KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

J. Bisikirskas, A. Bartkus

UML CASE įrankio įskiepis veiklos taisyklėms specifikuoti ribotos natūralios kalbos šablonais

Magistro darbas

Darbo vadovas
prof. dr. Lina Nemuraitė
2009-01-

KAUNAS, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

J. Bisikirskas, A. Bartkus

UML CASE įrankio įskiepis veiklos taisyklėms specifikuoti ribotos natūralios kalbos šablonais

Magistro darbas

Recenzentas
dr. Darius Šilingas
2009-01-

Vadovas
prof. dr. Lina Nemuraitė
2009-01-

Atliko
IFM-3/4 gr. stud.
Justinas Bisikirskas
Audrius Bartkus
2009-01-

KAUNAS, 2009

Turinys

SUMMARY

UML CASE TOOL PLUGIN FOR REPRESENTATION OF BUSINESS RULES USING LIMITED NATURAL LANGUAGE TEMPLATES

The main goal of this project was to develop template based business rules plug-in for CASE tool. We have chosen to create plug-in for CASE tool “Magic Draw UML”.

This plug-in extends “Magic Draw UML” abilities of constraints creation, because system partially ensures the correctness of constraint elements. Plug-in was developed for system analysts, who precisely specify requirements for systems.

The prototype we made successfully embodies main goals of project and can be improved in the future. Limited natural language templates can be specified using TBEProfile module. Business rules created with plug-in are formatted and clear to understand. It helps to improve project engineering level of automation. User using prototype is able to specify business rules in UML class, state protocol, state machine, sequence and activity diagrams.

System was created using JAVA programming language and “Magic Draw UML OpenAPI”.

1. ĮVADAS

Kuriant programinę įrangą, labai svarbu suprantamai ir išsamiai apibrėžti vartotojų poreikius ir sukurti efektyvų programinės įrangos modelį, kurį būtų galima automatiškai apdoroti. To siekia modeliais grindžiamas kūrimas (Model Driven Development). Automatizuoto kūrimo galimybes UML CASE įrankiais gali padidinti OCL (Object Constraint Language) ar kitos formalios kalbos, skirtos papildyti grafinius modelius veiklos taisyklių aprašais. Tačiau OCL ar kitos formalios kalbos yra per mažai naudojamos, kadangi programinės įrangos modelio kūrime dalyvaujantys specialistai dažnai nemoka modeliavimo kalbų.

Tyrimo tikslas – ištirti galimybes palengvinti veiklos taisyklių specifikavimą ir įvedimą į informacinių sistemų modelius, tam sukuriant reikalavimų lygio veiklos taisyklių sandaros modelį, skirtą IS analitikams ir projektuotojams, bei jo realizaciją UML CASE įrankyje (Magic Draw UML).

Sukurtas įskiepis leidžia naudoti natūraliai kalbai artimus veiklos taisyklių šablonus. Įvedimas yra dalinai automatizuotas ir korektiškesnis, lyginant su esamu įvedimu, kuris leidžia įvesti ribojimus su klaidomis. Specifikavimo kalba sukurta taip, kad būtų galima atlikti ja užrašytų veiklos taisyklių transformavimą į OCL.

Darbo rezultate MagicDraw UML įrankis yra papildytas galimybe įvesti struktūrizuotus ribojimus, sudarant juos iš dalykinės srities modelio elementų, operatorių ir funkcijų. Veiklos taisyklės įvedamos taikant natūraliai kalbai artimus šablonus. Šablonai yra nepriklausomi nuo kalbos, nauji šablonai lengvai sukuriami.

Darbe buvo sprendžiami šie **uždaviniai:**

- išanalizuoti veiklos taisyklių sąvokas, esamas klasifikacijas ir apibrėžimus;
- išanalizuoti veiklos taisyklių šablonus;
- išanalizuoti veiklos taisyklių metamodelius;
- sudaryti veiklos taisyklių struktūros metamodelį, kuris leistų užrašyti veiklos taisykles ribota natūralia kalba;
- suprojektuoti bei realizuoti taisyklių įvedimą pagal sukurtos struktūros modelį CASE įrankyje;
- įvertinti sukurtą įskiepį, jo naudingumą ir pranašumus.

Literatūros apžvalgai atlikti buvo analizuojami šių tipų literatūros šaltiniai:

- apžvalginiai straipsniai [3, 4, 5, 19, 20];
- techninės įrankių ir technologijų specifikacijos [13 ,14 ,15 ,16 ,17 ,18];

- knygos [1, 2, 6, 7, 8, 9, 10, 11, 12].

Darbo struktūra. Darbo analizės dalyje pateiktos ir apibendrintos penkių skirtingų autorių veiklos taisyklių klasifikacijos, išanalizuoti veiklos taisyklių šablonai ir metamodeliai, išnagrinėtas jų tinkamumas CASE įrankio funkcionalumui praplėsti. Taip pat atlikta įskiepio kūrimo technologijos analizė, apibrėžtas siekiamas sprendimas. Šios dalies pabaigoje pateiktas analizės rezultatų apibendrinimas ir suformuluoti reikalavimai tolesniam tyrimui.

Pagrindinėje darbo dalyje aprašyti veiklos taisyklių šablonų specifikavimo įrankio reikalavimai ir projektiniai sprendimai, kurie buvo pritaikyti įrankio kūrimo proceso metu. Taip pat šioje dalyje aprašomas realizuotas modulis, skirtas praplėsti šablonų panaudojimą UML CASE įrankyje. Pateikti veiklos taisyklių formavimo MagicDraw UML įrankyje metodai, aprašytas projekto modelis ir šablonizuotos kalbos, artimos natūraliai kalbai, metodika. Sistemos realizacijos poskyryje aprašomi sukurti įskiepio komponentai. Pateikiamas veiklos taisyklių ataskaitoms generuoti skirtas šablonas, bei įskiepio testavimo planas. Sukurtas prototipas leidžia įvesti veiklos taisykles į 5 UML diagramas. Pateikiami kontroliniai duomenys ir rezultatai. Projektinėje dalyje įvardijami prototipo tolimesni planai.

Ketvirtoje darbo dalyje aprašytas atliktas sukurto MagicDraw UML įskiepio naudingumo tyrimas, pateikti tyrimo rezultatai. Tyrimo tikslas - nustatyti sukurto prototipo, skirto veiklos taisyklėms specifikuoti pranašumus, lyginant su standartiniu kūrimo procesu Magic Draw UML įrankyje. Įvertinti šablonų panaudojimą, bei jų universalumą kalbos atžvilgiu.

Pabaigoje pateikiamos atlikto darbo pagrindinės išvados ir rezultatai.

Prieduose pateikiama straipsnis iš dalyvautos konferencijos bei veiklos taisyklėmis grindžiamo IS proceso pavyzdys, panaudojant įskiepį.

Darbo pasidalinimas

Audrius Bartkus:

- Analizės dalis
  - Veiklos taisyklių modelių raida
  - Veiklos taisyklių klasifikacijų apibendrinimas;
  - Veiklos taisyklių metamodelių analizė;
  - Esamų sprendimų analizė;
- Projekto dalis
  - Reikalavimų modelis – panaudojimo atvejai, dalykinės sritis.
  - Testavimo atvejai ir kontroliniai duomenys, Įskiepio diegimo aprašymas.
- Realizacijos dalis
  - VT įvedimo/redagavimo modulių pritaikymas skirtingoms diagramoms

- Įskiepio testavimas

Justinas Bisikirskas:

- Analizės dalis
  - Veiklos taisyklių klasifikacija pagal 5 autorius
  - Veiklos taisyklių šablonai
  - Įskiepio kūrimo metodika
- Projekto dalis
  - Reikalavimų modelis – vartotojo interfeiso modelis
  - Projektų modelis – analizės klasių diagramos, panaudojimo atvejų realizacijos, projekto klasių diagrama
  - Realizacijos modelis – komponentų ir įrangos diagramos
- Realizacijos dalis
  - Veiklos taisyklių reiškinio formavimo mechanizmas ir logika
  - Įskiepio grafinis interfeisas
  - Įskiepyje panaudoto TBE modulio sukūrimas ir realizacija
  - Įvedimas/Redagavimo mechanizmo bazinis modulis

2. VEIKLOS TAISYKLIŲ MODELIŲ IR AUTOMATIZAVIMO TECHNOLOGIJŲ ANALIZĖ

Šame skyriuje apibrėšime tyrimo problemą, sritį bei objektą. Pagal tyrimo problemą, nusistatysime tyrimo tikslus bei uždavinius. Atliksime šaltinių, technologijų bei panašių sistemų analizę. Aprašysime sudarytos veiklos taisyklių kalbos struktūrą, bei sandarą. Įvertinsime atliktos analizės tyrimo rezultatus, bei nustatysime tolimesnį sistemos plėtojimą.

2.1. Tyrimo tikslai ir uždaviniai

Tyrimo sritis – veiklos taisyklių įvedimo automatizavimas CASE įrankiuose.

Tyrimo objektas – veiklos taisyklių aprašymo ribota natūralia kalba procesas, atliekamas UML CASE įrankiu („MagicDraw UML“).

Šiuo metu veiklos atstovai susiduria su reikalavimų modelio specifikavimo supratimo problema. Įvairūs modeliai yra vaizduojami tam tikrais terminais, kuriuos lengvai supranta programuotojas ir analitikas, tačiau pagrindinis užsakovas, tiksliai neperprasdamas minties, negali šių modelių patvirtinti. Reikalavimų modelio specifikavimas artima natūraliai kalbai yra viena iš išeičių, palengvinančių darbą tiek analitikams išaiškinti kiekvieno termino prasmę, tiek užsakovams geriau suvokti modelius.

Tyrimo tikslas – palengvinti veiklos taisyklių specifikavimą ir įvedimą į informacinių sistemų modelius, tam sukuriant reikalavimų lygio veiklos taisyklių sandaros modelį, skirtą IS analitikams ir projektuotojams, bei jo realizaciją UML CASE įrankyje (Magic Draw UML).

Tyrimo uždaviniai:

- išanalizuoti veiklos taisyklių sąvokas, esamas klasifikacijas ir apibrėžimus;
- išanalizuoti veiklos taisyklių šablonus;
- išanalizuoti veiklos taisyklių metamodelius;
- sudaryti veiklos taisyklių struktūros metamodelį, kuris leistų užrašyti veiklos taisykles ribota natūralia kalba;
- realizuoti taisyklių įvedimą pagal sukurtos struktūros modelį CASE įrankyje;
- įvertinti sukurtą įskiepį, jo naudingumą bei pranašumus.

Tyrimo metu atliekamos analizės tikslai - visapusiškai ir nuodugniai išstudijuoti bei suprasti problemą, o tam spręsime tokius uždavinius:

- išanalizuosime esamus sprendimus, taikomus panašioms problemoms spręsti;
- išanalizuosime tyrimo objektą tam tikrais aspektais, kurie svarbūs sprendimui rasti;
- atliksime CASE įrankių, leidžiančių įvesti ribojimus, tiriamąją analizę.

Darbe taikysime konstruktyvųjį tyrimo metodą (angl. *Constructive Research Method*):

1. sukursime produktą (artefaktą), kuris išspręstų veiklos taisyklių specifikavimo palengvinimo problemą;
2. sukursime žinias apie tai, kaip ši problema gali būti išspręsta, ir jeigu egzistavo sprendimai, kokiu būdu naujesnis sprendimas yra naujesnis arba geresnis nei ankstesni.

Tyrimo uždavinio formuluotė pateikiama 1 lentelėje.

2.1 lentelė. Tyrimo formuluotė

Produkto (artefakto) tipas	**Srities problema**	**Sukuriamos žinios**
Veiklos taisyklių įskiepis skirtas analitikams	Kuriant informacines sistemas analitikams tenka įvedinėti veiklos taisykles tam tikra, sunkiai suprantama kalba, todėl tai sukelia problemų greitai ir efektyviai suprasti ir aprašyti tai, ko nori „verslo“ žmogus.	Natūraliai kalbai artima veiklos taisyklių struktūra, realizuojama UML CASE įrankiuose. Šios kalbos naudingumas lyginant su esamomis kalbomis

2.2. Veiklos taisyklių modelių raida

Veiklos taisyklės pirmą kartą buvo paminėtos 1984 metais. Nuo tada jas nagrinėjo daug autorių, kurie sukūrė įvairių taisyklių klasifikacijų ir modelių. Veiklos taisyklių apibrėžimai pagrindiniuose veikaluose pateikiami 2 lentelėje.

2.2 lentelė. Veiklos taisyklių apibrėžimai pagrindiniuose veikaluose.

Šaltinis	***Apibrėžimas***	***Komentarai***
Daniel S. Appleton, "Business Rules: The Missing Link," *Datamation*, Lapkričio 15, 1984, pp. 145–150.	"... Detalus ribojimo, egzistuojančio veiklos ontologijoje, apibrėžimas."[p. 146]	Šis straipsnis pirmą kartą literatūroje pateikia veiklos taisyklių terminą
Ronald G. Ross, *Entity Modeling: Techniques and Application,* Database Research Group, Inc., 1987.	"... Specifinės taisyklės (arba veiklos strategijos, kryptys), kurios daro įtaką .. (įmonės) elgesiui ir išskiria ją iš kitų. Šios taisyklės daro įtaką pasikeitimams pačios įmonės viduje." [p. 102]	Šis straipsnis pabandė aprėpti ir suklasifikuoti veiklos taisykles deklaruojant jas duomenų modeliais.
Ronald G. Ross, *The Business Rule Book*	"... Veiklos taisyklė gali būti laikoma vartotojo reikalavimu, kuris	Pagrindinis šio didelio veikalo indėlis (ir antrojo

(First Edition), Business Rule Solutions, LLC., 1994.	išreiškiamas neprocedūrine ir netechnine forma (paprastai teksto sakinys). Veiklos taisyklė išreiškia veiklos elgseną [p. 496]	leidimo 1997) tas, kad buvo surinkta ir suklasifikuota daugiau nei 500 veiklos taisyklių pavyzdžių, iliustruojant ir aprašant probleminės srities apimtį bei identifikuojant pagrindines veiklos taisyklių kategorijas ir šablonus.
"Defining Business Rules ~ What Are They Really?" (anksčiau žinomas kaip "GUIDE Business Rules Project Report," 1995), redaguotas David C. Hay ir Keri Anderson Healy, Business Rules Group, (trečias leidimas.), Liepa 2000.	"... Apibrėžimas, kuris apibūdina arba riboja kai kuriuos veiklos aspektus… kurie išreiškia veiklos struktūrą arba kontroliuoja arba įtakoja veiklos elgseną. Veiklos taisyklė negali būti suskaidoma ar dekomponuojama į dar smulkesnes veiklos taisykles… " [pp. 4-5]	Dažniausiai cituojamas kaip fundamentalus veiklos taisyklių straipsnis.
Ronald G. Ross, The Business Rule Book (Second Edition), Business Rule Solutions, LLC., 1997.	"Terminas, faktas ar taisyklė, kuri vaizduoja predikatą." [p. 380]	
Business Rules Group, 1998.	"Direktyva, kuri daro įtaką veiklos elgsenai. Tokios direktyvos egzistuoja veiklos strategijose, kurios formuluojamos kaip atsakas į rizikas, grėsmes ar galimybes. "	
Ronald G. Ross and Gladys S. W. Lam, Capturing Business Rules, 2000.	"Galimos vėl naudoti veiklos logikos atominė dalis, specifikuota deklaratyviai."	

Barbara von Halle, Business Rules Applied: Building Better Systems Using the Business Rule Approach, Wiley Computer Publishing, 2002.	"... Sąlygos, kurios daro poveikį veiklos įvykiui taip, kad jis būtų priimtinas veiklai." [p. 28]	Von Halle papildo GUIDE Project apibūdinimą (1995)
Tony Morgan, Business Rules and Information Systems, Addison-Wesley, 2002.	“Iš esmės, veiklos taisyklė yra glausta veiklos aspekto formuluotė”. Tai ribojimas ta prasme, kad veiklos taisyklė nustato, kas tam tikrais atvejais privalo ir kas neprivalo būti.	Morgan taip pat papildo GUIDE Project apibūdinimą (1995)
Ronald G. Ross, Principles of the Business Rules Approach, Addison-Wesley, 2003.	“… taisyklės tiesiogiai remiasi terminais ir faktais. Faktiškai, taisyklių pagalba turėtume suprasti privalomus ir neprivalomus terminus ir faktus, kurie jau buvo aprašyti duomenų modelyje ir sąvokų žodyne. Veiklos problemose, kuriose yra šimtai ar tūkstančiai taisyklių, neįmanoma pasiekti tokio didelio taisyklių skaičiaus neprieštaringumo be bendrinių faktų ir terminų žinių.”	
Semantics of Business Vocabulary and Business Rules (SBVR), OMG, 2005.	“… taisyklė, kuri priklauso veiklos jurisdikcijai.”	SBVR įvedė operatyvių taisyklių sąvokas prieš struktūrinių taisyklių sąvokas, paremtas privalomumo ir būtinumo logika.

2.3. Veiklos taisyklių klasifikacija

Šiame poskyryje apžvelgsime skirtingų autorių veiklos taisyklių klasifikacijas. Išnagrinėtos tokios klasifikacijos:

- taisyklių klasifikacijas pagal Guide projektą;

- objektinių taisyklių klasifikacija (pagal Haim Kilov Iand Simmonds)
- taisyklių klasifikacija pagal Odell
- taisyklių klasifikacija pagal RuleML
- taisyklių klasifikacija pagal UML

2.3.1. Taisyklių klasifikacija pagal Guide projektą

Pagal Guide projektą, veiklos taisyklės yra 4 kategorijų:

- *veiklos santykių apibrėžimas;*

  Veiklos taisyklių pagrindinis elementas yra kalba kuria tu jas išreiški. Kiekvienas inicijavimas veiklos taisyklės kartu yra ir direktyva, kuri apibūdina kaip žmonės galvoja ir kalba apie daiktus. Tai dažniausiai vaizduojama esybių/ryšių modeliuose.
- *faktai susiję su santykiais;*

  Tai iš organizacijos ar veiklos prigimties kilusios veiklos taisyklės.

  Dažniausiai aprašomi natūralia kalba arba tiesiog grafiniame modelyje vaizduojami kaip atributai, ryšiai, apibendrinimai.
- *ribojimai (taip vadinami veiksmo teiginiai);*

  Kiekviena įmonė riboja veiksmus tam tikru būdu, ir tai artimai susiję informacijos priėjimu, atnaujinimu.
- *išvedamos (ang. Derivations).*

  Veiklos taisyklės apibūdinančios kaip žinios vienoje srityje gali būti perkeltos į kitas žinias, kitokioje formoje.

Veiklos taisyklių tipai:

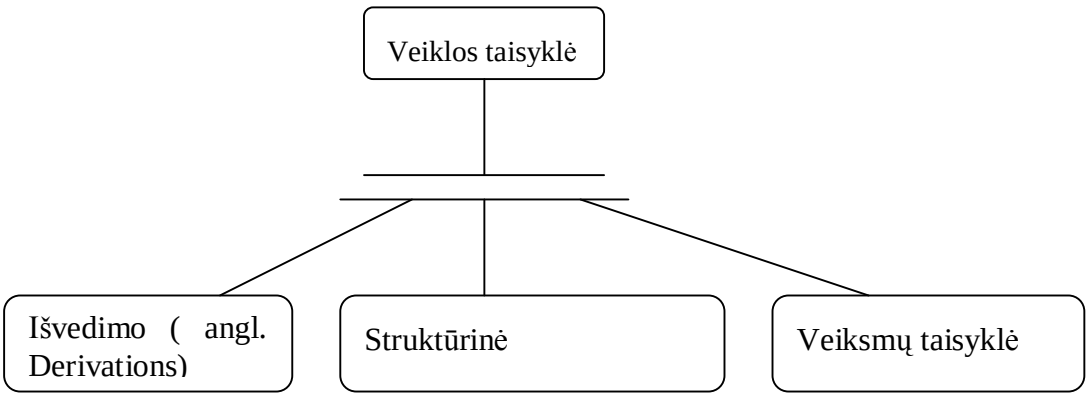
Kiekviena taisyklė turi būti viena iš (2.1 pav.):

- struktūrinių taisyklių;

  Apibūdina fakto idėją arba konstatavimą kurį galima išreikšti struktūriškai. Į tai įeina ir faktai, ir santykiai.
- veiksmų taisyklių;

  Išreiškia ribojimus ar sąlygas, kurie riboja organizacijos veiksmus.
- išvedimo taisyklių.

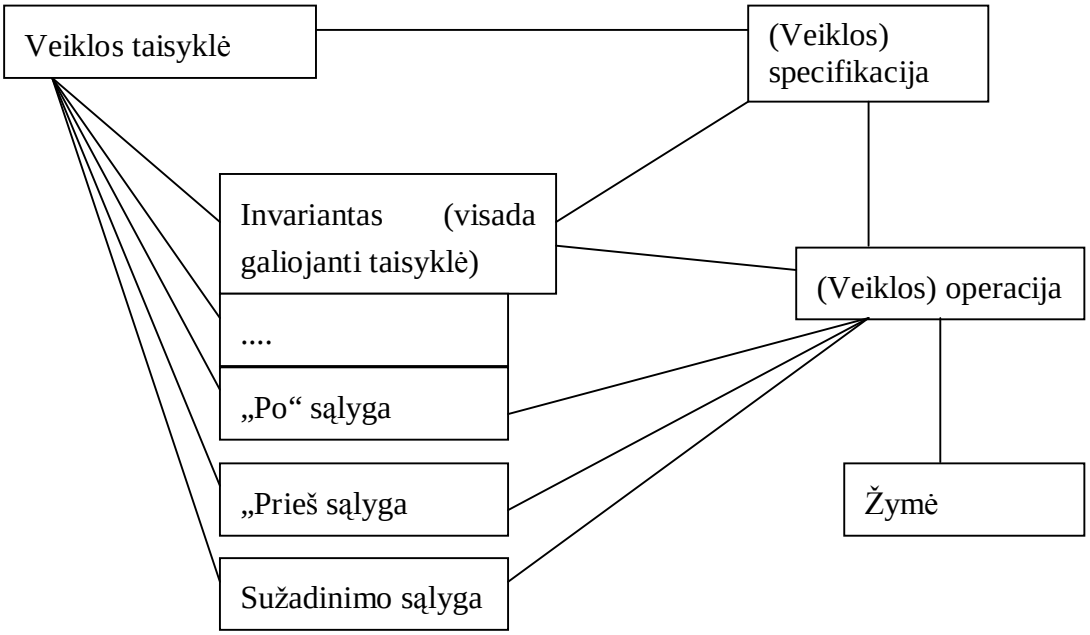
  Išreiškia žinias, gaunamas iš veiklos patirties.



2.1 pav. Taisyklė pagal Guide projektą

2.3.2. Objektinių taisyklių klasifikacija (pagal Haim Kilov and I. Simmonds)

Pagal jų klasifikaciją, veiklos taisyklės turi turėti veiklos pobūdį (2.2 pav.).

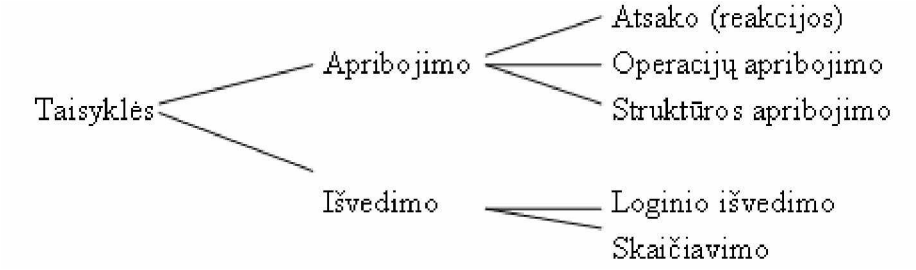


2.2 pav. Taisyklių klasifikacija pagal H. Kilov and I. Simmonds

Pagal šią klasifikaciją veiklos taisyklės skirstomos į potipius – invariantas, „po“ sąlyga, „prieš“ sąlyga, sužadinimo sąlyga.

2.3.3. Taisyklių klasifikacija pagal Odell

Odell taisykles skirsto į dvi kategorijas - ribojimų ir išvedimo [1]. Ribojimo taisyklės apibrėžia elgesio kryptis ar sąlygas, kurios riboja objektų struktūrą ir elgesį. Išvedimo taisyklės apibrėžia elgesio kryptis ar sąlygas faktų išvedimui ar apskaičiavimui, pasinaudojant kitais faktais (2.3 pav.).



2.3 pav. Taisyklių klasifikacija pagal Odell

Atsako taisyklės riboja elgesį. Jos aprašomos KAI ir TUOMET sąlygomis. KAI aprašo sąlygą, kuriai išsipildžius, reikia atlikti operaciją, kuri aprašyta sąlygoje TUOMET. Pavyzdžiui:

> "KAI produkto kiekis sandėlyje pasidaro mažesnis už būtiną minimumą,
> TUOMET užsakyti šį produktą".

Atsako taisyklės priklauso nuo konteksto. Taisyklės KAI sąlyga tikrinama tik tada, kai įvyksta tam tikro tipo įvykis.

Operacijų ribojimo taisyklės nurodo, kokia sąlyga turi būti patenkinta prieš operaciją, kad ją būtų galima atlikti (operacijos prieš sąlyga), ir kokia sąlyga turi būti patenkinta po operacijos, kad būtų užtikrintas jos atlikimo teisingumas (operacijos po sąlyga). Šie ribojimai gyvybiškai svarbūs operacijos atlikimui ir visiškai nepriklauso nuo konteksto, kuriame operacija kviečiama. Operacijos prieš ir po sąlygos nusako kontraktą, kuris susieja operaciją su jos užsakovais. Tai būtų galima užrašyti taip: "jei iškviesi operaciją, kai prieš-sąlyga įvykdyta, pažadame pateikti tokią galutinę būseną, kurioje bus patenkinta po-sąlyga".

Operacijos prieš-sąlygos taisyklės išreiškia tuos ribojimus, kuriuos patenkinus, operacija bus atlikta teisingai. Pavyzdys:

> "Pasiūlyti tarnautojui vadovo pareigas TIK TUOMET JEIGU tas tarnautojas nėra vadovas".

Operacijos po-sąlygos taisyklės garantuoja rezultatus. Kai operacija atlikta, sistema turi įgauti tam tikrą būseną. Pavyzdys:

> "Pasiūlyti tarnautojui vadovo pareigas ATLIKTA TEISINGAI TIK TUOMET JEIGU tas tarnautojas yra vadovas".

Struktūros ribojimo taisyklės aprašo ribojimus objektų tipams ir jų savybėms (atributams ir ryšiams), kurių negalima pažeisti.

Taisyklė gali riboti atributo reikšmes: "VISADA tarnautojo alga ne didesnė už vadovo algą".

Taisyklė gali riboti objekto tipo egzempliorius:

“VISADA Aukščiausio teismo teisėjų skaičius ne didesnis už 5”.

Taisyklė gali riboti ryšio kardinalumą:

“VISADA naujas vartotojas turi ne daugiau kaip 7 užsakymus”.

Struktūros ribojimo taisyklės nedaro nuorodų į operacijas, nes jos turi būti tenkinamos bet kuriomis veikimo aplinkybėmis. Nesvarbu, kad pasikeitė objekto būsena, jo struktūros ribojimo taisyklės turi būti patenkintos. Pavyzdžiui, pasikeitus tarnautojo amžiui ar atėjus naujam darbuotojui, reikia patenkinti tokią taisyklę:

“VISADA tarnautojo amžius neviršija 75 metų”.

Loginio išvedimo (išvadų) taisyklės sako, kad jei tam tikri faktai teisingi, galima daryti išvadas apie kitus faktus. Tokios taisyklės susiję su ekspertinėmis sistemomis.

“JEIGU daugiakampis turi perimetrą, TUOMET kvadratas turi perimetrą”.

Taisyklė gali išvesti objekto potipius:

Asmuo yra tarnautojas TADA IR TIK TADA, KAI jis dirba organizacijai.

Taisyklė gali išvesti objektų ryšius:

“Tarnautojas duoda ataskaitas vadovui TADA IR TIK TADA, KAI šis tarnautojas dirba padalinyje, kuriam vadovauja tas vadovas”.

Skaičiavimų taisyklės aprašo algoritmus, kurių pagalba galima gauti rezultatus. Jos išreiškiamos formulėmis.

Taisyklė gali aprašyti, kaip skaičiuoti skaitines reikšmes:

“Bendra produkto kaina SKAIČIUOJAMA TAIP: produkto kaina * (1 + mokesčių procentas / 100)”.

Taisyklė gali aprašyti objektų tipų skaičiavimą:

“Objekto tipas moteris SKAIČIUOJAMAS TAIP: visų moteriškos giminės žmonių ir suaugusių žmonių sankirta”.

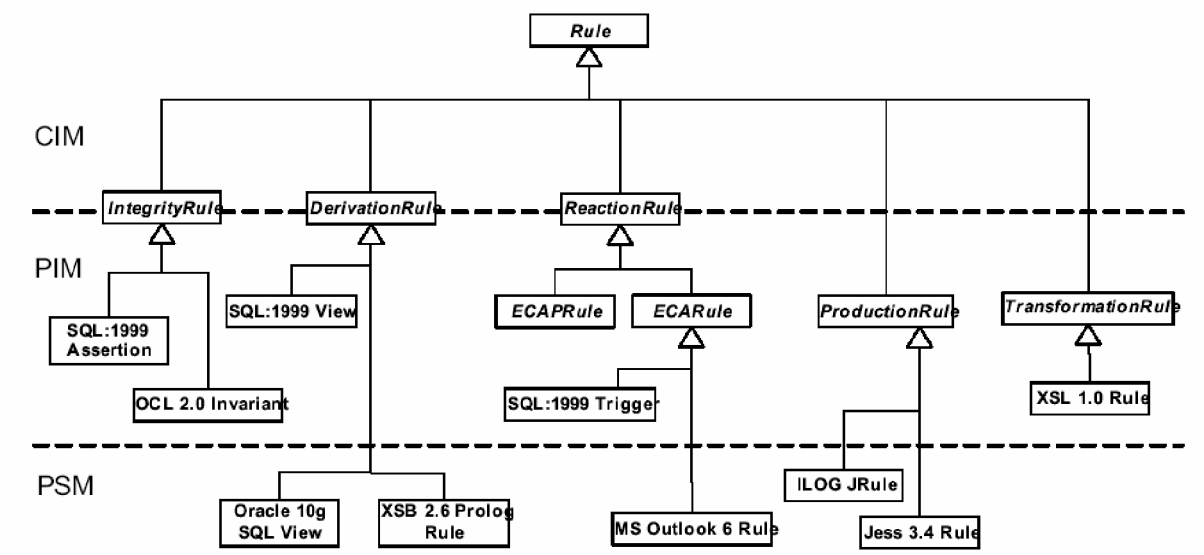
Taisyklė gali aprašyti ryšių skaičiavimus:

“Tėvų ryšys SKAIČIUOJAMAS TAIP: motinos ir tėvo ryšių sujungimas”.

Skaičiavimų taisyklės išreiškia, kaip iš faktų gauti rezultatą.

2.3.4. Taisyklių klasifikacija pagal RuleML

Tai dabartinė taisyklių klasifikacija artima klasikinei (2.4 pav.).



2.4 pav. Taisyklės klasifikacija pagal RuleML

Taisyklės yra skirtomos į

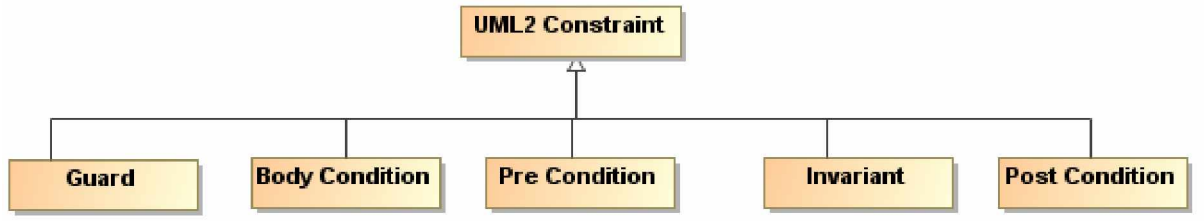
- Vientisumo (Integrity Rule)
- Išvedimo (Derivation Rule) – tai taisyklės įgalinančios išvesti sprendimus, kitas taisykles ar išskaičiuoti reikšmes;
- Reakcijos taisyklės – būna dviejų tipų - ECA (Įvykis, sąlyga, veiksmas), neturintis „po“ salygos, arba ECAP (Įvykis, sąlyga, veiksmas, „po“ sąlyga).
- Produkcinės taisyklės – CA taisyklės, susidedančios iš sąlygos ir veiksmo aprašymo.

Svarbu tai, kad šioje klasifikacijoje taisyklės skirstomos į CIM (nuo kompiuterizavimo nepriklausomas, grynas veiklos taisykles), PIM (nuo platformos nepriklausomas, sistemos lygio taisykles), ir PSM (specifinių realizacijos platformų taisykles).

