sintaksė yra tokia pati kaip apibrėžimo išraiškos, bet kiekvienas atributo ir operacijos apibrėžimas yra naudojamas su reikšminiu žodžiu def, kaip tai pavaizduota žemiau.

```
context Asmuo
    def: atlyginimas : Integer = self.Pareigos.Atlyginimas->sum()
    def: slapyvardis : String = 'Magistrinis darbas'
    def: turiPavadinimą(t : String) : Boolean = self.Pareigos->exists(Pavadinimas = t)
```

Atributų (operacijų) vardai apibrėžimo išraiškoje negali konfliktuoti su atitinkamais klasifikatoriaus atributų ir operacijų vardais.

Tokia apibrėžimo sintaksę yra identiška atributų (operacijų) sintaksei apibrėžiamai UML kalboje su stereotipu «OclHelper» ir prijungtu OCL ribojimu [20].

1.3.6.13 Tipų atitikimas

OCL yra tipų kalba ir iš pagrindinių reikšmių tipų yra sudaroma tipų hierarchija. Ši hierarchija apibrėžia skirtingų tipų atitikimą vienas kitam. Pavyzdžiui, negalima palyginti sveikųjų skaičių (angl. *integer*) tipo su loginiu (angl. *boolean*) arba tekstiniu (angl. *string*).

OCL išraiška, kurioje visi tipai dera yra teisinga išraiška. OCL išraiška, kurioje tipai nedera – neteisinga išraiška. Ji talpina tipo atitikimo klaidą. Tipas tipas1 dera su tipu tipas2 kuomet tipas1 egzempliorius gali būti panaudotas toje pačioje vietoje, kur ir tipas2 egzempliorius. Tipų atitikimo taisyklės klasių diagramoje yra paprastos:

- kiekvienas tipas atitinka savo supertipą,
- tipų atitikimas yra tranzityvus: jeigu tipas1 atitinka tipas2 ir tipas2 atitinka tipas3, tuomet tipas1 atitinka tipas3.

To pasekmė yra tai, kad tipas atitinka savo supertipui ir visiems supertipams esantiems aukščiau. Tipų atitikimo taisyklės tipams iš OCL standartų bibliotekos pateikiamos 5 lentelėje.

5 lentelė. Tipų atitikimo taisyklės [20]

Tipas	**Atitinka/yra pakaitalas**	**Sąlyga**
Set(T1)	Collection(T2)	Jei T1 atitinka T2
Sequence(T1)	Collection(T2)	Jei T1 atitinka T2
Bag(T1)	Collection(T2)	Jei T1 atitinka T2
Integer	Real	

Atitikimas ryšių tarp kolekcijos (angl. *collection*) tipų išlaikomas jeigu jis yra elementų tipų, kurie atitinka vienas kitą, kolekcija.

6 lentelėje pateikiami teisingų ir neteisingų išraiškų pavyzdžiai [20].

6 lentelė. Teisingos ir neteisingos išraiškos [20]

OCL išraiška	Teisingas?	Paaiškinimas
1+2*34	Taip	
1+‘motociklas‘	Ne	Tipas *String* neatitinka tipo *Integer*
23*false	Ne	Tipas *Boolean* neatitinka tipo *Integer*
12+13.5	Taip	

1.3.6.14 Pirmumo taisyklės

Operacijų pirmumo išrikiavimas OCL kalboje, nuo aukščiausios iki žemiausios:

1. @pre,
2. taško ir strėlės operacijos: „.“ ir „->“,
3. unary „not“ ir unary minus,
4. „*“ ir „/“,
5. „+“ ir „-„„
6. „if-then-else-endif“,
7. „<“, „>“, „<=“, „>=“,
8. „=“, „<>“,
9. „and“, „or“ ir „xor“,
10. „implies“.

Gali būti naudojami skliausteliai „(“ ir „)“ norint pakeisti pirmumą [20].

1.3.6.15 Infiksinių operatorių naudojimas

OCL kalboje yra leidžiamas infiksinių operatorių naudojimas. Operatoriai „+“, „–“, „*“, „/“„„<“, „>“, „<=“, „>=“ yra naudojami kai infiksiniai operatoriai. Jeigu tipas apibrėžia vieną iš šitų operatorių su teisingu žymėjimu, jie bus naudojami kaip infiksiniai operatoriai. Išraiška:

```
a + b
```

yra konceptualiai lygus išraiškai:

```
a.+(b)
```

taip yra naudojant „+“ operaciją tarp a ir b.

Infiksiniai operatoriai naudojami tipui apibrėžti, turi turėti tiksliai vieną parametrą. Infiksiniams operatoriams „<“, „>“, „<=“, „>=“, „=“, „<>“, „and“, „or“ ir „xor“ grąžinamas tipas turi būti loginis [20].

1.3.6.16 Reikšminiai žodžiai

Reikšminiai žodžiai OCL kalboje yra rezerviniai žodžiai. Tai reiškia, kad reikšminiai žodžiai negali atsirasti iš bet kur OCL išraiškoje, (pvz., paketo vardas, tipas, ar savybė). Reikšminių žodžių sąrašas:

and
attr
context
def
else
endif
endpackage
if
in
inv
let
not
oper
or
package
post
pre
then
xor [20]

1.3.6.17 Objektai ir savybės

OCL išraiškose gali nurodyti klasifikatorius (angl. *classifiers*), pavyzdžiui, tipus, klases, sąsajas, asociacijas ir duomenų tipus. Taip pat gali būti naudojami visi atributai, asociacijų pabaigos, metodai ir operacijos be šalutinio poveikio, kurie apibrėžiami šiais tipais ir pan. Šiame darbe atributai, asociacijų pabaigos, metodai be šalutinio poveikio ir operacijos laikomi savybėmis. Savybe laikoma:

- atributas,
- asociacijos pabaiga (angl. *association end*),
- operacija su teisinga isQuery,
- metodas su teisinga isQuery.

Savybės reikšmė objekte, kuris apibrėžtas klasių diagramoje yra nusakoma OCL išraiškoje su tašku ir po jo einančiu savybės vardu. Pavyzdžiui:

```
context Asmuo inv:
    self.yraVedęs
```

Jeigu pats yra nuoroda į objektą, tuomet self.turtas yra pats savybės turtas reikšmė [20].

1.3.6.18 Savybės: atributai

Pavyzdžiui asmens amžius yra užrašomas kaip self.Amžius:

```
context Asmuo inv:
    self.Amžius > 0
```

Subišraiškos self.Amžius reikšmė yra amžiaus atributo reikšmė tam tikrame asmens egzemplioriuje identifikuojamame pats. Šio subišraiškos tipas yra atributo amžius tipas, kuris yra standartinio sveikųjų skaičių tipo.

Naudojant atributus ir operacijas apibrėžtus pagrindiniais reikšmės tipais, galima išreikšti skaičiavimus ir kt. visame klasių modelyje. Pavyzdžiui, verslo taisyklė gali būti „Asmens amžius visada didesnis negu nulis“. Ši taisyklė gali būti aprašyta invariantu pateiktu anksčiau.

Atributai gali turėti kardinalumą (angl. *multiplicities*) UML modelyje. Kuomet atributų kardinalumas yra didesnis nei vienas, rezultato tipas yra reikšmių kolekcija. Kolekcijos OCL kalboje aprašomos žemiau esančiame skyriuje [20].

1.3.6.19 Apibrėžimo operacijos

Operacijos pačios gali būti apibrėžtos po sąlygų ribojimų. Tai yra ribojimas, kuris apibrėžiamas «postcondition» stereotipu. Objektas, kuris grąžinamas operacijos gali būti nurodomas rezultato. Tai užrašoma tokia forma:

```
context Asmuo::atlyginimas (D: Data) : Integer
    post: result = Amžius * 1000
```

Dešinėje šio apibrėžimo pusėje reikia nurodyti kokia operacija apibrėžiama (t.y. apibrėžimas gali būti grįžtamasis) kuomet grįžtamasis ryšys nėra begalinis. Viduje prieš arba po sąlygos galima taip pat naudoti operacijų parametrus. Rezultato (angl. *result*) tipas, kuomet operacija neturi perduodamų (angl. *in*) ir grąžinamų (angl. *out*) parametrų, yra operacijos grąžinimo tipas, kuris yra sveikas skaičius, anksčiau pateiktame pavyzdyje. Kuomet operacija turi perduodamus arba grąžinamus parametrus, rezultato tipas yra kortežas. Atlyginimas operacijai po sąlyga su grąžinamu parametru priedas bus tokios formos:

```
context Asmuo::atlyginimas (D: Data, Priedas: Integer) : Integer
    post: result = Tuple { priedas = ..., result = .... }
```

Siekiant parodyti operaciją ar metodą, kuris nenaudoja parametro, būtini skliausteliai su tuščiu argumentų sąrašu:

```
context Įmonė inv:
    self.akcijosVertė() > 0 [20]
```

1.3.6.20 Savybės: asociacijų pabaiga ir navigacija

Klasių diagramoje galima naudoti asociacijas, siekiant nurodyti kitus objektus ir jų savybes. Tam atlikti, nurodome asociacijas naudodami priešingybę – asociacijos pabaigą:

```
object.asociacijosPabaigosVardas
```

Šios išraiškos reikšmė yra objektų aibė, kitoje asociacijos pabaigos vardo asociacijos pusėje. Jeigu asociacijos pabaigos kardinalumas turi vieną iš reikšmių „0..1“ arba „1“, tuomet šios išraiškos reikšmė yra objektas. Naudojamame pavyzdyje (15 pav.), kuomet nagrinėjamas Įmonės kontekstas (t.y. *pats* yra Įmonės egzempliorius), jį galima užrašyti taip:

```
context Įmonė
    inv: self.Vadovas.YraBedarbis = false
    inv: self.Darbuotojas->notEmpty()
```

Pirmame invariante self.Vadovovas yra Asmuo, kadangi asociacijos kardinalumas yra vienetas. Antrame invariante self.Darbuotojas įvertinamas klasės Asmuo aibe. Pagal nutylėjimą, navigacija baigsis aibe.

Kolekcijos, tokios kaip aibės, išrikiuotos aibės ir OCL kalboje apibrėžiamos tipais. Jos turi didelį kiekį apibrėžtų operacijų. Kolekcijos savybė prieinama naudojant strėlę sekančią paskui savybės vardą. Sekančiame pavyzdyje pateikiamas kontekstas susijęs su Asmuo:

```
context Asmuo inv:
    self.Darbuotojas->size() < 3
```

Šis pavyzdys taikomas dydžio (angl. *size*) savybei self.Darbuotojas aibėje, kurios rezultatas yra skaičius Darbuotojų iš klasės Asmuo.

```
context Asmuo inv:
    self.Darbuotojas->isEmpty()
```

Šis pavyzdys taikomas yraTuščia (angl. *isEmpty*) savybei self.Darbuotojas aibėje. Tai įvertinama reikšme teisinga (angl. *true*), jeigu Darbuotojas aibė yra tuščia ir neteisinga (angl. netiesa) – priešingu atveju [20].

1.3.6.21 Navigacija tarp asociacijų su kardinalumu vienas arba nulis

Kadangi vaidmens Vadovas kardinalumas yra lygus vienetui, self.Vadovas yra tipo Asmuo objektas. Toks vienetinis objektas gali būti naudojamas kaip aibė. Tuomet jis elgiasi kaip aibė turinti vieną objektą. Naudojimas kaip aibės yra padaromas per strėlę sekančią po aibės savybės. Tai pateikta pavyzdyje:

```
context Įmonė inv:
    self.Vadovas->size() = 1
```

Subišraiška self.Vadovas naudojamas kaip aibė, kadangi strėlė naudojama pasiekti dydis savybę aibėje. Ši išraiška grąžina reikšmę tiesa.

```
context Įmonė inv:
    self.Vadovas.Amžius > 40
```

Subišraiška self.Vadovas naudojama kaip klasė Asmuo, kadangi taškas naudojamas prieiti prie Amžius savybės, esančios Asmuo esybėje.

Neprivalomos esybės (0..1 kardinalumas) atveju, ypač naudinga patikrinti ar egzistuoja objektas ar ne, kuomet atliekama navigacija asociacijoje. Pavyzdyje gali rašyti:

```
context Asmuo inv:
    self.Žmona->notEmpty() implies self.Žmona.Lytis = Lytis::Moteris
```
[20]

1.3.6.22 Savybių apjungimas

Savybės gali būti apjungtos siekiant užrašyti sudėtingesnes išraiškas. Svarbi taisyklė yra, kad OCL išraiška visada išreiškiama specifinio tipo specifiniu objektu. Po rezultato gavimo, galima visada taikyti kitą savybę rezultatui, norint gauti kitą rezultato reikšmę. Todėl kiekviena OCL išraiška gali būti skaitoma ir išreiškima iš kairės į dešinę.

Žemiau pateikiama keletas invariantų, kurie naudoja apjungtas savybes iš nagrinėjamos klasių diagramos (15 pav.):

[1] Vedusio asmens amžius >= 18

```
context Asmuo inv:
    self.Žmona->notEmpty() implies self.Žmona.Amžius >= 18 and
    self.Vyras->notEmpty() implies self.Vyras.Amžius >= 18
```

[2] Įmonės darbuotojų maksimalus skaičius <=50

```
context Įmonė inv:
    self.Darbuotojas->size() <= 50
```
[20]

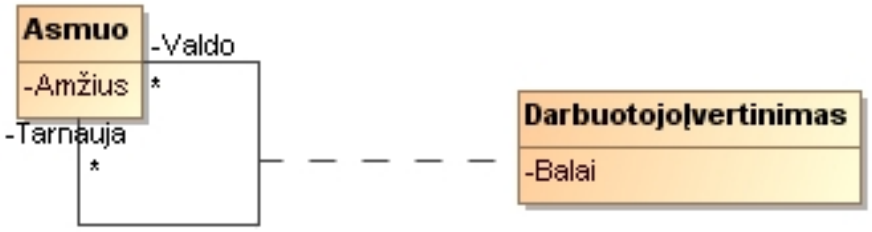
1.3.6.23 Navigacija į asociacijos klases

Norint apibrėžti navigaciją į asociacijų klases (pavyzdžiui į Pareigos ir Santuoka) OCL kalboje naudojamas taškas ir asociacijos klasės vardas rašomas mažosiomis raidėmis:

```
context Asmuo inv:
    self.Pareigos
```

Subišraiška self.Pareigos išreiškiama kaip visų Pareigų aibė, kurias Asmuo atlieka kompanijoje būdamas jos Darbuotoju. Asociacijos klasės atveju, klasių diagramoje neegzistuoja tikslių vaidmenų vardų. Vardas Pareigos naudojamas šioje navigacijoje yra asociacijos klasės vardas.

Grįžtamosios asociacijos atveju, kuomet asociacija yra į tą pačia klasę, nepakanka nurodyti tik asociacijos klasės vardą. Reikia išskirti kryptį į kurią asociacija yra nukreipta, taip pat kaip ir asociacijos klasės vardą. 16 paveikslėlyje pateikiamas modelis iliustruojantis aptartus aspektus.



16 pav. Ryšių priskyrimas grįžtamosios asociacijos klasei [20]

Kuomet ryšiai priskirti asociacijos klasei taip kaip darbuotojo Balai egzistuoja dvi galimybės priklausančios nuo krypties. Pavyzdžiui, aukščiau pateiktame pavyzdyje ryšį galima uždėti nuo Tarnauja pabaigos arba Valdo pabaigos. Naudojant tik asociacijos klasės vardą, šios dvi savybės negali būti atskirtos. Norint jas atskirti, krypties, į kurią norima dėti ryšį, vaidmens vardas pridedamas prie asociacijos klasės vardo, apskliausto laužtiniais skliaustais. Išraiška atrodo taip:

```
context asmuo inv:
    self.Balai[Valdo]->sum() > 0
```

self.Balai[Valdo] išreiškia aibę Balų priklausančią Valdo kolekcijai. Išraiškoje:

```
context Asmuo inv:
    self.Balai[Tarnauja]->sum() > 0
```

self.Balai[Tarnauja] išreiškia Balus priklausančius Tarnauja kolekcijai. Netinkamas asociacijos klasės vardo naudojimas yra neleidžiamas tokiose grįžtamosiose situacijose. Taigi, sekantis pavyzdys yra klaidingas:

```
context Asmuo inv:
    self.Balai->sum() > 0 -- KLAIDINGAS!
```

Negrįžtamojoje situacijoje, asociacijos klasės vardo pilnai pakanka, tai leidžia ir tinkamos versijos. Taigi pavyzdys pateiktas šio skyriaus pradžioje gali būti užrašytas taip:

```
context Asmuo inv:
    self.Pareigos[Darbuotojas] [20]
```

1.3.6.24 Navigacija iš asociacijos klasės

Galima apibrėžti navigaciją iš asociacijos klasės į objektą esantį asociacijoje. Tai atliekama naudojant taško žymėjimą ir vaidmenų vardus asociacijų pabaigose.

```
context Pareigos
    inv: self.Darbuotojas.DarbuotojųSkaičius >= 1
    inv: self.Darbuotojas.Amžius > 21
```

Navigacija iš asociacijos klasės į vieną iš objektų esančių asociacijoje visada pateikia tiksliai vieną objektą. Todėl, šios navigacijos rezultatas yra tiksliai vienas objektas, nors jis gali būti naudojamas aibėje naudojant strėlę („->“).[20]

1.3.6.25 Navigacija per sudėtines asociacijas

Sudėtinės (angl. *qualified*) asociacijos naudoja vieną arba daugiau sudėtinių atributų, norėdamos pasirinkti objektus iš kitos asociacijos pabaigos. Norint apibrėžti jų navigaciją, sudėtiniams atributams galima pridėti reikšmes. Tai atliekama naudojant laužtinius skliaustus einančius po vaidmens vardo. Leidžiama palikti sudėtines reikšmes, kuriose rezultatas bus visi objektai kitoje asociacijos pabaigoje. Pateiktame pavyzdyje pateikiama visų banko klientų aibė:

```
context Bankas inv:
    self.Klientas
```

Sekančiame pavyzdyje pateikiamas vienas asmuo, kurio sąskaitos numeris 2222:

```
context Bankas inv:
    self.SąskaitosNumeris[2222]
```

Jeigu egzistuoja daugiau negu vienas sudėtinis atributas, reikšmės atskiriamos kalbeliais, tokiu būdu, kuris yra apibrėžtas UML klasių modelyje. Neleidžiama dalinai apibrėžti sudėtinių atributų reikšmių. [20]

1.3.6.26 Kolekcijos

Viena asociacijos navigacija yra aibėje (angl. *set*), sudėtinė navigacija yra rinkinyje (angl. *bag*) ir navigacija per asociacijas su {*išrykiuota*} rezultatais yra išrikiuotoje aibėje (angl. *OrderSet*). Todėl kolekcijos tipai apibrėžiami OCL standartinėje bibliotekoje vaidina svarbų vaidmenį OCL išraiškose.