Šiame darbe bus nagrinėjamos taisyklės, priklausančios reikalavimų lygiui (riba tarp CIM ir PIM). Jos turi būti išreiškiamos struktūrizuotos natūralios kalbos šablonais, suprantamais verslo žmonėms, ir pakankamai formalios, kad būtų tiesiogiai vaizduojamos į vykdomąsias sistemos lygio taisykles.

2.3.5. Taisyklių klasifikacija pagal UML

UML 2 veiklos taisyklių klasifikacija, susijusi su siekiamo įrankio kūrimu (2.5.pav.):



2.5 pav. Taisyklių klasifikacija pagal UML

UML veiklos taisyklės vadinamos ribojimais (*Constraints*), kurie gali būti 4 tipų:

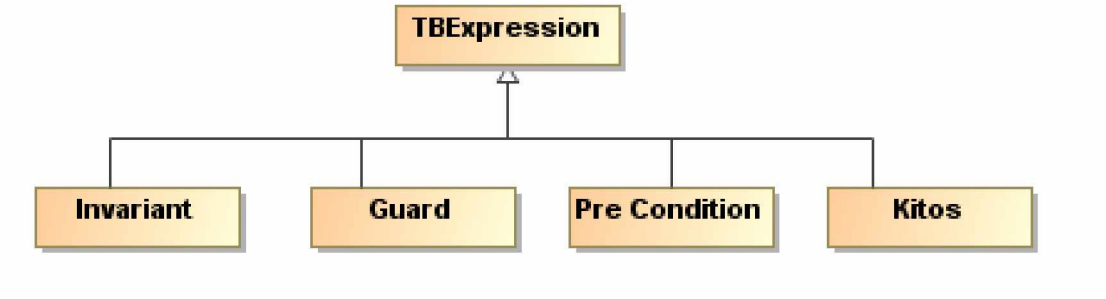
– ***Klasių invariantai***, nusakantys taisykles, kurios turi galioti visada;

– ***Operacijų prieš sąlygos***, nusakančios ribojimus, kuriems galiojant operacija įvykdoma teisingai ir taip, kaip aprašyta jos po sąlygoje;

– ***Operacijų po sąlygos***, nusakančios ribojimus, kurie turi galioti operacijai pasibaigus;

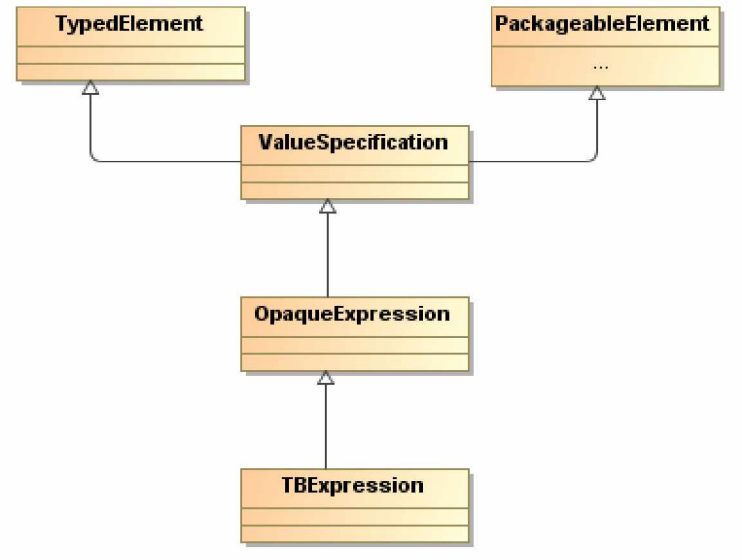
– ***Operacijų apibrėžimo sąlygos*** (*body conditions*), nusakančios operacijų apibrėžimus.

Atsižvelgus į UML taisyklių klasifikaciją, mūsų taisyklių klasifikacija atrodys taip (2.6 pav.):



2.6 pav. TBE taisyklių klasifikacija

Kiekvienas UML modelio elementas gali turėti ribojimų. Ribojimų reikšmes galima specifikuoti bet kuria kalba. Ribojimų reikšmių specifikacijos UML yra neinterpretuojami reiškiniai (*OpaqueExpression*). 2.7 pav. pavaizduotas UML metamodelis su mūsų šablonizuotais ribojimų reiškiniais:



2.7 pav. UML ribojimų reikšmės specifikuojamos neinterpretuojamais reiškiniais

2.3.6. Veiklos taisyklių klasifikacijų (taksonomijų) apibendrinimas

2.3 lentelė išvardina esminius kiekvienos taisyklių klasifikacijos principus:

2.3 lentelė. Veiklos taisyklių apibendrinimas

Šaltinis, autorius	***Sistematika, skirstymas***
Guide Business Rules Project	• Struktūrinis požiūris (santykiai, faktai) • Veiksmo požiūris (vientisi ribojimai, sąlygos, autorizavimas) • Išvedimai (skaičiavimai, išvados)
Ron Ross, Database Research Group	• Faktai • Santykiai • Taisyklės (ribojimai, sprendimo, išvados, laikinės, nuoseklios, euristinės)
James Odell, Independent Consultant	• Ribojimai: (atsako, operacijos ribojimo, struktūros ribojimo) • Išvedimo (loginio išvedimo, skaičiavimo)
Tom Romeo, IBM	• Struktūrinės (ryšiai, sritis), kardinalumas,sąlyginės, • Elgesio (priešsąlyginės, posąlyginės, išvedamos)
Usoft Corp.	• Draudimai (ribojimai, sritis) • Atimamos • Elgesio • Vaizduojamosios
Dan Tasker, Air New Zealand	• Veiksmo ribojamos • Veiksmo reguliuojamos (triggering) • Ribojimai
Brightware	• Veiklos • Elgsenos • Sekų • Euristinės sprendimui
Haim Kilov Ian Simmonds	• Invariantas • Prieš sąlyga • Po sąlyga • Sužadinimo sąlyga.
Vision Software	• Validavimo • Struktūrinės • Ryšių • Sąlygos veiksmų
SBVR	• Struktūrinė taisyklė • Operacinė taisyklė
UML	• Invariantas • Prieš sąlyga

• Po sąlyga

2.4 lentelė apibendrina veiklos taisyklių atitikimus skirtingose klasifikacijose:

2.4 lentelė. UML&TBExpression taisyklių atitikimas kitose klasifikacijose apibrėžiamų taisyklių kategorijoms

Guide	**Objektinės (pagal Odell)**	**RuleML**	**UML&TBexpressions**
Struktūriniai teiginiai	Klasių modelis	–	Klasių modelis
Išvedimo taisyklės	Loginio išvedimo taisyklės	Išvedimo taisyklės	Invariantai
Skaičiavimo taisyklės	Skaičiavimo taisyklės	Išvedimo taisyklės	Invariantai, operacijų po sąlygos, apibrėžimo sąlygos, rezultatai
Vientisumo ribojimai (veiksmų teiginiai)	Struktūriniai ribojimai	Vientisumo ribojimai	Invariantai
Sąlygos (veiksmų teiginiai)		ECA, ECAP dalis	TBexpressions reiškiniai
Veiksmų teiginiai (įgalintojai, kontroliuojantys teiginiai, veikiantys teiginiai, laikmačiai)	Operacijų ribojimai Reakcijos taisyklės	ECA, ECAP	Operacijų prieš ir po sąlygos. Būsenų mašinos, sekų ir veiklos diagramos su TBExpressions reiškiniais
Įgaliojimo taisyklės	Operacijų ribojimai	Sąlyga (ECA, ECAP taisyklės dalis)	Panaudojimo atvejų modelis, rolės klasių diagramoje ir pan.
	Operacijų ribojimai Reakcijos taisyklės	Produkcinės taisyklės	Invariantai, operacijų prieš ir po sąlygos
–	–	Transformacijų taisyklės	Invariantai arba operacijų prieš ir po sąlygos

2.4. Veiklos taisyklių šablonai

Pagal *Business Rule Solutions, LLC*, sakinio šablonas yra fundamentali struktūra, kuri gali būti naudojama tam tikro tipo taisyklės išreiškimui nuosekliu, gerai organizuotu būdu. Kiekvienas sakinio šablonas yra pritaikytas tam tikram taisyklės tipui. Šie taisyklių tipai yra paremti funkcinėmis kategorijomis, kurios vaizduoja atskiras būdingas operacijas ar efektus. Yra trys funkcinės kategorijos: atmetimo, produkcijos ir projektoriaus.

Sakinio šablonai nevaizduoja formalios kalbos taisyklių įvedimui sistemos lygmenyje

(taisyklių procesoriuje). Dažniausiai šablonų tikslas tinkamai išreikšti taisykles, taip pagerinant komunikavimą veiklos lygmenyje.

Labai svarbu suprasti taisyklių lygį, kuriame joms reikia pritaikyti šablonus. Artimos natūraliai kalbos veiklos taisyklių šablonai turi turėti tam tikrus elementus, kad juos galima būtų lengvai suprasti:

1. Kiekviena taisyklė turi funkcinę kategoriją:

   Funkcinė taisyklės kategorija parodo, kaip taisyklė reaguoja į įvykius. Pvz.:

   a) Atmetančios

   Kiekviena taisyklė, kuri neleidžia (atmeta) įvykį, jei taisyklė pažeidžiama.

   Atmetimai yra ribojimai, kurie apsaugo veiklą nuo neteisingų duomenų (ar būsenų) – t.y. nuo informacijos, kuri pažeidžia veiklos taisykles. Pavyzdžiui, atmetimo funkcinei kategorijai gali priklausyti taisyklė, kai užsakovui, kuris turi skolų, reikia uždrausti atsiskaitymus kreditine kortele.

   b) *Produkcijos*

   Bet kuri taisyklė, kuri nei atmeta, nei sukuria įvykius, bet dažniausiai matuoja arba skaičiuoja, arba yra išvedama automatiškai. Produkcijos taisyklės dar skaidomos į skaičiavimo ir diferencijavimo taisykles:

   Skaičiavimo taisyklės – bet kuri produkcijos tipo taisyklė, kuri skaičiuoja reikšmes automatiškai standartinėmis matematinėmis funkcijomis (pvz., suma, daugyba, vidurkis ir t.t.). Pavyzdžiui, skaičiavimo taisyklė gali būti duota užsakovų metiniams užsakymams skaičiuoti.

   Išvedimo taisyklės – bet kuri produkcinio tipo taisyklė, kuri išvedama automatiškai remiantis sąlygomis, kurios specifikuojamos detaliai. Pavyzdžiui, diferencijavimo taisyklė gali identifikuoti, ar asmuo laikomas moterimi, atsižvelgiant į asmens amžių ir lytį.

   c) Projekcinės

   Bet kuri taisyklė, kuri turi tendenciją atlikti kaip kuriuos veiksmus automatiškai, kai pasirodo tiesiogiai susijęs įvykis. Toks veiksmas gali būti duomenų sukūrimas ar ištrynimas, kitos taisyklės leidimas ar draudimas, kai kurių reikšmių nustatymas, procedūros ar programos vykdymas, kitaip sakant projektorius neatmeta įvykių, bet dažniausiai į juos atsižvelgdamas automatiškai sukuria naujus įvykius. Projektoriai dažniausiai nurodo, kaip turi elgtis

automatinės sistemos, pagerinančios darbininkų produktyvumą ir pan. Pavyzdžiui, projektorius gali būti specifikuotas taip, kad atnaujintų užsakymų sąrašą automatiškai, jei užsakymų kiekis per didelis.

Galimumo taisyklės – projektorius, kuris turi vieną iš šių efektų:

- Sukurti ar ištrinti duomenis
- Leisti ar uždrausti įvykius
- Leisti ar uždrausti operacijų ar procesų vykdymą.

Kopijuokliai

Paleidėjai.

2. Kiekviena taisyklė turėtų turėti subjektą

BRS *RuleSpeak*™ ragina naudoti aiškų subjektą kiekvienoje taisyklės sakinio pradžioje. Tai taip pat taikoma ir Funkcinių taisyklių kategorijoms. Tokiu būdu tai suteikia sakiniui aiškumo ir pastovumo. Sąlygos sakiniuose (Jei ... , tai ..) tikrasis subjektas atsiranda tik po „jei“ sakinio. Kartais tokia sakinio konstrukcija nėra gerai tinkama. Geriausias variantas skaičiavimo taisyklėms būtų iškelti skaičiavimo rezultatą (subjektą) į sakinio pradžią, todėl ką taisyklė skaičiuoja bus aišku nuo pat pradžių.

Taip pat į sąlygos sakinį (jei..., tai ...) neverta dėti ribojimų ar atmetimo sakinių. Pavyzdžiui, daug natūraliau skambėtų „Klientas privalo turėti adresą“, negu „Jei egzistuoja klientas, tai klientas turi turėti adresą“. Tai labai svarbus momentas, nes ribojimai yra labai artimi veiklos procesams, ir jų veikla turi daugybę.

3. Kiekviena taisyklė turėtų naudoti „taisyklės žodį“

Taisyklės pobūdis gali būti nustatomas pagal raktinius žodžius. Jie dar kitaip vadinami „taisyklės žodžiai“. Labai svarbu naudoti juos kiekvienoje taisyklėje:

- Privalo (arba turėtų);
- Neigimas (angl. NOT);
- Nė vienas;
- Tiktais (angl. only if).

4. Kiekviena atmetimo tipo taisyklė turi atvirkščią taisyklę

Kiekviena taisyklė turi atvirkščią taisyklę, kuri vadinama „leidimo taisykle“

Pavyzdžiui:

(Taisyklė) *Užsakymai kurių kreditas virš 1000 nepriimami be kredito patikrinimo.*

(Leidimo taisyklė) *Užsakymai kurių kreditas iki 1000 gali būti priimami be kredito patikrinimo*.

5. Bet kuri taisyklė gali būti ribojama

Bet kuri taisyklė gali turėti reguliavimus, pagal kuriuos taisyklė bus taikoma. Tokie ribojimai turėtų prasidėti žodžiais **„Jei“** arba **„Kai“**, prieš tai padėjus kablelį.

Pvz.:

Taisyklė: *Siuntimas turi būti apdraustas**, Jei** siuntimo verte didesnė nei 500 litų.*

„Jei“ turėtų būti naudojamas, kai taisyklė yra taikoma nuolatos. „Kai“ naudojamas tuomet, kai taisyklė taikoma tam tikru atveju.

Pvz.:

*Semestro mokestis turėtų būti nustatytas 2000, **Kai** studentas užsiregistruoja semestrui.*

6. Kiekviena taisyklė gali būti reguliuojama naudojant laiko ribas

Laiko ribą galima atpažinti taisyklėje pagal tokius požymius:

- prieš (laikas);
- per ar prieš pat (laikas);
- per (įvardintas laikas);
- iki (laikas);
- po (laikas).

Pvz.:

*Studentas negali įstoti į klubą, **per ar prieš pat** galą registracijos.*

*Studentas privalo gyventi stovykloje **per** vasarą.*

*Studentas privalo būti įtrauktas į bent 2 modulius, **po** registracijos pabaigos.*

7. Bet kuri taisyklė gali būti susieta su tam tikra konstanta.

Pagal sutarimą konstantą patariama žymėti viengubom kabutėm.

Pvz.:

Paskutinė_mokesčių_grąžinimo_data privalo būti nustatyta ‚gegužės 1‘.

2.5. Veiklos taisyklių metamodelių analizė

RuleML

Apžvelgsime taisykles trijuose skirtinguose abstrakcijos lygiuose:

1. Veiklos srities lygmenyje taisyklės yra formuluotės, kurios išreiškia veiklą (pvz. apibrėžia veiklos srities pagrindinius terminus, ribojimus, operacijas ir kt.) deklaratyviai, dažniausiai natūralia kalba ar vizualiai. Pavyzdžiui:

(T1) „Vairuotojas, kuris nuomojasi automobilį, turi būti vyresnis nei 25 metų“.

(T2) „Aukso pirkėjas, privalo turėti daugiau nei vieną milijoną dolerių depozite“.

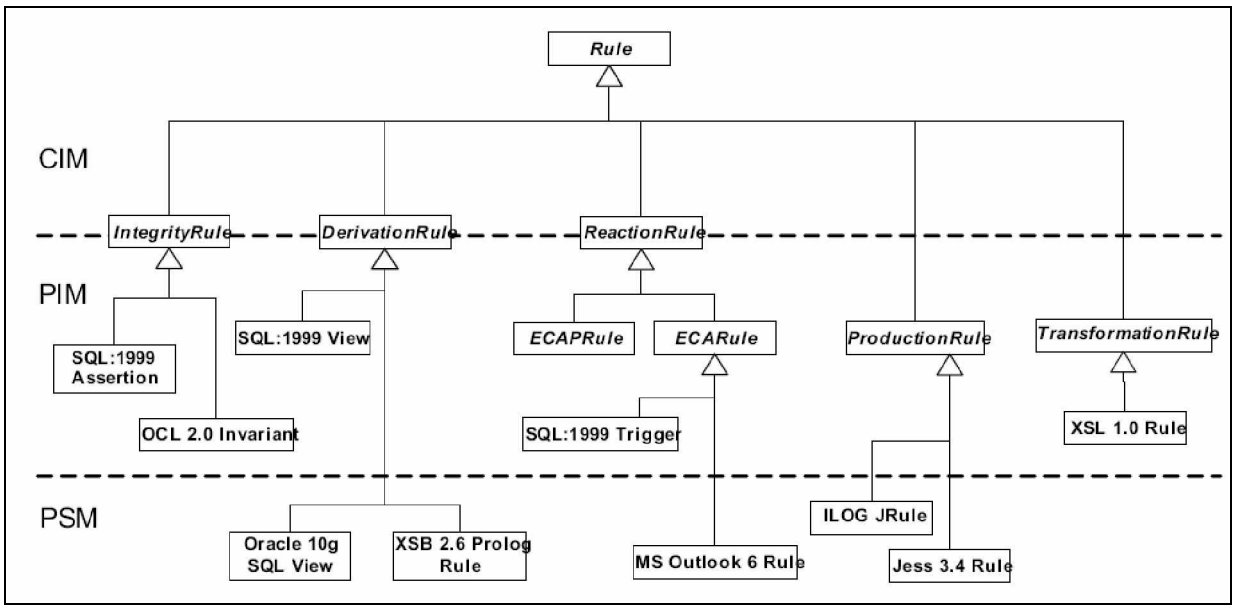
(T3) „Kai akcijos kaina krinta daugiau nei 5% ir investicijoms netaikomas pelno mokestis, tai akcijas parduoti“.

T1 yra vientisumo taisyklė, T2 diferencijavimo taisyklė, T3 reakcijos taisyklė. Tai pagrindinės semantinės veiklos taisyklių kategorijos. Faktiškai dauguma veiklos taisyklių yra reakcijos taisyklės, kurios nusako veiklos kryptis.

2. Nuo platformos nepriklausančiame lygyje, taisyklės yra formalios, išreikštos formalizmu ar skaičiavimo paradigma. Tai tarsi visos veiklos srities abstrakcija. Taisyklių kalbos naudojamos šiame lygmenyje yra SQL:1999, OCL 2.0 ir ISO Prolog.
3. Nuo platformos priklausančiame lygmenyje, taisyklės nusakomos specifinėse vykdomosios programos kalbose, tokiose kaip Oracle 10g, Jess 3.4, XSB 2.6 Prolog ar kt.

Bendrai, taisyklės apima žinias, kuriomis aprašomi samprotavimai. Jos gali specifikuoti:

- statinius ar dinaminius vientisumo ribojimus;
- diferencijavimą;
- reakcijas.



2.8 pav. Taisyklių sąvokos ir išreiškimai skirtinguose abstrakcijos lygiuose

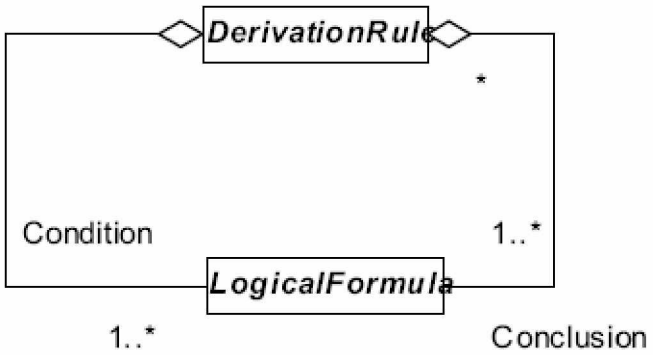
2.8 paveiksle rodoma išvedimo taisyklių sintaksė MOF/UML modelio formoje. Apžvelgsime pagrindines modelio taisykles:

Integralumo (vientisumo) taisyklės

Integralumo taisyklės, taip pat žinomos kaip integralumo ribojimai, susideda iš loginio sakinio. Taisyklė T1 (aprašyta aukščiau) yra statinio ribojimo pavyzdys. Dinaminio ribojimo pavyzdys: “Nuomos rezervacijos patvirtinimas turi būti susijęs su automobilio, iš tam tikros automobilių grupės, priskyrimu atsižvelgiant į pareikalavimo datą ir laikant ją prioritetu nuomojant automobilį.” Gerai žinomos kalbos ribojimų išreiškimui yra SQL ir OCL.

Išvedimo taisyklės (Derivation Rules)

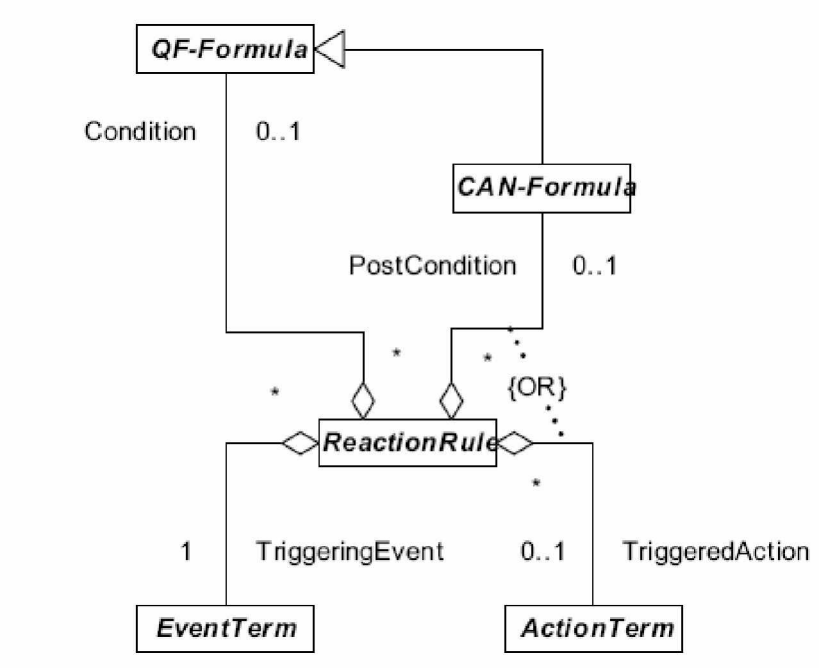
Išvedimo taisyklės susideda iš vienos ar daugiau sąlygų ir vienos ar daugiau išvadų, kurios abi išreiškiamos loginių formulių (angl. LogicalFormula) tipu (2.9 pav.).



2.9 pav. Išvedimo taisyklių modelis

Reakcijos taisyklės

Reakcijos taisyklių tipas yra laikomos svarbiausiu veiklos taisyklių tipu. Jos susideda iš privalomo sužadinančio įvykio, nebūtinos sąlygos, ir vykdomo veiksmo, kurie yra EventTerm, LogicalFormula ir ActionTerm, tipo (2.10 pav.).



2.10 pav. Reakcijos taisyklių modelis

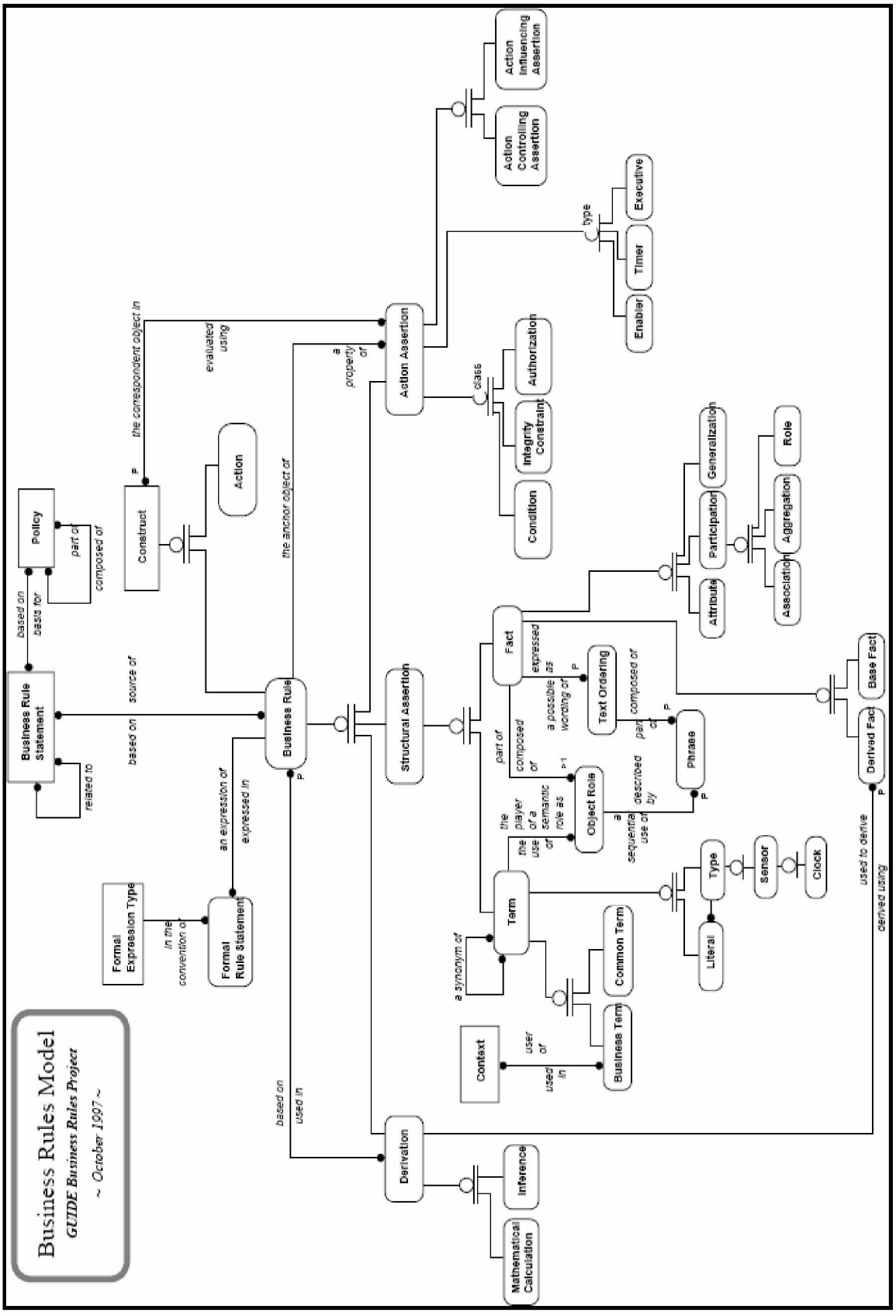
Reakcijos taisyklės po sąlyga gali būti atominė sąlyga, arba neigiama atominei sąlyga arba jų sąjunga. Taip žymima „CAN-Formula“ (2.10 pav.)

Pagal RuleML 0.85 sąlygomis laikomos be kvantorių (quantifier-free) loginės formulės su silpnu ir stipriu neigimu. Jos vadinamos QF-Formula (2.10 pav.)

.

GUIDE BUSINESS RULES PROJECT

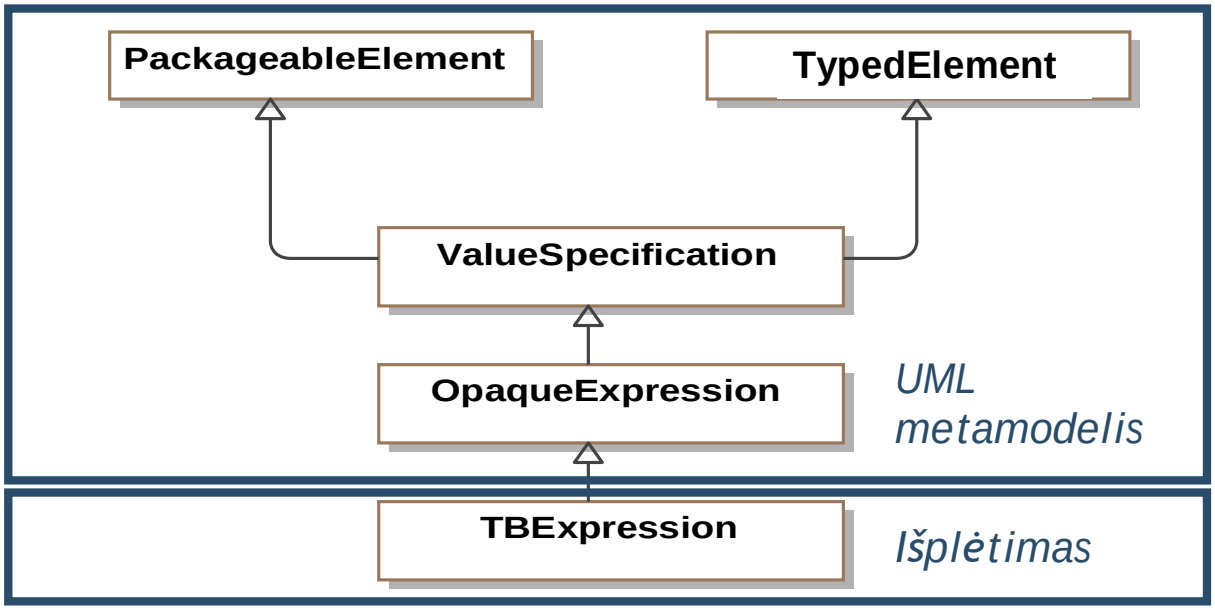
GUIDE veiklos taisyklių projektas aprašė ir apibūdino veiklos taisykles ir su jomis susijusius konceptus, taip išreikšdami terminą kas yra ir kas nėra veiklos taisyklė. Aprašydami pilną koncepcinį veiklos taisyklių modelį, jie išreiškė kas yra veiklos taisyklė ir kaip ji taikoma informacinėms sistemoms (2.11 pav.).



2.11 pav. Guide veiklos taisyklių metamodelis [5]

TBExpression metamodelis:

Mūsų siūlomas metamodelis yra UML metamodelio praplėtimas. Kiekvienas UML modelio elementas gali turėti ribojimų. Ribojimų reikšmes galima specifikuoti bet kuria kalba. Ribojimų reikšmių specifikacijos yra UML neinterpretuojami reiškiniai (OpaqueExpression) .



2.6. Įskiepio kūrimo technologijos analizė

Įskiepis “MagicDraw UML“ pakete yra vienintelis būdas CASE įrankių, taip pat ir MagicDraw UML funkcionalumui papildyti. Pagrindinis įskiepio architektūros tikslas yra išplėsti MagicDraw UML funkcines galimybes. Kai kurių MagicDraw UML funkcinių savybių keisti negalima.

Įskiepis turi apimti tokius resursus:

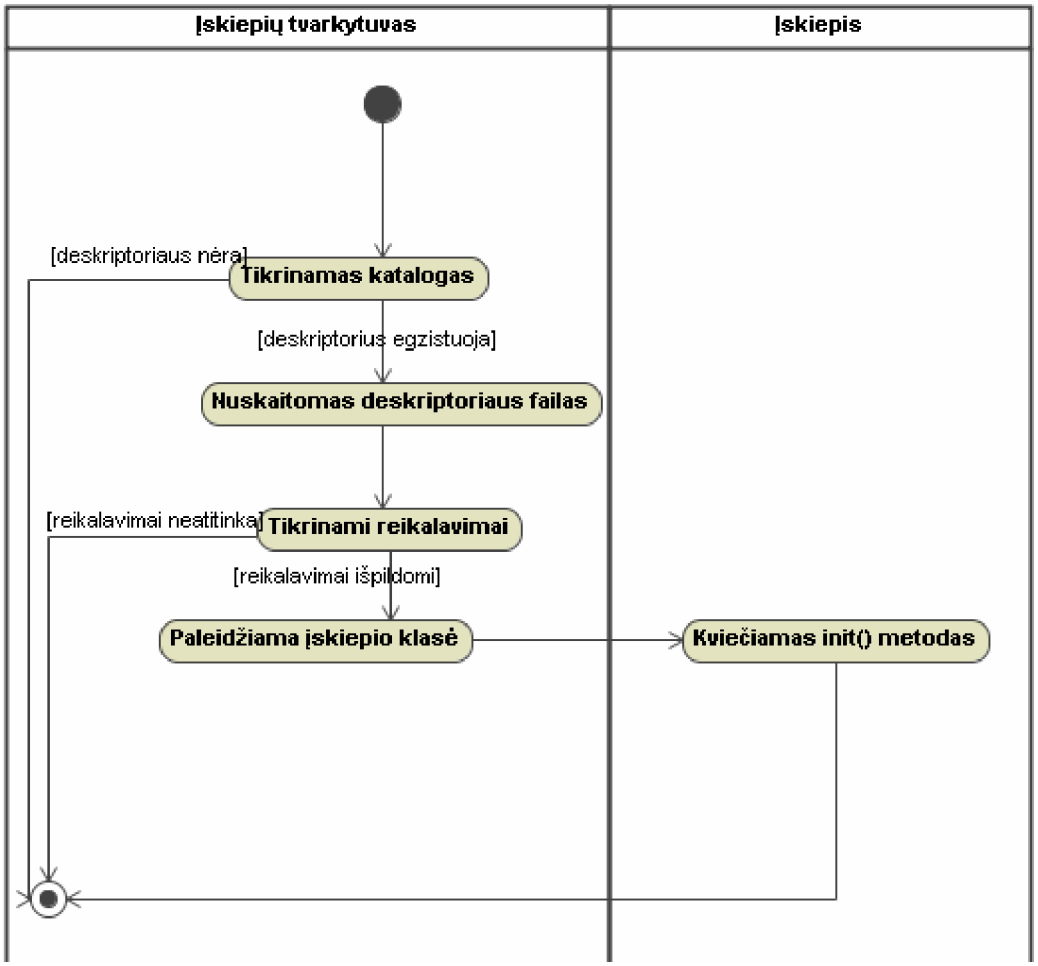
- katalogą;
- jar failą, sudarytą iš java programinių failų;
- įskiepio deskriptoriaus failą;
- papildomus failus, kuriuos naudoja įskiepis.