Kolekcijų tipai yra apibrėžiami OCL kalboje. Kolekcijų tipas apibrėžia didelį kiekį operacijų leidžiančių OCL išraiškos autoriui (modeliuotojui) manipuliuoti kolekcijomis. Atitinkamai su OCL, kaip išraiškų kalbos apibrėžimu, kolekcijos operacijos niekada nekeičia kolekcijos. Jos gali įtakoti kolekciją, bet vietoj to, kad pakeistų originalią kolekciją, jos sukuria naują.

Kolekcija yra abstraktus tipas su konkrečiais tipais kaip jos potipiais. OCL išskiria tris skirtingus kolekcijų tipus: aibė, seka ir rinkinys. Aibė yra matematinė aibė. Joje elementai nedubliuojami. Rinkinys yra kaip aibė, tačiau jame gali būti dubliuojami elementai (t.y. tas pats elementas rinkinyje gali būti du arba daugiau kartų). Seka yra kaip rinkinys, kuriame elementai yra išrikiuoti. Tiek aibė tiek rinkinys neturi jokio rikiavimo.

1.3.6.27 Kolekcijų aprašai

Aibės, sekos ir rinkiniai gali būti užrašyti OCL kalboje. Kontūriniai skliaustai dedami aplink kolekcijos elementus. Tarp kolekcijos elementų dedami kableliai. Kolekcijos tipas užrašomas prieš kontūrinius skliaustus.

Aibių pavyzdžiai:

```
Set { 1 , 2 , 5 , 88 }
Set { 'obuolys,' 'apelsinas,' 'žemuogė' }
```

Sekų pavyzdžiai:

```
Sequence { 1, 3, 45, 2, 3 }
Sequence { 'riešutas,' 'uoga' }
```

Rinkinio pavyzdys:

```
Bag {1, 3, 4, 3, 5 }
```

Dėl nuoseklių sveikųjų skaičių sekų naudingumo, jų sukūrimui egzistuoja keletas aprašų. Elementai viduje kontūrinių skliaustų gali būti pakeisti intervalo specifikacija, kuri susideda iš dviejų sveikųjų skaičių tipo išraiškų *Int-expr1* ir *Int-expr2*, atskirtų dviem taškais. Tai žymi visus sveikųjų skaičių tipo skaičius tarp *Int-expr1* ir *Int-expr2* reikšmių, įskaitant ir pačius *Int-expr1* ir *Int-expr2*:

```
Sequence{ 1..(6 + 4) }
Sequence{ 1..10 }
-- abu yra identiški:
Sequence{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
```

Kolekcija gali būti apibrėžta ir taip, kaip pavaizduota žemiau. Vienintelis kitas kelias gauti kolekciją yra naudotis navigacija. Tiksliau sakant vienintelis būdas gauti *Set*, *OrderedSet*, *Sequence* arba *Bag* yra:

1. tiesioginis, jis įtakos aibes, išrikiuotas aibes, sekas arba rinkinius:

```
Set {2 , 4, 1 , 5 , 7 , 13, 11, 17 }
OrderedSet {1 , 2, 3 , 5 , 7 , 11, 13, 17 }
Sequence {1 , 2, 3 , 5 , 7 , 11, 13, 17 }
Bag {1, 2, 3, 2, 1}
```

2. navigacija prasidedanti nuo vieno objekto gali virsti kolekcija:

```
context Company inv:
    self.employee
```

3. operacijos su kolekcija gali sukurti naują kolekciją:

```
kolekcija1->union(kolekcija2)
```
[20]

1.3.6.28 Kolekcijų kolekcijos

UML 1.4 versijoje OCL kalboje kolekcija negalėjo talpinti kitų kolekcijų kaip elementų. Šis ribojimas panaikintas UML 2.0 versijoje. OCL leidžia kolekcijų elementams patiems būti kolekcijoms. OCL standartinė biblioteka talpina specialias operacijas skirtas kolekcijoms. [20]

1.3.6.29 Ankstesnės reikšmės po sąlygose

OCL gali būti naudojamas apibrėžti prieš ir po sąlygas operacijoms ir metodams UML kalboje. Po sąlygose išraiška gali nurodyti reikšmę kiekvienai objekto savybei dviem laiko momentais:

- savybės reikšmę operacijos ar metodo pradžioje,
- savybės reikšmę operacijos ar metodo užbaigimo metu.

Savybės reikšmė po sąlygoje yra reikšmė po operacijos užbaigimo. Nurodant savybės reikšmę operacijos pradžioje reikia nurodyti savybės vardą su reikšminiu žodžiu *@pre*:

context Asmuo::Gimtadienis()
post: Amžius = Amžius@pre + 1

Savybė Amžius nurodo į Asmens egzemplioriaus savybę, kuri vykdo operaciją. Savybė *Amžius@pre* nurodo į Asmuo savybės Amžius reikšmę, kuri įvykdo operaciją operacijos pradžioje.

Jei savybė turi parametrus, *@pre* rašomas po savybės vardo, prieš parametrus.

context Įmonė::SamdomasDarbuotojas(A : Asmuo)
post: Darbuotojas = Darbuotojas@pre->including(A) **and** akcijosVertė()= akcijosVertė@pre() + 10

Postfiksas *@pre* leidžiamas naudoti tik OCL išraiškose, kurios yra po sąlygos dalis. [20]

1.3.6.30 Kortežai

Įmanoma sukurti keletą reikšmių korteže (angl. *tuple*). Kortežas susideda iš dalių turinčių pavadinimą ir duomenų tipą. Keletas kortežų pavyzdžių:

Tuple {Vardas: String = ‘Jonas,’ Amžius: Integer = 10}
Tuple {a: Collection(Integer) = Set{1, 3, 4}, b: String = ‘foo,’ c: String = ‘bar’}

Tai taip pat būdas užrašyti kortežų simbolius OCL kalboje. Kortežų simboliai yra apskliaudžiami kotūriniais skliaustais ir atskiriami vienas nuo kito kableliais. Tipų vardai yra neprivalomi ir jų dalių išrykiavimas nėra svarbus. Taigi:

Tuple {Vardas: String = ‘Jonas,’ Amžius: Integer = 10} yra ekvivalentu
Tuple {Vardas = ‘Jonas,’ Amžius = 10} ir ekvivalentu
Tuple {Amžius = 10, Vardas = ‘Jonas’}[20]

1.4. Aprašytų metodų palyginimas

Aprašytų metodų ir modelių palyginimas vykdomas dvejais aspektais:

- Kokią loginę struktūrą verslo taisyklių modeliavimui suteikia metodas?
- Kuris verslo taisyklių tipas gali būti sumodeliuotas?

Septintoje lentelėje pateikiami abu kriterijai. Modeliuojant pavyzdžius, paminėtus aukščiau, kartais naudojama modelių loginė struktūra. Reikia prisiminti, kad iš principo visi taisyklių tipai gali būti apibūdinti žodžiais. Metodų charakterizavimas, atsižvelgiant į jų galimybę atvaizduoti tam tikrus verslo taisyklių tipus, priklauso nuo to, ar papildomas žodinis apibūdinimas yra neatskiriama metodo dalis.

7 lentelė. Aptartų metodų palyginimas

Metodas	Modeliavimo loginės struktūros					Verslo taisyklių tipai	
	Objektai	Taisyklės	Įvykiai	Sąlygos	Veiksmai	Nuoseklumo taisyklės	Veiklos taisyklės
DFD	Duomenų saugyklos, išoriniai objektai	---	Atvaizduojami dalinai duomenų srautais	---	Procesai	Tiktai mini specializacijose	Labai suvaržytas
CPM	---	---	Išoriniai/ vidiniai įvykiai	Sinchronizacija	Operacija	Taip	Taip
STD	---	---	(parėjimo žymės)	Perėjimo žymės	Perėjimo žymės	Taip	Taip
Perėjimo grafai	--- (apribotos iki vieno objekto)	---	---	Perėjimo žymės	---	Tarpinės, dinaminės	Ne
Išplėstas PN	---	---	Vietos	Vietos	Perėjimai	Taip	Taip
ERM, NIAM	Esybių tipai, ryšio tipai (atributai)	---	---	Kardinalumai, raktai, sritys	---	Statinės (ribotos)	Ne
ER-RM	Esybių tipai, ryšio tipai (atributai)	Taisyklių tipas	Tiesioginiai lankai	Numatytos taisyklės tipo	Tiesioginiai lankai	Taip	Labai ribotos
BIER	Esybių tipai, ryšio tipai (atributai)	---	Vietos	---	Perėjimai	Tarpinės, statinės, dinaminės	Taip
ELH	Esybių tipai (apribota iki vieno esybės tipo)	---	Įvykiai (išoriniai)	---	Operacijos	Tarpinės (ribotos)	Labai ribotos
Įvykių schema iš OOA&D	--- (nukreiptos į objekto modelį)	Trigerio taisyklės	Įvykio tipai	Valdymo sąlygos	Operacijos	Taip	Taip
UML klasių diagrama + OCL	Esybių tipai	Trigerio taisyklės	Įvykio tipai	Kardinalumai, raktai, valdymo sąlygos	Operacijos	Taip	Taip

2. OCL išraiškų užrašymo metodo patobulinimas

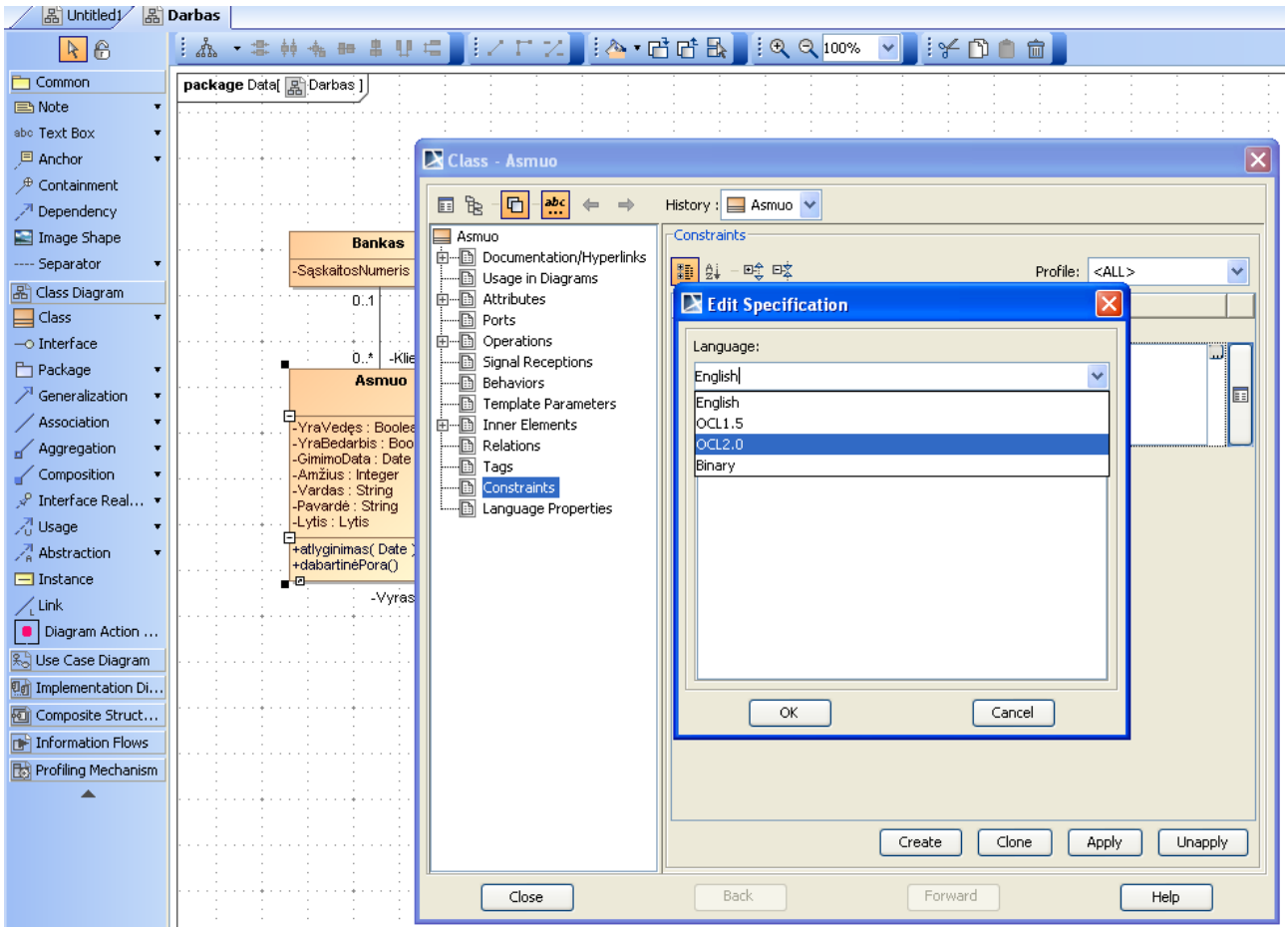
Pagrindinės problemos, su kuriomis susiduria informacinės sistemos projektuotojas, rašydamas OCL išraiškas:

1) sudėtinga rasti *MagicDraw* priemonėje esantį OCL išraiškų rašymo įrankį
2) nors UML yra objektinė kalba, ji naudoja sintaksiškai sudėtingus OCL ribojimus, kurių sudarymo struktūros naudotojas turi mokytis,
3) nėra sukurta priemonė, kuri užtikrintų užrašytos OCL išraiškos teisingumą,

2.1. OCL išraiškų rašymas MagicDraw aplinkoje

OCL kalbos išraiškų užrašymo metodas bus patobulintas *MagicDraw* įrankyje, kuris palaiko UML 2.0 diagramas ir suteikia OCL išraiškų užrašymo galimybę.

Šiuo metu *MagicDraw* aplinkoje esantis OCL išraiškų rašymo įrankis (17 pav.) suteikia galimybę užrašyti OCL išraiškas tik šią kalbą mokančiam informacinių sistemų specialistui. OCL sintaksės tikrinimo priemonė tik patikrina, ar teisinga išraiškos sintaksė, tačiau nepatikrina pačios išraiškos teisingumo.



17 pav. OCL išraiškų užrašymas *MagicDraw* aplinkoje

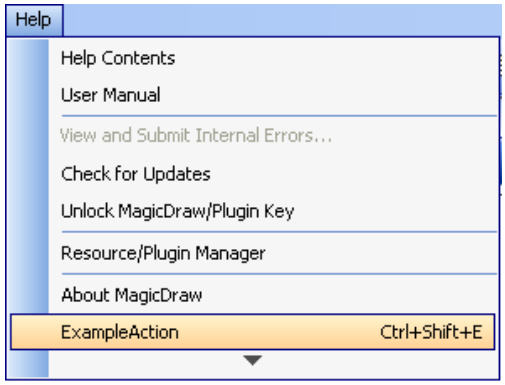
2.2. MagicDraw funkcijų išplėtimas

Vienintelis būdas, leidžiantis išplėsti MagicDraw funkcijas, yra papildinio (angl. *plug-in*) sukūrimas. Papildinys turi turėti žemiau pateiktus išteklius:

- katalogą;
- sukompiliuotą *java* failą (**.jar*);
- papildinio aprašymo failą (angl. *descriptor file*);
- papildomus papildinio naudojamus failus.

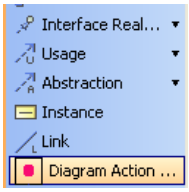
Papildomas funkcijas galima pridėti prie:

- pagrindinio meniu (angl. *main meniu*);

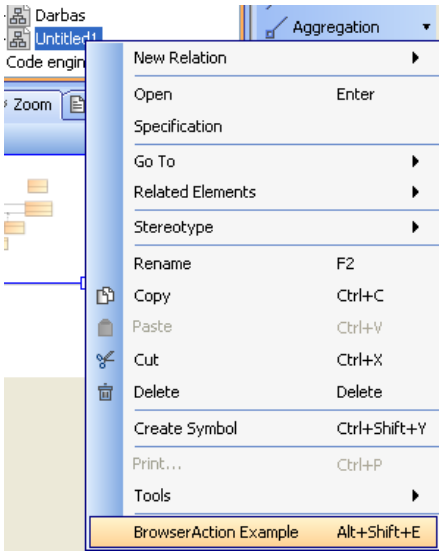


- pagrindinės mygtukų juostos (angl. *main toolbar*);

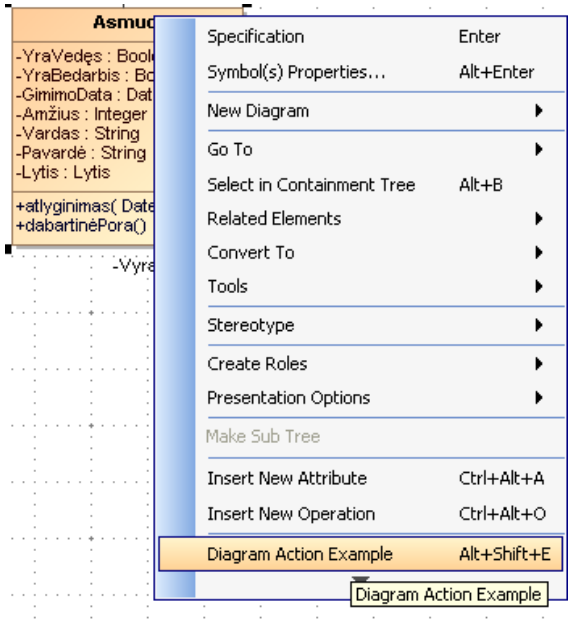
- diagramos mygtukų juostos (angl. *diagram toolbar*);



- naršyklės turinio meniu (angl. *browser context meniu*);



- diagramos turinio meniu (angl. *diagram context meniu*).



2.3. Papildinio aprašymo failas

Papildinio aprašymo failas yra parašytas XML kalba ir pavadintas plugin.xml. Kiekvienas papildinio aprašymo failas turi tik vieno papildinio savybių. Paleidžiant MagicDraw programą, yra patikrinami visi papildinio aprašymo failai, esantys <MagicDraw įdiegimo katalogas>/plugins.