Įskiepio veikimas

MagicDraw UML kiekvieno paleidimo metu skanuoja įskiepių katalogą ir ieško jame subkatalogų (2.12 pav.):

- jei subkataloge yra įskiepio deskriptoriaus failas, įskiepių tvarkytuvas nuskaito jį;
- jei reikalavimai deskriptoriaus faile yra tenkinami, įskiepių tvarkytuvas paleidžia nustatytą klasę. Ši nustatyta klasė turi būti išvestinė *com.nomagic.magicdraw.plugins.Plugin* klasei. Toliau vykdomas nustatytos klasės

init() metodas. *Init()* metodas naudojamas įskiepio vartotojo sąsajos paleidimui.



2.12 pav. MagicDraw UML įskiepio paleidimo schema

Pagrindiniai MagicDraw UML **įskiepio kūrimo** etapai:

1. MagicDraw UML įskiepių kataloge sukuriamas įskiepio subkatalogas.
2. Rašomas įskiepio programinis kodas.

   Įskiepyje privalo būti bent viena klasė išvesta iš *com.nomagic.magicdraw.plugins.Plugin* klasės.
3. Kompiliuojamas įkiepio kodas ir sudaromas jar failas.

   Kad sukompiliuotumėme programos kodą, reikia įkelti MagicDraw UML klases į java klasių biblioteką. Visos įskiepio kūrimui reikalingos klasės yra sudėtos į šias bibliotekas:

   <MagicDraw installation directory>/lib/md.jar

   <MagicDraw installation directory>/lib/uml2.jar

   <MagicDraw installation directory>/lib/javax_jmi-1_0-fr.jar

   <MagicDraw installation directory>/lib/cmof14.jar

   <MagicDraw installation directory>/lib/y.jar

Iš sukompiliuoto kodo padaromas jar failas.

4. Sudaromas įskiepio deskriptoriaus failas.

   Įskiepio deskriptoriaus failas pavadinamas *plugin.xml* ir patalpinamas susikurtame kataloge.

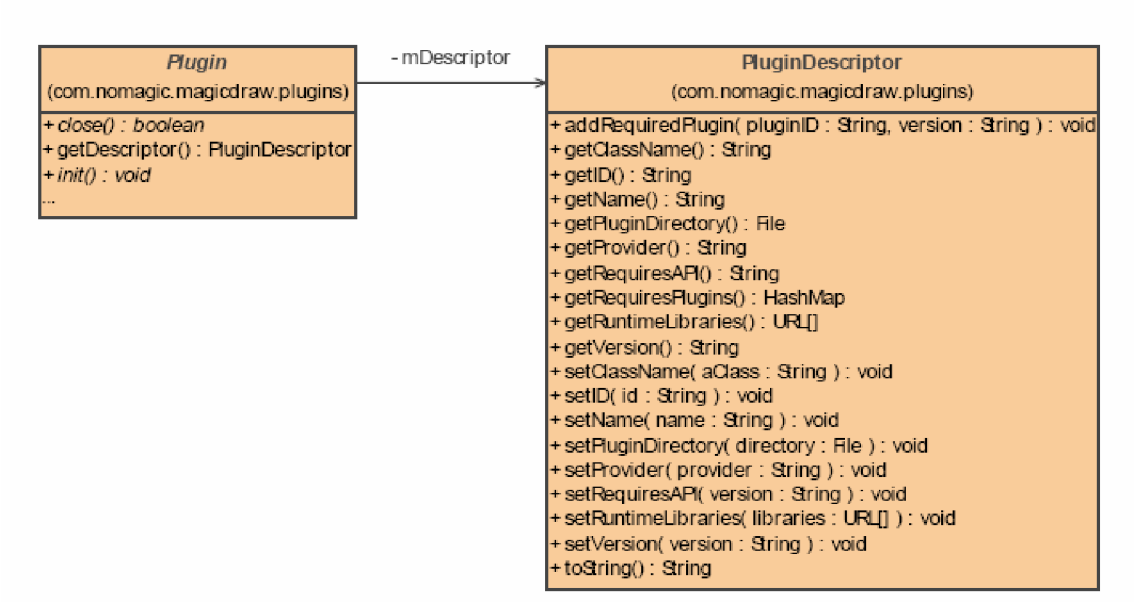
Įskiepio deskriptorius yra failas, parašytas XML kalba ir pavadintas *plugin.xml* vardu. Deskriptoriuje gali būti tik vieno įskiepio ypatybės, t.y. kelių įskiepių vienas deskriptorius aprašyti negali.

Lentelėje 2.5 aprašoma *plugin.xml* struktūra:

2.5 lentelė plugin.xml struktūra

Elementas	Aprašymas	
	Pavadinimas	Aprašymas
plugin	Atributai	
	id	Įskiepio ID turi būti unikalus. Naudojamas įskiepio identifikavimui.
	name	Įskiepio pavadinimas. Šio atributo aprašymą neriboja jokios taisyklės.
	version	Įskiepio versija.
	provider-name	Įskiepio autorius.
	class	Pagrindinė įskiepio klasė, kurioje turi būti init() metodas.
	Įdėtieji elementai	
	requires	MagicDraw UML API versija, reikalinga įskiepiui.
	runtime	dinaminės vartotojo bibliotekos, reikalingos įskiepiui.
requires	Įdėtieji elementai	
	api	Reikalinga MagicDraw UML API versija
	required-plugin	Reikalingi kiti įskiepiai, kurie gali būti naudojami įskiepio paleidimui.
api	Atributai	
	version	Reikalinga MagicDraw UML API versija
required-plugin	Atributai	
	id	Reikalingo įskiepio ID
	version	Reikalingo įskiepio versija
runtime	Įdėtieji elementai	
	library	dinaminė biblioteka, reikalinga įskiepio paleidimui.
library	Atributai	
	name	Reikalingos bibliotekos pavadinimas

Įskiepio klasės pavaizduotos 2.13 paveiksle.



2.13 pav. Įskiepio klasės

2.13 paveiksle *Plugin* yra pagrindinė abstrakti klasė, būdinga kiekvienam MagicDraw UML įskiepiui. Vartotojo sukurtas įskiepis turi būti išplėstinis iš šios klasės. Įskiepyje turi būti du specialūs metodai:

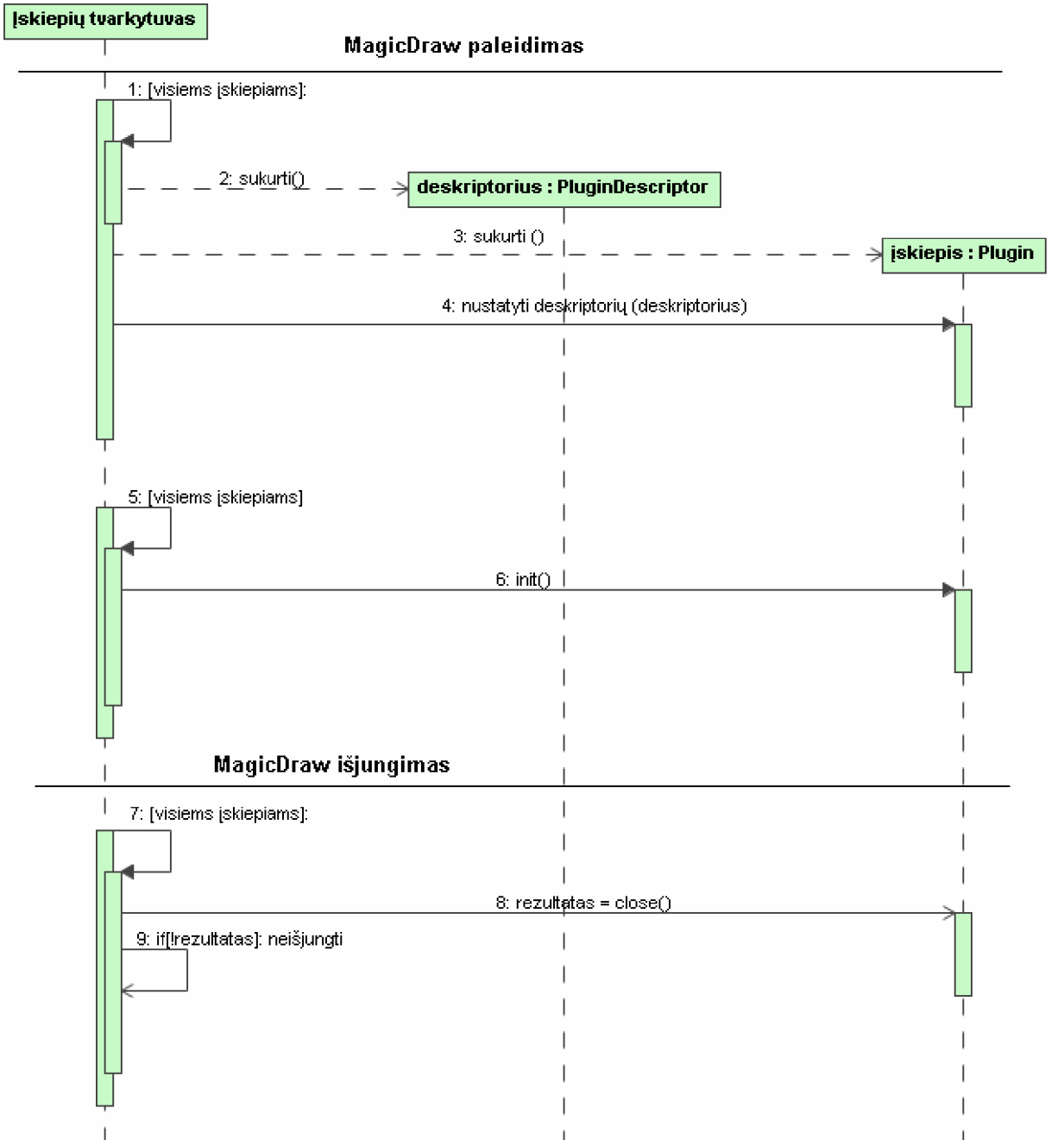
- *public abstract void init()*

  Šis metodas yra iškviečiamas kiekvienu MagicDraw UML paleidimo metu. Iškvietus šį metodą galima naudotis funkcinėmis galimybėmis, kurias suteikia įskiepis.

- *public abstract boolean close()*

  Šis metodas yra iškviečiamas darbo su MagicDraw UML pabaigoje. Jei galima baigti darbą su įskiepiu, metodas turėtų grąžinti *true*, kitu atveju – *false* (tokiu atveju MagicDraw UML išjungimas atšaukiamas).

PluginDescriptor yra klasė, aprūpinanti informacija įskiepį. Informacija užkraunama iš *plugin.xml* failo (deskriptoriaus).



2.14 pav. MagicDraw UML paleidimo ir išjungimo sekų diagrama

2.7. Esamų sprendimų analizė

Pažangiausi CASE įrankiai leidžia užrašyti veiklos taisykles formalia OCL kalba, tačiau IS kūrimo praktikoje ji mažai naudojama. Kalba, naudojama bendravimui tarp sistemos analitiko ir tam tikros srities eksperto, tam, kad išanalizuoti ir dokumentuoti sistemos reikalavimus, neturėtų būti „techninė“. Ji turėtų būti vizuali ir/arba sudaroma natūralios kalbos ar taisyklių reiškiniais, kurie gali būti suprantami tam tikro eksperto be specialaus techninio pasiruošimo. UML modeliuose skirta naudoti kalba OCL leidžia aprašyti invariantus, prieš ir po sąlygas, tačiau ji yra tolima natūraliai kalbai ir sunkiai suprantama ne

tik veiklos atstovams, bet ir projektuotojams ir analitikams.

CASE įrankių, leidžiančių įvesti veiklos taisykles paprastesne kalba, yra nedaug. Galima paminėti URML ir CASE įrankį „Strelka“, reikalavimų valdymo įrankį LeapSE, BROOD požiūrį.

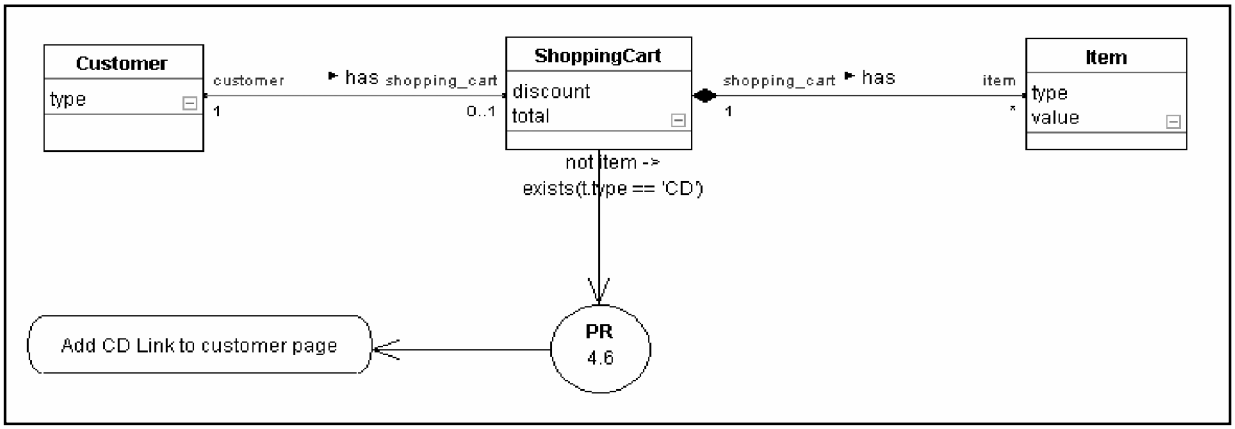
URML ir CASE įrankis „***Strelka“***

Tam, kad būtų galima vizualiai modeliuoti taisykles, REWERSE darbo grupė sukūrė:

- UML paremtą taisyklių modeliavimo kalbą (URML), kuri paveldi UML klasių modelius ir prideda prie jų taisykles;
- Strelka – įrankį, kuriuo galima sudaryti URML modelius.

Grafinis taisyklių vaizdavimas, kai modelis nedidelis, yra puikus reikalavimų specifikavimo būdas. Bet jei modelis didelis ir taisyklių daug, grafinis taisyklių vaizdavimas labai išplės modelį ir jį bus sunkiau analizuoti. Be to, „Strelka“ nagrinėja tik produkcines taisykles.

„Strelka“ modelio pavyzdys pateiktas 2.15 paveiksle.



2.15 pav. „Strelka“ modeliavimo pavyzdys

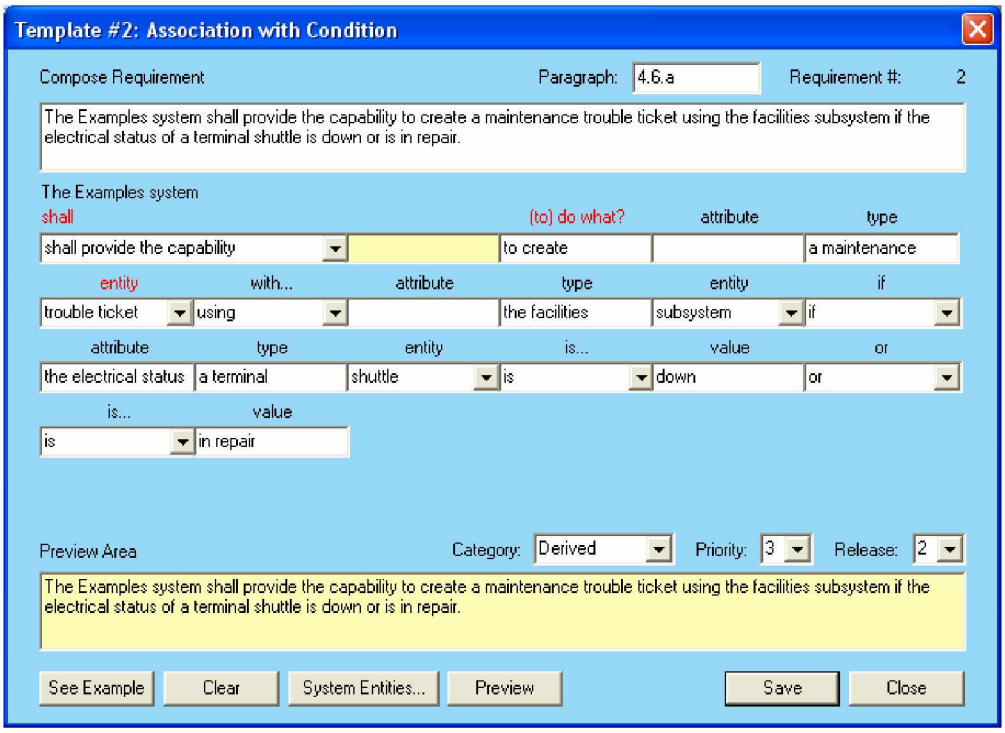
2.15 paveiksle parodyta taisyklė „Jei pirkėjas (customer) neturi pozicijų su CD tipu (type) savo pirkimo kortelėje, tada reikia pridėti CD puslapio nuorodą į pirkėjo puslapį“.

*CASE įrankis **„Leap SE“***

Leap SE yra CASE įrankis, kuris paverčia sistemos reikalavimus tiesiogiai į objekto modelius programinės įrangos projektavimui. Daugiau nei reikalavimų valdymo programa, Leap SE

sutrumpina sistemos kūrimo ciklą ir paspartina greitąjį programų kūrimą (RAD – rapid application development). Leap SE suteikia 22 šablonus greitam ir paprastam reikalavimų sudarymui (18 funkcinių šablonų ir 4 struktūriniai šablonai).

Pavyzdys. 2.16 pav. pavaizduotas "Asociacijos su Sąlyga" šablonas. Šis funkcinis šablonas naudojamas suskaidant reikalavimą, kuris pabrėžia bendradarbiavimą tarp sistemos esybių paremtų sąlygomis. Kada <if> yra nurodytas ir reikalavimas yra išsaugotas, objekto modelio duomenų bazėje yra sukuriamas asociacijos ryšys.



2.16 pav. Leap SE funkcinio šablono pavyzdys

Leap SE trūkumai – labai specifiniai, neišsamūs šablonai, neapimantys taisyklių įvairovės.

„BROOD" (Business Rules-driven Object Oriented Design) požiūris

BROOD tikslas - suteikti kūrimo aplinką, kur veiklos analizės ir sistemos projektavimo sritys yra palaikomos veiklos taisyklių modeliavimu taip palengvinant efektyvų programų vystymą. BROOD sukurti taisyklių šablonai. BROOD šablonai sukurti klasikiniams taisyklių tipams: atributų ir ryšių ribojimams, veiksmų teiginiams, skaičiavimams ir išvedimo taisyklėms.
BROOD metamodelis taip pat aprašo veiklos taisyklių organizavimo bei tvarkymo elementus. Tokie elementai yra taisyklių rinkinys, veiklos procesas ir savininkas. Taisyklių rinkinys naudojamas veiklos taisyklių grupavimui. Kiekvienas veiklos taisyklės modelis privalo turėti vienintelį taisyklių rinkinį, kuris laikomas šakniniu taisyklių rinkiniu. Toks taisyklių rinkinys

turi mažiausiai vieną taisyklių teiginį arba kitą taisyklių rinkinį. Veiklos taisyklių rinkinius galima priskirti procesams. Tinkamas veiklos taisyklių priskyrimas procesams palengvina veiklos taisyklių išrinkimą, kai veiklos taisyklių rinkinys yra didelis. Kiekviena veiklos taisyklė privalo turėti savininką. Savininkas gali būto organizacijos vienetas, individualus vartotojas, vartotojų grupė ar rolė.

Taigi, BROOD požiūryje taisyklės iš karto susiejamos su programiniais komponentais – vienu konkrečiu sprendimu.

Mūsų tyrimo tikslas yra išbandyti galimybes sukurti bendresnę artimą natūraliai veiklos taisyklių specifikavimo kalbą, kurią būtų galima taikyti objektiniuose modeliuose ir ateityje transformuoti į OCL.

2.8. Siekiamas sprendimas

- Kuriamas sprendimas skirsis nuo URML tuo, kad taisyklių modelis bus kuriamas UML Constraints pagrindu. Šis UML elementas geriausiai atitinka veiklos taisyklės sąvoką, taip pat UML ribojimus lengviau transformuoti į OCL;
- Taisyklių struktūra skirsis nuo taisyklių šablonų, skirtų veiklos atstovams – jos bus griežtesnės, artimesnės analitikams ir projektuotojams, kuriantiems reikalavimų ir projektinius modelius;
- Taisyklės papildys IS projektavime naudojamas diagramas ir bus įvedamos UML metamodelyje numatytuose taškuose:
  - klasėse kaip ribojimų (Constraint) tipo elementai;
  - operacijose kaip prieš sąlyga (Pre condition) ir po sąlyga (PostCondition);
  - State elementuose kaip Guard Tipo elementai;
  - Call Action elementuose kaip Constraint tipo elementai;
  - Message elementuose kaip Guard tipo elementai.
- Taisyklių struktūros dalis bus vaizduojama grafiškai;
- Toks būdas artimas įprastam kūrimui, kuris bus išplėstas taisyklių specifikavimu;
- Veiklos taisyklių reiškiniai turi būti formuojami naudojant dalykinės srities terminų žodyną, kuris aprašomas UML esybių klasių diagramomis
- Veiklos taisyklių reiškiniai turi būti struktūrizuoti, rekursiškai plečiami;
- Jie turi būti lengvai suprantami vartotojui;
- Šie reiškiniai turi turėti galimybę ateityje transformuoti juos į OCL ir į kitas formalias kalbas, t. y., turi būti paremti logine reiškinių struktūra.

2.9. Analizės išvados

1. Informacinių sistemų projektavimo įrankiai neturi patogių priemonių veiklos taisyklėms specifikuoti.
2. Veiklos taisyklių klasifikacijų ir metamodelių analizė parodė, kad visos klasifikacijos ir metamodeliai turi tam tikrą bendrą karkasą, o įvairūs autoriai skirtingai jas plėtoja.
3. Natūraliai kalbai artimų veiklos taisyklių šablonų analizė parodė, kad esami šablonai skirti veiklos taisyklių specifikavimui tekstu nenaudojant grafinių simbolių.
4. Šis tyrimas skirtas sistemos projektavimui palengvinti, naudojant grafinius simbolius ir tekstinius reiškinius, todėl mūsų šablonai bus skirti veiklos taisyklėms įvesti papildant UML diagramas.
5. UML taisyklių vaizdavimo galimybių analizė parodė, kad aktualiausia būtų įvesti veiklos taisykles klasių ir elgsenos diagramose (būsenos, sekų ar veiklos).

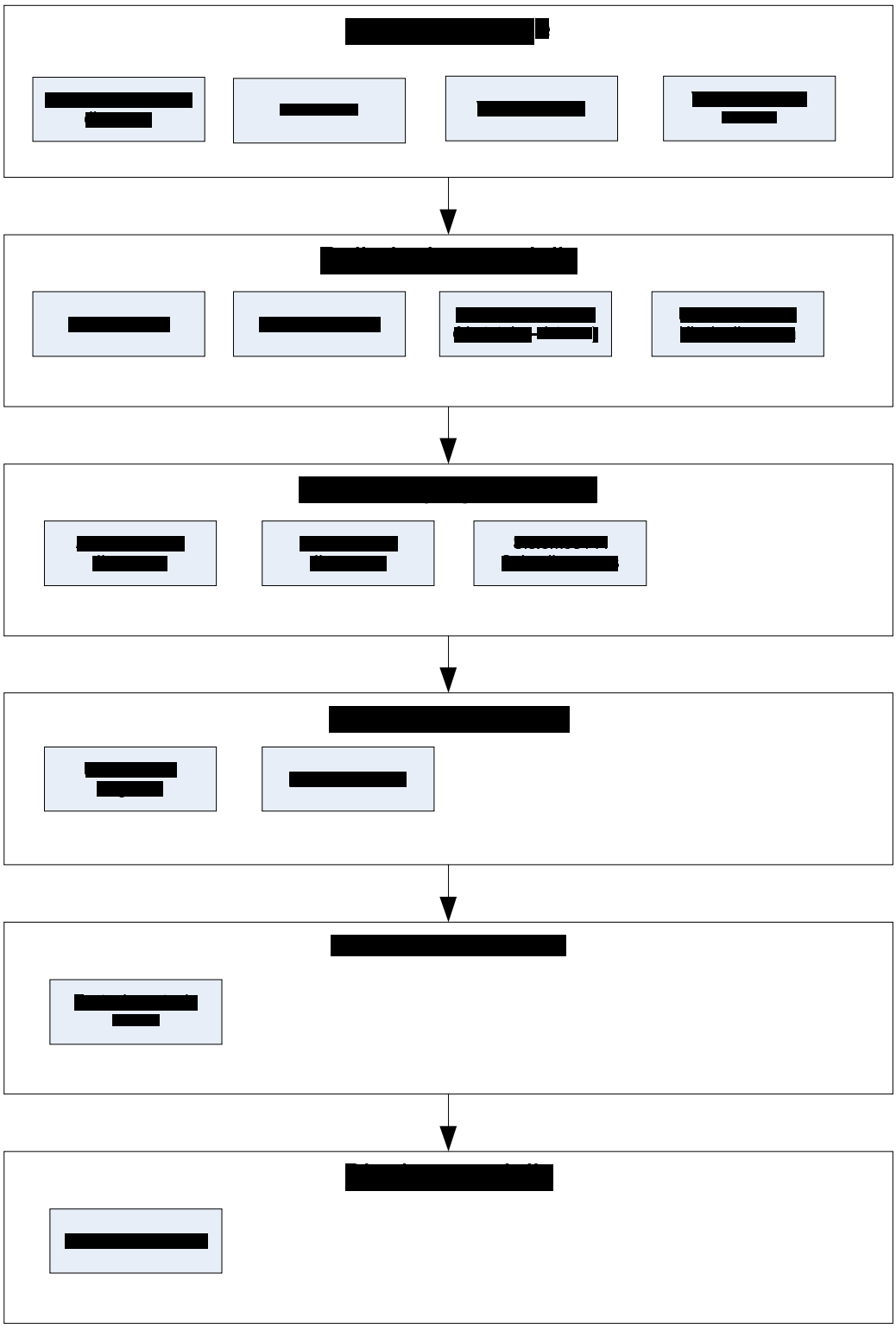
3. OBJEKTINIŲ VEIKLOS TAISYKLIŲ SPECIFIKAVIMO MODELIS IR JO REALIZACIJA

Šioje dalyje aprašyti veiklos taisyklių šablonų specifikavimo įrankio reikalavimai ir projektiniai sprendimai, kurie buvo pritaikyti įrankio kūrimo proceso metu. Taip pat šioje dalyje aprašomas realizuotas modulis, skirtas praplėsti šablonų panaudojimą UML CASE įrankyje. Pateikti veiklos taisyklių formavimo MagicDraw UML įrankyje metodai, aprašytas projekto modelis ir šablonizuotos kalbos, artimos natūraliai kalbai metodika. Sistemos realizacijos poskyryje aprašomi sukurti įskiepio komponentai. Pateikiamas veiklos taisyklių ataskaitoms generuoti skirtas šablonas, bei įskiepio testavimo planas. Sukurtas prototipas leidžia įvesti veiklos taisykles į 5 UML diagramas. Pateikiami kontroliniai duomenys ir rezultatai. Projektinėje dalyje įvardijami prototipo tolimesni planai

3.1. Taisyklių specifikavimo įrankio projektavimo procesas

Šio projekto tikslas – papildyti MagicDraw UML įrankį galimybe įvesti struktūrizuotus ribojimus, sudarant juos iš dalykinės srities modelio elementų, operatorių ir funkcijų. Įrankis padidins kūrimo proceso automatizavimo laipsnį. Tokiu būdu įskiepis padės analitikui, nemokančiam specialių modeliavimų kalbų, lengviau įvesti veiklos taisykles ir ribojimus, taip padidinant modelių išsamumą ir teisingumą.

Projektavimui buvo pasirinkta RUP metodika su keletu specifinių modifikacijų. Kai kurie modeliai nebuvo panaudoti dėl to, kad negali būti sudaryti dėl esamos projekto stadijos arba projekto specifikos. Projektavimo eiga suskirstyta į 6 etapus: aplinkos analizė, reikalavimų modelis, analizė ir projektavimas, realizacijos modelis, testavimo modelis ir diegimo modelis. Detalesnis projektavimo etapų vaizdas su konkrečiais modeliais pateiktas 3.1 pav.



3.1 pav. Projektavimo etapai

Realizacijos etape nėra duomenų bazės diagramos, nes duomenys bus laikomi projektiniame MagicDraw UML faile.

Projektavimo įrankiu buvo pasirinktas programinis paketas MagicDraw UML 15.5. Šis paketas buvo pasirinktas dėl to, kad jam skirtas magistrinis darbas, taip pat įrankis palaiko RUP metodiką, teigiamai vertinamas projektuotojų ir vienas iš esminių kriterijų buvo turima

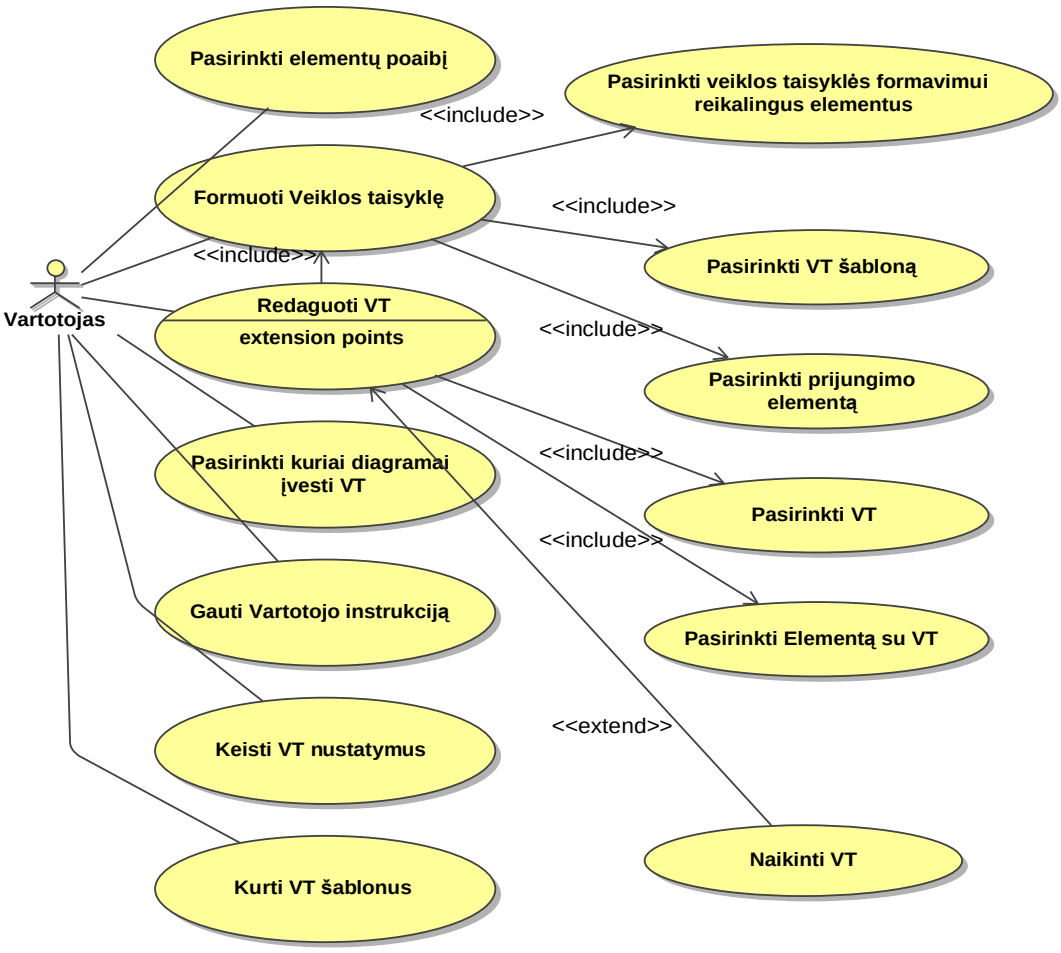
patirtis dirbant šiuo įrankiu.