Siekiant patobulinti OCL išraiškų rašymą MagicDraw aplinkoje buvo sukurtas plugin.xml failas, kuriame yra tokia informacija:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
        id="123456"
        name="OCL plugin"
        version="2.0"
        provider-name="No Magic"
        class="myPlugin.ActionsExample">
        - <runtime>
  <library name="OCL.jar" />
 </runtime>
 </plugin>
```

Čia: *ID* – unikalus papildinio numeris;

name – papildinio pavadinamas;

version – papildinio versija;

provider-name – papildinio autoriaus arba įmonės pavadinimas, pavyzdžiui: „No Magic“;

class – klasės, kuri bus iškviečiama paleidžiant MagicDraw, pavadinimas ir kelias iki jos iš <MagicDraw įdiegimo katalogas>/plugins katalogo.

Library name – bibliotekos, kuri patalpinta <MagicDraw įdiegimo katalogas>/plugins kataloge, vardas.

2.4. OCL papildinys

Kuriant OCL papildinį pagrindinis dėmesys buvo skiriamas:

- OCL išraiškų rašymui palengvinti;
- OCL išraiškų rašymo teisingumo kontrolei įvesti.

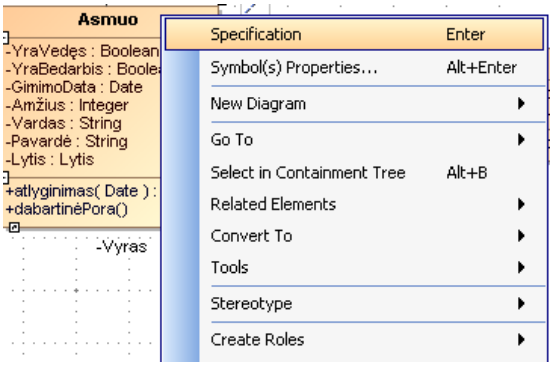
Pasirinkti keturi pagrindinės OCL išraiškos, kurių rašymas turėjo būti palengvintas sukurto papildinio, tai:

1. OCL invarianto išraiška;
2. prieš ir po sąlygų išraiška;
3. užklausos išraiška;
4. išvedimo (angl. *derivation*) išraiška.

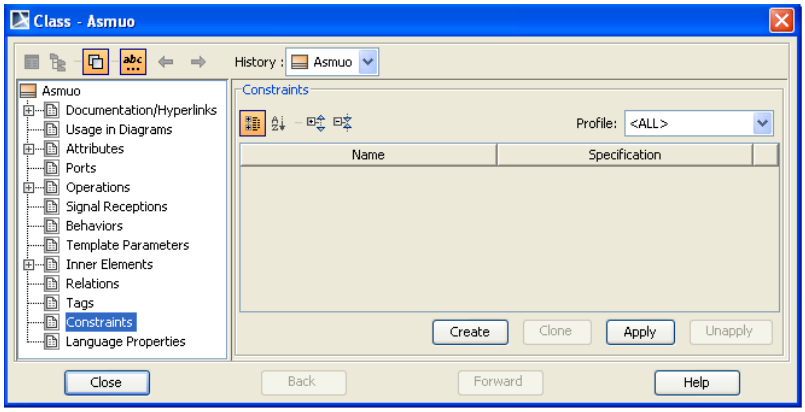
2.4.1. Prieigos prie OCL išraiška rašymo įrankio palengvinimas

Pirma problema, kurią pavyko išspręsti siekiant patobulinti OCL išraiška rašymą, buvo prieigos prie OCL išraiškos rašymo įrankio palengvinimas. Nepatobulintame MagicDraw, norint pasiekti OCL išraiškos įrankį, reikėjo atlikti tokius veiksmus:

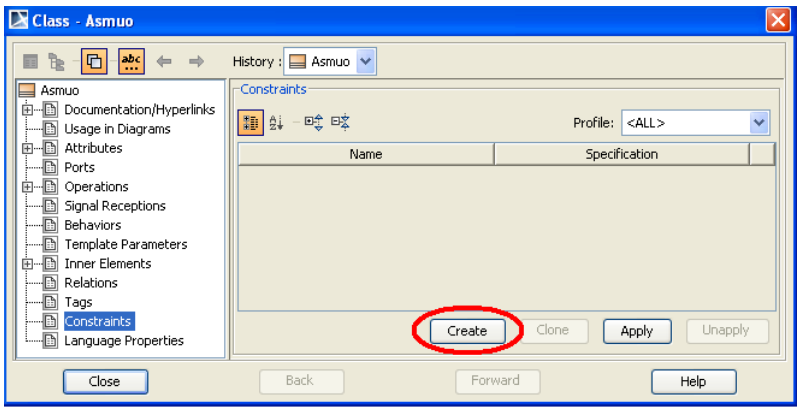
1) ties pasirinkta klase paspausti dešinįjį pelės mygtuką ir pasirinkti „Specifikacija“ (angl. *Specification*);



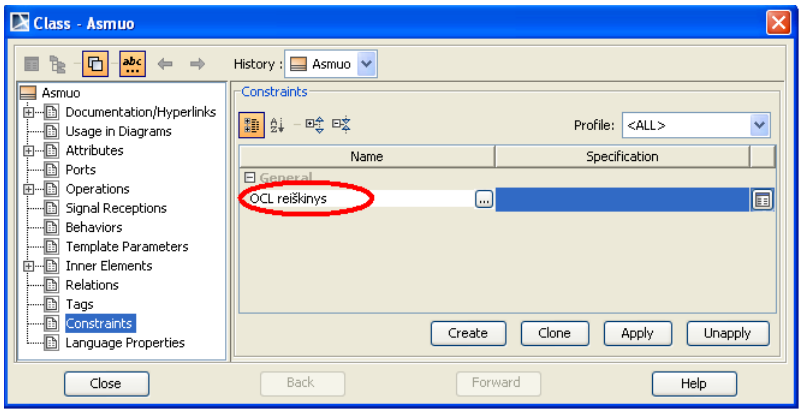
2) atsidariusiame lange pasirinkti „Ribojimai“ (angl. *Constraints*);



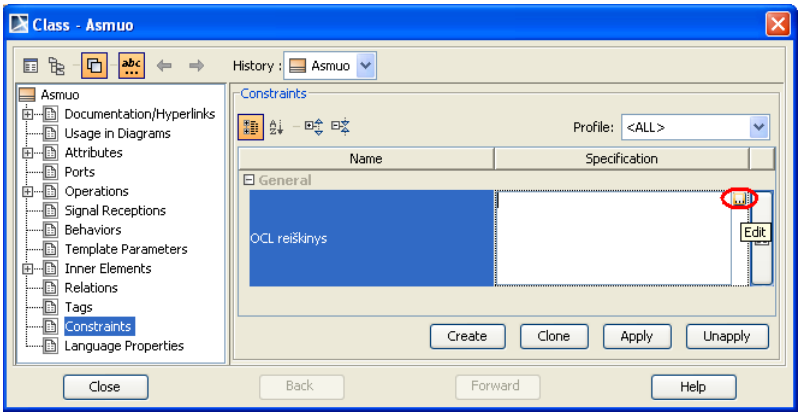
3) paspausti mygtuką „Sukurti“ (angl. *Create*);



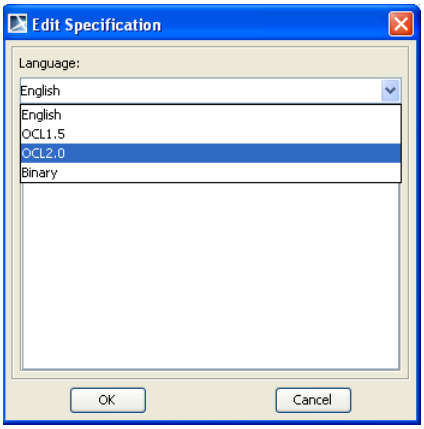
4) įrašyti pavadinimą;



5) susirasti ir paspausti mygtuką „Redaguoti“ (angl. *edit*);



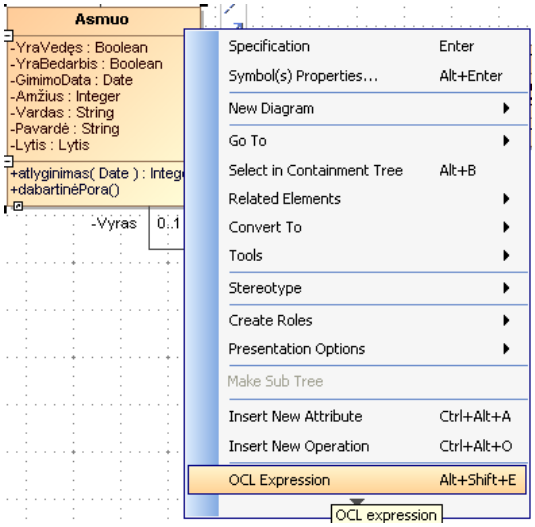
6) atsidariusiame lange pasirinkti ribojimo rašymo kalbą OCL2.0 arba kitą;



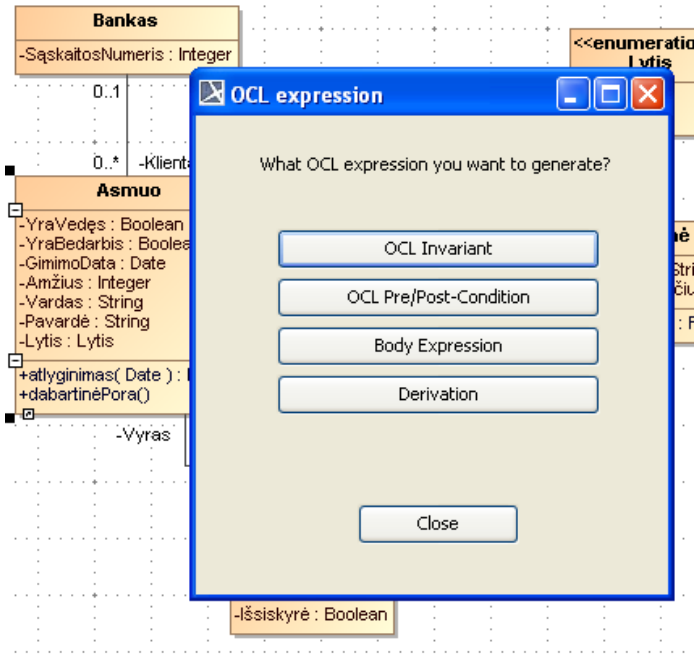
7) rašyti OCL išraišką.

Įdiegus papildinį, OCL išraiškos rašymo įrankį galima pasiekti atlikus tokius veiksmus:

1. ties pasirinkta klase paspausti dešinįjį pelės mygtuką ir pasirinkti „OCL išraiška“ (angl. *OCL Expression*);



2. pasirinkti reikiamą OCL išraišką;



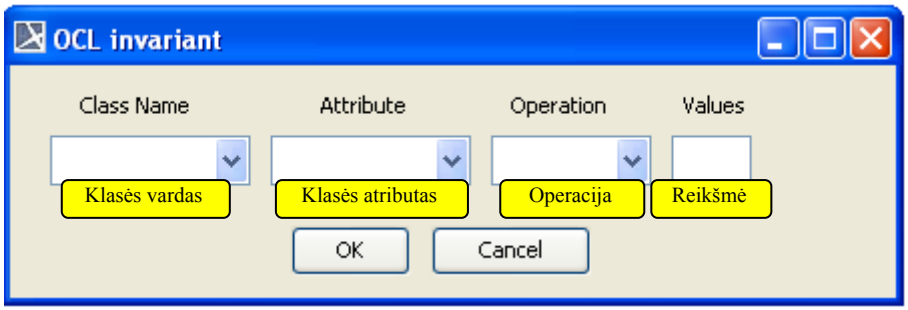
3. pasirinkti norimus kriterijus.

2.4.2. OCL invarianto išraiškos generavimas

OCL invariantas – dažniausiai naudojama OCL išraiška, jo sintaksė nėra labai sudėtinga, tačiau reikalauja tam tikrų OCL kalbos žinių. Apskritai OCL invarianto išraiška galėtų atrodyti taip:

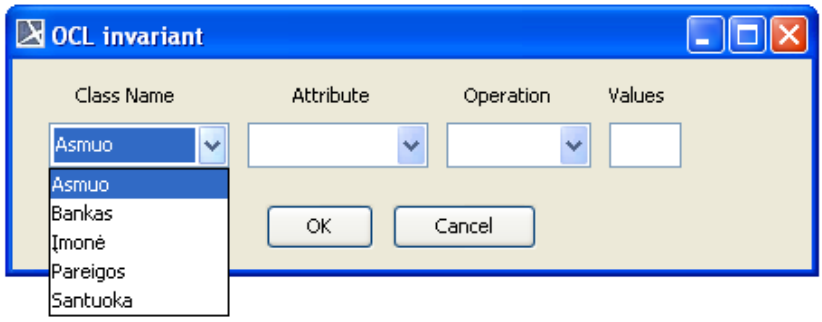
context KlasėsVardas **inv**:
[self.]KlasėsAtributas operacija reikšmė

Remiantis ankščiau pateiktu pavyzdžiu, buvo sukurtas įrankis, leidžiantis užrašyti tokio tipo išraiškas (18 pav.).



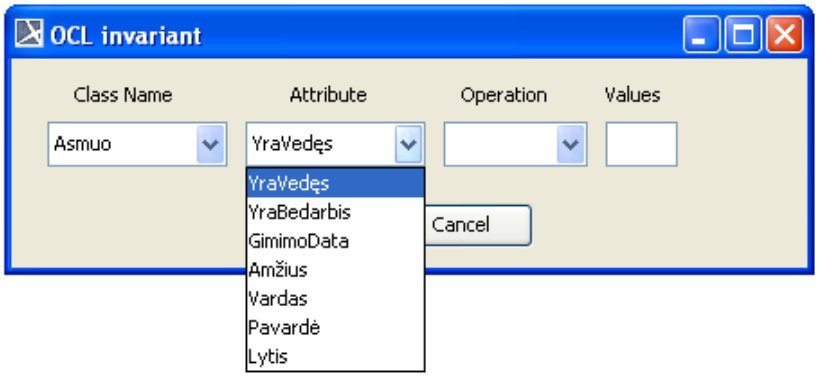
18 pav. OCL invarianto išraiškos generavimo įrankis

Klasės vardai yra nuskaitomi iš *MagicDraw* klasės diagramos, pateiktos nagrinėjant analitinę dalį (15 pav.). Naudotojas, pasirinkęs OCL invarianto išraiškų generavimo įrankį, galės pasirinkti iš tokių klasių, kaip pateikta 19 paveiksle:



19 pav. Nagrinėjamu atveju galimos klasės vardų reikšmės

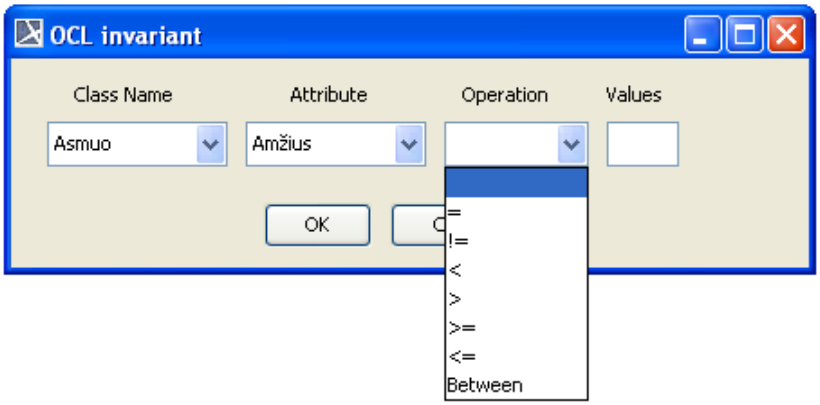
Priklausomai nuo pasirinktos klasės, klasės atributų laukelyje pateikiamos atributų reikšmės (20 pav.).



20 pav. Klasės „Asmuo“ atributų sąrašas

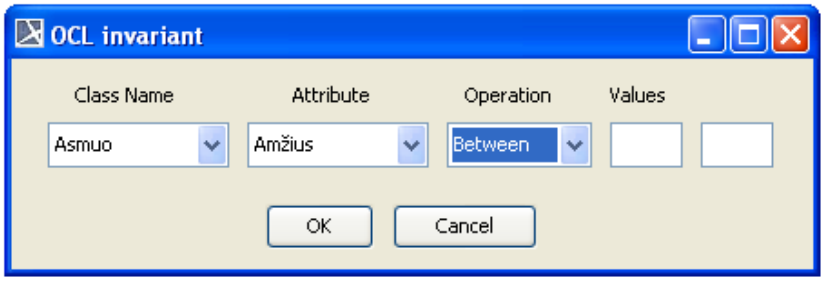
Priklausomai nuo pasirinktos atributo reikšmės tipo yra pateikiamas operacijų sąrašas (20 pav.):

- jeigu pasirinkto atributo reikšmė yra loginio (angl. *boolean*) tipo, galima operacijos ir reikšmės laukelį palikti tuščią (jei reikšmė „TEISINGA“ (angl. *TRUE*)) arba operacijos laukelyje pasirinkti „=“, o reikšmės laukelyje įrašyti „TEISINGA“ arba „NETEISINGA“ (angl. *FALSE*);
- jeigu pasirinkto atributo reikšmė yra tekstinė, tuomet galima pasirinkti tik „=“ operaciją;
- jeigu pasirinkto atributo reikšmė yra skaitinė, tuomet galima pasirinkti visas reikšmes, kurios pavaizduotos 21 paveiksle.



21 pav. Galimos klasės „Asmuo“ atributo „Amžius“ operacijų reikšmės

Jeigu iš operacijų sąrašo pasirenkama „Tarp“ (angl. *Between*) reikšmė, tuomet OCL invarianto išraiškos generavimo įrankis atrodo taip, kaip pateikta 22 paveiksle.

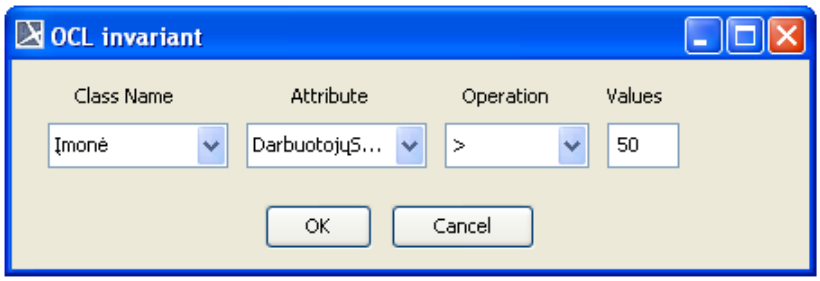


22 pav. OCL invarianto išraiškos generavimo įrankis, pasirinkus operaciją „Tarp“

Nagrinėjant literatūros apžvalgą, surastas OCL invarianto išraiškos pavyzdys:

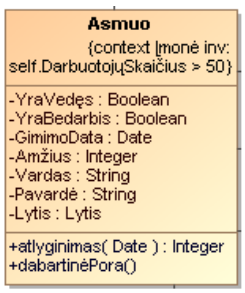
context Įmonė **inv**:
self.DarbuotojųSkaičius > 50

gali būti užrašytas taip, kaip tai pateikta 23 paveiksle.



23 pav. Verslo taisyklės „Įmonės darbuotojų skaičius turi būti didesnis negu 50“ užrašymo OCL invarianto išraiškos generavimo įrankiu pavyzdys

Pasirinkus tokius kriterijus, kaip pavaizduoja 23 paveiksle, ir paspaudus mygtuką „GERAI“ (angl. *OK*), klasių diagramoje gaunama OCL invarianto išraiška, pateikta 24 paveiksle.



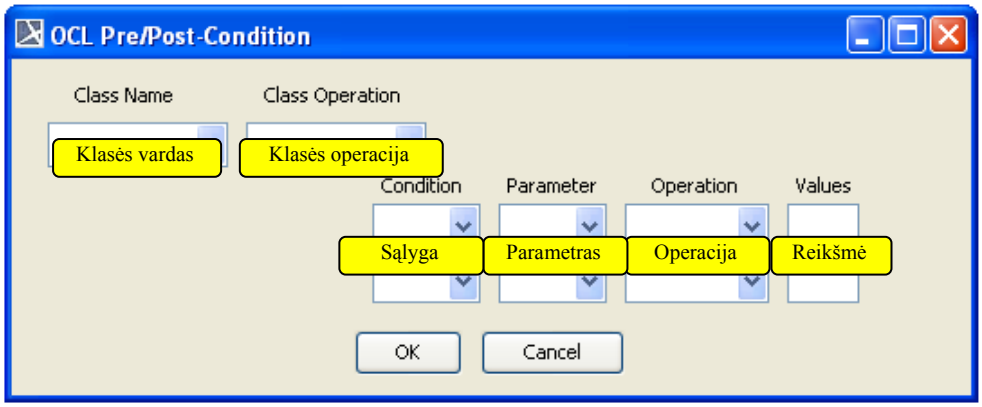
24 pav. Verslo taisyklės „Įmonės darbuotojų skaičius turi būti didesnis negu 50“ atvaizdavimas klasėje „Asmuo“

2.4.3. OCL prieš ir po sąlygų išraiškos generavimas

OCL prieš ir po sąlygos užrašomos, remiantis tokia sintakse:

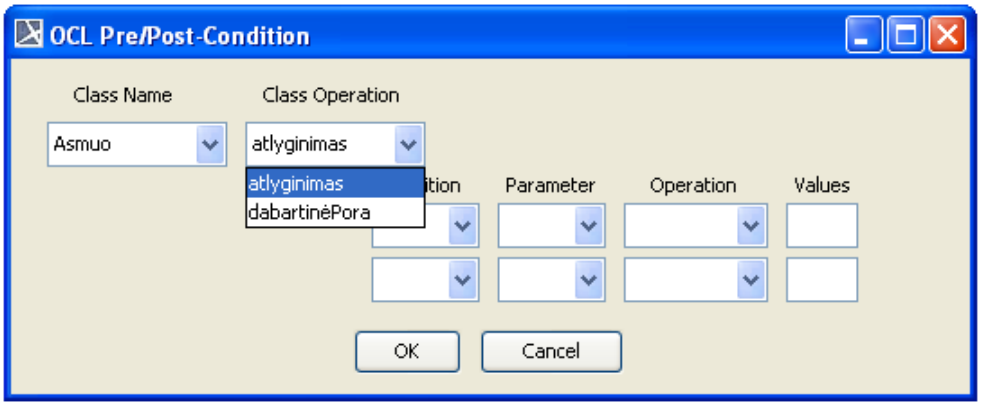
context KlasėsVardas::OperacijosVardas(Parametras1 : Tipas1, ...): GrąžinamasTipas
pre : parametras1 > ...
post: rezultatas = ...

Remiantis šia sintakse, buvo sukurtas prieš ir po sąlygų išraiškos generavimo įrankis (25 pav.)



25 pav. OCL prieš ir po sąlygų išraiškos generavimo įrankis

Klasės vardai, kaip ir OCL invarianto išraiškos atveju ir kaip žemiau pateiktais atvejais, nuskaitomi iš klasės diagramos. Klasės operacijos taip pat nuskaitomos iš klasės diagramos, kaip pavaizduota 26 paveiksle.

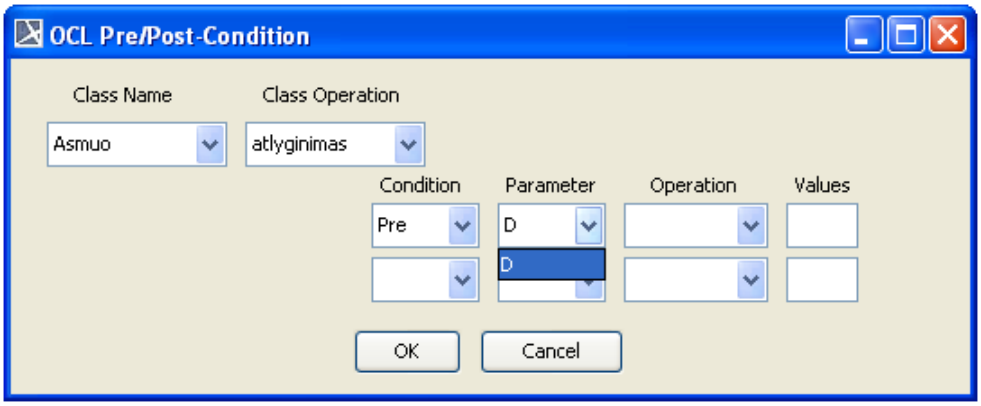


26 pav. Klasės „Asmuo“ operacijos

Klasė privalo turėti operaciją, kitu atveju tolesni veiksmai negalimi.

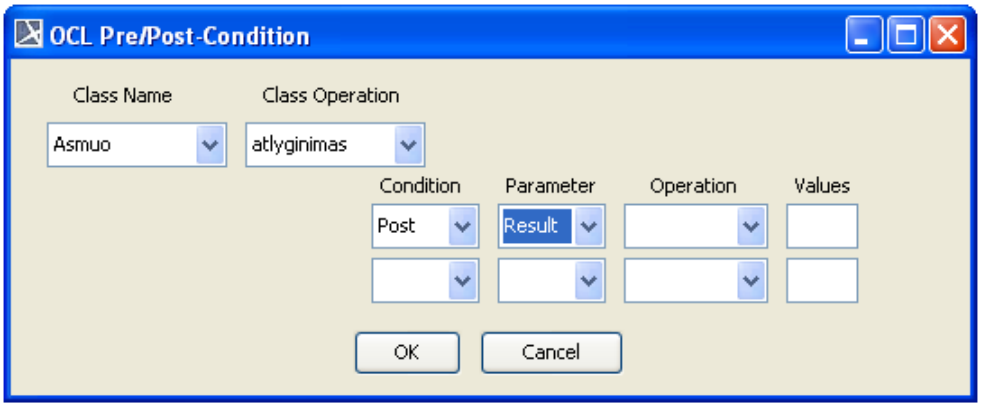
Jeigu klasė turi operaciją, toliau galima pasirinkti vieną iš dviejų sąlygų:

- prieš (27 pav.);
- po (28 pav.).



27 pav. Prieš sąlygos sudarymo pavyzdys

Pavyzdyje (27 pav.) parametro reikšmė lygi „D“. Ši reikšmė gaunama iš klasės diagramos operacijos „atlyginimas“ reikšmės tipo „Data“ (angl. *date*), imant tik pirmąją raidę.



28 pav. Po sąlygos sudarymo pavyzdys

Pavyzdyje (28 pav.) parametro reikšmė lygi „Rezultatas“ (angl. *result*). „Rezultatas“ yra rezervinis po sąlygos kintamasis, kuris nurodo po operacijos rezultatą.

Šiame įrankyje leidžiama pasirinkti po vieną prieš sąlygą ir po sąlygą arba dvi prieš sąlygas, arba dvi po sąlygas.

Operacijoms galioja lygiai tokios pat sąlygos kaip ir OCL invarianto išraiškos atveju.

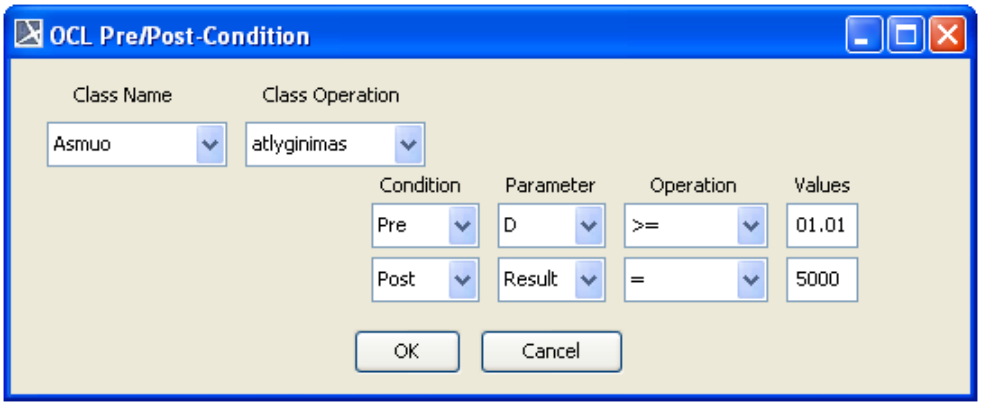
Analitinėje dalyje nagrinėtą po sąlygos pavyzdį:

context Asmuo::atlyginimas(D : Date) : Integer
post: result = 5000

galima papildyti prieš sąlyga: „data turi būti didesnė už arba lygi 2009.01.01“. Taip gaunama tokia prieš ir po sąlygų išraiška:

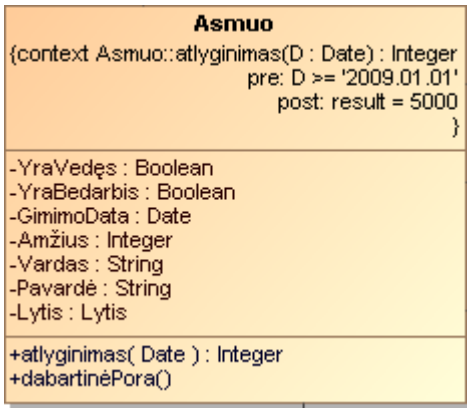
context Asmuo::atlyginimas(D : Date) : Integer
pre: D >= ‘2009.01.01’
post: result = 5000

Šių prieš ir po sąlygų užrašymo pavyzdys generavimo įrankyje pateikiamas 28 paveikslėlyje.



29 pav. Po sąlygos pavyzdys

Rezultatas, gautas paspaudus mygtuką „GERAI“, pateikiamas 29 paveiksle.



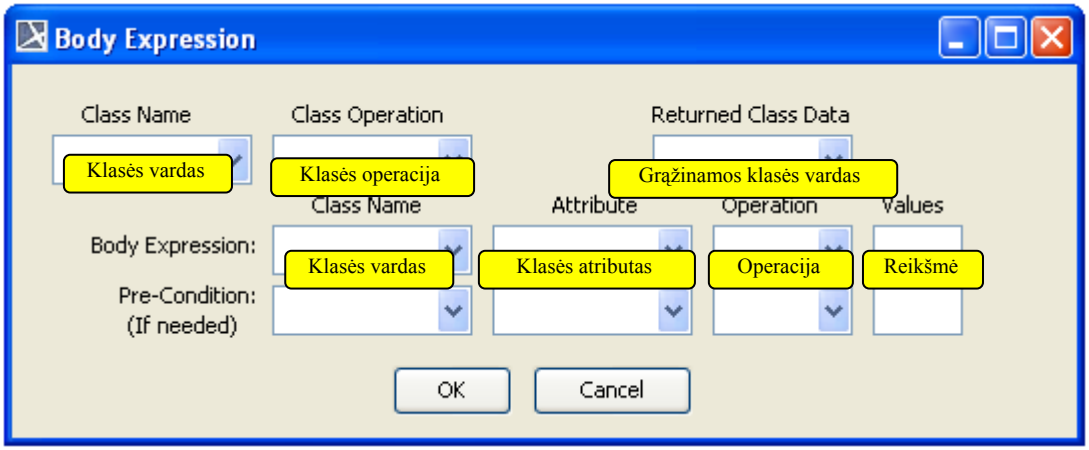
30 pav. Prieš ir po sąlygos atvaizdavimas klasėje „Asmuo“

2.4.4. OCL užklausos išraiškos generavimas

OCL išraiška gali būti naudojama užklausos operacijos rezultatų nustatymui. Tai atliekama vartojant tokią sintaksę:

context KlasėsVardas::OperacijosVardas(parametras1 : tipas1, ...): GrąžinamasKlasėsVardas
body: -- tam tikra užklausos operacijos išraiška

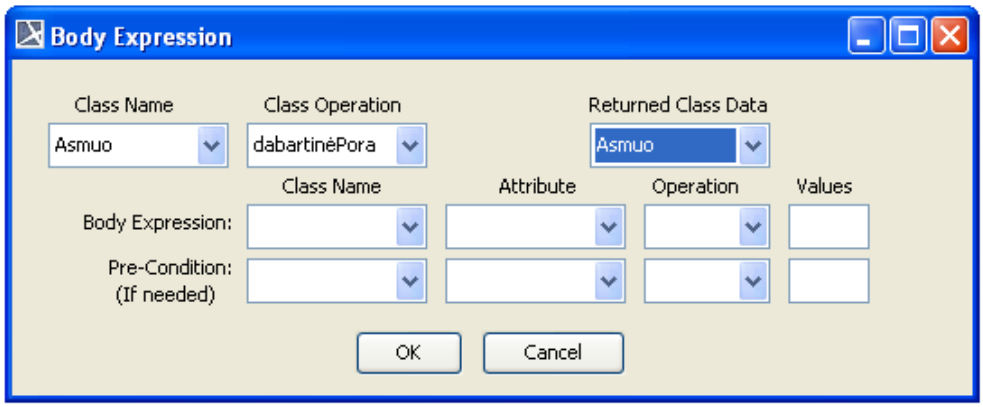
Remiantis šia sintakse ir užklausos išraiškų pavyzdžiais, nagrinėtais analitinėje dalyje, buvo sukurtas užklausos išraiškos generavimo įrankis, pateiktas 31 paveiksle.



31 pav. Užklausos išraiškos generavimo įrankis

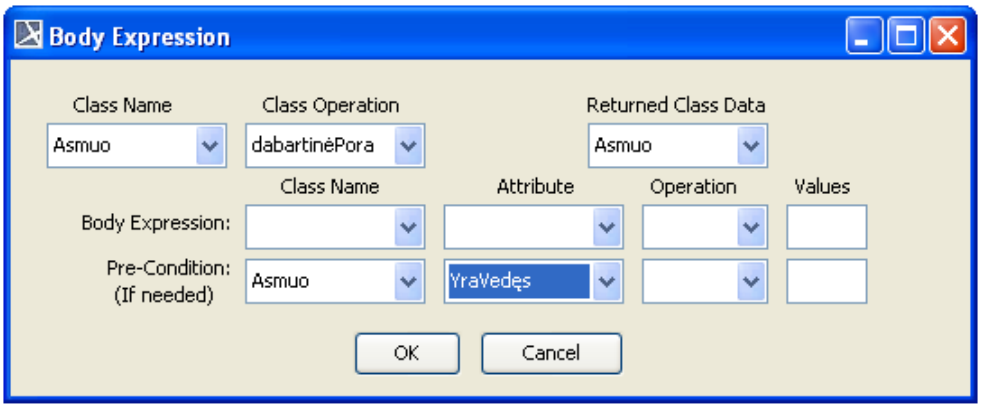
Užklausos išraiška dažnai naudojama su prieš sąlyga, dėl to šiame įrankyje suteikiama ir prieš sąlygos užrašymo galimybė.

Visų pirma pasirenkamas klasės vardas, klasės atributas ir klasės, kurios duomenis norima gauti vardas. Tarkime, norima surasti dabartinę asmens porą. Tam reikia pasirinkti klasės asmuo operaciją „dabartinėPora“. Grąžinama klasė taip pat turėtų būti „Asmuo“ (32 pav.).



32 pav. Klasės vardo, atributo ir grąžinamos klasės pasirinkimas

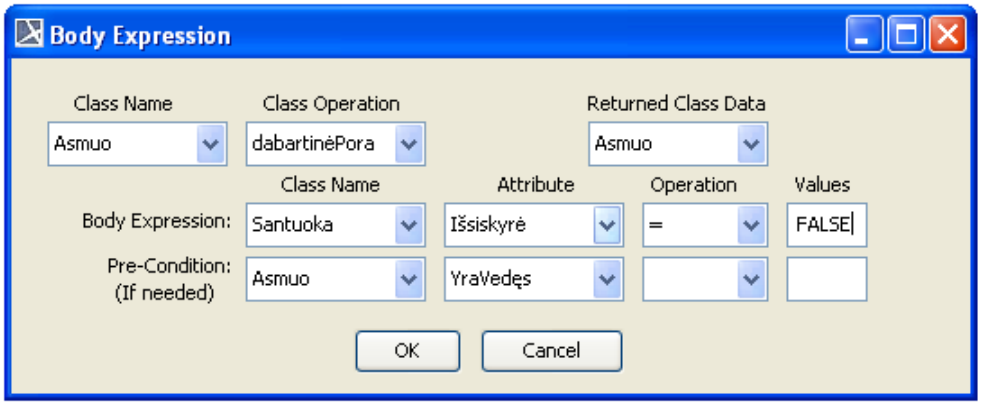
Kad būtų galima patikrinti, kokia dabar yra asmens pora, jis turi būti vedęs. Taigi šiuo atveju reikalinga prieš sąlyga, kurioje reikia patikrinti, ar klasės „Asmuo“ atributas „YraVedęs“ turi reikšmę „TEISINGA“. Tai pateikta 33 paveiksle. Programa, generuodama užklausos išraišką, automatiškai prideda žodelį „Pre“, jeigu yra užpildytas bent vienas prieš sąlygos laukelis.



33 pav. Klasės vardo, atributo ir grąžinamos klasės pasirinkimas

Operacijos ir reikšmės laukeliai paliekami tušti, nes atributas „YraVedęs“ turi reikšmę „TEISINGA“.

Toliau yra sudaromas užklausos išraiška. Kadangi užklausos išraiškoje naudojama užklausa, tai ir tolimesni veiksmai turi būti atliekami atsižvelgiant į tai. Taigi, norint surasti asmens porą, reikia iš klasės „Santuoka“ išrinkti tas asmens santuokas, kurių atributo „Išsiskyrė“ reikšmė yra „NETEISINGA“ (34 pav.). Tokia reikšmė gali būti tik viena, todėl užklausa yra gana paprasta.

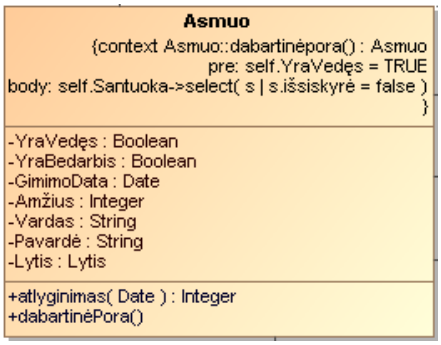


34 pav. Sudaryta užklausos išraiška su prieš sąlyga

Taip sudaryta OCL užklausos išraiška OCL kalboje užrašoma taip:

context Asmuo::dabartinėpora() : Asmuo
pre: self.YraVedęs = TRUE
body: self.Santuoka->select(S | S.Išsiskyrė = false)

Paspaudus mygtuką „GERAI“, būtent tokia OCL išraiška sugeneruojama į klasės diagramos klasę „Asmuo“ (35 pav.)



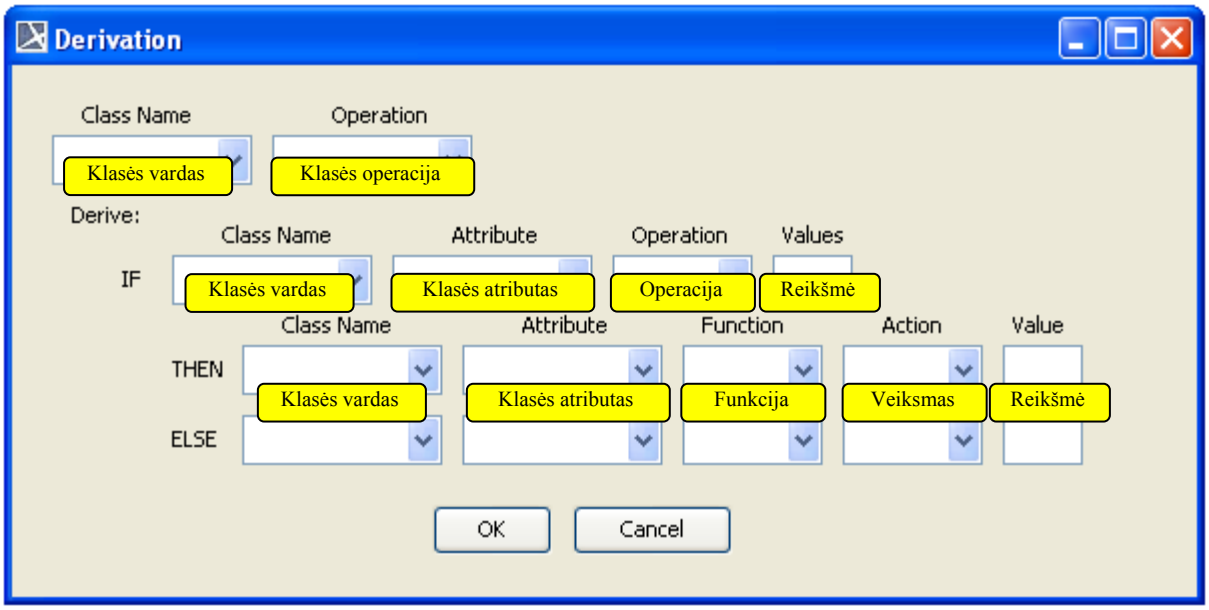
35 pav. Užklausos išraiškos atvaizdavimas klasėje „Asmuo“

2.4.5. OCL išvedimo išraiškos generavimas

OCL išraiška gali būti naudojama atributų gautoms reikšmėms parodyti. Tam vartojama tokia sintaksė:

context KlasėsVardas::KlasėsAtributas: Tipas
derive: -- išraiškos atvaizduojančios išvedimo taisyklę

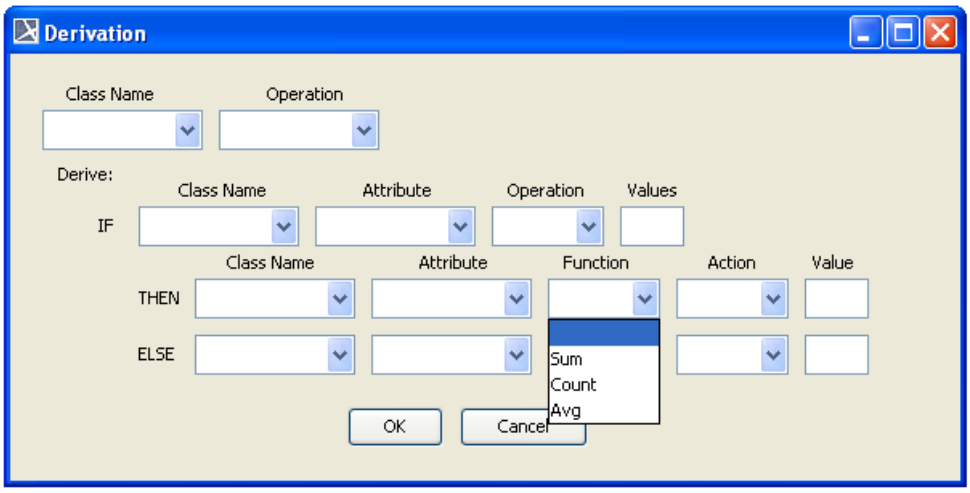
Remiantis šia sintakse ir pavyzdžiais, nagrinėtais analitinėje dalyje, buvo sudarytas įrankis, generuojantis OCL išvedimo išraišką (36 pav.)



36 pav. OCL išvedimo išraiškos generavimo įrankis

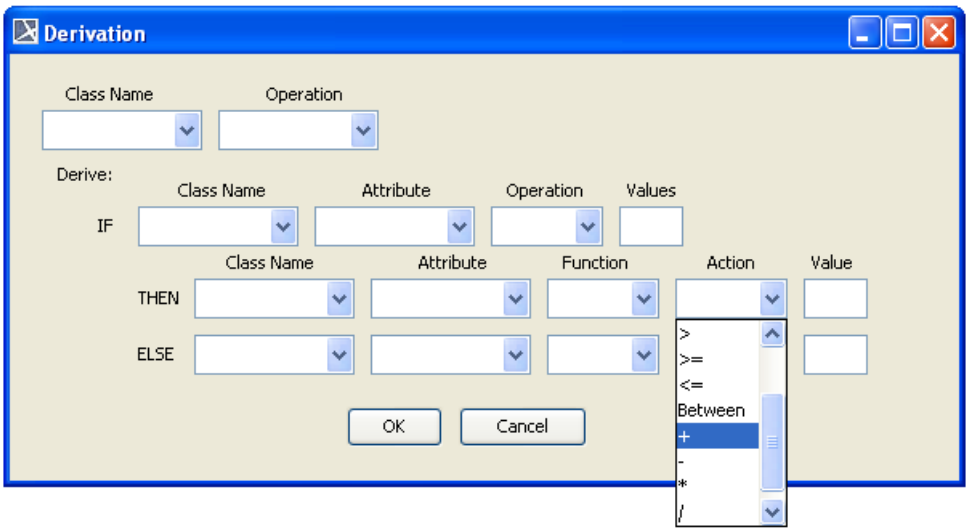
Šiame įrankyje pridėti du papildomi parametrų laukai:

- funkcija (36 pav.);
- veiksmas (37 pav.).



37 pav. OCL išvedimo išraiškos funkcijos parametro laukas

Šiame pavyzdyje (37 pav.) pateikiamos tik pagrindinės funkcijos: suma, kiekis, vidurkis, tačiau šis funkcijų sąrašas gali būti lengvai praplėstas.



38 pav. OCL išvedimo išraiškos parametro „Veiksmas“ laukas

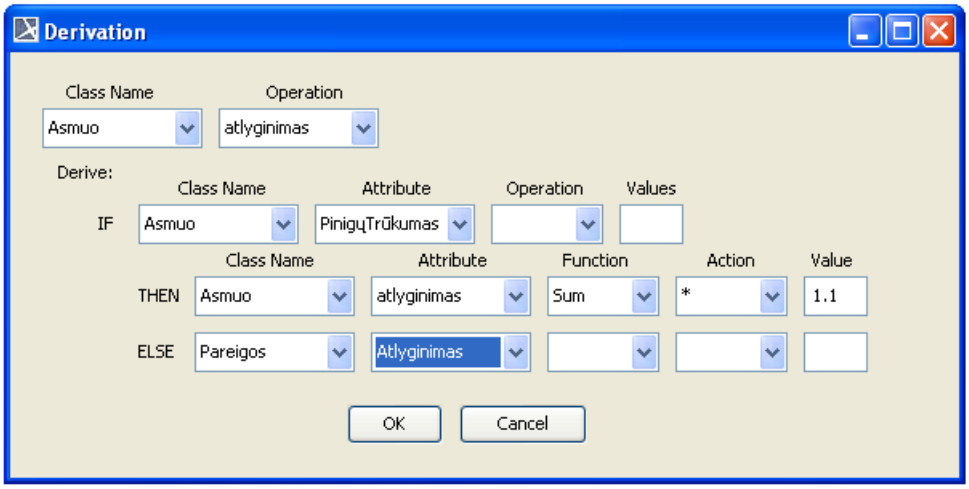
Iš 38 paveikslo matyti, kad veiksmo laukas – tai standartinės operacijos, kurios buvo pateiktos visuose anksčiau nagrinėtuose įrankiuose, praplėstos tokiais veiksmais, kaip: sudėtis, atimtis, daugyba ir dalyba. Jų reikalingumas bus iliustruotas žemiau pateiktame pavyzdyje.

Iš anksčiau pateiktų (36, 37 ir 38) paveikslėlių matyti, kad išvedimo išraiška turi standartinę ir gerai žinomą IF-THEN-ELSE struktūrą. Todėl galima iš karto pereiti prie konkretaus pavyzdžio ir pabandyti jį sugeneruoti OCL išvedimo išraiškos generavimo įrankiu.

Bus naudojamas analitinėje dalyje nagrinėtas pavyzdys:

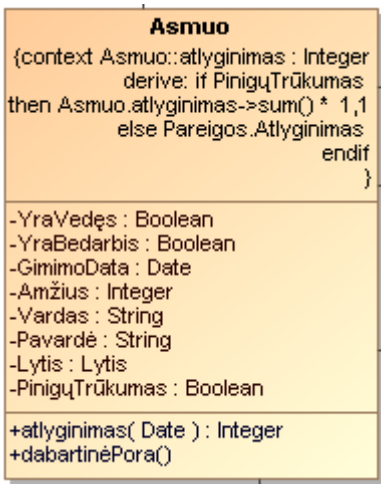
```
context Asmuo::atlyginimas : Integer
    derive: if PinigųTrūkumas
    then Asmuo.atlyginimas->sum() *  1,1
    else Pareigos.Atlyginimas
    endif
```

Šiame pavyzdyje nagrinėjama klasės „Asmuo“ operacija „atlyginimas“, tai ir turėtų būti užrašyta pirmojoje išvedimo išraiškos generavimo įrankio eilutėje. Be to, iš pavyzdžio matyti, kad atributais išvedimo išraiškoje gali būti laikomos ir operacijos. Išvedimo sąlyga turėtų skambėti taip: „Jeigu yra pinigų trūkumas, tuomet išvedamas bendras asmens atlyginimas, padidintas dešimčia procentų, jeigu nėra pinigų trūkumo, tuomet išvedamas asmens pareigų atlyginimas“. Visa tai iliustruota 39 paveiksle.



39 pav. Anksčiau pateiktas pavyzdys, užrašytas OCL išvedimo išraiškos generavimo įrankyje

Paspaudus mygtuką „GERAI“, būtent tokia OCL išraiška sugeneruojama į klasės diagramos klasę „Asmuo“ (40 pav.).



40 pav. Sugeneruota OCL išvedimo išraiška klasėje „Asmuo“

IŠVADOS

1. Buvo nustatyta, kad verslo taisyklės gali būti išreikštos savo komponentais, t.y. įvykiais, sąlygomis ir veiksmais. Tinkamas koncepcinis modelis turi atvaizduoti bent vieną iš šių komponentų. Be to, turi būti įmanoma apibrėžti taisyklių poveikį objektams ir priklausomybėms tarp verslo taisyklių. Vadovaujantis šiais kriterijais buvo išanalizuoti dažniausiai naudojami verslo taisyklių vaizdavimo metodai, jų privalumai ir trūkumai.
2. Palyginus visus išanalizuotus metodus, buvo nustatyta, kad esybių ryšių metodas yra vienintelis, kuris suteikia aiškią verslo taisyklių modeliavimo konstrukciją. Tačiau šis metodas gali būti naudojamas tiktai struktūriniams taisyklių aspektams, o dinamiškos tarpusavio priklausomybės ignoruojamos. Verslo taisyklių modeliavimas jų komponentais geriausiai matomas OOA&D metodo schemose, kurios yra artimos CPM Merise. Galingiausia sistemos struktūros ir dinamikos modeliavimo kombinacija pateikta BIER metode. Tačiau pagrindinės verslo taisyklės yra pateikiamos tik netiesiogiai.

   Iš palyginimo rezultatų nustatyta, kad beveik visi išnagrinėti metodai ir jų koncepciniai modeliai nepajėgia visiškai atvaizduoti visų įmonėje egzistuojančių verslo taisyklių. Tačiau, remiantis šiais rezultatais, nustatyta, kad pats tinkamiausias metodas taisyklėms atvaizduoti yra UML klasių diagrama naudojama kartu su OCL kalba.
3. Išnagrinėjus priemones, kuriomis galima užrašyti OCL išraiškas, buvo nustatytos tokios problemos:
   3.1. sudėtinga rasti *MagicDraw* priemonėje esantį OCL išraiškų rašymo įrankį,
   3.2. nors UML yra objektinė kalba, ji naudoja sintaksiškai sudėtingus OCL ribojimus, kurių sudarymo struktūros naudotojas turi mokytis,
   3.3. nėra sukurta priemonė, kuri užtikrintų užrašytos OCL išraiškos teisingumą,
4. Dėl minėtų priežasčių buvo pasiūlytas įrankis, kuriuo galima lengvai ir patogiai sugeneruoti teisingas OCL išraiškas nežinant OCL kalbos sintaksės. Įrankis leidžia generuoti keturias dažniausiai naudojamas OCL išraiškas:
   4.1. OCL invariantą,
   4.2. prieš ir po sąlygos išraiškas,
   4.3. užklausos išraišką,
   4.4. išvedimo išraišką.

ŽODYNAS

Infiksinis operatorius – operatorius rašomas tarp dviejų operandų

Invariantas – programos teiginys, kuris atliekant programą turi būti visada teisingas. Vartojamas programos teisingumui patikrinti. Yra kai kuriose moderniose programavimo kalbose

Literalai – ženklai, skaičiai, žodžiai, tekstai, žymintys jų pačių reikšmes. Literalų pavyzdžiai: 'a', 125, 'žmogus', 'aš ir tu'. Įprasta, kad skaičiai vaizduoja patys save. Raidės, žodžiai gali būti kitų reikšmių žymenys. Kai jie vartojami kaip literalai, tai kokiu nors būdu išskiriami, dažniausiai rašomi tarp kabučių arba apostrofų

Postfiksinis žymuo – operacijos ženklas, rašomas po operandų

Sintropija –antros kartos objektiškai orientuotas analizės ir programinės įrangos projektavimo metodas, sukurtas *Object Designers Limited* Jungtinėje Karalystėje 1990 m.

LITERATŪRA

1. Herbst H., Knolmayer G., Myrach T. ir Schlesinger M. The Specification of Business Rules: A Comparison of Selected Methodologies [interaktyvus]. Liumburgo universitetas, 1994. 29 – 46 psl. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.wi-inf.uni-duisburg-essen.de/MobisPortal/upload/cris94.pdf
2. Le Vie D. S. Understanding Data Flows Diagrams [interaktyvus]. Integrated Concepts, Inc., 2000. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.stc.org/confproceed/2000/PDFs/00098.pdf
3. Tannier C., Josselin D., Bruch H., Frankhauser P. Realization of Conceptual Data and Processing Models Using the MERISE Formalism [interaktyvus]. Kinijos universitetas, 1998. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://thema.univ-fcomte.fr/IMG/pdf/HongKong1998.pdf
4. Fowler M. UML Distilled [interaktyvus]. 3 rd. ed. Addison Wesley Professional, 2003. 83 p. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://galeb.etf.bg.ac.yu/~pd030119d/ir4ps/Addison%20Wesley%20-%20UML%20Distilled%20(3rd%20edition).pdf
5. Klauck C., Muller H. Formal Business Process Engineering Based on Graph Grammars [interaktyvus]. Bremeno universitetas, 1996. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://citeseer.ist.psu.edu/cache/papers/cs/419/ftp:zSzzSzftp.agki.tzi.dezSzpubzSzpaperszSzckformpeogg.pdf/ch96formal.pdf
6. Harel D., Gery E. Executable Object Modeling with Statecharts [interaktyvus]. Weizmann‘o mokslo institutas, 1997. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://citeseer.ist.psu.edu/cache/papers/cs/21230/http:zSzzSzselab.korea.ac.krzSzselabzSzresourceszSzsezSzpaperszSzobjectszSzharel97.pdf/harel97executable.pdf
7. De Backer M., Snoeck M. Business Process Verification: a Petri Net Approach [interaktyvus]. Leuven‘o katalikiškas universitetas, 2007. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.econ.kuleuven.be/fetew/pdf_publicaties/KBI_0705.pdf
8. Gomik D. Entity Relationship Modeling with UML [interaktyvus]. IBM, 2003. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www-128.ibm.com/developerworks/rational/library/content/03July/2500/2785/2785_uml.pdf
9. Chen P. Entity – Relationship Model [interaktyvus]. Luizianos universitetas, 2002. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą:

http://www.csc.lsu.edu/~chen/pdf/Chen_Pioneers.pdf

10. Hay D., Anderson Healy K. Defining Business Rules [interaktyvus]. GUIDE verslo taisyklių projektas, 2000. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://rewerse.net/downloads/BRG-whatisBR_3ed.pdf
11. Tavater K., Wagner G. Agent-Oriented Business Rules: Deontic Assignments [interaktyvus]. Eindhoveno universitetas, 2001. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.informatik.tu-cottbus.de/~gwagner/papers/OES-SEO.pdf
12. Vasilecas O., Smaižys A. Business Rule Based Configuration Management and Software System Implementation Using Decision Tables [interaktyvus]. Vilniaus Gedimino technikos universitetas, 2007. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.adbis.org/docs/lp/3.pdf
13. Vasilecas O., Lebedys E. Business Rules Repository for Business Rules Represented Using UML [interaktyvus]. Tarptautinė kompiuterių sistemų ir technologijų konferencija, 2005. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://ecet.ecs.ru.acad.bg/cst05/Docs/cp/SII/II.5.pdf
14. Halpin T. Verbalizing Business Rules: Part 7 [interaktyvus]. Northface'o universitetas, 2004. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.orm.net/pdf/VBR7.pdf
15. Halpin T. Entity Relationship Modeling From an ORM Perspective: Part 3 [interaktyvus]. Microsoft korporacija, 2000. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.orm.net/pdf/JCM13.pdf
16. Kardasis P., Loucopoulos P. Expressing and Organising Business Rules [interaktyvus]. Atėnų universitetas, 2003. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0B-4BV48JP-1&_user=986143&_coverDate=09%2F01%2F2004&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000049865&_version=1&_urlVersion=0&_userid=986143&md5=85cf88246aaa04bb0181751f6a4be53d
17. Arsanjani A. Rule Object 2001: A Pattern Language for Adaptive and Scalable Business Rule Construction [interaktyvus]. IBM, 2001. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/AArsanjani1/PLoP2001_AArsanjani1_0.pdf
18. Dorsey P. The Business Rules Approach to Systems Development [interaktyvus]. Dulcian, Inc., 2002. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą:

http://www.dulcian.com/BRIM%20Documents/The%20Business%20Rules%20Approach%20to%20Systems%20Development.htm

19. Sengupta A., Mohan S., Doshi R. XER – Extensible Entity Relationship Modeling [interaktyvus]. DeepX Ltd., 2006. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.idealliance.org/papers/dx_xml03/papers/06-01-01/06-01-01.pdf
20. Object Management Group Object Constraint Language version 2.0 [interaktyvus]. Object Management Group, 2006. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.omg.org/docs/formal/06-05-01.pdf
21. Toni Mancini Finite Satisfiability of UML class diagrams by Constraint Programming [interaktyvus]. Romos universitetas, 2004. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.dis.uniroma1.it/~tmancini/research/allpubs/WI5%20%5Bmancini,04,dl%5D.pdf
22. Robert C. Martin UML Tutorial: Part 1 – Class Diagrams [interaktyvus]. Object Mentor, 1997. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.objectmentor.com/resources/articles/umlClassDiagrams.pdf
23. Mandar Chitnis, Pravin Tiwari ir Lakshmi Ananthamurthy The UML Class Diagram: Part 1 [interaktyvus]. Developer Solutions, 2003. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.developer.com/design/article.php/2206791
24. Holger Eichelberger ir Jurgen Wolff von Gudenberg UML Class Diagrams - State of the Art in Layout Techniques [interaktyvus]. Wurzburg‘o universitetas, 2003. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://webhome.cs.uvic.ca/~mstorey/vissoft2003/submissions/eichelberger_pospaper.pdf
25. Terry Halpin Verbalizing Business Rules: Part 9 [interaktyvus]. Northface‘o University, 2005. [žiūrėta 2008 m. birželio 22 d.]. Prieiga per internetą: http://www.orm.net/pdf/VBR9.pdf
26. David C. Hay Object Oriented Data Modeling: Entity Life Histories [interaktyvus]. Essential Strategies, Inc. [žiūrėta 2008 m. sausio 13 d.]. Prieiga per internetą: http://www.essentialstrategies.com/documents/elhs.pdf
27. David C. Hay Requirements Analisys. From Bussiness Views to Architecture. New Jersey, 2003. 304 – 339 psl.

PRIEDAS. Papildinio programinis kodas

Klasės ActionExample programinis kodas