3.2. Objektinių veiklos taisyklių specifikavimo įrankio reikalavimai

3.2.1. Specifikavimo įrankio panaudojimo atvejai

Kuriamoje sistemoje vartotojai neskirstomi į tam tikrus lygius – egzistuoja vienas vartotojas. Vartotojas paleisdamas įskiepį atidaro naują langą, kuriame jis gali atlikti veiklos taisyklių įvedimo, redagavimo, peržiūrėjimo, šalinimo funkcijas. Įvedant ribojimus sistema pasiūlo pasirinkti sudedamąsias ribojimo dalis, bei prie kurio diagramos elemento „prikabinti“ ribojimą. Formuojant veiklos taisyklę, galima pasirinkti siūlomų elementų apimtį, pagal projekto hierarchiją. Redaguojant ribojimus pirmiausia reikia pasirinkti diagramos elementą, kurio ribojimus norime redaguoti. VT šablonus ir VT sąlygas galima kurti bei redaguoti, nes jos yra kartu su MagicDraw UML projektu.

Panaudojimo atvejų schema pateikta 3.2 paveiksle.



3.2 pav. Panaudojimo atvejų diagrama

3.2.2. Panaudojimo atvejų specifikacijos

Pagrindiniai vartotojo panaudojimo atvejai specifikuoti pagal specifikavimo lentelės šabloną, kurioje nurodomi pagrindiniai panaudojimo atvejo parametrai: pavadinimas, sąlyga prieš, sąlyga po, susiję panaudojimo atvejai, pagrindinis įvykių srautas ir sistemos reakcija. Šių lentelių informacija papildo vartotojo PA modelį. Panaudojimo atvejų specifikacijos pateiktos lentelėse: Lentelė 3.1, Lentelė 3.2.

3.1 Lentelė PA "Formuoti veiklos taisyklę" specifikacija

PA „Formuoti veiklos taisyklę“		
Prieš sąlyga		Sukurtas projektas. Atidarytas pagrindinis programos langas. Pasirinkta diagrama.
Sužadinimo sąlyga		Vartotojas nori sukurti ribojimą
Susiję panaudojimo atvejai	Išplečia PA	
	Apima PA	Pasirinkti VT šabloną Pasirinkti VT formavimui reikalingus elementus Pasirinkti prijungimo elementą
	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas pasirenka diagramos elementą ribojimui kurti. 2. Vartotojas pasirenka šabloną arba kuria savo naują. 3. Vartotojas renkasi diagramos elementus. 4. Vartotojas patvirtina ribojimo saugojimą		1. Sistema pažymi pasirinktą elementą. 2. Sistema į pagrindinį ribojimo langą įrašo pasirinktą vartotojo šabloną 3. Sistema filtruoja ir pateikia sąrašą galimų diagramos elementų ribojime. 4. Sistema užsaugo vartotojo sudarytą ribojimą prie diagramos elemento.
Po sąlygą		Diagramoje pasirinktam elementui sukuriamas ribojimas.

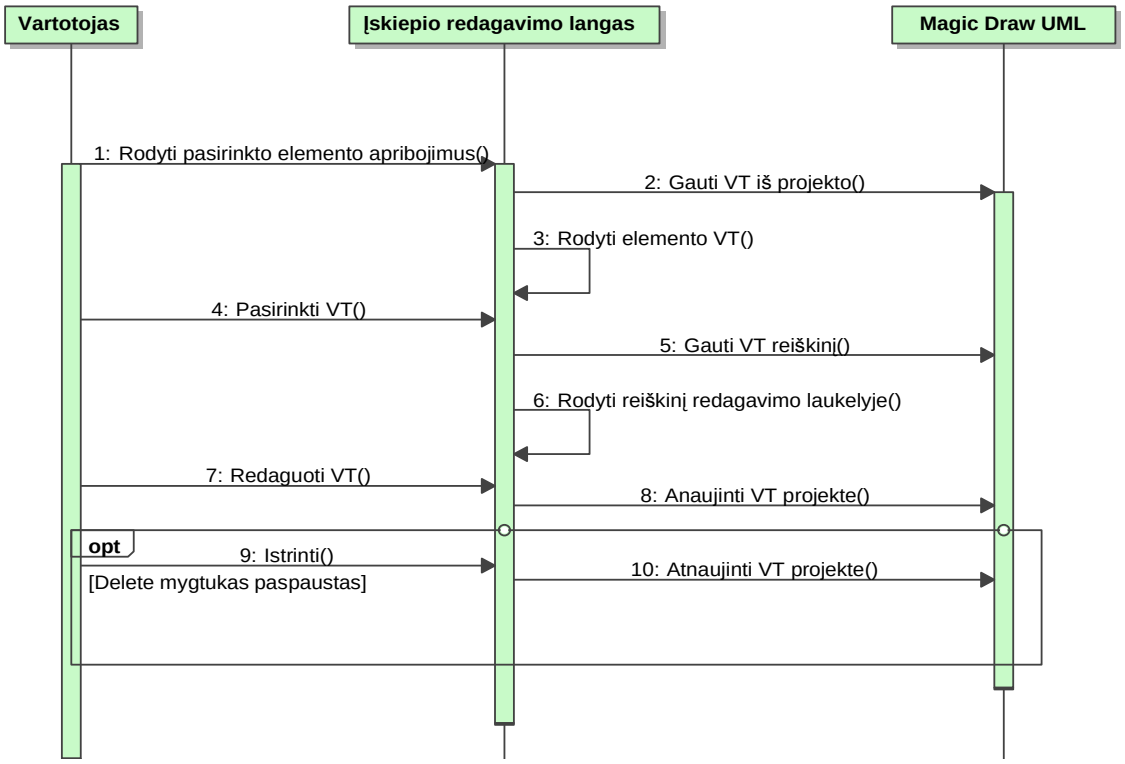
Lentelė 3.2 PA "Redaguoti VT" specifikacija

PA „Redaguoti VT“		
Prieš sąlyga		Sukurtas projektas. Atidarytas pagrindinis programos langas. Pasirinkta diagrama.
Sužadinimo sąlyga		Vartotojas nori redaguoti veiklos taisyklę
Susiję panaudojimo atvejai	Išplečia PA	Ištrinti pasirinktą VT
	Apima PA	Formuoti VT Pasirikti konkrečią VT Pasirinkti elementą su VT
	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai

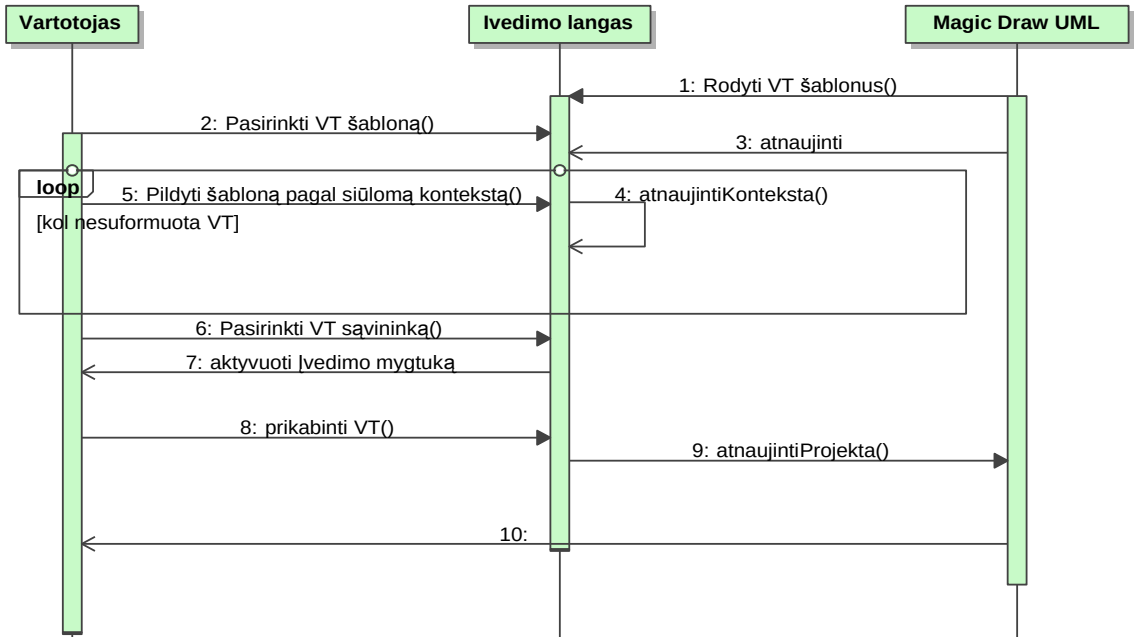
1. Vartotojas pasirenka diagramos elementą su ribojimu. 2. Vartotojas pasirenka konkrečią VT redagavimui. 3. Vartotojas renkasi diagramos elementus. 4. Vartotojas patvirtina redaguoto ribojimo saugojimą	1. Sistema pažymi pasirinktą diagramos elementą ir išveda visus jo ribojimus. 2. Sistema įveda pasirinktą ribojimą į redagavimo komponentą. 3. Sistema filtruoja galimus įrašyti į šabloną elementus. 4. Sistema užsaugo redaguotą ribojimą.
Po sąlygą	Diagramoje vaizduojamas redaguotas ribojimas.

3.2.3. Vartotojo ir sistemos sąveikos diagramos

Šio etapo sekų diagramose modeliuojamas bendravimas tik tarp sistemos vartotojo ir pačios sistemos. Sistemos komponentai nėra išskiriami ir į sistemą. Vartotojas atlieka veiksmus įskiepio languose (redagavimo arba įvedimo) ir jo atlikti veiksmai atsispindi MagicDraw UML sistemoje. Vartotojo ir sistemos sekų diagramų tikslas yra parodyti kokius veiksmus, kiek jų ir kokia tvarka turės atlikti vartotojas norėdamas atlikti vieną panaudojimo atvejį. Sekų diagramos yra sudarytos pagrindiniams panaudojimo atvejams ir pateiktos paveiksluose: 3.3 ir 3.4.



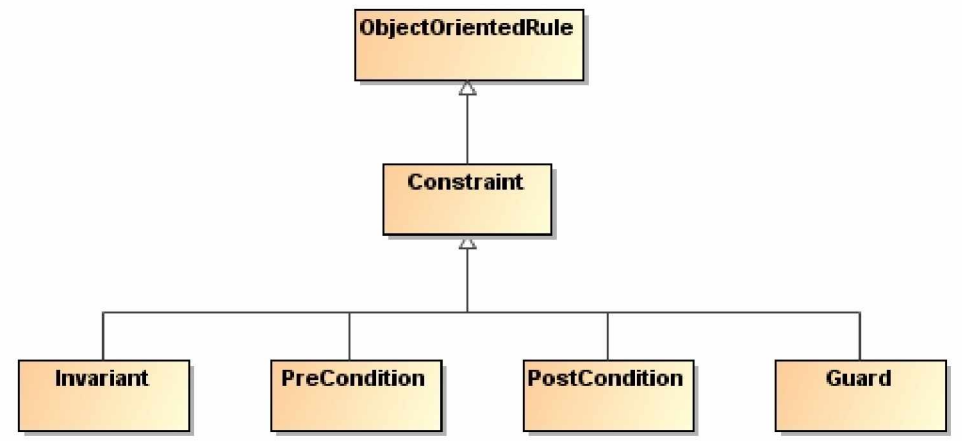
3.3 pav. Redaguoti VT panaudojimo atvejų sekų diagrama



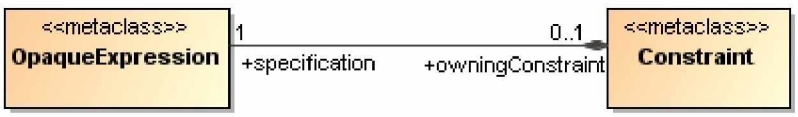
3.4 pav. Įvesti VT panaudojimo atvejų sekų diagrama

3.2.4. Objektinių veiklos taisyklių metamodelis

3.5 paveiksle pavaizduoti objektinių veiklos taisyklių tipai. Visos objektinės taisyklės yra ribojimai, vaizduojami UML metaklase „Constraint“. Galimi veiklos taisyklių ribojimų tipai yra invariantai, „prieš“ sąlygos, „guard“ sąlygos ir „po“ sąlygos. Taisyklių reiškiniai užrašomi naudojant metaklasės ValueSpecification poklasę OpaqueExpression, kadangi šie reiškiniai neinterpretuojami UML (3.6 pav.).



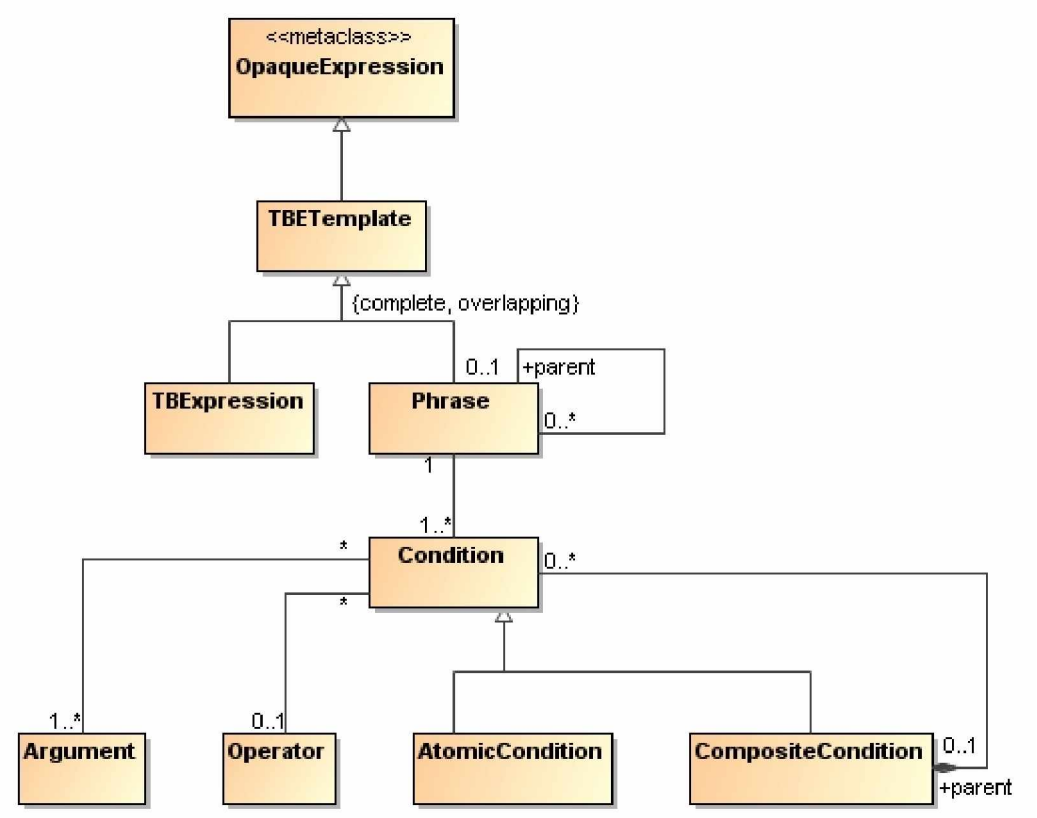
3.5 pav. Objektinės taisyklės – UML ribojimai



3.6 pav. Ribojimų reiškiniai UML metamodelyje

Įskiepiu įvedami ir redaguojami ribojimų reiškiniai yra struktūrizuoti. Ribojimo

reiškinio struktūra pavaizduota 3.7 paveiksle.



3.7 pav. Veiklos taisyklės reiškinių struktūra

Ribojimų reiškiniai susideda iš argumentų, operatorių, sąlygų ir frazių. :

- ***Argumentas***

  Argumentas – tai svarbus ribojimo reiškinio elementas, kuris nurodo konkrečių klasės diagramos elementų panaudojimą. Mūsų sukurtame įskiepyje argumentai žymimi specialiu simboliu – laužtiniais skliaustais iš abiejų pusių, pavyzdžiui „[Užsakymas]“.

  Argumentas visada vaizduoja ribojimo sakinio veiksnį – tam tikrą elementą, kuriam yra taikomi tam tikri ribojimai. Argumentu gali būti bet kuris modelio elementas. Ribojime argumento reikšmė bus elemento vardas.

- ***Operatorius***
  - Palyginimo operatoriai: >,>=,=, <=,<, !=
  - Jungimo ir išskyrimo operatoriai: IR, ARBA, ARBA NE.
  - Kitų: egzistuoja, neegzistuoja, ->tusciaAibe, ir kt.

  Operatoriai įskiepyje nėra išskiriami, jie rašomi tiesiogiai, arba yra pasirenkami iš sąlygų formos.

- ***Sąlyga***

  Sąlygos tai argumentų ir operatorių derinys. Jos susideda iš įvairiausių sąlygų

sakinių, tokių kaip lyginimas dviejų argumentų, išskyrimas vieno argumento su specialia sąlyga ir kt. Mūsų sukurtame įskiepyje sąlygos yra vaizduojamos su specialiu simboliu, skliaustu, iš abiejų pusių, pvz.: „([Užsakymas] egzistuoja)“

- ***Frazė:***

  Frazės yra didžiausias ribojimo elementas. Tai įvairūs sąlygų deriniai, kurie aprėminami specialiu simboliu – figūriniai skliaustais {}. Frazės atitinka veiklos taisyklių ruošinius.

Tik naudojant tokias struktūrizuotas taisykles galima realizuoti veiklos taisyklių įskiepį.

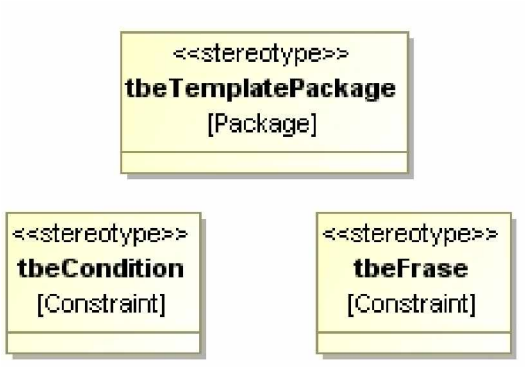
Ribojimo elementai realizuoti įskiepyje.

3.2.5. Įskiepio kalbos elementų profilio panaudojimas TBE šablonams kurti

Įskiepis MagicDraw UML projekte naudoja pagalbinį modulį, skirtą surinkti bei valdyti sukurtus ribojimų šablonus. Panaudojant šį modulį, naujų šablonų ir sąlygų įvedimas tampa labai paprastas. Įskiepis pagal panaudotus stereotipus leidžia tiesiogiai panaudoti naujus elementus ribojimų įvedimo dialoge (3.7 pav.).

Pagrindiniai TBEProfile elementai:

- tbeTemplatePackage Stereotipas. Šio stereotipo metaklasė yra paketas. Stereotipas skirtas identifikuoti paketą, kuriame būtų sudėti visi naudojami TBE šablonai ir sąlygos. Tokiu būdu TBE įskiepis atpažįsta šablonus ir sąlygas ir rodo jas įvedimo dialoge.
- tbeCondition Stereotipas. Skirtas taikyti Constraint tipo elementams. Šie elementai aprašo tam tikrą sąlygos šabloną, pvz., „[argumentas1] > [argumentas2]“ . Visi sukurti nauji sąlygos šablonai turi būti paketo su tbeTemplatePackage stereotipu viduje.
- TbeFrase Stereotipas. Skirtas taikyti Constraint tipo elementams. Šie elementai aprašo tam tikrą ribojimo frazę, pvz. „ If ((condition) then {fraze1} else { show System.Error message}”. Sukurti šablonai turi būti padėti į stereotipizuotą tbeTemplatePackage stereotipu paketą.

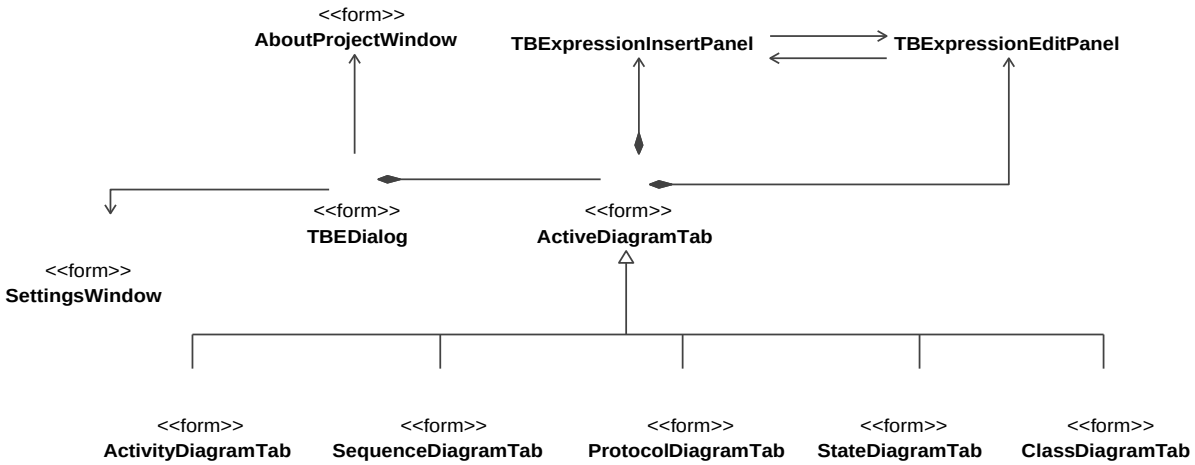


3.7 pav. TBE papildomo profilio elementai

Naudojant TBEProfile modulį, ribojimų įvedimo įskiepis tampa mažai priklausomas nuo struktūrizuotos ribojimų kalbos. Ribojimų kalbą gali pasirinkti pats projektuotojas ar analitikas ir taikyti ją projekte, naudodamas stereotipus.

3.2.6. Vartotojo interfeiso modelis – navigavimo planas

Veiklos taisyklių įvedimo įskiepio navigavimo plane pateikiami visi pagrindiniai langai reikalingoms funkcijoms atlikti. Modulis neišskiria vartotojų tipų, todėl schema yra bendra visiems vartotojams. Sistema suprojektuota taip, kad vartotojas nesunkiai pereitų nuo vienos funkcijos prie kitos. Meniu juostoje pateikta parametrų nustatymo funkcija. Navigavimo planas pateikiamas 3.8 paveiksle.



3.8 pav. Vartotojo interfeiso modelis - navigavimo planas

3.2.7. Vartotojo interfeiso modelis- formos

Įskiepio interfeiso modelis turės pagrindines dvi formas:

- Veiklos taisyklių įvedimo forma

  Veiklos taisyklių įvedimo formą sudaro 5 pagrindinės zonos (3.9 pav.).

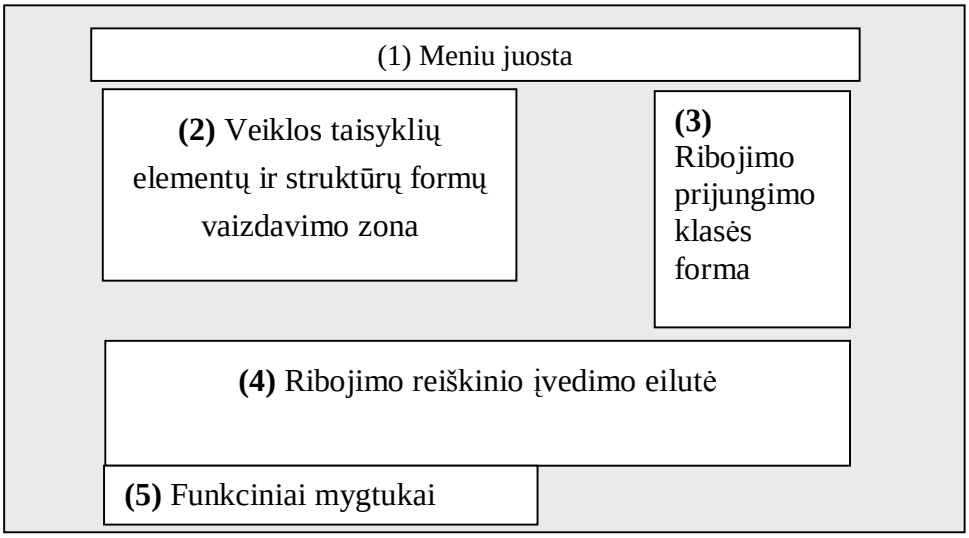
§ (1) meniu juosta;

§ (2) veiklos taisyklių elementų ir struktūrų formų vaizdavimo zona;

§ (3) ribojimo prijungimo klasės forma;

§ (4) ribojimo reiškinio įvedimo eilute;

§ (5) funkciniai mygtukai.



3.9 pav. Veiklos taisyklių įvedimo forma

- Veiklos taisyklių redagavimo – šalinimo.

Veiklos taisyklių redagavimo - šalinimo formą sudaro 6 pagrindinės zonos: (3.10 pav.)

§ (1) meniu juosta;

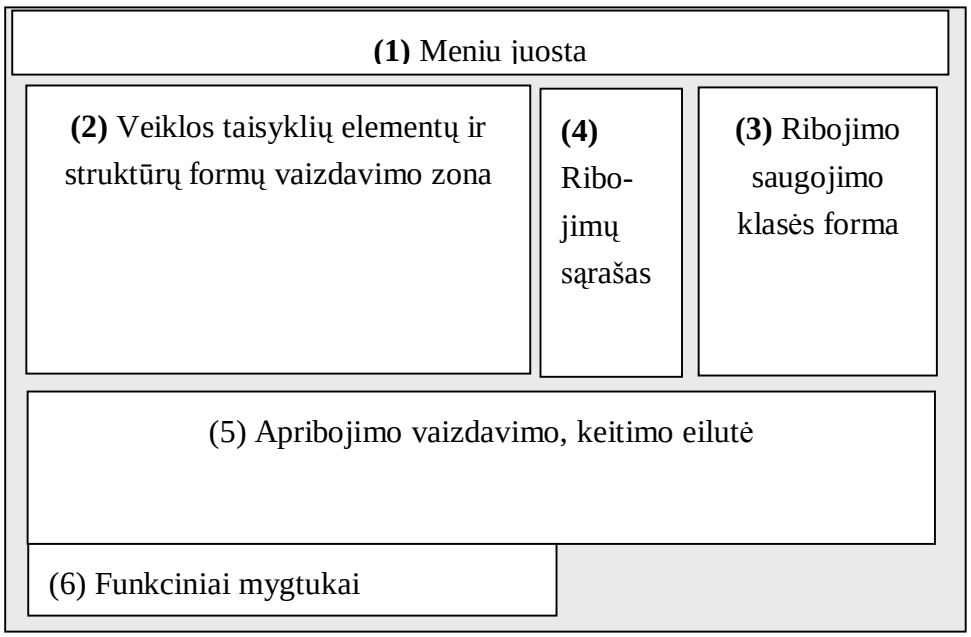
§ (2) veiklos taisyklių elementų ir struktūrų formų vaizdavimo zona;

§ (3) ribojimo saugojimo klasės forma;

§ (4) ribojimo pasirinkimo forma;

§ (5) ribojimo vaizdavimo, keitimo eilutė;

§ (6) funkciniai mygtukai;

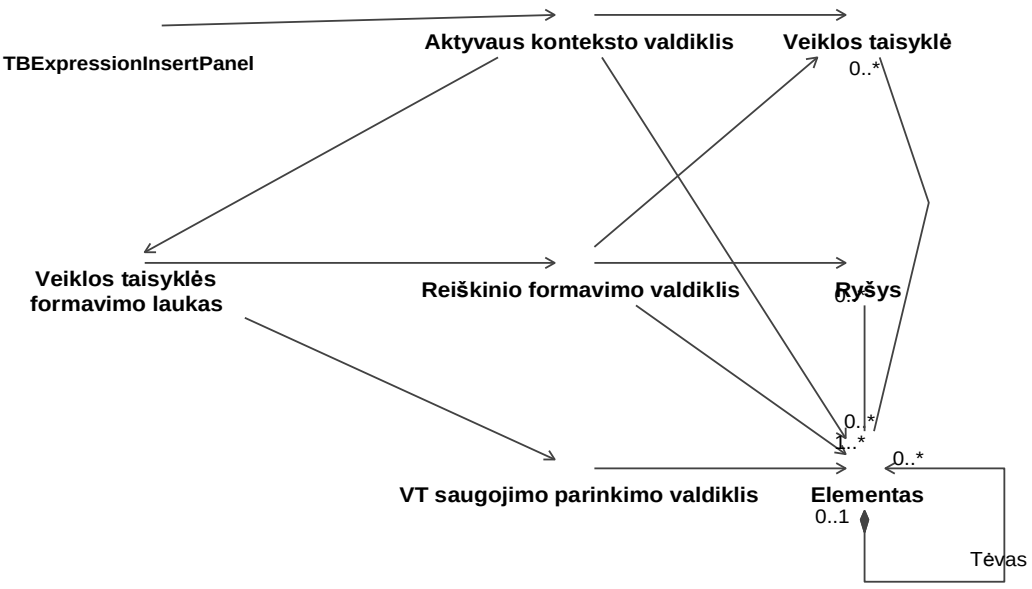


3.10 pav. Veiklos taisyklių redagavimo-šalinimo forma

3.3. Sistemos projektas

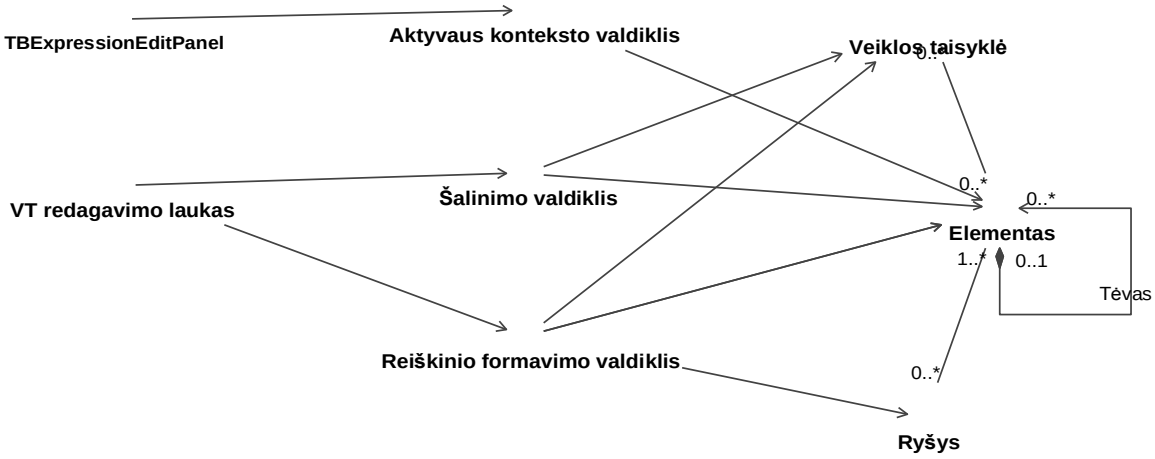
3.3.1. Analizės klasių diagramos

Analizės klasių diagramose pateiktas sistemos vaizdas kuriame susiejami trys sistemos architektūros lygiai: vartotojo sąsajos, valdymo ir esybių klasės. Veiklos taisyklių įskiepio analizės klasių diagramos pateiktos 3.11 ir 3.12 paveiksluose.