```
package myPlugin;

import java.util.List;
import javax.swing.ImageIcon;
import com.nomagic.magicdraw.actions.*;
import com.nomagic.magicdraw.plugins.Plugin;
import com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction;
import com.nomagic.magicdraw.uml.DiagramType;

public class ActionsExample extends Plugin
{
   public void init()
   {
      ActionsConfiguratorsManager manager = ActionsConfiguratorsManager.getInstance();

      // ADDING ACTION TO CLASS DIAGRAM
      DiagramAction dAction = new DiagramAction();
      dAction.setLargeIcon( new ImageIcon(getClass().getResource("diagram_toolbar_icon.gif")));
      dAction.setDescription("OCL expression");
      DiagramConfigurator diagramConfigurator = new DiagramConfigurator(dAction);
      manager.addDiagramContextConfigurator(DiagramType.UML_CLASS_DIAGRAM, diagramConfigurator);
      manager.addDiagramShortcutsConfigurator(DiagramType.UML_CLASS_DIAGRAM, diagramConfigurator);

      manager.addDiagramToolbarConfigurator(DiagramType.UML_CLASS_DIAGRAM, diagramConfigurator);
   }
   public boolean close()
   {
      return true;
   }
   public boolean isSupported()
   {
      return true;
   }
}
```

Klasės BrowserAction programinis kodas

```
package myPlugin;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction;
import com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement;
import com.nomagic.magicdraw.uml.symbols.PresentationElement;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import java.awt.Image;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.util.Iterator;
import java.util.List;

public class BrowserAction extends DefaultBrowserAction
{

   public BrowserAction()
   {
      super("", "BrowserAction Example", KeyStroke.getKeyStroke(KeyEvent.VK_E,
KeyEvent.ALT_MASK+KeyEvent.SHIFT_MASK), null);
   }
```

```
   public void actionPerformed(ActionEvent e)
   {
      Image img = new ImageIcon(BrowserAction.class.getResource("mduml.jpg")).getImage();
      JFrame pagrindinis = new Pagrindinis();
      pagrindinis.setTitle("OCL expression");
      pagrindinis.setIconImage(img);

       com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class checkClass;
        String classList = "";
        DiagramPresentationElement diagramPresentationElement =
Application.getInstance().getProject().getActiveDiagram();
        List diagramelements = diagramPresentationElement.getPresentationElements();

        for(Iterator it = diagramelements.iterator();it.hasNext();){
           PresentationElement classPressentationElement = (PresentationElement) it.next();
           Element el = classPressentationElement.getActualElement();
           if (el instanceof com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class){
              checkClass = (com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class) el;
              classList = classList + "\n" + checkClass.getName();
           }
        }
        JOptionPane.showMessageDialog(null, classList);

  }
   @Override
   public void updateState()
   {
      setEnabled(getTree().getSelectedNode() != null );
   }

}
```

Klasės Pagrindinis programinis kodas

```
package myPlugin;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Pagrindinis extends JFrame{
   public Pagrindinis(){

      Container c;
   JButton inv;
   JButton pre;
   JButton deriv;
   JButton cancel;
   JButton body;
      c = getContentPane();
      c.setLayout(null);//new FlowLayout());
      JLabel klase = new JLabel("What OCL expression you want to generate?");
      c.add(klase);
      klase.setBounds(40, 20, 300, 20);
      //mygtukas ok pasirinkus visus kriterijus sugeneruojams txt failiukas

      inv = new JButton("OCL Invariant");
      c.add(inv);
      inv.setBounds (new Rectangle (50, 70, 200, 25));
      inv.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {
                 Image img = new ImageIcon(BrowserAction.class.getResource("mduml.jpg")).getImage();
```

```
            JFrame invariant = new Invariantas ();
            invariant.setTitle("OCL invariant");
            invariant.setIconImage(img);
            Pagrindinis.this.dispose();

          }
        }
       );
   pre = new JButton("OCL Pre/Post-Condition");
   c.add(pre);
   pre.setBounds (new Rectangle (50, 100, 200, 25));
   pre.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
           Image img = new ImageIcon(BrowserAction.class.getResource("mduml.jpg")).getImage();
           JFrame precondition = new PreCondition ();
           precondition.setTitle("OCL Pre/Post-Condition");
           precondition.setIconImage(img);
           Pagrindinis.this.dispose();

          }
        }
       );
   deriv = new JButton("Derivation");
   c.add(deriv);
   deriv.setBounds (new Rectangle (50, 160, 200, 25));
   deriv.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
           Image img = new ImageIcon(BrowserAction.class.getResource("mduml.jpg")).getImage();
           JFrame derive = new Derivation();
           derive.setTitle("Derivation");
           derive.setIconImage(img);
           Pagrindinis.this.dispose();

          }
        }
       );
   body = new JButton("Body Expression");
   c.add(body);
   body.setBounds (new Rectangle (50, 130, 200, 25));
   body.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
           Image img = new ImageIcon(BrowserAction.class.getResource("mduml.jpg")).getImage();
           JFrame BodyExpression = new BodyExp();
           BodyExpression.setTitle("Body Expression");
           BodyExpression.setIconImage(img);
           Pagrindinis.this.dispose();