3.11 Įvedimo analizės klasių diagrama

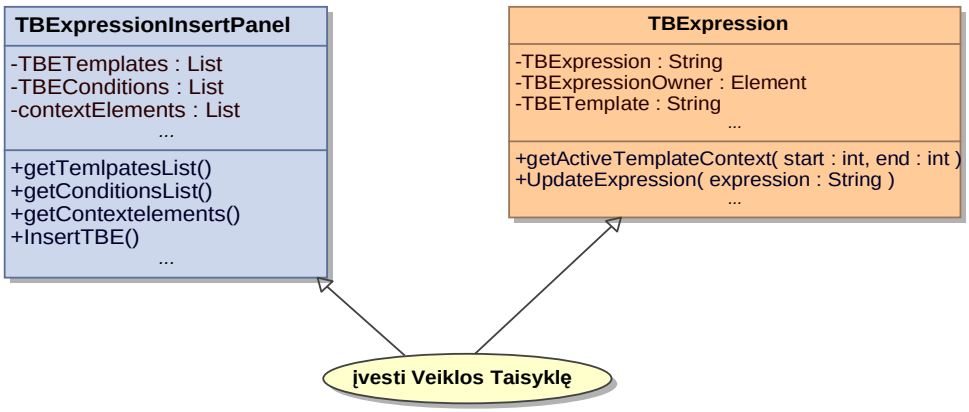
Redagavimo analizės klasių diagrama (3.12 pav.):



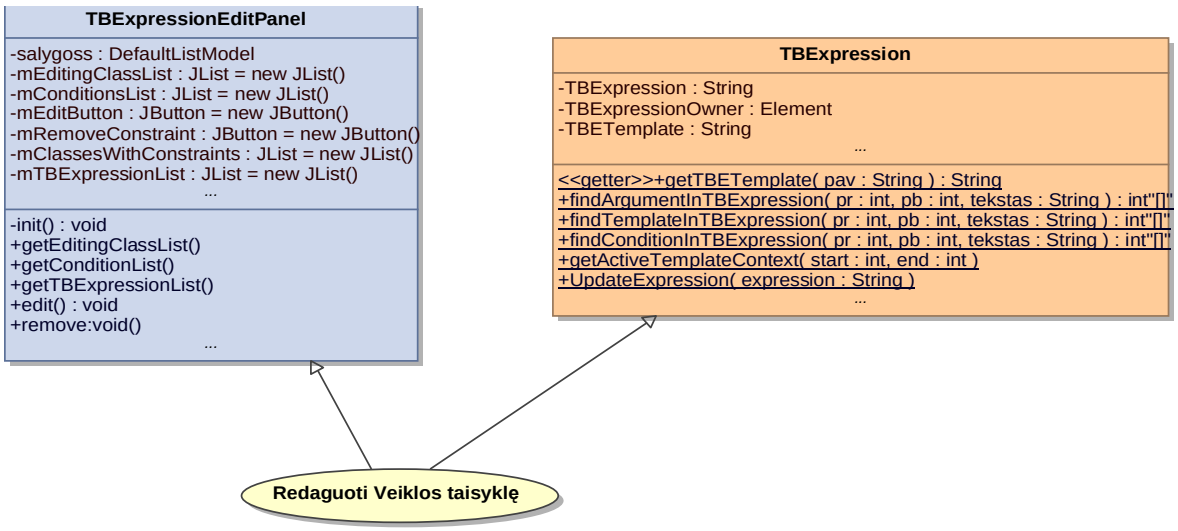
3.12 pav. Redagavimo analizės klasių diagrama

3.3.2. Panaudojimo atvejų realizacijos

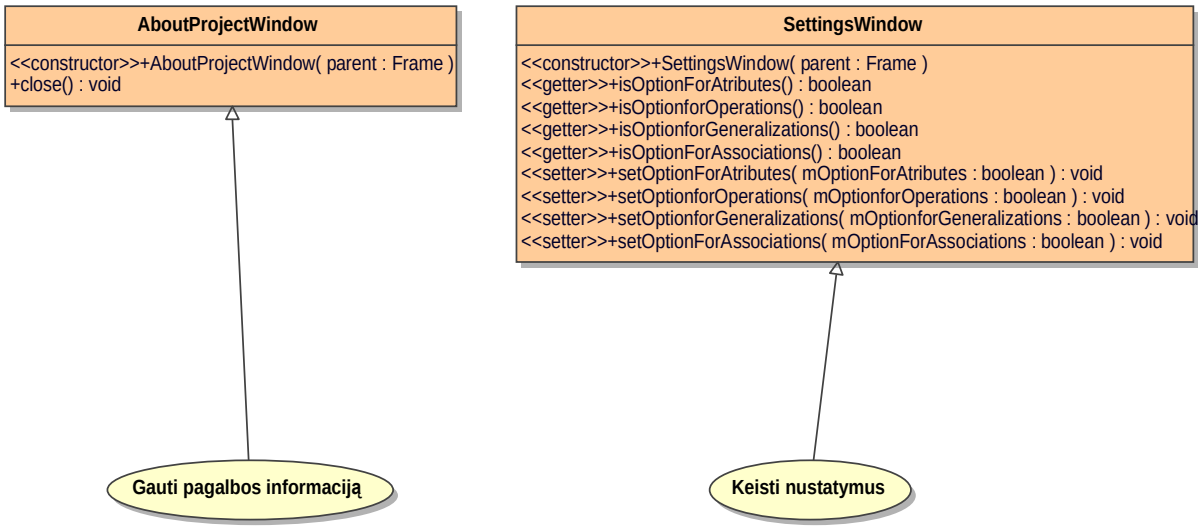
Panaudojimo atvejų realizacijų diagramos vaizduoja juos realizuojančias klases.



3.13 pav. Panaudojimų atvejo „Įvesti veiklos taisyklę“ realizacijos diagrama



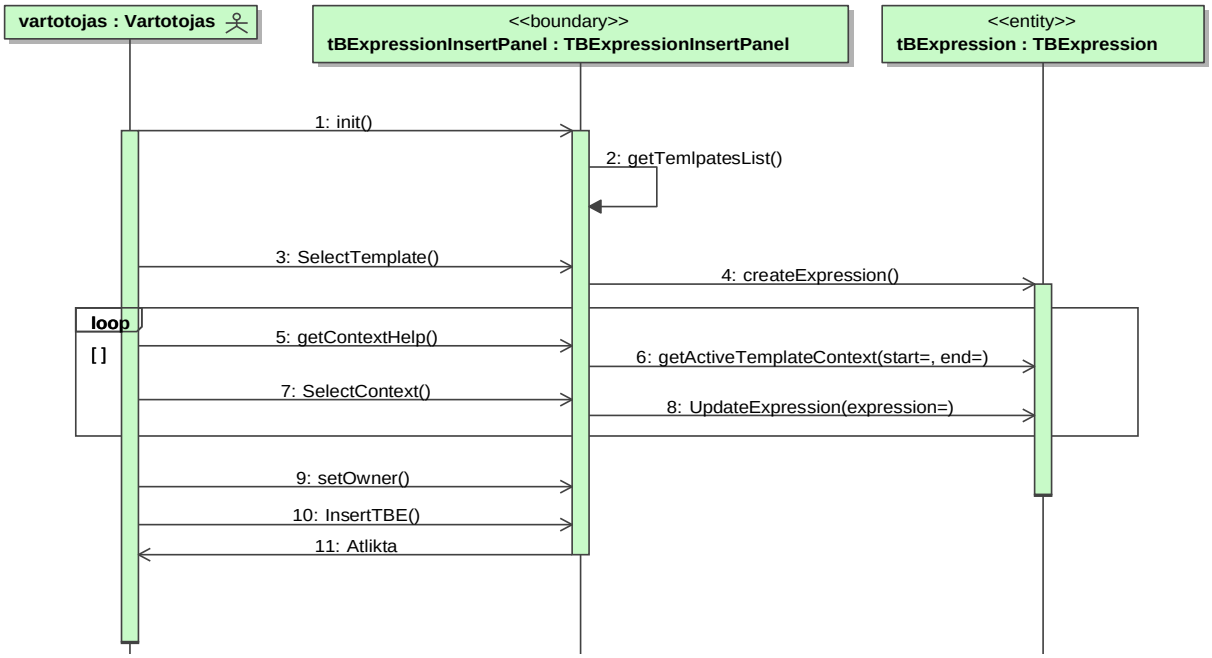
3.14 pav. Panaudojimo atvejo „Redaguoti veiklos taisyklę“ realizacijos diagrama



3.15 pav. Panaudojimo atvejų „Gauti pagalbos informaciją“ ir „Keisti nustatymus“ realizacijos diagrama

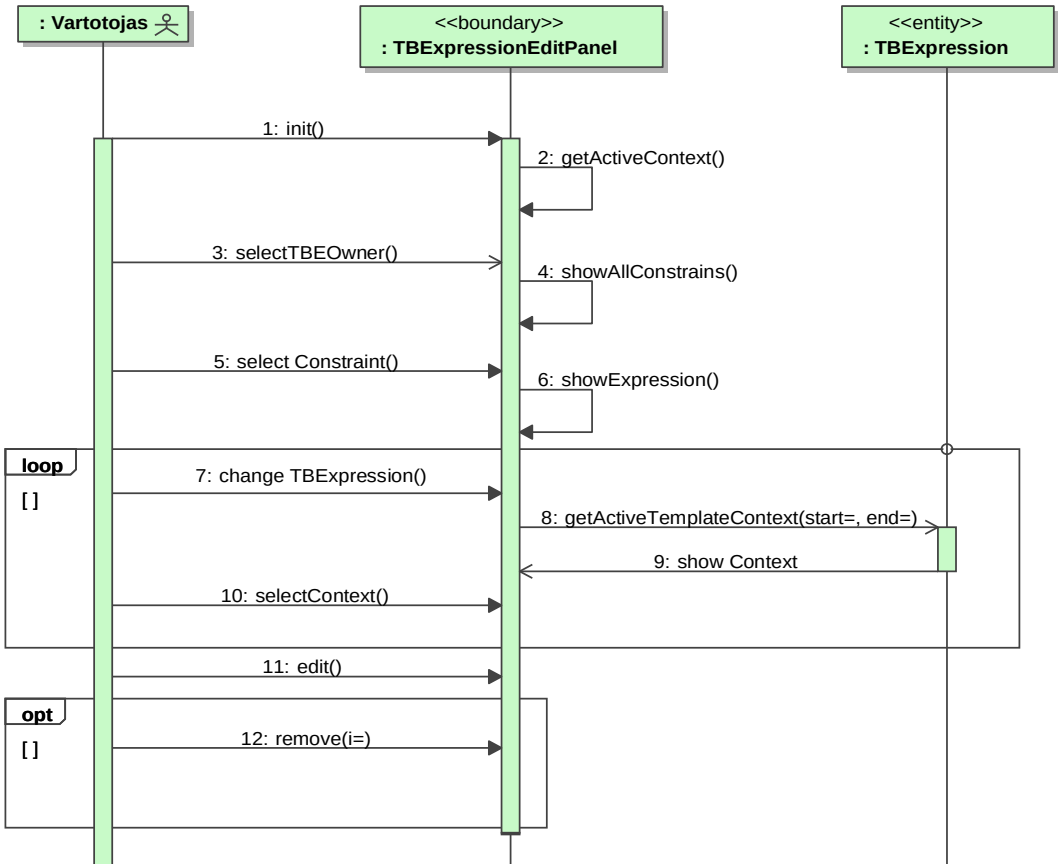
3.3.3. Sistemos panaudojimo atvejų realizacijų sekų diagramos

Pagrindinių panaudojimų atvejų realizacijų sekų diagramos pavaizduotos 3.16 ir 3.17 pav. Veiklos taisyklių įvedimo seka pateikta 3.16 pav. Ji apibrėžia pagrindinių vykdomų funkcijų sekas sukurtame prototipe.



3.16 pav. Veiklos taisyklės įvedimo sekų diagrama

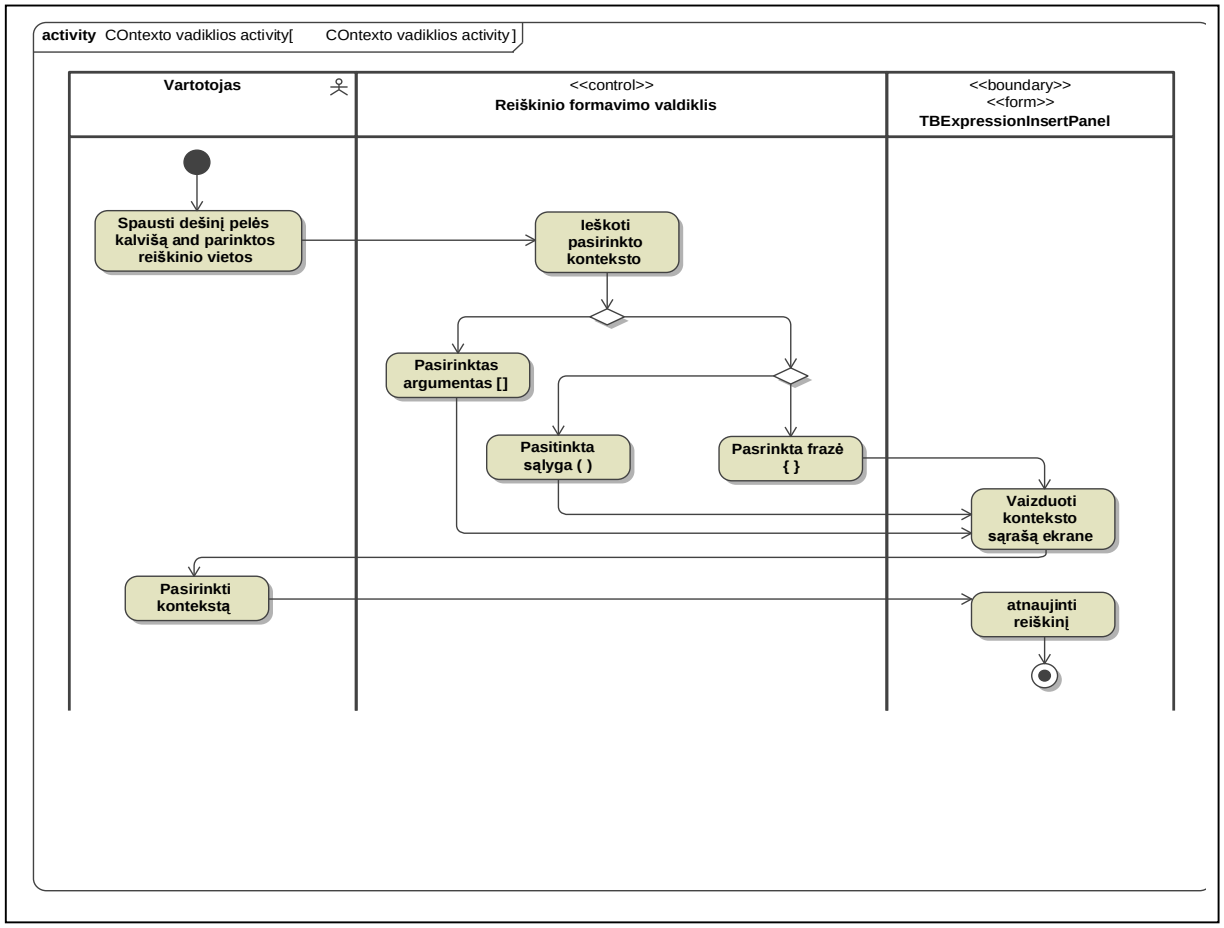
Redaguoti veiklos taisyklę panaudojimo atvejo realizacijos sekų diagrama pateikta 3.17 pav. Ji apibrėžia pagrindinius vartotojo ir sistemos veiksmus pasirinkus redagavimo atvejį. Redagavimo atveju vartotojas atidarydamas TBExpressionEditPanel langą, išsikviečia redaguojamo elemento ribojimą, bei jam taiko redagavimui skirtus veiksmus – ribojimo pakeitimo bei šalinimo.



3.17 pav. Veiklos taisyklės redagavimo sekų diagrama

3.3.4. Reiškinio formavimo naudojant kontekstinę pagalbą veiklos diagrama

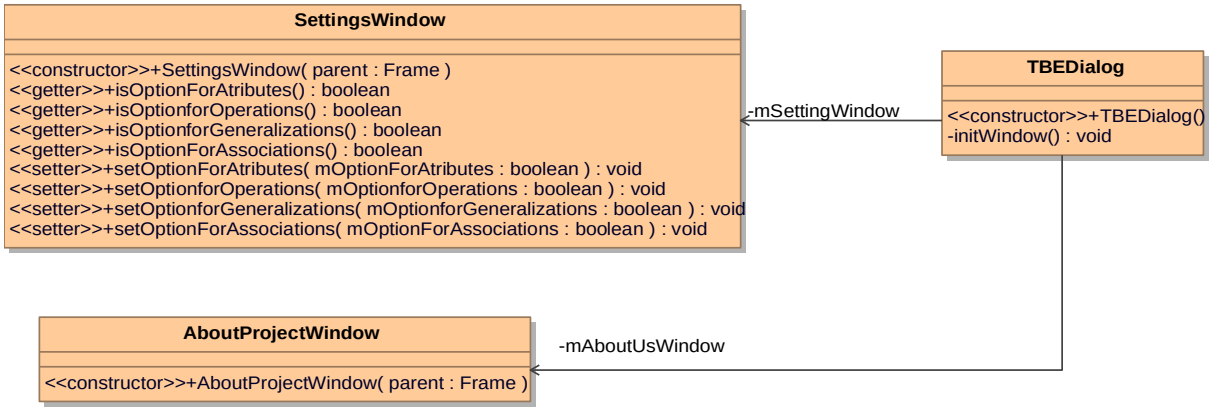
3.18 paveiksle pateikta reiškinio formavimo, naudojant kontekstinę pagalbą, veiklos diagrama. Sistema veikia intuityviai. Paspaudus dešinį pelės klavišą, sistema ieško pasirinkto konteksto, tikrina vartotojas pasirinktą elementą ir pateikia tam kontekstui galimus elementų variantus.



3.18 pav. Reiškinio formavimo veiklos diagrama

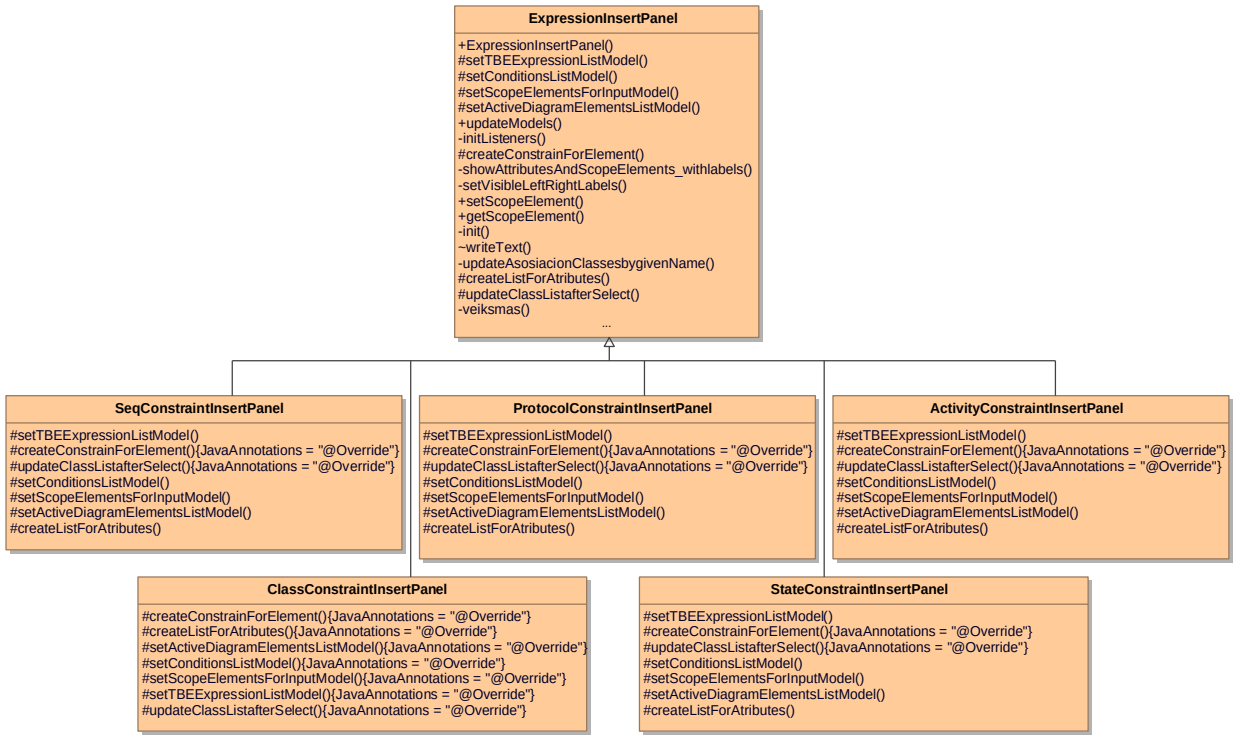
3.3.5. Projekto klasių diagrama

Sistema suprojektuota, taikant trijų lygių architektūros modelį. Remiantis šiuo modeliu ribojimų įvedimo sistemą sudaro vartotojo, veiklos bei duomenų paslaugos, kurios sugrupuojamos į atskirus paketus (3.19 pav.). Duomenų lygmenį sudaro MagicDraw UML XMI failas, vaizdavimo logiką atlieka įskiepio dialogai ir panelės.



3.19 pav. Pagrindinių komponentų klasių diagrama

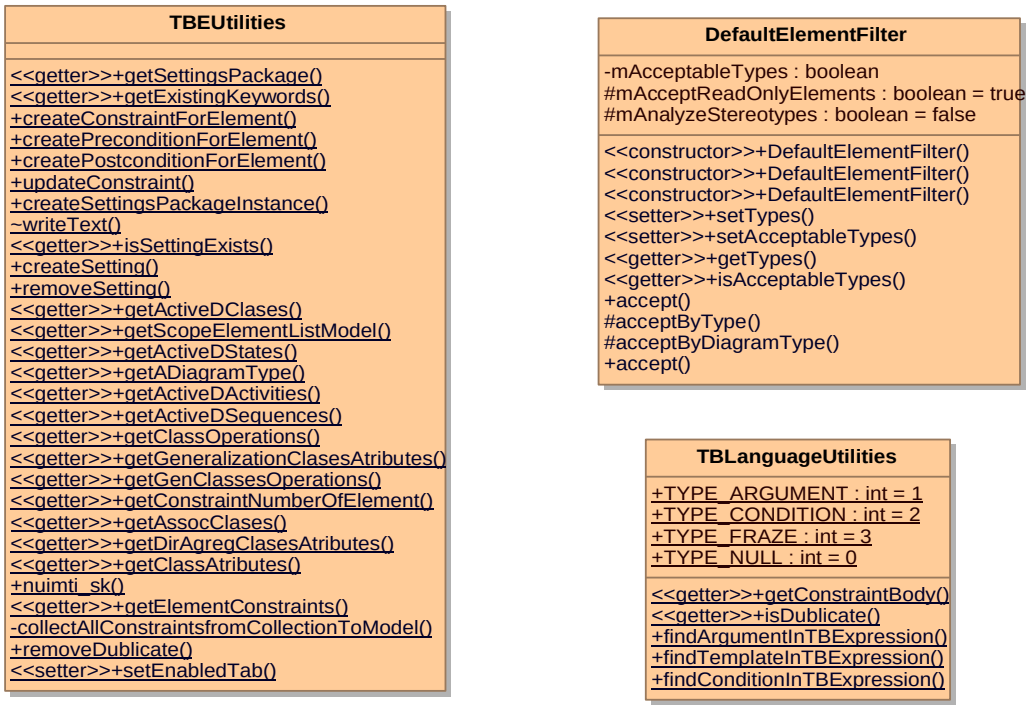
Įskiepio projektas yra objektinis. Įvedimo modulį sudaro pagrindinė pilna ExpressionInsertPanel klasė (ExpressionEditPanel – redagavimui). Kiekvienai specifinei diagramai yra pritaikomi tik atskiri metodai:



3.20 pav. Veiklos taisyklių įvedimo komponentų hierarchinė klasių diagrama

3.21 pav. Veiklos taisyklių redagavimo komponentų hierarchinė klasių diagrama

Pagalbinių įskiepio klasių, skirtų padėti atlikti tam tikrus, dažnai taikomus metodus, diagrama (3.22 pav.):

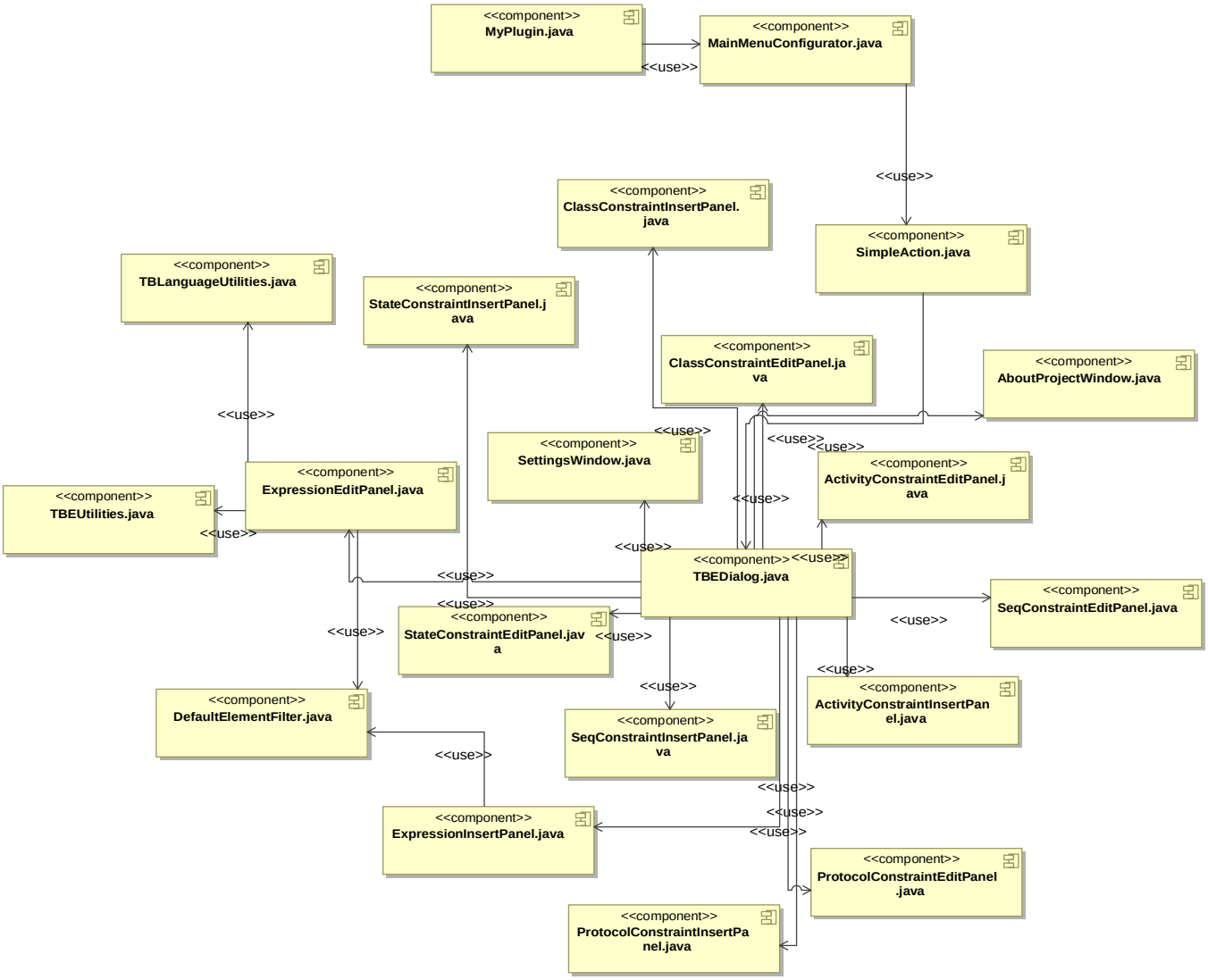


3.22 pav. Pagalbinių klasių diagrama

3.4. Sistemos realizacija

3.4.1. Komponentų ir įrangos diagramos

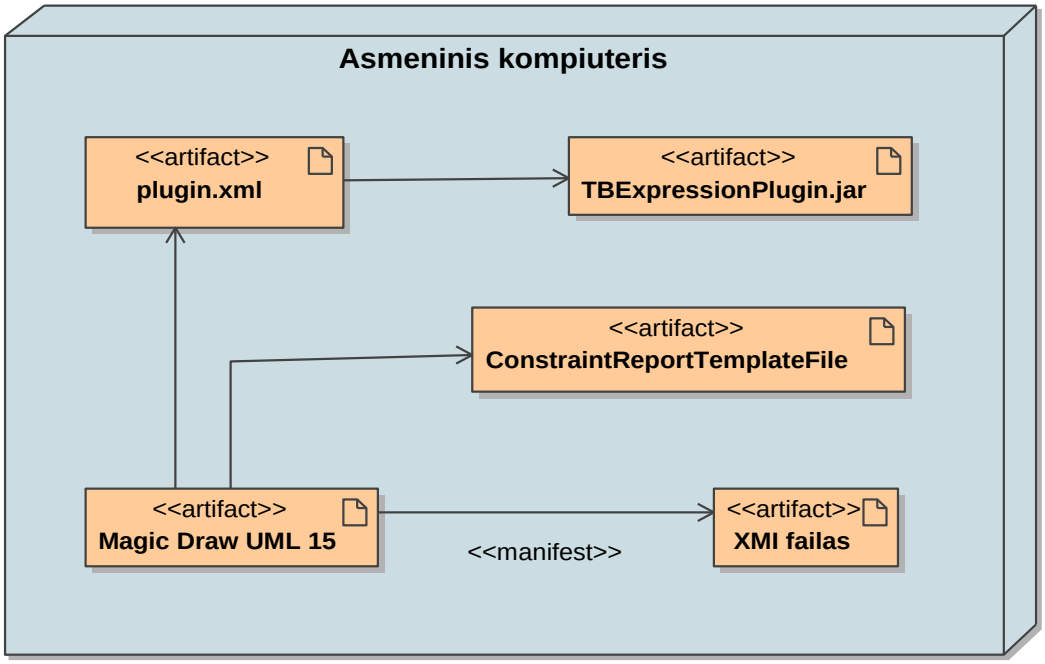
Veiklos taisyklių kūrimo įskiepis projektuotas objektiniu principu. Šis modulis sudarytas iš mažų komponentų, kurie įeina į didesnius komponentus. Šie savo ruoštu sudaro visą sistemą. Sistemos komponentų diagrama pateikiama 3.23 paveiksle.



3.23 pav. Įskiepio komponentų diagrama

3.3 lentelė. Sistemos komponentų struktūros diagramos aprašymas

Komponentas	Aprašymas
MyPlugin	Pagrindinis įskiepio paleidimo komponentas. Paleidžia įskiepio funkcionalumą MagicDraw UML sistemoje.
MainMenuConfigurator	Sukuria įskiepio dialogų paleidimo mygtukus ir įdeda į MagicDraw UML meniu juostą.
SimpleAction	Įskiepio paleidimo veiksmo komponentas. Paleidžia įskiepio dialogą.
TBLanguageUtilities	Su TB kalba susijęs komponentas. Atlieka specifinius metodus.
StateConstraintInsertPanel	Įvesti į būsenų mašinų diagramą ribojimus skirtas komponentas.
ClassConstraintInsertPanel	Įvesti į klasių diagramą ribojimus skirtas komponentas.
ClassConstraintEditPanel	Skirtas redaguoti klasių diagramoje panaudotus ribojimus komponentas.
AboutProjectWindow	Dialogas, skirtas aprašyti įskiepio projektą.
TBEUtilities	Einamų metodų, skirtų atlikti įskiepio funkcionalumą realizavimas.
ExpressionEditPanel	Redaguoti sukurtus ribojimus skirtas komponentas.
SettingsWindow	Įskiepio nustatymų tvarkymo langas.
ActivityConstraintEditPanel	Veiklos diagramoje sukurtų ribojimų redagavimo komponentas
TBEDialog	Pagrindinis įskiepio veikimo dialogas.
SeqConstraintEditPanel	Sekų diagramoje įvestų ribojimų redagavimo komponentas.
StateConstraintEditPanel	Būsenų mašinų diagramoje sukurtų ribojimų redagavimo komponentas.
DefaultElementFilter	Komponentas skirtas vaizduoti MagicDraw UML modelio elementus gražia, apdorota forma.
SeqConstraintInsertPanel	Ribojimų įvedimo į sekų diagramą komponentas.
ActivityConstraintInsertPanel	Ribojimų įvedimo į veiklos diagramą komponentas.
ExpressionInsertPanel	Pagrindinis ribojimų įvedimo į MagicDraw UML modelį komponentas
ProtocolConstraintEditPanel	Ribojimų redagavimas protokolų būsenų diagramose.
ProtocolConstraintInsertPanel	Ribojimų įvedimas į protokolų būsenų diagramas.



3.24 pav. Artefaktų įdiegimo diagrama

Asmeniniame kompiuteryje įdiegiamas MagicDraw UML paketas ir papildomai įdiegiamas veiklos taisyklių įvedimo įskiepis (10 lentelė). MagicDraw UML sistema suradusi plugin.xml failą paleidžia parametrus nurodytus jo viduje. Vidoje saugomas kelias iki įskiepio paleidimo klasės. Įskiepis saugomas faile TBEExpressionPlugin.jar. Jame yra 21 klasė. MagicDraw UML duomenis diske saugoja XMI duomenų faile. Duomenys į įskiepį paimami per TBEUtilities klasę. Visą įskiepio veiksmų seką reguliuoja TBEDialog, ExpressionInsertPanel, ExpressionEditPanel. SimpleAction, MainMenuConfiguator ir MyPlugin klasės atsakingos už įskiepio paleidimą.

Papildomai prie Ribojimų įvedimo įskiepio yra sukurtas ribojimų ataskaitų šablonas.

3.4 lentelė. Komponentų ir įdiegimo diagramos specifikacija

Elementas	Specifikacija
TBEExpressionPlugin.jar failas	Šis failas reikalingas paleisti įskiepį MagicDraw UML programos metu
XMI failas	MagicDraw UML duomenų failas, kuriame bus vaizduojami išsaugoti ribojimai, kartu su visais projekto elementais.
MagicDraw	MagicDraw UML sistema
plugin.xml	Deskriptorius skirtas aprašyti pagrindinius įskiepio paleidimo parametrus, bei aprašyti įskiepį.
ConstraintReportTemplateFile.rtf	Rtf formatu parašytas ribojimų ataskaitos šablonas. Pagal jį, ataskaitų variklis sugeneruoja ataskaitą apie visus įvestus ribojimus.

3.4.2. Ataskaitų generavimo šablonas

MagicDraw UML įrankis turi įdiegtą įskiepį, skirtą ataskaitų generavimui. Ataskaitų generavimo variklis naudoja šablonus, parašytus VTL kalba. Šablonas parašytas VTL kalba naudoja MagicDraw modelio duomenis, gaunamus per ataskaitų generavimo variklį. Tokiu būdu buvo sukurtas specialus šablonas, skirtas aprašyti visas įvestas į modelį veiklos taisykles. Ataskaitos šablonas parašytas RTF formatu, todėl jį geba atidaryti daugelis teksto tvarkymo įrankių.

Fragmentas iš sukurto šablono sintaksės:

Įvestos veiklos taisyklės į modelio elementus:

```
#set($ElementList=$array.createArray())
#foreach($klase in $Class)
#set($tmp=$ElementList.add($klase))
#end
#foreach($act in $CallBehaviorAction)
#set($tmp=$ElementList.add($act))
#end
#foreach($bus in $State)
#set($tmp=$ElementList.add($bus))
#end
#foreach($mess in $Message)
#set($tmp=$ElementList.add($mess))
#end
#foreach($tr in $Transition)
#set($tmp=$ElementList.add($tr))
#end
#foreach($fl in $ControlFlow)
#set($tmp=$ElementList.add($fl))
#end
#foreach($kintamasis in $sorter.sort($ElementList, "name"))
#set($lisC = false)
#set($lisC = $kintamasis.get_constraintOfConstrainedElement())
#set($guard = false)
#set($guard = $kintamasis.getGuard())
#if (($lisC && $lisC.size()>0) || $guard)
```

Gautos sugeneruotos ataskaitos pavyzdys:

Įvestos veiklos taisyklės

Element: Class Darbuotojas

Constraint 1

[Darbuotojas.Menedzeris.Padalinys] turi buti lygus [Darbuotojas.Padalinys]

Element: Class PlanoUzduotis

Constraint 1

[PlanoUzduotis.Planas.Produktas] turi buti lygus [ProduktoOperacija.Produktas]

Constraint 2

Jei (x = [PlanoUzduotis]) Tai {x.[Planas.Produktas] = x.[ProduktoOperacija.Produktas]}

Element: Class Uzsakymas

1.3. Constraint 2

3.4.3. Sistemos testavimo planas

Sistemos testavimui yra sudaromas testavimo planas, kuriame atsispindi, kokius veiksmus reikia atlikti testuojant sistemą, taip pat, kokia turi būti sistemos reakcija į konkrečius veiksmus. Sistemos testavimo planas pateiktas 11 lentelėje:

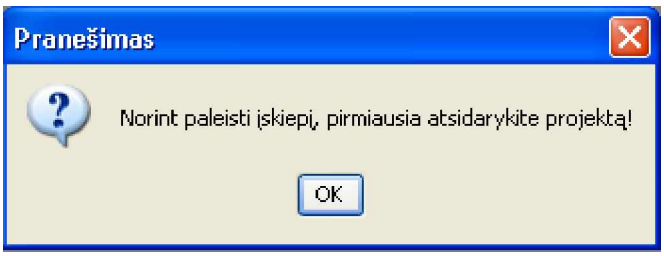
3.5 lentelė

NR.	TESTAVIMO VEIKSMŲ SEKA	LAUKIAMA SISTEMOS REAKCIJA
	1. ĮSKIEPIO PALEIDIMAS	
1.1.	Paleisti MagicDraw UML įkiepį, kai nėra atidarytas projektas.	Įskiepio paleidimo mygtukas yra neaktyvus
1.2.	Paleisti MagicDraw UML įskiepį, kai atidarytas projektas	Atidaromas įskiepio pagrindinis langas.
	2. RIBOJIMO ĮVEDIMAS	

2.1. 2.2	Pasirinkti veiklos taisyklių šabloną Paspaudžiamas dešinys pelės klavišas ant argumento	Sistema į veiklos taisyklių įvedimo eilutę įveda pasirinktą šabloną Sistema išveda pasirinktos apimties elementų sąrašą.
2.3 2.4 2.5 2.6 2.7 2.8	Pasirenkama elementas argumentų sąraše Pasirenkama asociacijos elementas, kuris nebuvo ribojimo elemento sudarymo cikle (įvedant ribojimus į klasių diagramas) Pasirenkama asocijuojanti klasė, kuri buvo ribojimo elemento cikle Pasirenkama klasė, turinti tėvinę klasę Pasirenkamas elementas, kuriam bus taikomas ribojimas Paspaudžiamas mygtukas „Įvesti“	Sistema atidaro klasių sąrašą. Pasirinkus vieną klasę iš sąrašo atidaromi klasės atributai bei operacijos. Taip pat tėvo klasės bei atributai. Išvedamas sąrašas klasių, kurios jungiasi su pasirinkta klase asociacijos ryšiais. Sistema išveda visus asociacijos klasės atributus bei operacijas. Taip pat išmetami visi tėvo atributai bei operacijos (jei klasė turi tėvinę klasę). Sistema neišveda asociacijos klasės atributų bei operacijų. Pasirinktos klasės atributų ir veiklos taisyklių sąraše matomi ne tik jos elementai, bet ir tėvinės klasės elementai. Sistema suteikia galimybę mygtuko „Įvesti“ paspaudimui. Sistema užsaugo ribojimą prie pasirinkto elemento.
3. RIBOJIMO REDAGAVIMAS		
3.1 3.2 3.3	Pasirenkamas elementas, kurio ribojimą norime redaguoti Iš veiklos taisyklių sąrašo pasirenkame vieno ribojimo pavadinimą Išsaugomas redaguotas ribojimas.	Programa išveda sąrašą veiklos taisyklių pavadinimų, kuriuos elementas turi. Sistema išveda pasirinkto ribojimo reiškinį. Pasikeičia ribojimo reiškinys MagicDraw UML diagramoje.
4. RIBOJIMO ŠALINIMAS		
4.1	Spaudžiamas mygtukas „Išvalyti“	Pašalinamas ribojimas iš elemento

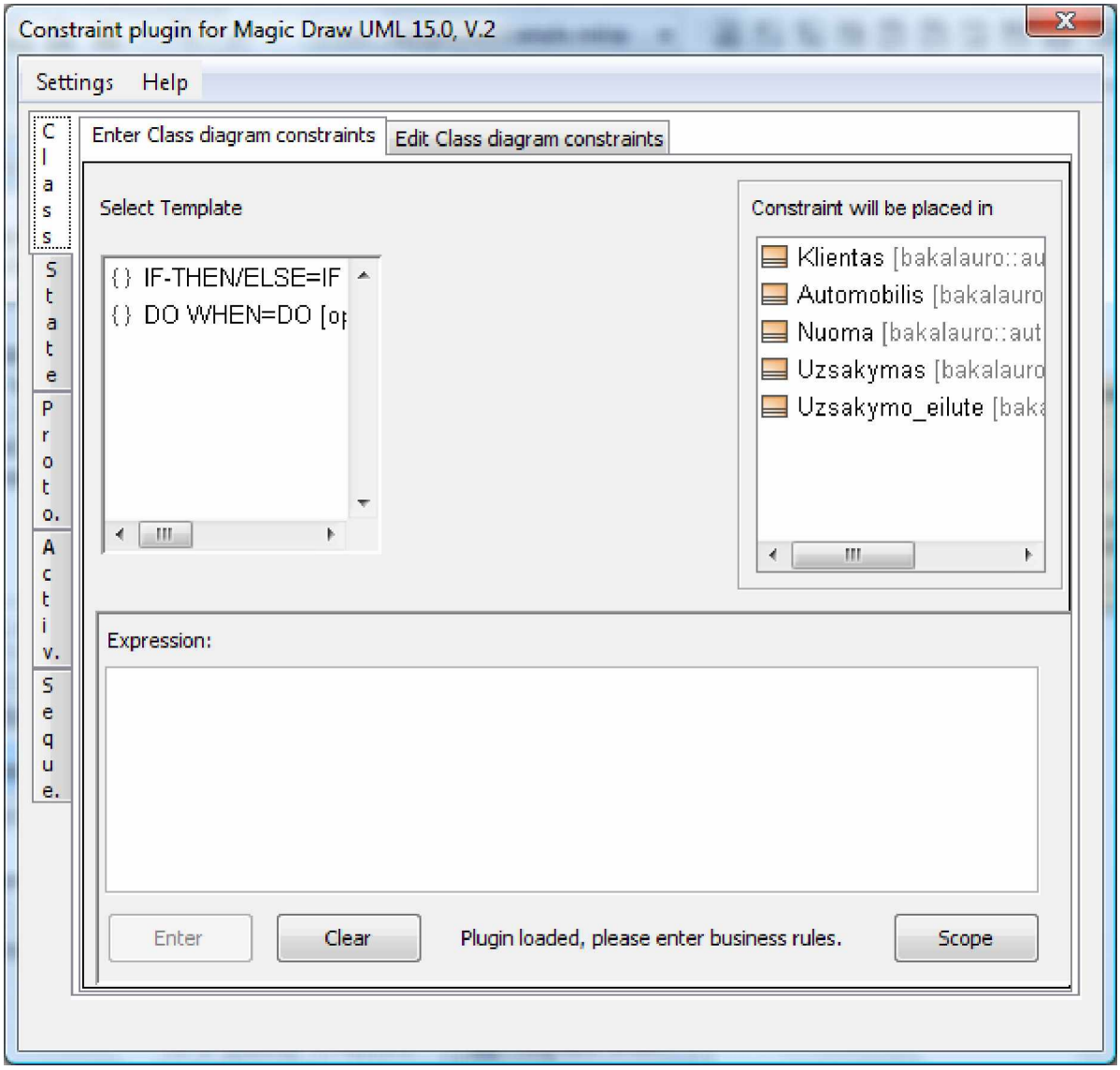
1. Įskiepio paleidimo testavimas

1.1. Bandome paleisti įskiepį, kai neatidarytas projektas. Matome tokį vaizdą (3.25 pav.):



3.25 pav. Sistemos pranešimas, kai neatidarius projekto paleidžiamas įskiepis

1.2 Jei projektas atidarytas, paspaudus įskiepio aktyvavimo mygtuką atidaromas pagrindinis įskiepio langas.

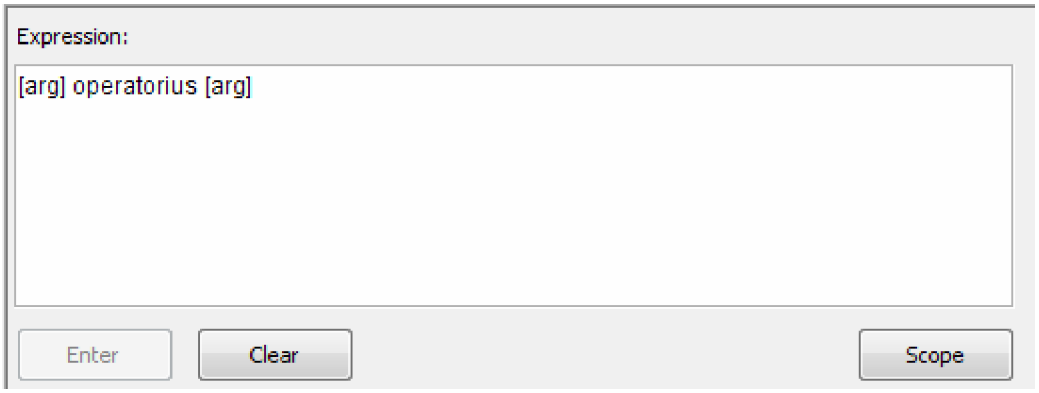


3.26 pav. Pagrindinis įskiepio langas

2. Veiklos taisyklių įvedimo testavimas

2.1 Pasirenkamas veiklos taisyklių šablonas. Sistema į veiklos taisyklių įvedimo eilutę

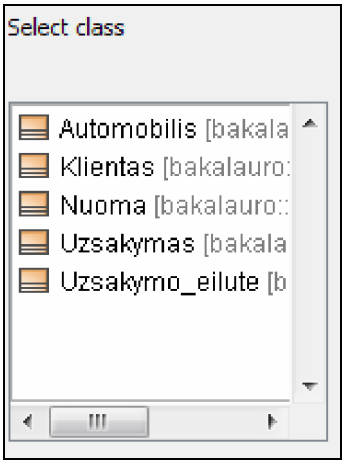
sėkmingai įveda pasirinktą šabloną (3.27 pav.):



3.27 pav. išvedamas šablonas

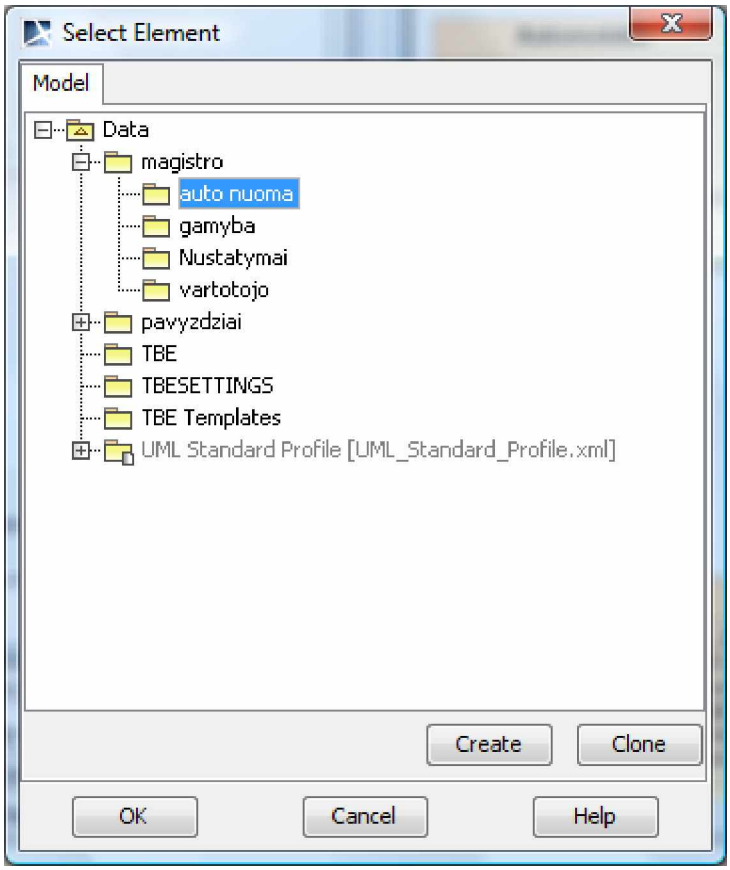
Šiuo atveju pasirinktas ribojimo šablonas - [arg] operatorius [arg]. Pasirinkus kitus ribojimo šablonus, sistema taip pat veikia teisingai.

2.2 Spaudžiamas dešinys pelės klavišas ant argumento (bet kur tarp laužtinių skliaustų). Programa sėkmingai išveda į sąrašą visas klases, kurios yra pasirinktame pakete (3.28 pav.):



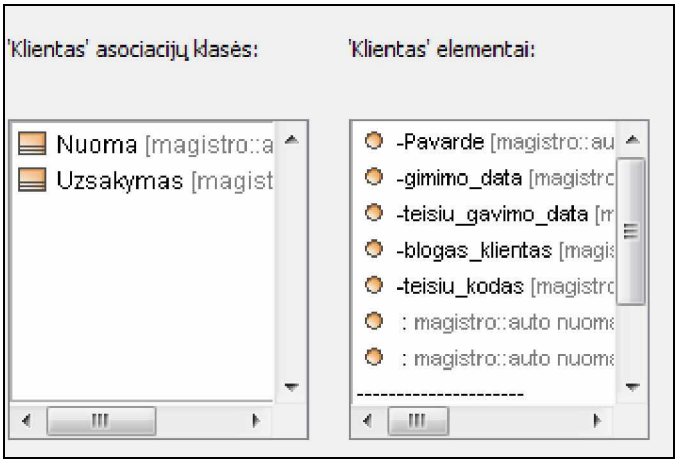
3.28 pav. išvedimo forma

Paketas pasirenkamas mygtuko „scope“ pagalba. Atsidariusiame lange pasirenkamas modelio paketas, kurio elementus naudosime (3.29 pav.):

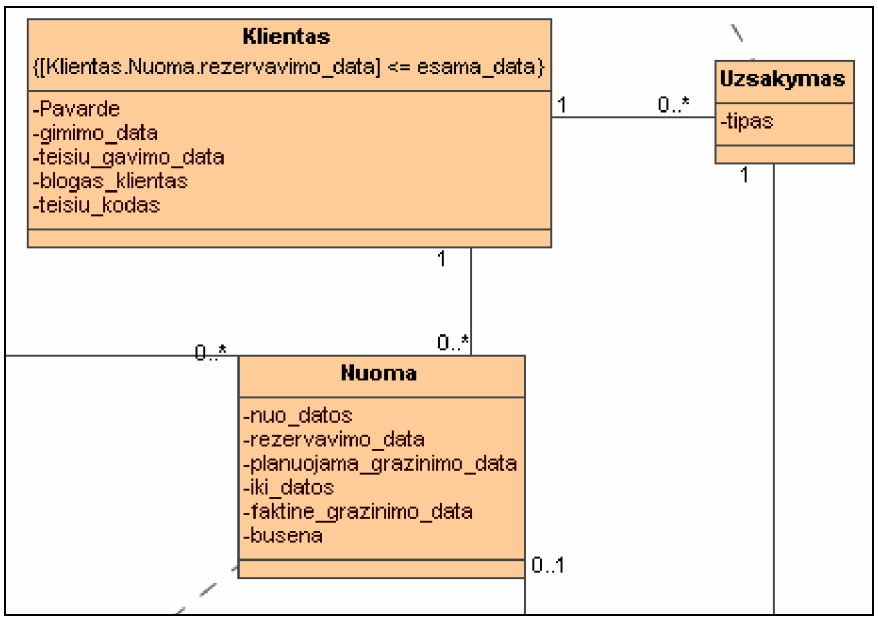


3.29 pav. Apimties pasirinkimas

2.3. Iš klasių sąrašo pasirenkame konkrečią klasę, kurios elementus norėtumėme įtraukti į ribojimą (pasirenkame klasę „Klientas“). Programa atidaro du langus. Kairiajame lange matome klases, kurios jungiasi su mūsų pasirinkta klase „Klientas“ per asociacijos ryšius. Dešiniajame - visus pasirinktos klasės „Klientas“ metodus bei operacijas (3.30 pav.).



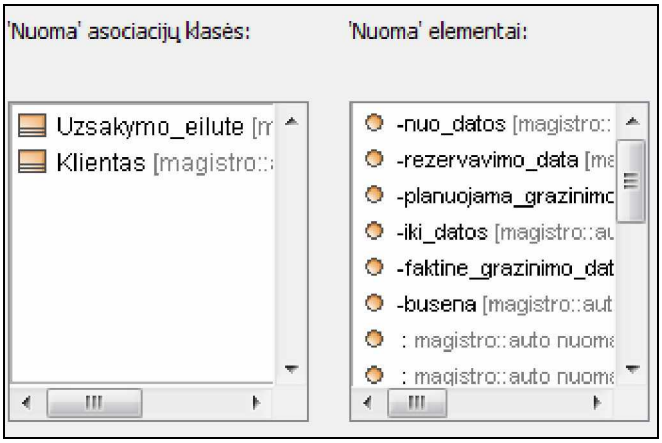
3.30 pav. Išvedamos duomenų formos



3.31 pav. įvedimo pavyzdys

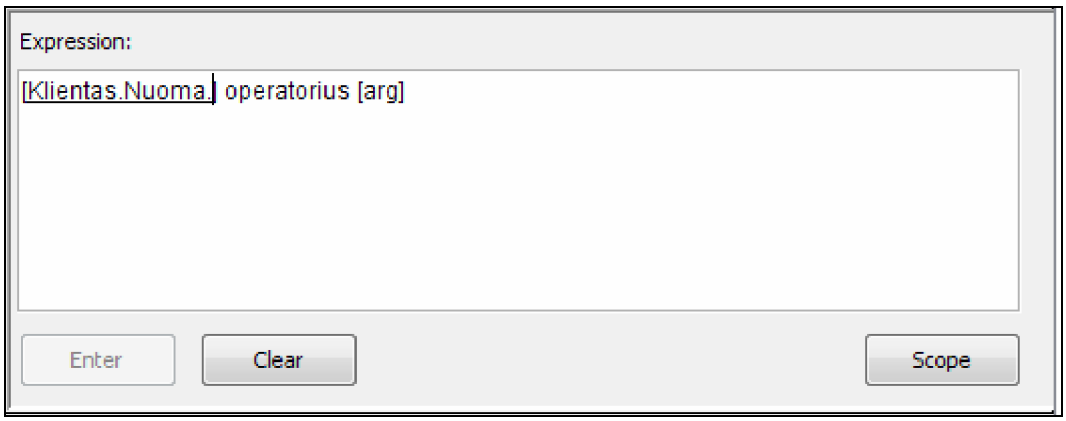
Pagal modelį (3.31 pav.) matome, kad įskiepis asociacijos klases išrenka teisingai.

2.4 Pasirenkama asociacijos klasė, kuri nebuvo ribojimo elemento sudarymo cikle. Pasirenkame klasę „Nuoma“. Kairiajame lange matome klases, kurios jungiasi su mūsų pasirinkta klase „Nuoma“ per asociacijos ryšius. Dešiniajame - visus pasirinktos klasės „Nuoma“ metodus bei operacijas (3.32 pav.).



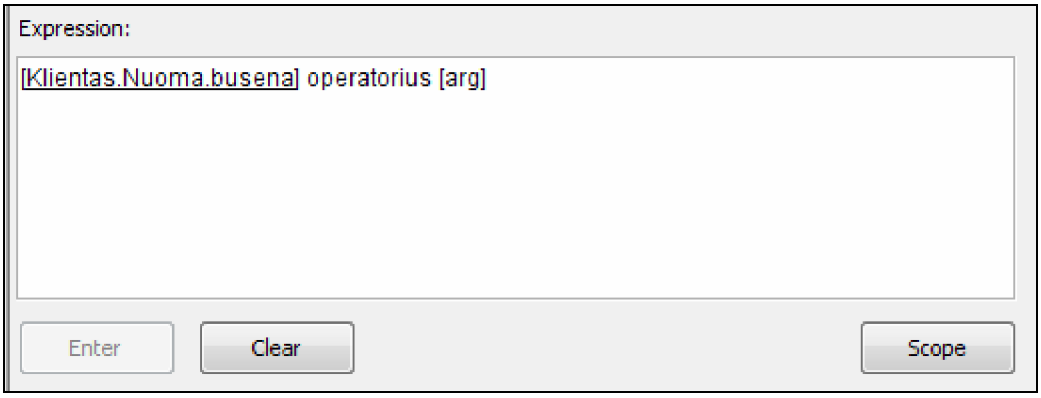
3.32 pav. su „Nuoma“ susieti elementai

Ribojimo reiškinio eilutė pasirinkus klasę atrodo taip (3.33 pav.):



3.33 pav. Ribojimo reiškinio eilutė pasirinkus klasę

Jei toliau asociacijos ryšiais neiname, o pasirenkame klasės atributą ar operaciją, veiklos taisyklių eilutė atrodo taip (3.34 pav.):

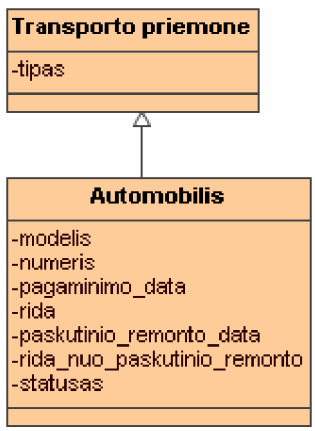


3.34 pav. Pasirinktas atributas

Taigi, matome, kad testuojant programa atlieka veiksmus teisingai.

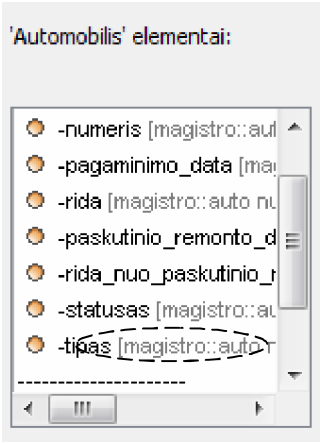
2.5 Pasirenkama asocijuojanti klasė, kuri buvo ribojimo elemento cikle. Testavimo atveju toks atvejis susidaro tada, jei mes nepasirenkame klasės „Nuoma“ atributo, o pasirenkame klasę „Klientas“. Ši klasė jau buvo pasirinkta, todėl negali įeiti į ribojimo elemento reiškinį, t.y. „Klientas.Nuoma.Klientas“ reiškinys negalimas.

2.6 Pasirenkama klasė, turinti tėvinę klasę (3.35 pav.).



3.35 pav. Klasė turinti tėvinę klasę

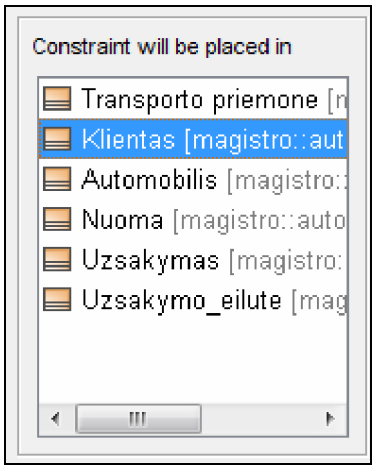
Testuojant pasirenkama klasė „Automobilis“. Pasirodo toks klasės „Automobilis“ atributų ir operacijų sąrašas (3.36 pav.):



3.36 pav. „Automobilis“ vidiniai elementai

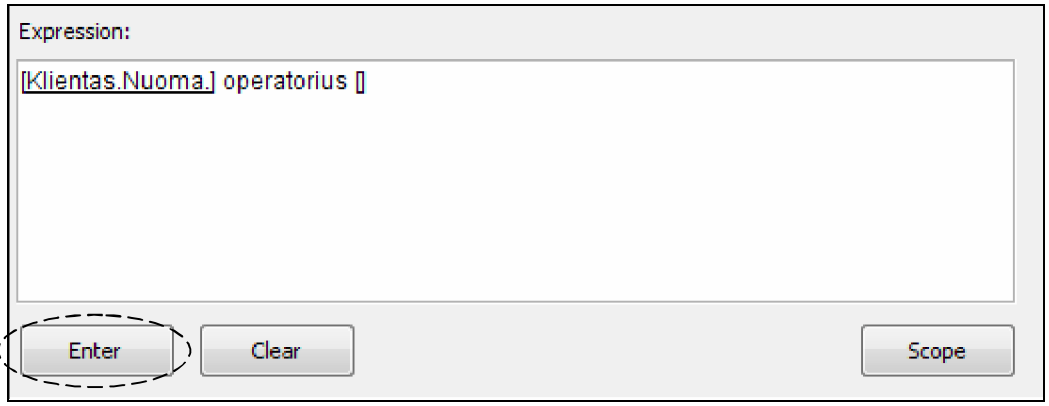
Pastebime, kad veiksmai su apibendrinimo ryšiais atliekami teisingai. Prie klasės „Automobilis“ atsiranda tėvinės klasės „Transporto priemonė“ atributas – „tipas“.

2.7 Pasirenkama klasė, kuriai bus taikomas ribojimas (3.37 pav.).



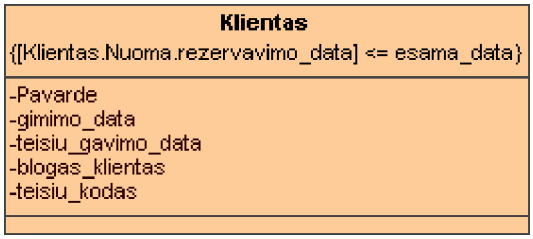
3.37 pav. Priskyrimo klasės pasirinkimas

Paspaudžiame ant klasės „Klientas“ ir kaip ir buvo numatyta sistemoje jau galima pasirinkti įvedimo mygtuką (3.38 pav.).



3.38 pav. Pasirinkus reiškinio saugojimo elementą, aktyvuojasi mygtukas

2.8 Paspaudus mygtuką „Įvesti“ MagicDraw UML diagramoje prie pasirinkto ribojimo saugojimo elemento pasirodo ribojimo reiškinys (3.39 pav.):

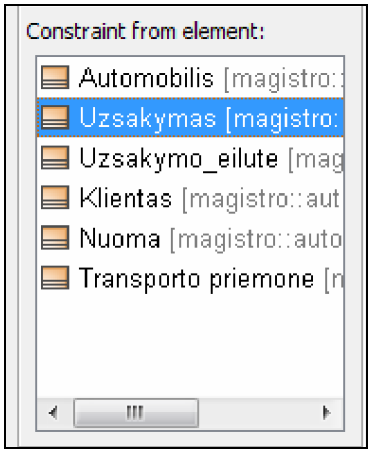


3.39 pav. Ribojimas įrašytas į klasę

Taigi, sistema atlieka ribojimo įvedimą teisingai.

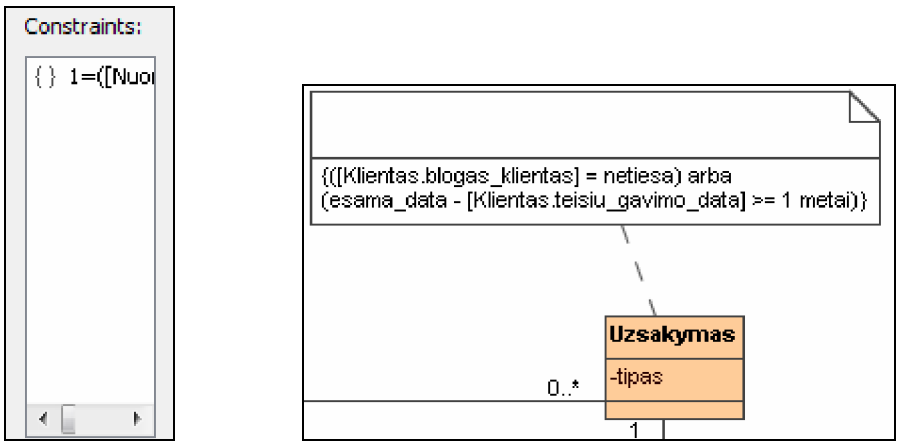
3. Ribojimo redagavimo testavimas

3.1 Pasirenkama klasė, kurios ribojimą norime pašalinti (3.40 pav.).



3.40 pav. Pasirinkimas klasės

Atidaromas langas, su visais klasės veiklos taisyklių pavadinimais (šiuo atveju klasė turi tik vieną ribojimą)(3.41 pav.):

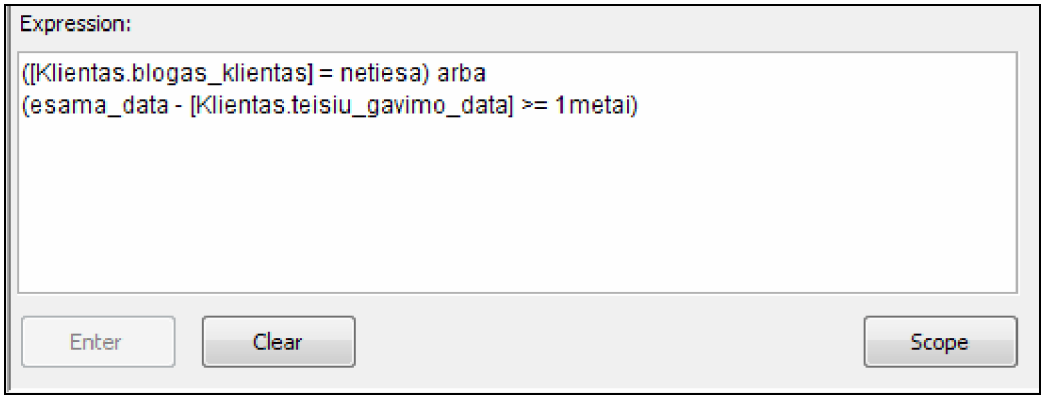


3.41 pav. Veiklos taisyklių vaizdavimo langas ir jo vaizdas modelyje

Taigi, sistema veiklos taisyklių išrinkimą iš klasės atlieka teisingai.