          }
        }
       );
   //Cancel mygtukas
   cancel = new JButton("Close");
        cancel.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent ae) {
     Pagrindinis.this.dispose();
   }
  });
   c.add(cancel);
   cancel.setBounds (new Rectangle (100, 240, 100, 25));
setSize(300,330);
show();
}
```

```
}
```

Klasės Invariantas programinis kodas

```
package myPlugin;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class Invariantas extends JFrame{
   private JComboBox klasiuPavadinimai;
   private JComboBox atributuSarasas;
   private JComboBox operacijuSarasas;
   private Container c;
   private JButton ok;
   private JButton cancel;
   private JTextField text1, text2;
   private String kPavadinimai[]={"Asmuo","Bankas","Įmonė","Pareigos", "Santuoka"};
   private String atributai[]={"YraVedęs", "YraBedarbis", "GimimoData", "Amžius", "Vardas","Pavarde",
"Asmens_kodas", "Lytis"};
   private String operacijos[]={"","=", "!=","<", ">", ">=", "<=", "Between"};

   public Invariantas(){
      c = getContentPane();
      c.setLayout(null);//new FlowLayout());
      klasiuPavadinimai = new JComboBox(kPavadinimai);
      //////////////////////////
      atributuSarasas = new JComboBox (atributai);
      c.add(atributuSarasas);
      atributuSarasas.setBounds(130, 35, 100, 25);
      klasiuPavadinimai.addActionListener(
            new ActionListener() {
             public void actionPerformed(ActionEvent e) {
                if (klasiuPavadinimai.getSelectedIndex() == 0) {
                atributuSarasas.removeAllItems();
                atributuSarasas.addItem("YraVedęs");
                atributuSarasas.addItem("YraBedarbis");
                atributuSarasas.addItem("GimimoData");
                atributuSarasas.addItem("Amžius");
                atributuSarasas.addItem("Vardas");
                atributuSarasas.addItem("Pavardė");
                atributuSarasas.addItem("Lytis");
               }
                else if (klasiuPavadinimai.getSelectedIndex() == 1) {
                atributuSarasas.removeAllItems();
                atributuSarasas.addItem("SąskaitosNumeris");
                }
                else if (klasiuPavadinimai.getSelectedIndex() == 2) {
                atributuSarasas.removeAllItems();
                atributuSarasas.addItem("Pavadinimas");
                atributuSarasas.addItem("DarbuotojųSkaičius");
                }
                else if (klasiuPavadinimai.getSelectedIndex() == 3) {
                atributuSarasas.removeAllItems();
                atributuSarasas.addItem("Pareigų_pav");
                atributuSarasas.addItem("PradžiosData");
                atributuSarasas.addItem("Atlyginimas");
                }
                else if (klasiuPavadinimai.getSelectedIndex() == 4) {
                atributuSarasas.removeAllItems();
                atributuSarasas.addItem("Vieta");
```

```
            atributuSarasas.addItem("Data");
            atributuSarasas.addItem("Išsiskyrė");
            }
        }
     }
    );

JLabel klase = new JLabel("Class Name");
c.add(klase);
klase.setBounds(35, 10, 100, 20);
JLabel atributas = new JLabel("Attribute");
c.add(atributas);
atributas.setBounds(155, 10, 100, 20);
JLabel operacija = new JLabel("Operation");
c.add(operacija);
operacija.setBounds(250, 10, 80, 20);
JLabel reiksme = new JLabel("Values");
c.add(reiksme);
reiksme.setBounds(330, 10, 100, 20);
c.add(klasiuPavadinimai);
klasiuPavadinimai.setBounds(20, 35, 100, 25);

operacijuSarasas = new JComboBox (operacijos);
c.add(operacijuSarasas);
operacijuSarasas.setBounds(240, 35, 80, 25);

text1 = new JTextField(20);
c.add(text1);
text1.setBounds(330, 35, 40, 25);

//text field 2 of 4
operacijuSarasas.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if (operacijuSarasas.getSelectedIndex() == 7) {
             text2 = new JTextField(20);
             c.add(text2);
             text2.setBounds(380, 35, 40, 25);}
           //else if (operationsList.getSelectedIndex() == 0) c.remove(text2);
        }
}
);

//mygtukas ok pasirinkus visus kriterijus sugeneruojams txt failiukas

ok = new JButton("OK");
c.add(ok);
ok.setBounds (new Rectangle (140, 80, 60, 25));
   ok.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            FileOutputStream out;
            PrintStream p;
              try
            {
                out = new FileOutputStream("myfile.txt");
                p = new PrintStream( out );
                      Invariantas.this.dispose();
            }
              catch (Exception ezz)
            {
             System.out.println (ezz.getMessage());
          }
```

```
            }
          }
         );

      //Cancel mygtukas. Uzdaromas langas
        cancel = new JButton("Cancel");
             cancel.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        Invariantas.this.dispose();
      }
     });
      c.add(cancel);
      cancel.setBounds (new Rectangle (210, 80, 80, 25));

      //Helpas
      //c.add(new JButton("Help"));

   setSize(450,150);
   show();

   }
}
```

Klasės PreCondition programinis kodas

```
package myPlugin;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;


 class PreCondition extends JFrame{
   private JComboBox classesNameList;
   private JComboBox operationsNameList;
   private JComboBox operationsList;
   private JComboBox operationsList2;
   private JComboBox returnTypeList;
   private JComboBox typeList;
   private JComboBox conditionTypeList;
   private JComboBox conditionTypeList2;
   private JComboBox parameterList;
   private JComboBox parameterList2;
   private Container c;
   private JButton ok;
   private JButton cancel;
   private JTextField text1;
   private JTextField text2;
   private JTextField text3;
   private JTextField text4;
   private String conditionType[]={"Pre","Post"};
   private String conditionType2[]={"","Pre","Post"};
   private String classesName[]={"Asmuo","Bankas","Įmonė","Pareigos", "Santuoka"};
   private String operationsName[]={"Atlyginimas"};
   private String operations[]={"","=", "!=","<", ">", ">=", "<=", "Between"};
   private String returnType[]={"Integer", "Char", "Real", "Date", "Boolean"};
   private String parameter[]={"Date"};
   private String parameter2[]={""};
   private String type[]={"Date"};
```

```
public PreCondition(){
    c = getContentPane();
    c.setLayout(null);//new FlowLayout());
    classesNameList = new JComboBox(classesName);
    c.add(classesNameList);
    classesNameList.setBounds(20, 35, 100, 25);
    operationsNameList = new JComboBox (operationsName);
    c.add(operationsNameList);
    operationsNameList.setBounds(130, 35, 100, 25);
    classesNameList.addActionListener(
          new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              if (classesNameList.getSelectedIndex() == 0) {
                operationsNameList.removeAllItems();
                operationsNameList.addItem("atlyginimas");
                operationsNameList.addItem("dabartinėPora");}
               if (classesNameList.getSelectedIndex() == 1) {
                operationsNameList.removeAllItems();
              }
              if (classesNameList.getSelectedIndex() == 2) {
                operationsNameList.removeAllItems();
              }
           }
    }
    );
    typeList = new JComboBox(type);
    returnTypeList = new JComboBox(returnType);


    operationsNameList.addActionListener(
          new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              if (operationsNameList.getSelectedIndex() == 0) {
              typeList.removeAllItems();
              typeList.addItem("Date");
              returnTypeList.addItem("Integer");}
              else if (operationsNameList.getSelectedIndex() != 0)
              {
              typeList.removeAllItems();
              typeList.addItem("");
              returnTypeList.removeAllItems();
              returnTypeList.addItem("");
               }
           }
    }
    );

    //condition types list
    conditionTypeList = new JComboBox(conditionType);
    c.add(conditionTypeList);
    conditionTypeList.setBounds(200, 80, 60, 25);

    //parameter list
    parameterList = new JComboBox(parameter);
    c.add(parameterList);
    parameterList.setBounds(270, 80, 60, 25);
    conditionTypeList.addActionListener(
          new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              if (conditionTypeList.getSelectedIndex() == 0) {
                 parameterList.removeAllItems();
                 parameterList.addItem("D");}
              if (conditionTypeList.getSelectedIndex() == 1) {
```

```
                parameterList.removeAllItems();
                parameterList.addItem("Result");}
            }
}
);

//operation list
operationsList = new JComboBox (operations);
c.add( operationsList);
operationsList.setBounds(340, 80, 80, 25);

//Text fields
//text field 1 of 4
text1 = new JTextField(20);
c.add(text1);
text1.setBounds(430, 80, 40, 25);

//text field 2 of 4
operationsList.addActionListener(
        new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            if (operationsList.getSelectedIndex() == 6) {
              text2 = new JTextField(20);
              c.add(text2);
              text2.setBounds(480, 80, 40, 25);}
            //else if (operationsList.getSelectedIndex() == 0) c.remove(text2);
        }
}
);

//second condition types list
conditionTypeList2 = new JComboBox(conditionType2);
c.add(conditionTypeList2);
conditionTypeList2.setBounds(200, 110, 60, 25);

//second parameters list
parameterList2 = new JComboBox(parameter2);
c.add(parameterList2);
parameterList2.setBounds(270, 110, 60, 25);
conditionTypeList2.addActionListener(
        new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            if (conditionTypeList2.getSelectedIndex() == 1) {
                parameterList2.removeAllItems();
                parameterList2.addItem("Data");}
            if (conditionTypeList2.getSelectedIndex() == 2) {
                parameterList2.removeAllItems();
                parameterList2.addItem("Result");}
            if (conditionTypeList2.getSelectedIndex() == 0) {
                parameterList2.removeAllItems();
                }
        }
}
);
//second operation list
operationsList2 = new JComboBox(operations);
c.add(operationsList2);
operationsList2.setBounds(340, 110, 80, 25);

//text field 3 of 4
 text3 = new JTextField(20);
 c.add(text3);
 text3.setBounds(430, 110, 40, 25);
```

```
//text field 4 of 4
operationsList2.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if (operationsList2.getSelectedIndex() == 6) {
             text4 = new JTextField(20);
             c.add(text4);
             text4.setBounds(480, 110, 40, 25);}

       }
}
);


//Labels
//Class label
JLabel classLabel = new JLabel("Class Name");
c.add(classLabel);
classLabel.setBounds(35, 10, 100, 20);

//condition label
JLabel conditionLabel = new JLabel("Condition");
c.add(conditionLabel);
conditionLabel.setBounds(205, 60, 100, 20);

//parameter label
JLabel parameterLabel = new JLabel("Parameter");
c.add(parameterLabel);
parameterLabel.setBounds(275, 60, 100, 20);

//Operation label
JLabel operationLabel = new JLabel("Operation");
c.add(operationLabel);
operationLabel.setBounds(355, 60, 80, 20);

//Value label
JLabel valueLabel = new JLabel("Values");
c.add(valueLabel);
valueLabel.setBounds(435, 60, 100, 20);

//class operation label
JLabel classOperation = new JLabel("Class Operation");
c.add(classOperation);
classOperation.setBounds(140, 10, 100, 20);

//OK button generate OCL expression
ok = new JButton("OK");
c.add(ok);
ok.setBounds (new Rectangle (190, 150, 60, 25));
   ok.addActionListener(
       new ActionListener() {
         public void actionPerformed(ActionEvent e) {
             FileOutputStream out;
             PrintStream p;
               try
             {
                 out = new FileOutputStream("precondition.txt");
                 p = new PrintStream( out );
                       p.println("pre: result"); //+ operacijuSarasas.getSelectedItem() + " " + tekstinis.getText());
                       PreCondition.this.dispose();
              }
                catch (Exception ezz)
```

```
                {
                System.out.println (ezz.getMessage());
            }
           }
          }
         );



      //Cancel mygtukas. Uzdaromas langas
        cancel = new JButton("Cancel");
             cancel.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        PreCondition.this.dispose();
      }
     });
      c.add(cancel);
      cancel.setBounds (new Rectangle (260, 150, 80, 25));

      //Helpas
      //c.add(new JButton("Help"));

   setSize(540,220);
   show();

   }
}
```

Klasės BodyExp programinis kodas

```
package myPlugin;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;


 class BodyExp extends JFrame{
   private JComboBox classesNameList;
   private JComboBox operationsNameList;
   private JComboBox operationsList;
   private JComboBox operationsList2;
   private JComboBox returnTypeList;
   private JComboBox typeList;
   private Container c;
   private JButton ok;
   private JButton cancel;
   private JTextField text1;
   private JTextField text2;
   private JTextField text3;
   private JTextField text4;
   private String classesName[]={"Asmuo","Bankas","Įmonė","Pareigos", "Santuoka"};
   private String operationsName[]={"atlyginimas","dabartinėPora"};
   private String operations[]={"","","=", "!=","<", ">", ">=", "<=", "Between"};
   private String attributes[]={"YraVedęs", "YraBedarbis", "GimimoData", "Amžius", "Vardas","Pavarde",
"Asmens_kodas", "Lytis"};
   private JComboBox bodyClassesList;
   private JComboBox returnedClassesList;
   private JComboBox attributesList;
   private JComboBox preClassesList;
   private JComboBox preAttributesList;
```

```
public BodyExp(){
   c = getContentPane();
   c.setLayout(null);//new FlowLayout());

//Body class mane list
   bodyClassesList = new JComboBox(classesName);
   c.add(bodyClassesList);
   bodyClassesList.setBounds(130, 80, 100, 25);
   attributesList = new JComboBox (attributes);
   c.add(attributesList);
   attributesList.setBounds(240, 80, 100, 25);
   bodyClassesList.addActionListener(
         new ActionListener() {
          public void actionPerformed(ActionEvent e) {
             if (bodyClassesList.getSelectedIndex() == 0) {
              attributesList.removeAllItems();
              attributesList.addItem("YraVedęs");
              attributesList.addItem("YraBedarbis");
              attributesList.addItem("GimimoData");
              attributesList.addItem("Amžius");
              attributesList.addItem("Vardas");
             attributesList.addItem("Pavardė");
              attributesList.addItem("Lytis");
             }
              else if (bodyClassesList.getSelectedIndex() == 1) {
              attributesList.removeAllItems();
              attributesList.addItem("SąskaitosNumeris");
              }
              else if (bodyClassesList.getSelectedIndex() == 2) {
              attributesList.removeAllItems();
              attributesList.addItem("Pavadinimas");
              attributesList.addItem("DarbuotojųSkaičius");
              }
              else if (bodyClassesList.getSelectedIndex() == 3) {
              attributesList.removeAllItems();
              attributesList.addItem("Pareigų_pav");
              attributesList.addItem("PradžiosData");
              attributesList.addItem("Atlyginimas");
              }
              else if (bodyClassesList.getSelectedIndex() == 4) {
              attributesList.removeAllItems();
              attributesList.addItem("Vieta");
              attributesList.addItem("Data");
              attributesList.addItem("Išsiskyrė");
              }
          }
       }
      );
   returnedClassesList = new JComboBox(classesName);
   c.add(returnedClassesList);
   returnedClassesList.setBounds(320, 35, 100, 25);
 //Main class name list
   classesNameList = new JComboBox(classesName);
   c.add(classesNameList);
   classesNameList.setBounds(20, 35, 100, 25);
   operationsNameList = new JComboBox (operationsName);
   c.add(operationsNameList);
   operationsNameList.setBounds(130, 35, 100, 25);
   classesNameList.addActionListener(
         new ActionListener() {
          public void actionPerformed(ActionEvent e) {
             if (classesNameList.getSelectedIndex() == 0) {
```

```
            operationsNameList.removeAllItems();
            operationsNameList.addItem("atlyginimas");
            operationsNameList.addItem("dabartinėPora");}
           if (classesNameList.getSelectedIndex() == 1) {
            operationsNameList.removeAllItems();
          }
          if (classesNameList.getSelectedIndex() == 2) {
            operationsNameList.removeAllItems();
          }
        }
}
);
//PRE class mane list
preClassesList = new JComboBox(classesName);
c.add(preClassesList);
preClassesList.setBounds(130, 110, 100, 25);
preAttributesList = new JComboBox (attributes);
c.add(preAttributesList);
preAttributesList.setBounds(240, 110, 100, 25);
preClassesList.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if ( preClassesList.getSelectedIndex() == 0) {
           preAttributesList.removeAllItems();
           preAttributesList.addItem("YraVedęs");
           preAttributesList.addItem("YraBedarbis");
           preAttributesList.addItem("GimimoData");
           preAttributesList.addItem("Amžius");
           preAttributesList.addItem("Vardas");
           preAttributesList.addItem("Pavardė");
           preAttributesList.addItem("Lytis");
          }
           else if (preClassesList.getSelectedIndex() == 1) {
           preAttributesList.removeAllItems();
           preAttributesList.addItem("SąskaitosNumeris");
           }
           else if (preClassesList.getSelectedIndex() == 2) {
           preAttributesList.removeAllItems();
           preAttributesList.addItem("Pavadinimas");
           preAttributesList.addItem("DarbuotojųSkaičius");
           }
           else if ( preClassesList.getSelectedIndex() == 3) {
           preAttributesList.removeAllItems();
           preAttributesList.addItem("Pareigų_pav");
           preAttributesList.addItem("PradžiosData");
           preAttributesList.addItem("Atlyginimas");
           }
           else if (preClassesList.getSelectedIndex() == 4) {
           preAttributesList.removeAllItems();
           preAttributesList.addItem("Vieta");
           preAttributesList.addItem("Data");
           preAttributesList.addItem("Išsiskyrė");
           }
        }
     }
    );
//operation list
operationsList = new JComboBox (operations);
c.add( operationsList);
operationsList.setBounds(350, 80, 70, 25);