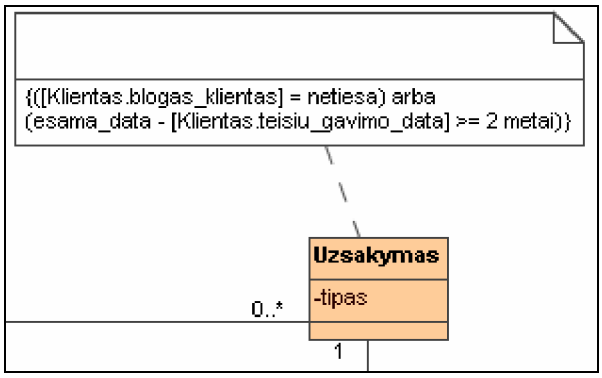
3.2 Iš veiklos taisyklių sąrašo pasirenkame vieno ribojimo pavadinimą, kurio specifikaciją norėtumėm keisti. Sistema į ribojimo lauką sėkmingai išveda pasirinkto ribojimo specifikaciją

į redagavimo lauką (3.42 pav.):



3.42 pav. Ribojimo tekstas redaktoriuje

3.3 Pakeičiame ribojimo specifikaciją ir spaudžiame mygtuką „Enter“. MagicDraw UML diagramoje matome, kad klasės ribojimas pasikeitė (šiuo atveju ribojimo specifikacijoje pakeitėme metus – iš 1 į 2) (3.43 pav.):

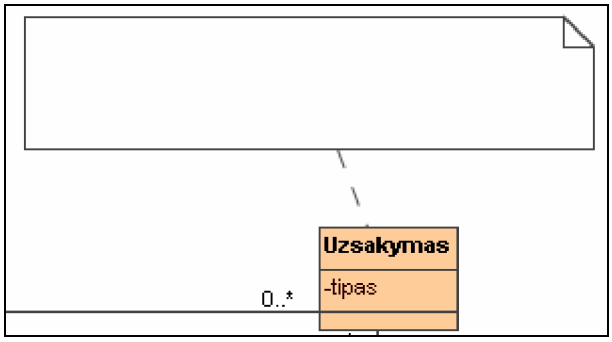


3.43 pav. Ribojimas po redagavimo

Kaip matosi iš testavimo, sistema sėkmingai atlieka visus ribojimo redagavimo veiksmus.

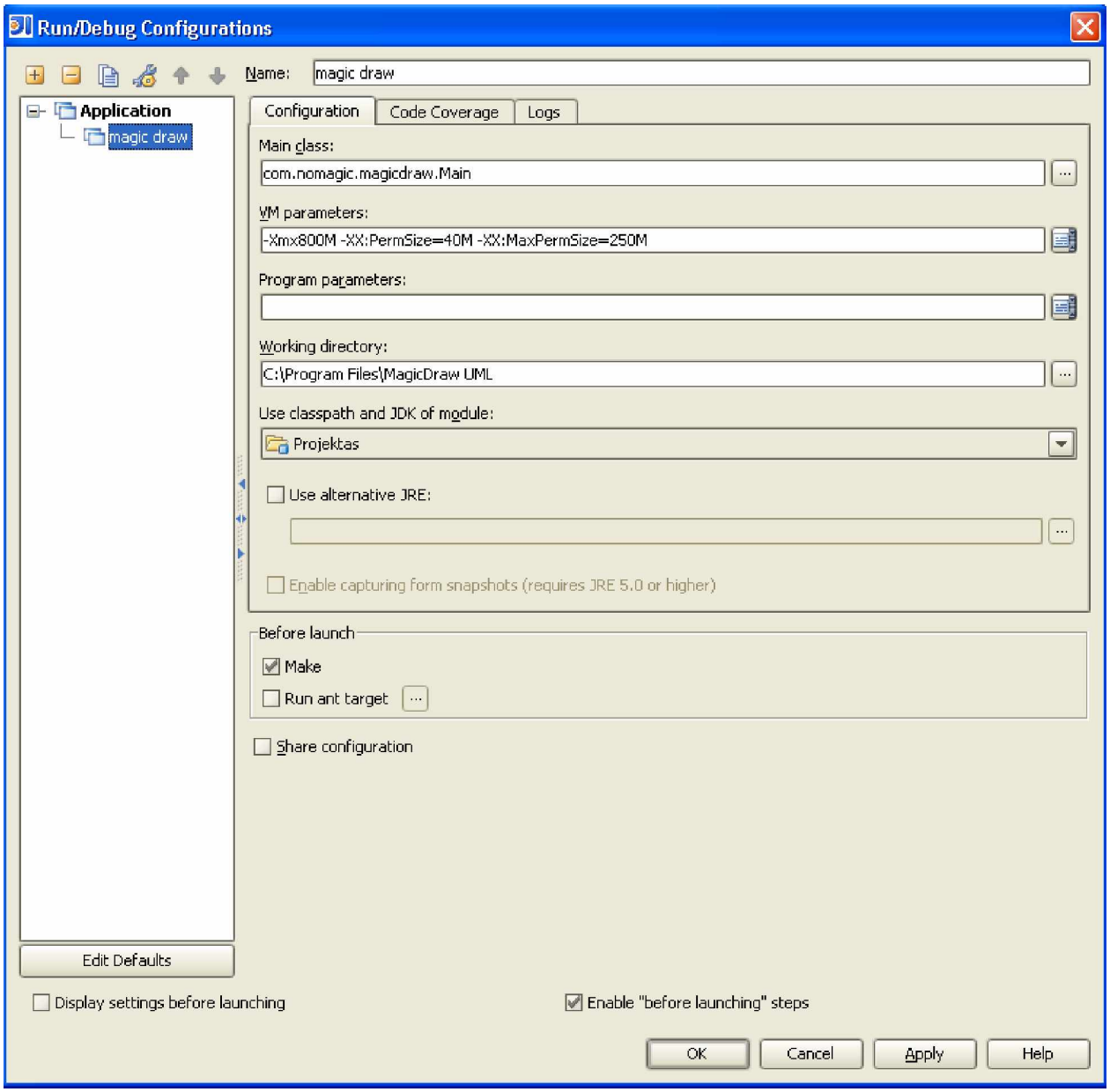
4. Ribojimo šalinimas

4.1 Pradiniai veiksmai buvo atlikti taip kaip ir redagavime (3.1 ir 3.2 punktai), tik vietoj redagavimo ir užsaugojimo spaudžiame mygtuką „Ištrinti ribojimą“. Programa sėkmingai atlieka ribojimo šalinimą. Ribojimo pašalinimas atsispindi MagicDraw UML diagramoje (3.44 pav.):



3.44 pav. Ribojimo ištrynimas

3.4.4. Įskiepis ir IntelliJ IDEA 6.0



3.45 pav. IntelliJ konfigūracija MagicDraw UML paketui

Kuriant programą iškyla problemos su įskiepio kodo ir MagicDraw UML derinimu (*debbuging*). Jei pakeičiamas programos kodas, turime sukompiliuoti jar failą ir įkelti į mūsų įskiepio katalogą. Tada turime perkrauti MagicDraw UML programą ir tik tada matomi kodo pakeitimai. Tai labai nepatogu ir atima daug laiko, kadangi MagicDraw UML reikalauja daug kompiuterio resursų.

Kurdami įskiepį, mes išsprendėme šią problemą. Programos kodą rašėme IntelliJ IDEA pakete. Susikonfigūravome MagicDraw UML paleidimą iš IntelliJ IDEA programos. Visa konfigūracija matoma konfigūracijos paveiksle. Tokios konfigūracijos pagalba, per IntellIJ paleidžiame MagicDraw UML. Jei pakeičiame programos kodą, perkompiliuojame įskiepio

kodą ir kodo pakeitimai iškart atsispindi įskiepyje (3.45 pav.).

3.4.5. Sistemos įdiegimo aprašymas

Įskiepis, kurį sudaro 3 sisteminiai failai, įdiegiamas labai paprastai. Reikia nepamiršti kad diegiant įskiepį į kompiuterį, reikia turėti administratoriaus teises.

Įdiegimo etapai būtų tokie:

- MagicDraw UML įskiepių įdiegimo kataloge (pvz. D:\Program Files\MagicDraw UML\plugins), susikurti naują katalogą „TBEplugin“.
- į šį katalogą perkopijuoti 3 sisteminius failus:
  - ***plugin.xml;***
  - ***TBEPlugin.jar;***
  - ***pagalba.chm***.
- Įdiegimas baigtas. Norint, kad įskiepis pasileistų, reikia perkrauti MagicDraw UML paketą.

Įskiepio diegimas reikalauja 5MB disko atminties.

4. EKSPERIMENTINIS VEIKLOS TAISYKLIŲ ĮSKIEPIO TYRIMAS

Sistemos reikalavimų bei sistemos projekto specifikavimo procesą atlieka analitikas. Sukurtas TBE įskiepis iš dalies automatizuoja veiklos taisyklių formavimą bei įvedimą į modelį. Reiškinių formavimas panaudojant šablonus turėtų sumažinti sistemos projektavimo laiką. Sukurti šablonai yra artimi natūraliai kalbai, todėl tiek analitikas, tiek projektuotojas, taikydamas įskiepį, lengviau suprastų reiškinius, jiems nereikėtų specialių žinių perprasti OCL kalbą.

Šis tyrimas atliekamas siekiant nustatyti dalinai automatizuoto veiklos taisyklių formavimo ir įvedimo pranašumus prieš įprastą kūrimo būdą. Taip pat formavimo kalbų sudėtingumo palyginimas, lyginant su OCL kalba. Įvertinamas ir įskiepio projektas, jo galimybė plėstis.

Reikia ištirti:

- ar įskiepis leidžia specifikuoti taisykles įvairiose diagramose?
- ar įskiepis pritaikomas IS kūrimo procese?
- ar juo lengviau užrašyti taisykles negu jo neturint?

Modelių projektavimui naudojamas 15.5 versijos MagicDraw UML CASE įrankis. Veiklos taisyklėms įvesti į modelį naudojamas TBE įskiepis.

Atliktas įskiepio galimybių tyrimas parodė, kad TBE įskiepis leidžia įvesti ribojimus į UML klasių, sekų, veiklos, protokolų būsenų ir būsenų mašinų diagramas, taip pat įskiepis gali būti panaudotas IS kūrimo procese. Tai iliustruoja 4.1 ir 4.2 poskyrių pavyzdžiai.

4.1. Įskiepio taikymas veiklos taisyklių ir vientisumo ribojimų įvedimui

Kalbant apie informacinių sistemų koncepcinių modelių ribojimus, naudojami terminai „veiklos taisyklės“ ir „vientisumo ribojimai“. Veiklos taisyklės priklauso veiklai, jos ateina iš veiklos. Tuo tarpu vientisumo ribojimai yra susiję su koncepcinio modelio teisingumu, jo atitikimu modeliuojamai sričiai. Sukurtu ribojimų įvedimo įskiepiu galima įvesti:

- Veiklos taisykles;
- Vientisumo ribojimus.

Nors vientisumo ribojimai ir veiklos taisyklės atrodo skirtingi, jie yra tarpusavyje susiję. Vientisumo ribojimai priklauso nuo veiklos taisyklių, pavyzdžiui, „Darbuotojo ir menedžerio padalinys turi sutapti“ yra ir veiklos taisyklė, ir vientisumo ribojimas. Veiklos taisyklės gali

būti tokios, kad darbuotojui gali vadovauti menedžeris iš kito padalinio, ir tuomet vientisumo ribojimas tampa nereikalingas.

Pirmasis pavyzdys (4.1 pav.) vaizduoja automobilių nuomos sistemos klasių modelį ir naudojant įskiepį įvestus veiklos taisyklių ribojimus.

Įvesti ribojimai:

Šio tipo ribojimas realizuoja alternatyvias veiklos taisykles:

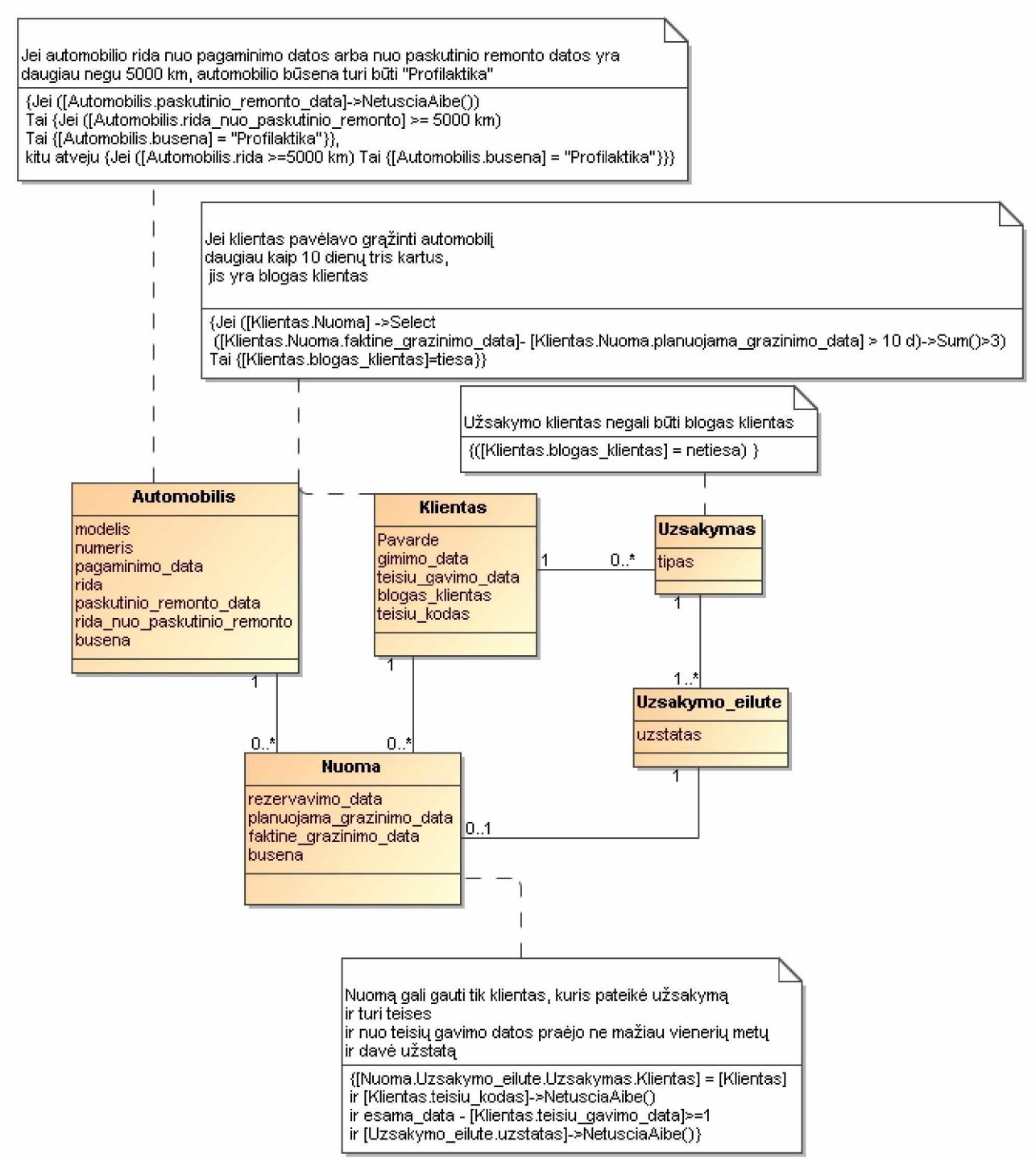
```
{Jei ([Automobilis.paskutinio_remonto_data] ->NetusciaAibe())
 Tai {jei ([Automobilis.rida_nuo_paskutinio_remonto] >= 5000 km)
        Tai {[Automobilis.busena] = "Profilaktika"}}.
Kitu atveju {Jei ([Automobilis.rida] >= 5000km)
        Tai {[Automobilis.busena] = „Profilaktika"}}}
```

Tai privaloma veiklos taisyklė skirta užsakymui:

```
{ [Klientas.blogas_klientas] = netiesa}
```

Ribojimas, realizuojantis privalomų ribojimų junginį klasei „Nuoma“:

```
{[Nuoma.uzsakymo_eilute.Uzsakymas.Klientas] = [Klientas]}
 IR [Klientas.teisiu_kodas]->NetusciaAibe()
 IR  esama_data –[Klientas.teisiu_gavimo_data] >=1 metai
 IR [Uzsakymo_eilute.uzstatas]->NetusciaAibe()}
```



4.1 pav. Veiklos taisyklių įvedimas, naudojant įskiepį

Vientisumo ribojimo pavyzdys (4.2 pav.):

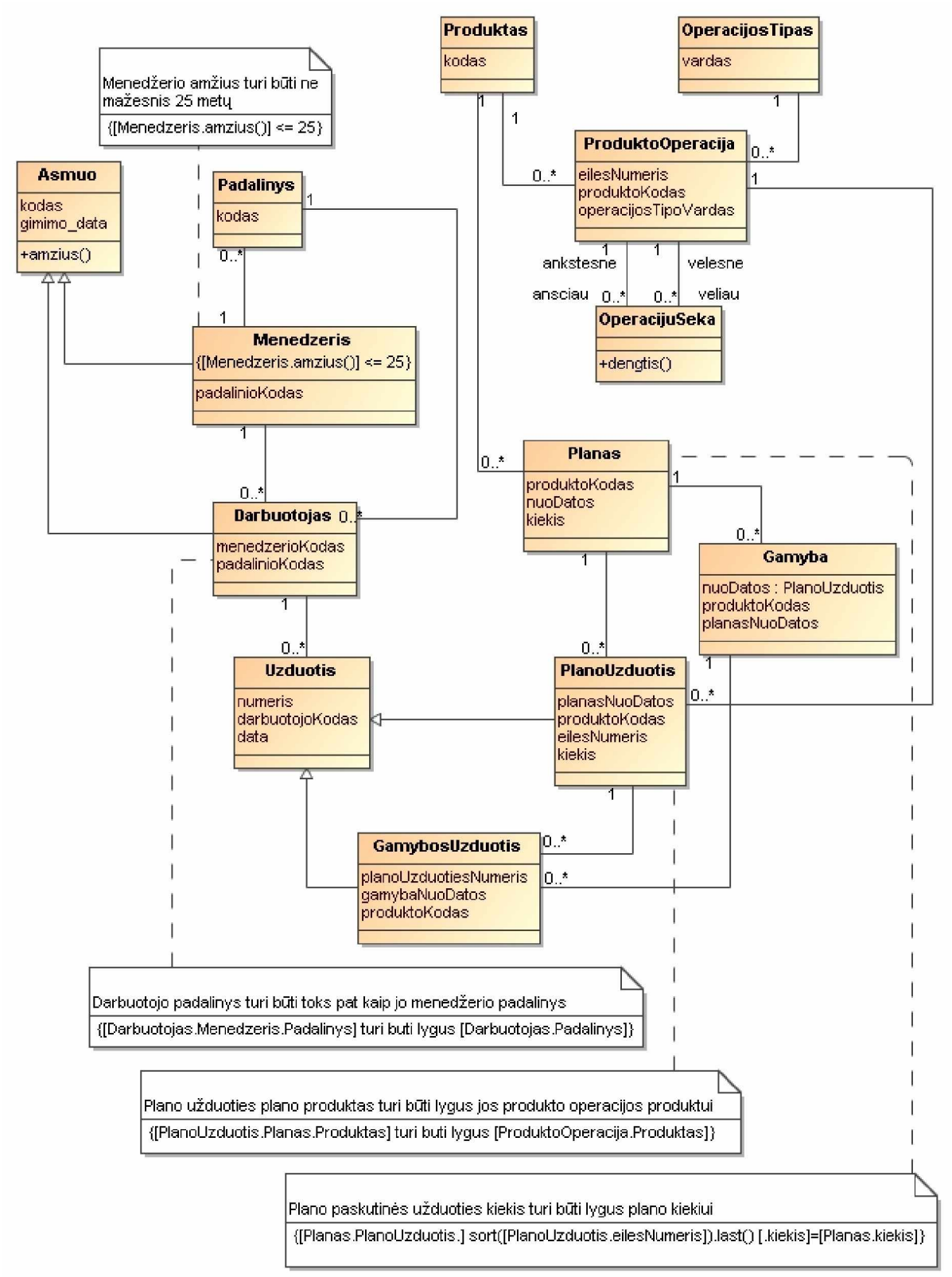
klasės „Menedzeris“ egzemplioriai turi amžiaus ribojimą

```
{[Menedzeris.amzius()] <= 25}
```

Ribojimas, taikomas apibrėžiant objektų ryšius:

```
{[Darbuotojas.Menedzeris.Padalinys] turi būti lygus[Darbuotojas.Padalinys]}
```

.

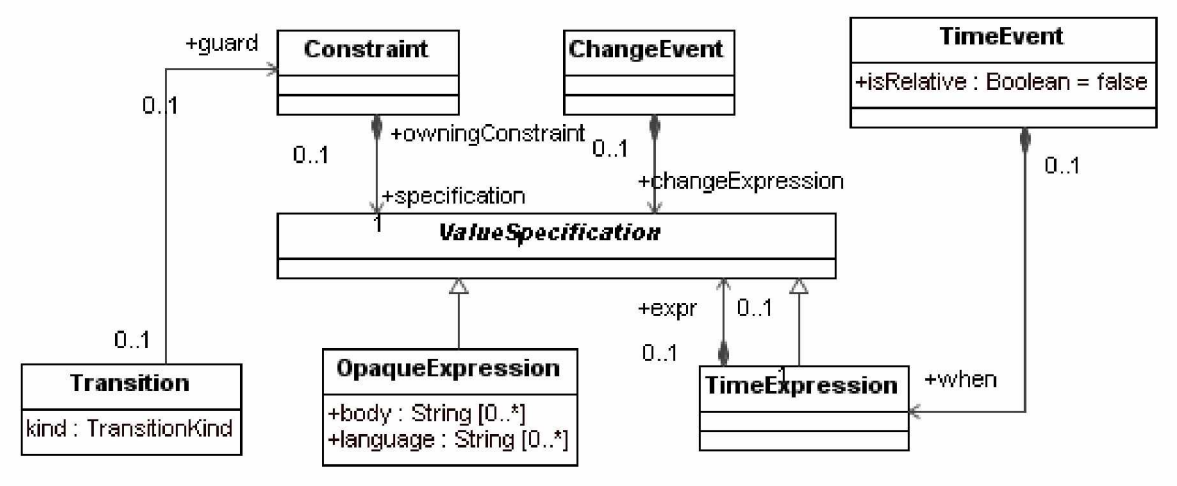


4.2 pav. Vientisumo ribojimų įvedimas, naudojant įskiepį

Būsenų diagrama:

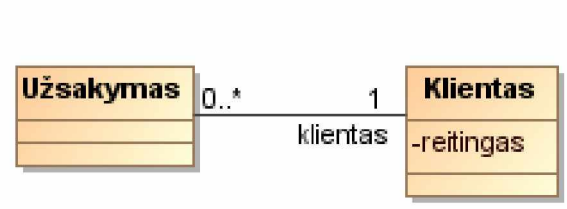
Būsenų diagramoje galima užrašyti būsenų invariantus, pasikeitimo įvykius, laiko įvykius ir perėjimų ribojimus. Jei būsena reiškia veiksmą, ji vadinama tą veiksmą kviečiančios

operacijos vardu. Activity, actions ir states, reiškiančios operacijas, taip pat vadinamos tų operacijų vardais.

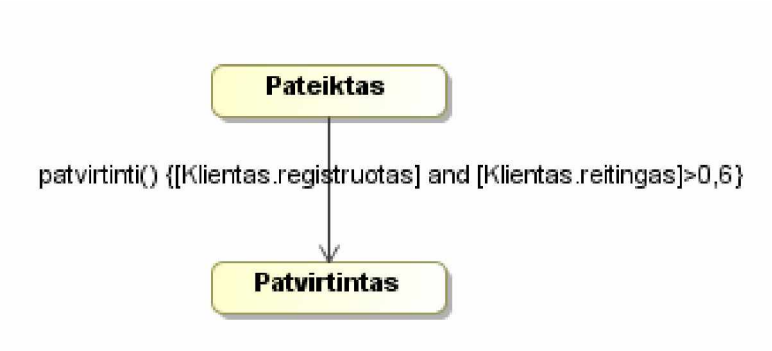


4.3 pav. Būsenų diagramos ribojimų ir įvykių modelis

Būsenų diagramos pavyzdys parodytas 4.5 paveiksle (pavyzdžio klasės parodytos 4.4 pav.):



4.4 pav. Užsakymo klasės



4.5 pav. Užsakymo būsenų diagrama

Būsenų diagramos leidžia pavaizduoti objektines taisykles - operacijų prieš ir po sąlyga, taip pat produkcines ir įvairias reakcijos taisykles – ECA, ECAC, CA

4.1 lentelė. Perėjimo pavaizdavimas ECAC taisykle:

Įvykis	Sąlyga	Veiksmas	Po sąlyga
CallOperationAction	Perėjimo pradžios būsenos reiškinys „and“	callOperation action (jei nėra efekto); arba effect (OpaqueAction); arba callBehaviorAction (tai gali būti	Perėjimo pabaigos būsenos

	sąlygos guard reiškinys	būsenų mašinos, activity ar interaction kvietimas, aprašytas kaip efektas); arba funkcinės elgsenos vykdymas; arba do activity, pavaizduotos pabaigos būsenoje, kvietimas. Bet kokiu atveju tai yra operacijos vardas	reiškinys
sendSignalAction	Tas pats	Effect action; Pabaigos būsenos do activity action	
approve()	self.oclInState(Patvirtintas) and self.Klientas.oclInState(Registruotas) and self Klientas.reitingas >0,6		

Produkcinė taisyklė bus CAC (gaunama iš ECAC, atmetus įvykį) arba CC taisyklė. Condition-postcondition taisyklės gaunamos iš if-then arba implies reiškinių (invariantų). Pvz.,

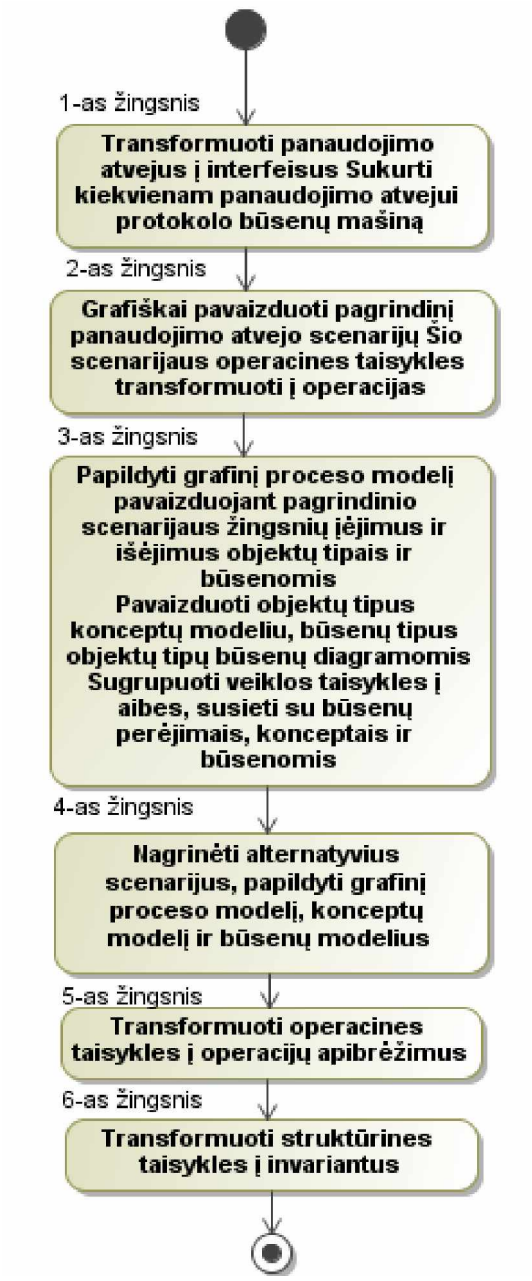
```
If Klientas.registracija=true then Klientas.registruotas=true
```

Atitinkamas OCL reiškinys:

```
Klientas inv inv1:
    self.registracija=true implies self.oclInState(registruotas)
```

4.2. Įskiepio taikymas veiklos taisyklėmis grindžiamo IS kūrimo procese

Panagrinėsime, kaip galima vykdyti taisyklėmis grindžiamą kūrimą sudarant tam tikrų tipų PIM modelius. Veiklos taisyklėmis grindžiamo kūrimo proceso modelis paslaugų sistemoms projektuoti pateikiamas 4.6 paveiksle. Procesams modeliuoti čia taikomos protokolų būsenų mašinos, kadangi jos tiesiogiai išreiškia operacijų vykdymo logiką, tačiau panašiai galima naudoti ir veiklos arba sekų diagramas ir paprastas būsenų mašinas. Panašūs procesai taikomi kuriant grynai objektines sistemas arba naudojant RUP stereotipus (valdiklius ir esybes).



4.6pav. Taisyklėmis grindžiamas kūrimo procesas sudarant paslaugų sistemos modelį

Pailiustruosime veiklos taisyklėmis grindžiamą kūrimo procesą paskolos gavimo pavyzdžiu. Asmuo, norintis gauti paskolą, turi kreiptis į banką ir prašyti paskolos. Paskola yra galima, jei asmuo pateikia užstatą ir jis pats yra užstato savininkas. Jei pats asmuo negali pateikti užstato, jis gali pateikti kito asmens (rėmėjo) nuosavybę kaip užstatą, tačiau tuomet užstatas turi turėti rėmėjo, kuris yra užstato savininkas, garantiją. Be to, garantijos pradžios data turi būti nedidesnė nei paskolos suteikimo data ir garantijos pabaigos data turi būti nemažesnė nei planuojama paskolos grąžinimo data. Kad būtų galima išduoti paskolą, bankas turi atlikti šiuos veiksmus:

– tikrinti paskolos galimumą;

– tikrinti paskolos patikimumą (ir asmens patikimumą, nes paskola yra patikima, jei asmuo patikimas);

– išduoti paskolą;

– pervesti paskolą.

Gavęs paskolą, asmuo turi ją grąžinti. Asmens patikimumą leidžia patikrinti bankų sistema, kuri tikrina visas asmens paskolas jai žinomuose bankuose. Asmuo yra patikimas, jei visos jo paskolos yra grąžintos laiku.

Paskolos davimo veiksmus bankas gali atlikti ir kita tvarka, pavyzdžiui:

– tikrinti paskolos patikimumą (ir asmens patikimumą);

– tikrinti paskolos galimumą;

– išduoti paskolą;

– pervesti paskolą,

arba

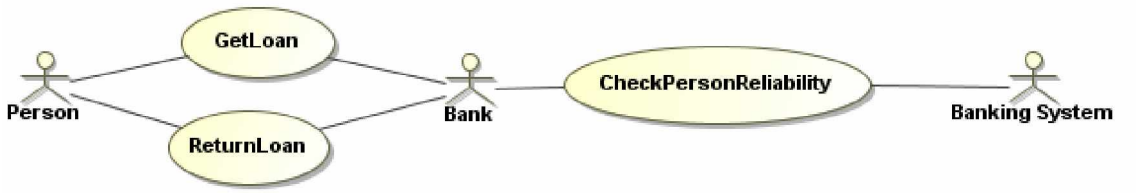
– lygiagrečiai tikrinti paskolos galimumą ir patikimumą (ir asmens patikimumą);

– išduoti paskolą;

– pervesti paskolą.

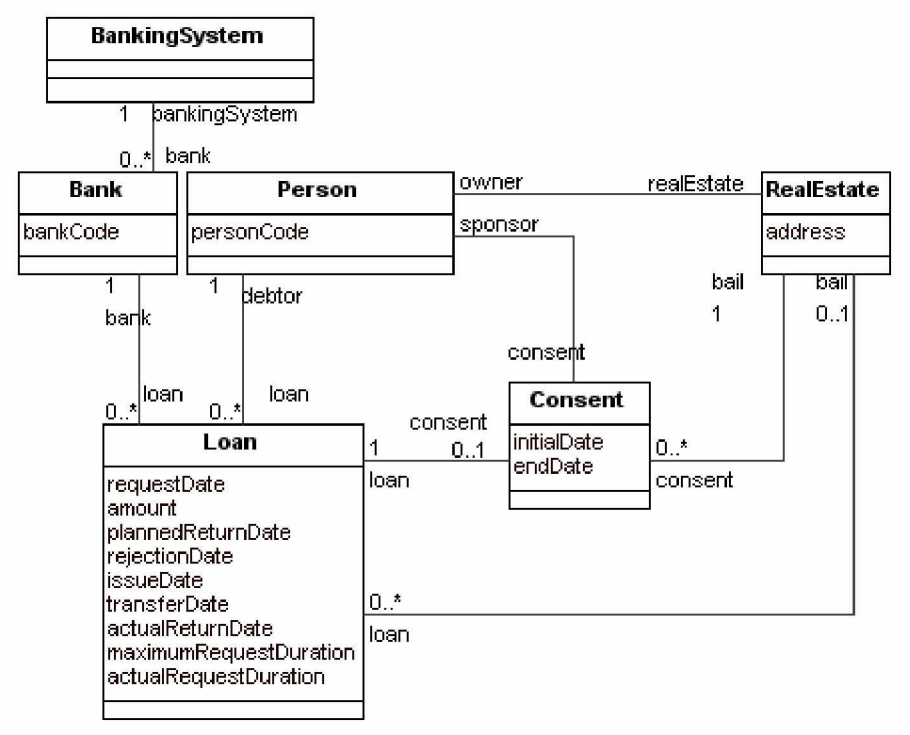
Tarkime, kompiuterizuojame paskolų gavimo procesą ir pasirenkame pirmą proceso variantą. Aprašykime panaudojimo atvejus, kartu apibrėžkime veiklos žodyno terminus ir veiklos taisyklių žodyno taisykles. Panaudojimo atvejai apibrėžia įvykių sekas, pagal kurias sudaromi panaudojimo atvejų vykdomų proceso modeliai.