//Text fields
//text field 1 of 4
```

```
text1 = new JTextField(20);
c.add(text1);
text1.setBounds(430, 80, 45, 25);

//text field 2 of 4
operationsList.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if (operationsList.getSelectedIndex() == 7) {
             text2 = new JTextField(20);
             c.add(text2);
             text2.setBounds(480, 80, 40, 25);}
           //else if (operationsList.getSelectedIndex() == 0) c.remove(text2);
        }
}
);

//second operation list
operationsList2 = new JComboBox(operations);
c.add(operationsList2);
operationsList2.setBounds(350, 110, 70, 25);

//text field 3 of 4
text3 = new JTextField(20);
c.add(text3);
text3.setBounds(430, 110, 45, 25);

//text field 4 of 4
operationsList2.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if (operationsList2.getSelectedIndex() == 7) {
             text4 = new JTextField(20);
             c.add(text4);
             text4.setBounds(480, 110, 40, 25);}

        }
}
);
//Labels
//Class label
JLabel classLabel = new JLabel("Class Name");
c.add(classLabel);
classLabel.setBounds(35, 15, 100, 20);

//condition label
JLabel conditionLabel = new JLabel("Class Name");
c.add(conditionLabel);
conditionLabel.setBounds(150, 60, 100, 20);

//parameter label
JLabel parameterLabel = new JLabel("Attribute");
c.add(parameterLabel);
parameterLabel.setBounds(270, 60, 100, 20);

//Operation label
JLabel operationLabel = new JLabel("Operation");
c.add(operationLabel);
operationLabel.setBounds(355, 60, 80, 20);

//Value label
JLabel valueLabel = new JLabel("Values");
c.add(valueLabel);
```

```
valueLabel.setBounds(435, 60, 100, 20);

//class operation label
JLabel classOperation = new JLabel("Class Operation");
c.add(classOperation);
classOperation.setBounds(140, 15, 100, 20);

JLabel Condition = new JLabel("Pre-Condition:");
c.add(Condition);
Condition.setBounds(54, 105, 80, 20);
JLabel ifNeeded = new JLabel("(If needed)");
c.add(ifNeeded);
ifNeeded.setBounds(58, 120, 70, 20);

JLabel Parameter = new JLabel("Param.");
c.add(Parameter);
Parameter.setBounds(240, 115, 70, 20);

JLabel BodyExpres = new JLabel("Body Expression:");
c.add(BodyExpres);
BodyExpres.setBounds(40, 80, 100, 20);

//returned class data label
JLabel returnedClass = new JLabel("Returned Class Data");
c.add(returnedClass);
returnedClass.setBounds(320, 15, 100, 20);


//OK button generate OCL expression
ok = new JButton("OK");
c.add(ok);
ok.setBounds (new Rectangle (190, 150, 60, 25));
   ok.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            FileOutputStream out;
            PrintStream p;
              try
            {
                out = new FileOutputStream("precondition.txt");
                p = new PrintStream( out );
                     BodyExp.this.dispose();
            }
              catch (Exception ezz)
            {
             System.out.println (ezz.getMessage());
          }
         }
       }

       );
//Cancel mygtukas. Uzdaromas langas
   cancel = new JButton("Cancel");
        cancel.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ae) {
  BodyExp.this.dispose();
}
});
c.add(cancel);
cancel.setBounds (new Rectangle (260, 150, 80, 25));

//Helpas
//c.add(new JButton("Help"));
```

```
    setSize(540,220);
    show();

    }
 }
```

Klasės Derivation programinis kodas

```
package myPlugin;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

class Derivation extends JFrame{
    private JComboBox classesNameList;
    private JComboBox operationsNameList;
    private JComboBox operationsList;
    private JComboBox returnTypeList;
    private JComboBox typeList;
    private Container c;
    private JButton ok;
    private JButton cancel;
    private JTextField text1,text2, text3, text4, text5, text6;
    private String classesName[]={"Asmuo","Bankas","Įmonė","Pareigos", "Santuoka"};
    private String operationsName[]={"atlyginimas", "dabartinėPora"};
    private String operations[]={"","=", "!=","<", ">", ">=", "<=", "Between"};
    private String attributes[]={"YraVedęs", "YraBedarbis", "GimimoData", "Amžius", "Vardas","Pavarde",
"Asmens_kodas", "Lytis","PinigųTrūkumas","atlyginimas","dabartinėPora"};
    private JComboBox ifClassesList;
    private JComboBox thenClassesList;
    private JComboBox elseClassesList;
    private JComboBox actionList;
    private JComboBox actionList2;
    private JComboBox attributesList;
    private JComboBox thenAttributesList;
    private JComboBox elseAttributesList;
    private JComboBox functionList, functionList2;
    private String actions[]={"","=", "!=","<", ">", ">=", "<=", "Between","+","-","*","/"};
    private String functions[]={"","Sum", "Count","Avg"};

    public Derivation(){
        c = getContentPane();
        c.setLayout(null);//new FlowLayout());

    //IF class mane list
        ifClassesList = new JComboBox(classesName);
        c.add(ifClassesList);
        ifClassesList.setBounds(80, 95, 100, 25);
        attributesList = new JComboBox (attributes);
        c.add(attributesList);
        attributesList.setBounds(190, 95, 100, 25);
        ifClassesList.addActionListener(
              new ActionListener() {
               public void actionPerformed(ActionEvent e) {
                  if (ifClassesList.getSelectedIndex() == 0) {
                  attributesList.removeAllItems();
                  attributesList.addItem("YraVedęs");
                  attributesList.addItem("YraBedarbis");
                  attributesList.addItem("GimimoData");
                  attributesList.addItem("Amžius");
```

```
            attributesList.addItem("Vardas");
            attributesList.addItem("Pavardė");
            attributesList.addItem("Lytis");
            attributesList.addItem("PinigųTrūkumas");

           }
            else if (ifClassesList.getSelectedIndex() == 1) {
            attributesList.removeAllItems();
            attributesList.addItem("SąskaitosNumeris");
            }
            else if (ifClassesList.getSelectedIndex() == 2) {
            attributesList.removeAllItems();
            attributesList.addItem("Pavadinimas");
            attributesList.addItem("DarbuotojųSkaičius");
            }
            else if (ifClassesList.getSelectedIndex() == 3) {
            attributesList.removeAllItems();
            attributesList.addItem("Pareigų_pav");
            attributesList.addItem("PradžiosData");
            attributesList.addItem("Atlyginimas");
            }
            else if (ifClassesList.getSelectedIndex() == 4) {
            attributesList.removeAllItems();
            attributesList.addItem("Vieta");
            attributesList.addItem("Data");
            attributesList.addItem("Išsiskyrė");
            }
         }
      }
     );
//THEN class mane list
thenClassesList = new JComboBox(classesName);
c.add(thenClassesList);
thenClassesList.setBounds(115, 140, 100, 25);
thenAttributesList = new JComboBox (attributes);
c.add(thenAttributesList);
thenAttributesList.setBounds(225, 140, 100, 25);
thenClassesList.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if ( thenClassesList.getSelectedIndex() == 0) {
            thenAttributesList.removeAllItems();
            thenAttributesList.addItem("YraVedęs");
            thenAttributesList.addItem("YraBedarbis");
            thenAttributesList.addItem("GimimoData");
            thenAttributesList.addItem("Amžius");
            thenAttributesList.addItem("Vardas");
            thenAttributesList.addItem("Pavardė");
            thenAttributesList.addItem("Lytis");
            thenAttributesList.addItem("atlyginimas");
            thenAttributesList.addItem("dabartinėPora");
           }
            else if ( thenClassesList.getSelectedIndex() == 1) {
            thenAttributesList.removeAllItems();
            thenAttributesList.addItem("SąskaitosNumeris");
            }
            else if ( thenClassesList.getSelectedIndex() == 2) {
            thenAttributesList.removeAllItems();
            thenAttributesList.addItem("Pavadinimas");
            thenAttributesList.addItem("DarbuotojųSkaičius");
            }
            else if ( thenClassesList.getSelectedIndex() == 3) {
            thenAttributesList.removeAllItems();
```

```
            thenAttributesList.addItem("Pareigų_pav");
            thenAttributesList.addItem("PradžiosData");
            thenAttributesList.addItem("Atlyginimas");
            }
            else if ( thenClassesList.getSelectedIndex() == 4) {
            thenAttributesList.removeAllItems();
            thenAttributesList.addItem("Vieta");
            thenAttributesList.addItem("Data");
            thenAttributesList.addItem("Išsiskyrė");
            }
        }
     }
    );

//ELSE class mane list
elseClassesList = new JComboBox(classesName);
c.add(elseClassesList);
elseClassesList.setBounds(115, 175, 100, 25);
elseAttributesList = new JComboBox (attributes);
c.add(elseAttributesList);
elseAttributesList.setBounds(225, 175, 100, 25);
elseClassesList.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (elseClassesList.getSelectedIndex() == 0) {
            elseAttributesList.removeAllItems();
            elseAttributesList.addItem("YraVedęs");
            elseAttributesList.addItem("YraBedarbis");
            elseAttributesList.addItem("GimimoData");
            elseAttributesList.addItem("Amžius");
            elseAttributesList.addItem("Vardas");
            elseAttributesList.addItem("Pavardė");
            elseAttributesList.addItem("Lytis");
            elseAttributesList.addItem("atlyginimas");
            elseAttributesList.addItem("dabartinėPora");
           }
            else if (elseClassesList.getSelectedIndex() == 1) {
            elseAttributesList.removeAllItems();
            elseAttributesList.addItem("SąskaitosNumeris");
            }
            else if (elseClassesList.getSelectedIndex() == 2) {
            elseAttributesList.removeAllItems();
            elseAttributesList.addItem("Pavadinimas");
            elseAttributesList.addItem("DarbuotojųSkaičius");
            }
            else if (elseClassesList.getSelectedIndex() == 3) {
            elseAttributesList.removeAllItems();
            elseAttributesList.addItem("Pareigų_pav");
            elseAttributesList.addItem("PradžiosData");
            elseAttributesList.addItem("Atlyginimas");
            }
            else if (elseClassesList.getSelectedIndex() == 4) {
            elseAttributesList.removeAllItems();
            elseAttributesList.addItem("Vieta");
            elseAttributesList.addItem("Data");
            elseAttributesList.addItem("Išsiskyrė");
            }
        }
     }
    );

/*returnedClassesList = new JComboBox(classesName);
c.add(returnedClassesList);
```

```
  returnedClassesList.setBounds(320, 35, 100, 25); */
//Main class name list
  classesNameList = new JComboBox(classesName);
  c.add(classesNameList);
  classesNameList.setBounds(20, 35, 100, 25);
  operationsNameList = new JComboBox (operationsName);
  c.add(operationsNameList);
  operationsNameList.setBounds(130, 35, 100, 25);
  classesNameList.addActionListener(
        new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            if (classesNameList.getSelectedIndex() == 0) {
              operationsNameList.removeAllItems();
              operationsNameList.addItem("atlyginimas");
              operationsNameList.addItem("dabartinėPora");}
             if (classesNameList.getSelectedIndex() == 1) {
              operationsNameList.removeAllItems();
            }
            if (classesNameList.getSelectedIndex() == 2) {
              operationsNameList.removeAllItems();
            }
         }
  }
  );
  //operation list
  operationsList = new JComboBox (operations);
  c.add( operationsList);
  operationsList.setBounds(300, 95, 70, 25);

  //Text fields
  //text field 1 of 4
  text1 = new JTextField(20);
  c.add(text1);
  text1.setBounds(380, 95, 40, 25);

  //text field 2 of 4
  operationsList.addActionListener(
        new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            if (operationsList.getSelectedIndex() == 7) {
              text2 = new JTextField(20);
              c.add(text2);
              text2.setBounds(430, 95, 40, 25);}
            //else if (operationsList.getSelectedIndex() == 0) c.remove(text2);
         }
  }
  );
  //function list
  functionList = new JComboBox (functions);
  c.add(functionList);
  functionList.setBounds(335, 140, 70, 25);

  //action list
  actionList = new JComboBox (actions);
  c.add(actionList);
  actionList.setBounds(415, 140, 70, 25);

  //Text fields
  //text field 1 of 4
  text5 = new JTextField(20);
  c.add(text5);
  text5.setBounds(495, 140, 40, 25);
```

```
//text field 2 of 4
actionList.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if (actionList.getSelectedIndex() == 7) {
             text6 = new JTextField(20);
             c.add(text6);
             text6.setBounds(545, 140, 40, 25);}
           //else if (operationsList.getSelectedIndex() == 0) c.remove(text2);
        }
}
);

//function list
functionList2 = new JComboBox (functions);
c.add(functionList2);
functionList2.setBounds(335, 175, 70, 25);
//action list
actionList2 = new JComboBox (actions);
c.add(actionList2);
actionList2.setBounds(415, 175, 70, 25);

//Text fields
//text field 1 of 4
text3 = new JTextField(20);
c.add(text3);
text3.setBounds(495, 175, 40, 25);

//text field 2 of 4
actionList2.addActionListener(
       new ActionListener() {
        public void actionPerformed(ActionEvent e) {
           if (actionList2.getSelectedIndex() == 7) {
             text4 = new JTextField(20);
             c.add(text4);
             text4.setBounds(545, 140, 40, 25);}
           //else if (operationsList.getSelectedIndex() == 0) c.remove(text2);
        }
}
);


//Labels
//Class label
JLabel mainclassLabel = new JLabel("Class Name");
c.add(mainclassLabel);
mainclassLabel.setBounds(35, 15, 100, 20);

//if class name label
JLabel ifclassLabel = new JLabel("Class Name");
c.add(ifclassLabel);
ifclassLabel.setBounds(105, 75, 70, 20);

//then class name label
JLabel thenclassLabel = new JLabel("Class Name");
c.add(thenclassLabel);
thenclassLabel.setBounds(135, 120, 70, 20);

//Derivation label
JLabel derivLabel = new JLabel("Derive:");
c.add(derivLabel);
derivLabel.setBounds(30, 65, 100, 20);
```

```
//if attribute label
JLabel ifattributeLabel = new JLabel("Attribute");
c.add(ifattributeLabel);
ifattributeLabel.setBounds(220, 75, 100, 20);

//then attribute label
JLabel thenattributeLabel = new JLabel("Attribute");
c.add(thenattributeLabel);
thenattributeLabel.setBounds(255, 120, 100, 20);

JLabel Function = new JLabel("Function");
c.add(Function);
Function.setBounds(345, 120, 70, 20);

JLabel actionLabel = new JLabel("Action");
c.add(actionLabel);
actionLabel.setBounds(435, 120, 70, 20);

JLabel valueLabel = new JLabel("Value");
c.add(valueLabel);
valueLabel.setBounds(500, 120, 70, 20);



//Operation label
JLabel operationLabel = new JLabel("Operation");
c.add(operationLabel);
operationLabel.setBounds(310, 75, 80, 20);

//Value label
JLabel value2Label = new JLabel("Values");
c.add(value2Label);
value2Label.setBounds(385, 75, 100, 20);

//class operation label
JLabel classattribute = new JLabel("Operation");
c.add(classattribute);
classattribute.setBounds(160, 15, 100, 20);



JLabel ifas = new JLabel("IF");
c.add(ifas);
ifas.setBounds(55, 95, 40, 25);

JLabel Condition = new JLabel("THEN");
c.add(Condition);
Condition.setBounds(80, 140, 40, 25);

//returned class data label
JLabel returnedClass = new JLabel("ELSE");
c.add(returnedClass);
returnedClass.setBounds(80, 175, 40, 25);


//OK button generate OCL expression
ok = new JButton("OK");
c.add(ok);
ok.setBounds (new Rectangle (210, 220, 60, 25));
   ok.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            FileOutputStream out;
```

```
                PrintStream p;
                  try
                 {
                    out = new FileOutputStream("precondition.txt");
                    p = new PrintStream( out );
                         Derivation.this.dispose();
                 }
                   catch (Exception ezz)
                 {
                  System.out.println (ezz.getMessage());
              }
             }
            }

           );



    //Cancel mygtukas. Uzdaromas langas
      cancel = new JButton("Cancel");
           cancel.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
     Derivation.this.dispose();
    }
   });
    c.add(cancel);
    cancel.setBounds (new Rectangle (280, 220, 80, 25));

    //Helpas
    //c.add(new JButton("Help"));

  setSize(600,300);
  show();

  }
}
```