1 žingsnis. Išskiriame panaudojimo atvejus. Panaudojimo atvejų modelis pateikiamas 4.7 paveiksle.

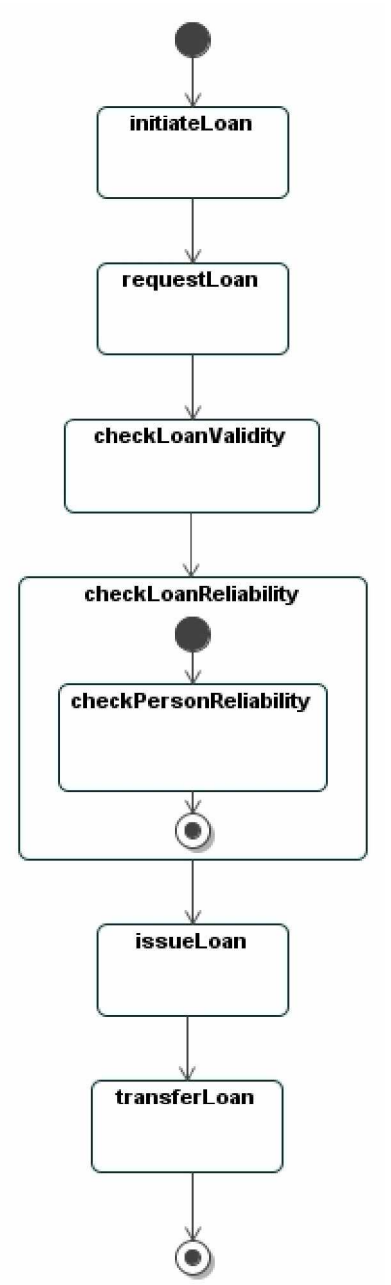


4.7 pav. Paskolos gavimo panaudojimo atvejai

2 žingsnis. Panaudojimo atvejai pavaizduojami abstrakčiais interfeisais, kuriems sukuriamos protokolo būsenų mašinos. Pavyzdžiui, pavaizduokime pagrindinį sėkmingą panaudojimo atvejo „Get Loan“ scenarijų (4.8 pav.), t. y. operacinių taisyklių faktų tipus pavaizduokime veiksmais (operacijų kvietimais) ir išdėstykime taip, kaip pasirinktame scenarijuje, nekreipdami dėmesio į alternatyvius scenarijus, kurie atsiranda dėl neišpildytų sąlygų ar užklausų trukmės ribojimų. Natūralios kalbos vardai transformuojami į UML modeliams būdingus vardus (naikinami tarpai, jungiamas žodis pradedamas didžiąja raide, operacijų vardai pradedami mažąja raide, konceptų ir esybių būsenų – didžiosiomis). Paskolų srities esybių klasių modelis pateikiamas 4.8 paveiksle.

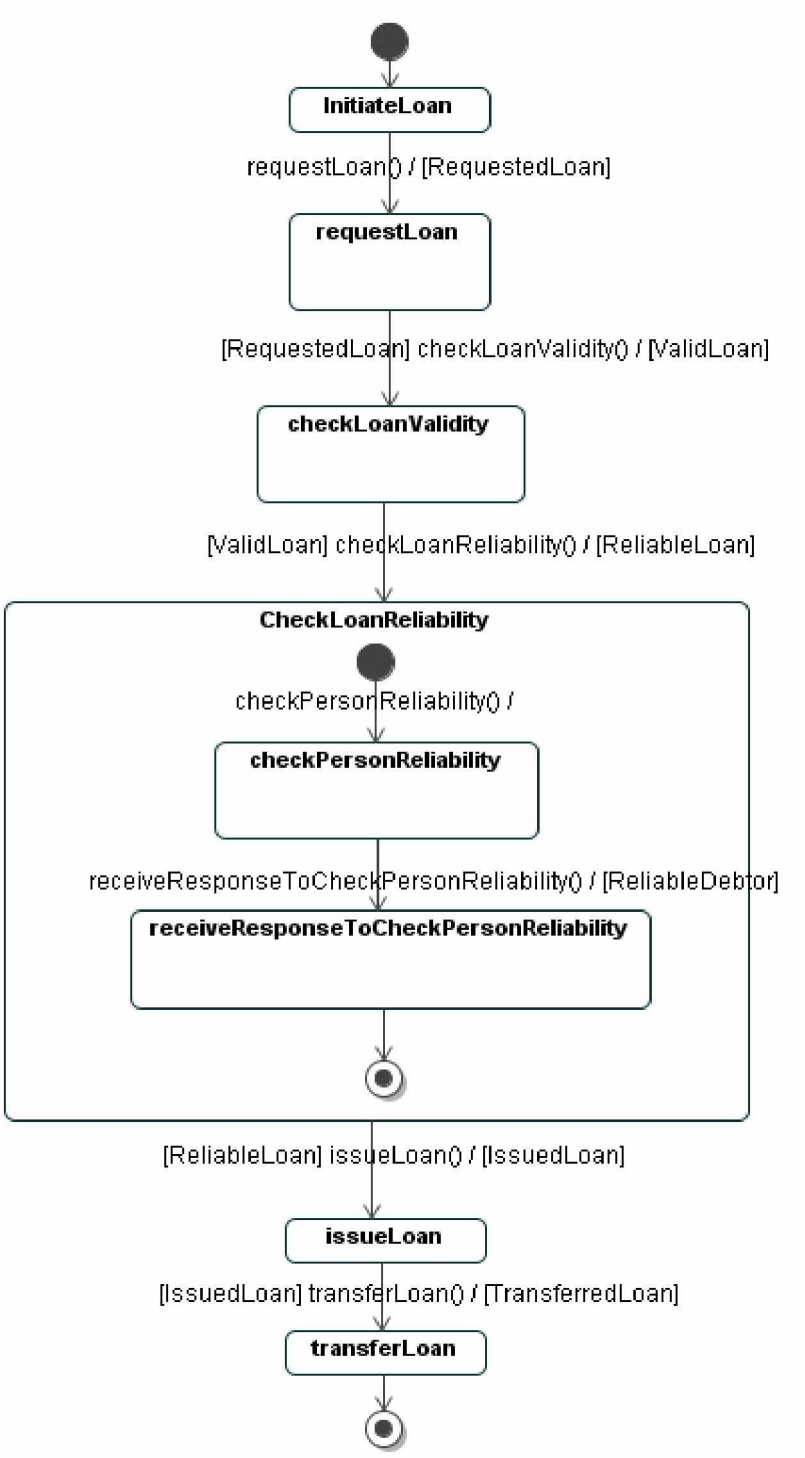


4.8 pav. Paskolų esybių klasių modelis

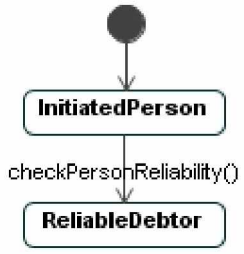


4.9 pav. Pagrindinis paskolų gavimo (panaudojimo atvejo „Get Loan“) sėkmės scenarijus

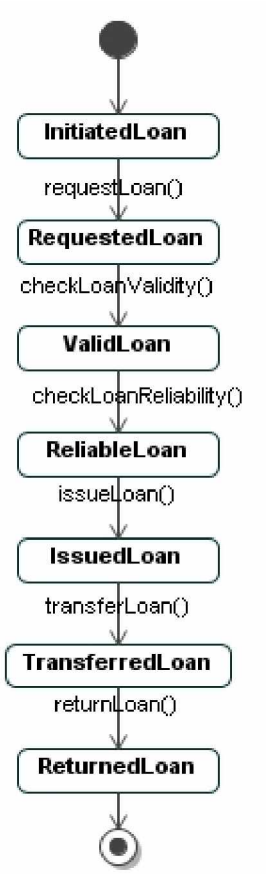
3 žingsnis. Nustatome kiekvieno žingsnio įėjimus, išėjimus, *prieš* ir *po* sąlygas, identifikuojame pagrindinius konceptus, kuriuos įtraukiame į veiklos konceptų žodyną. Papildome veiklos proceso modelį, sukuriame veiklos esybių būsenų diagramas (4.10 pav.), užrašome pagrindines veiklos taisykles. Pagrindinės paskolos davimo taisyklės anglų kalba pateikiamos 4.2 lentelėje, lietuvių kalba – 4.3 lentelėje. Operacinės taisyklės atspindi veiksmus ir veiksmų ribojimus, apibrėžimai – struktūrinius ribojimus. Šios taisyklės neapibrėžia proceso vykdymo tvarkos.



4.10 pav. Papildyta veiklos proceso „Get Loan“ diagrama



4.11 pav. Veiklos esybės „Person“ pradinė būsenų diagrama



4.12 pav. Veiklos esybės „Loan“ pradinė būsenų diagrama

4.2 lentelė. Pagrindinės paskolos gavimo veiklos taisyklės anglų kalba

<Operative business rule> It is permissible that debtor requests a loan.
<Operative business rule> It is obligatory that bank checks validity of each loan.
<Operative business rule> It is obligatory that bank checks reliability of each loan.
<Operative business rule> It is obligatory that during checking reliability of the loan bank requests to check debtor reliability from banking system.
<Operative business rule> It is obligatory that bank issues a loan if the loan is the valid loan and the reliable loan.
<Operative business rule> It is obligatory that bank transfers a loan if the loan is the issued loan.
<Definition> a loan is the valid loan if the debtor is the owner of the bail or the bail has a consent of the sponsor who is the owner of the bail and the initial date of the consent is not greater than the issue date of the loan and the end date of the consent is not less than the planned return date of the loan.

<Definition> A loan is reliable loan if the debtor is the reliable debtor.
<Definition> The debtor is reliable if each loan of the debtor is returned loan and return date of the loan is not greater than planned return date of the loan.

4.3 lentelė. Pagrindinės paskolos gavimo veiklos taisyklės lietuvių kalba

<Operative business rule> Leidžiama, kad skolininkas prašytų paskolos.
<Operative business rule> Įpareigojama, kad bankas tikrintų kiekvienos paskolos galimumą
<Operative business rule> Įpareigojama, kad bankas tikrintų kiekvienos paskolos patikimumą
<Operative business rule> Įpareigojama, kad paskolos patikimumo tikrinimo metu bankas kreiptųsi į bankų sistemą patikrinti skolininko patikimumą
<Operative business rule> Įpareigojama, kad bankas suteiktų paskolą, jei paskola yra galima ir patikima
<Operative business rule> Įpareigojama, kad bankas pervestų paskolą, jei paskola yra suteikta.
<Definition> Paskola yra galima, jei skolininkas yra užstato savininkas arba užstatas turi garantiją iš rėmėjo, kuris yra užstato savininkas ir garantijos pradžios data yra nedidesnė negu paskolos suteikimo data ir garantijos pabaigos data yra nemažesnė negu planuojama paskolos grąžinimo data.
<Definition> Paskola yra patikima, jei skolininkas yra patikimas
<Definition> Skolininkas yra patikimas, jei kiekviena skolininko skola yra grąžinta ir jos grąžinimo data yra nedidesnė negu numatyta grąžinimo data.

4 žingsnis. Analizuojame alternatyvius scenarijus, kurie atsiranda dėl to, kad:

– paskola gali būti negalima, t. y. neišpildytos su užstatu susijusios sąlygos;

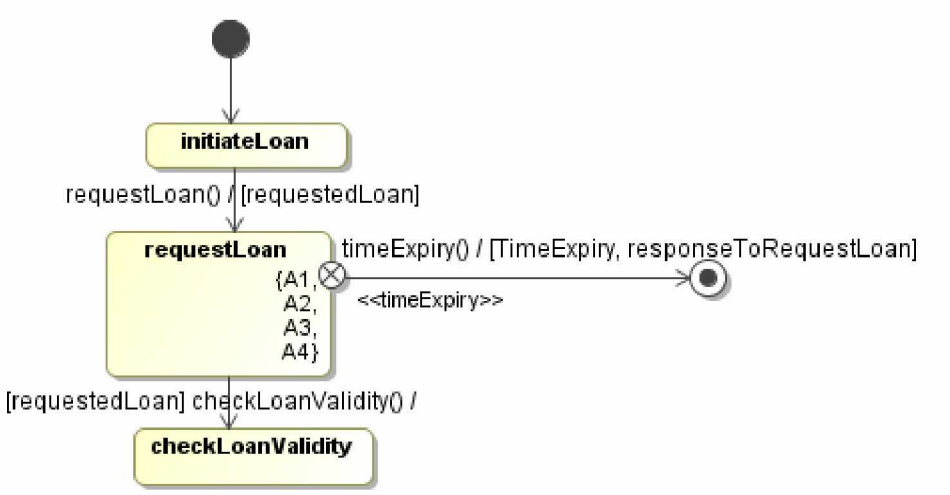
– asmuo gali būti nepatikimas;

– gali būti prisijungimo per tinklą problemų, todėl užklausų jungimosi trukmė yra ribota. Peržengus šią ribą, užklausa nutraukiama. Nagrinėjamame pavyzdyje dėl paprastumo toks ribojimas taikomas tik paskolos prašymo užklausai, bet realiai analogiški ribojimai turėtų būti taikomi ir bankų sistemos užklausai asmens patikimumui nustatyti. Taip pat turėtų būti nustatyta, kiek kartų arba kiek laiko bankas pakartotinai kreipiasi į bankų sistemą, jei prie jos prisijungti nepavyksta.

Šiame žingsnyje papildomi procesų ir esybių būsenų modeliai, veiklos žodynas ir veiklos

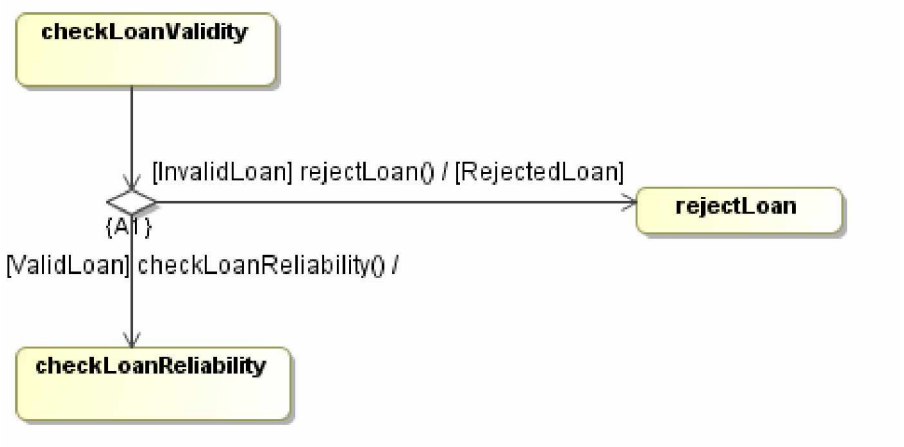
taisyklių žodynas. Veiklos taisyklės sugrupuojamos į aibes. Užbaigiama panaudojimo atvejų specifikacija, kurioje nurodomos su kiekvienu žingsniu susijusios taisyklių aibės.

Paimkime, pavyzdžiui, proceso žingsnį „requestLoan“ (4.8 pav). Tai paprastas žingsnis, kurio metu sukuriamas prašomos paskolos objektas. Šis objektas turi turėti nurodytą skolininką, prašymo datą, sumą ir planuojamą grąžinimo datą. Užklausos priėmimo laikas yra ribotas, todėl žingsnis turi alternatyvų žingsnį: skolininkui grąžinamas atsakymas „`time expiry`“. Sukurtoji diagrama yra protokolo būsenų mašina, ant jos perėjimo nurodomos operacijos *prieš* ir *po* sąlygos. Palaikantys faktų tipai operacijos apibrėžime virsta *po* sąlyga, kuri apibrėžia prašomos paskolos objekto sukūrimą. Vienos operacijos *po* sąlyga virsta kitos operacijos *prieš* sąlyga. Kitas tipinis žingsnis su alternatyviu žingsniu (neišpildyta sąlyga) pateikiamas 4.13 paveiksle. Taip sudaroma visa proceso būsenų mašina (4.14 pav.)



4.13 pav. Proceso žingsnis su tipiniu alternatyviu žingsniu – operacijai skirto laiko ribojimu

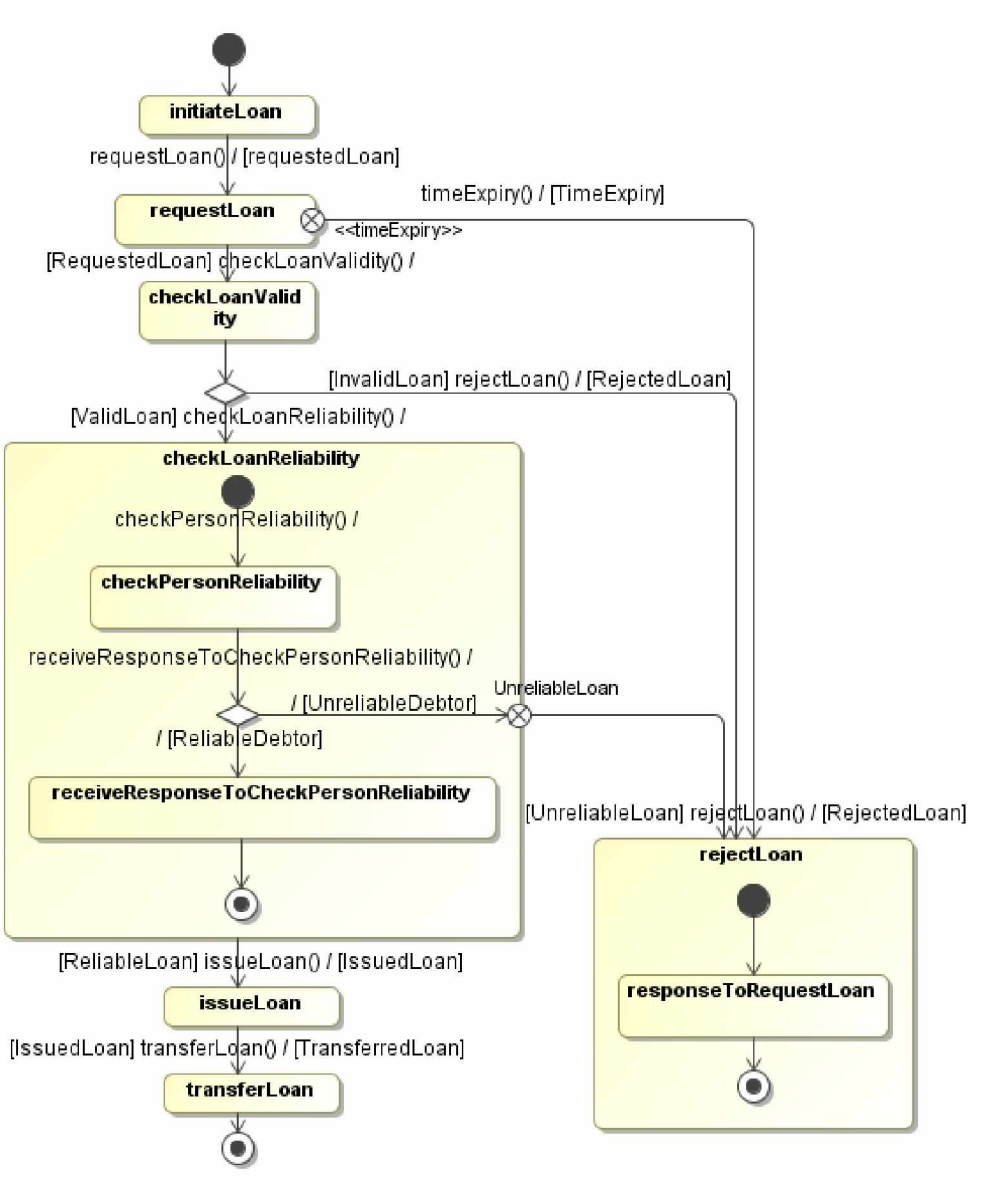
1.8 pav. būsenos „requestLoan“ apribojimai:
A1:{([Loan.debtor] is defined) IF ([Loan.state] = [LoanState.loanRequest])}
A2:{([Loan.requestDate] is defined) IF ([Loan.state] = [LoanState.loanRequest])}
A3:{ ([Loan.amount]>0) IF ([Loan.state] = ([LoanState.loanRequest])}
A4:{ ([Loan.plannedReturnDate] is defined) IF ([Loan.state] = [LoanState.loanRequest])}



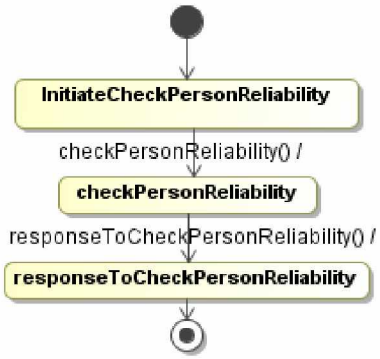
4.14 pav. Proceso žingsnis su tipiniu alternatyviu žingsniu – sąlygos neišpildymu

A1:{([Loan.state] = [LoanState.validRequest]) IF ([Loan.debtor] = [Loan.bail.owner]) or ([Loan.consent.sponsor] = [Loan.consent.bail.owner]) and ([Loan.consent.fromDate] <= [Loan.requestDate]) and ([Loan.consent.toDate] >= [Loan.plannedReturnDate])}

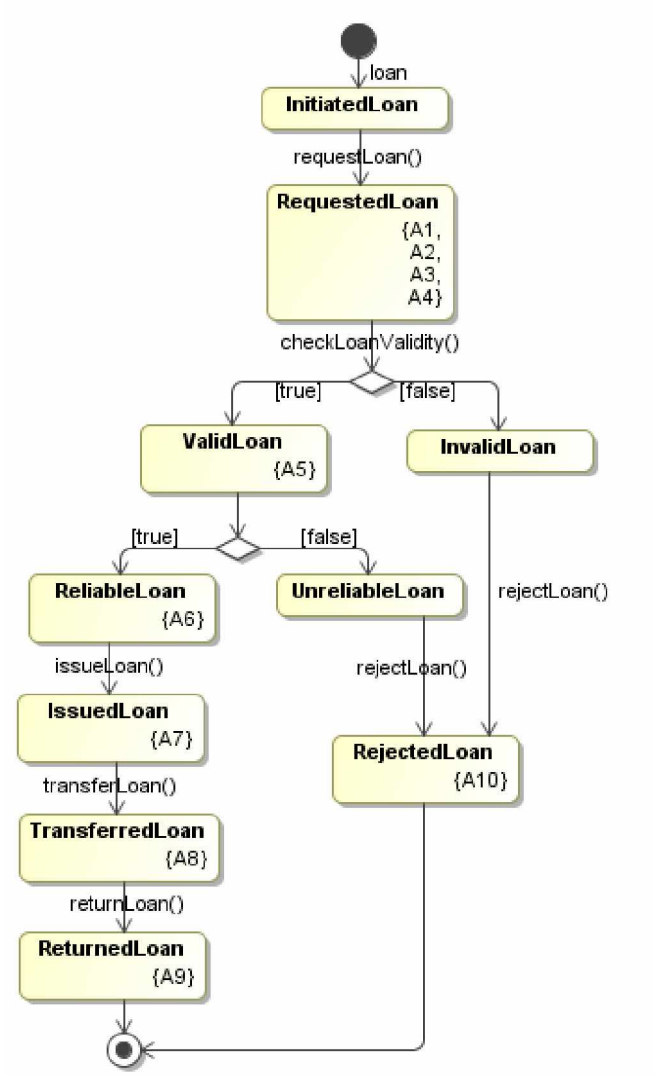
Panaudojimo atvejo specifikacija būsenų diagrama, atspindinti pagrindinį scenarijų, pateikiama 4.15 paveiksle. Taip pat sudaromos paskolos ir asmens (skolininko) būsenų diagramos.



4.15 pav. Panaudojimo atvejo „getLoan“ veiklos procesas



4.16 pav. Panaudojimo atvejo „CheckPersonReliability“specifikacija būsenų diagrama



4.17 pav. Papildyta paskolos „Loan“ būsenų diagrama (ribojimai pažymėti jų vardais)

Ribojimai TBE kalba:

A1:{([Loan.debtor] is defined) IF ([Loan.state] = [LoanState.loanRequest])}

A2:{([Loan.requestDate] is defined) IF ([Loan.state] = [LoanState.loanRequest])}

A3:{ ([Loan.amount]>0) IF ([Loan.state] = ([LoanState.loanRequest])}

A4:{ ([Loan.plannedReturnDate] is defined) IF ([Loan.state] = [LoanState.loanRequest])}

A5:{([Loan.state] = [LoanState.validRequest]) IF ([Loan.debtor] = [Loan.bail.owner]) or ([Loan.consent.sponsor] = [Loan.consent.bail.owner]) and ([Loan.consent.fromDate] <= [Loan.requestDate]) and ([Loan.consent.toDate] >= [Loan.plannedReturnDate])}

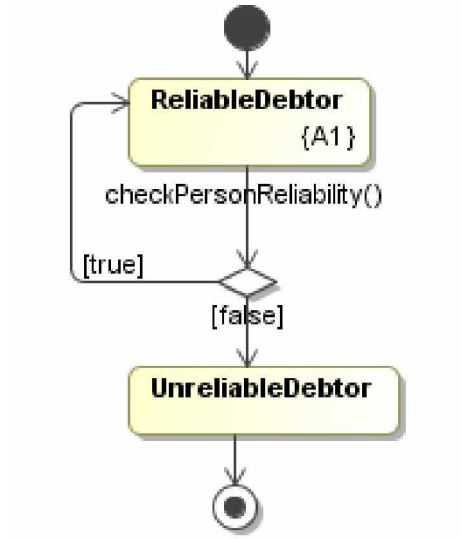
A6:{([Loan.state]=[LoanState.reliableLoan]) IF ([Loan.debtor.state] = [PersonState.reliablePerson])}

A7:{([Loan.issueDate] is defined) IF ([Loan.state] = [LoanState.issuedLoan])}

A8:{([Loan.transferDate] is defined) IF ([Loan.state] = [LoanState.transferedLoan])}

A9:{([Loan.actualReturnDate] is defined) IF ([Loan.state] = [LoanState.returnedLoan])}

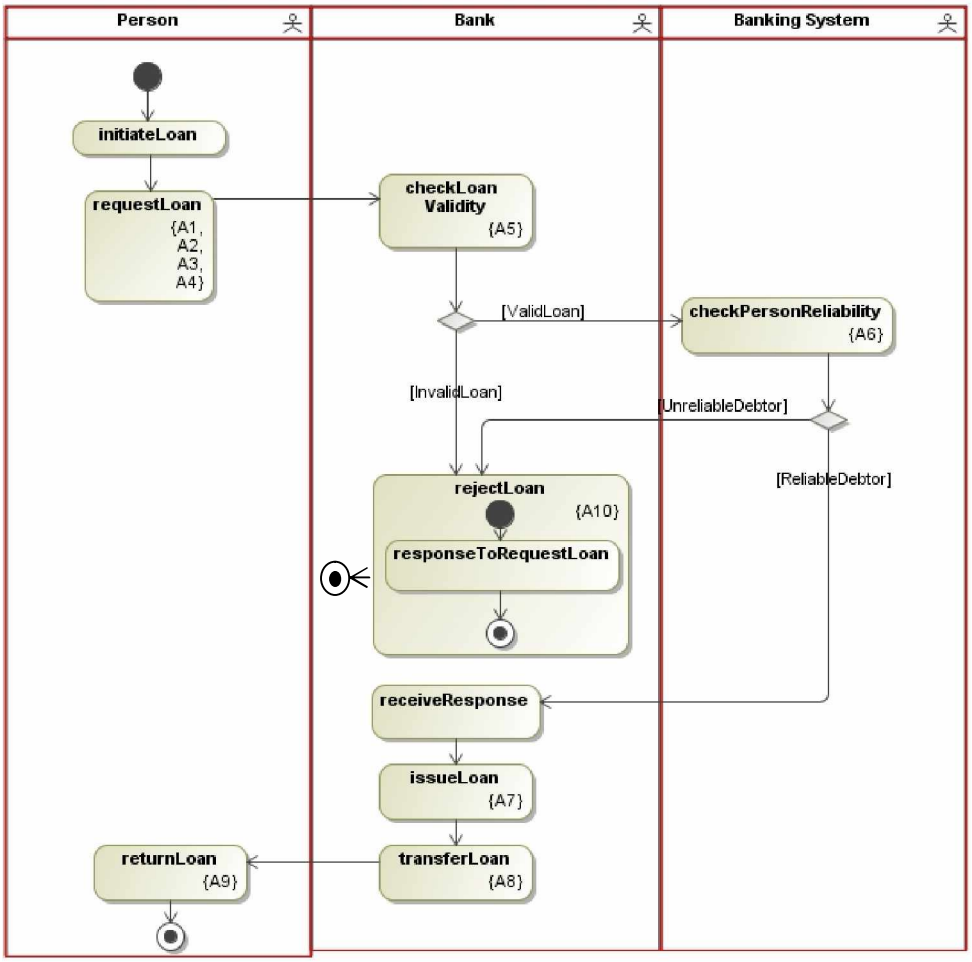
A10:{([Loan.rejectionDate] is defined) IF ([Loan.state] = [LoanState.rejectedRequest])}



4.18 pav. Papildyta asmens „Person“ būsenų diagrama

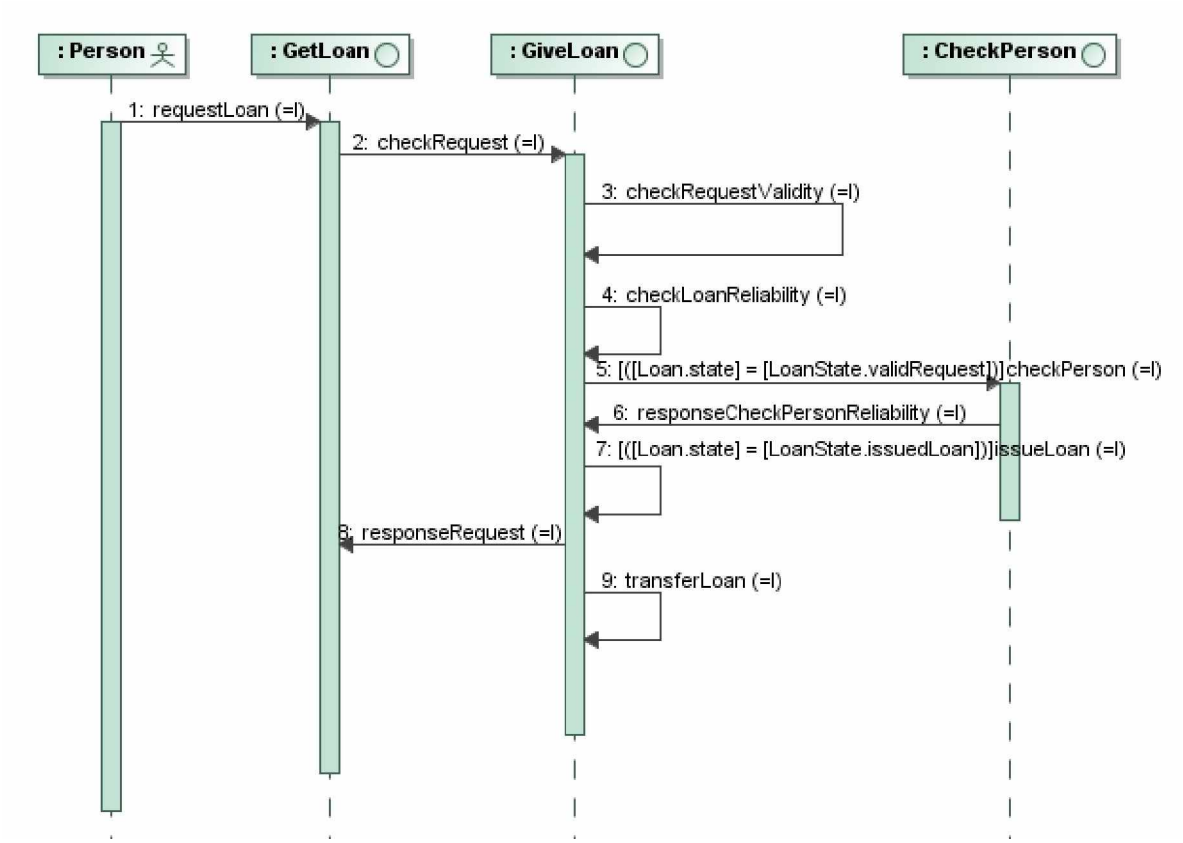
A1: {([Loan.debtor.state] = [PersonState.reliablePerson]) IF {(for all ([Person.loan.state]=[LoanState.returnedLoan])) and (Person.loan.actualReturnDate)<=[Person.plannedReturnDate]}}

Įskiepiu galima įvesti apribojimus ir į veiklos diagramas. Paskolos veiklos diagramos pavyzdys parodytas 4.19 pav. Veiklos diagramoje aiškiau matoma, kokie vaidmenys atlieka konkrečius veiksmus.



4.19 pav. Paskolos veiklos diagramos pavyzdys

4.20 pav. parodytas paskolos sekų diagramos pavyzdys. Joje aiškiai matoma procesų vykdymo seka.

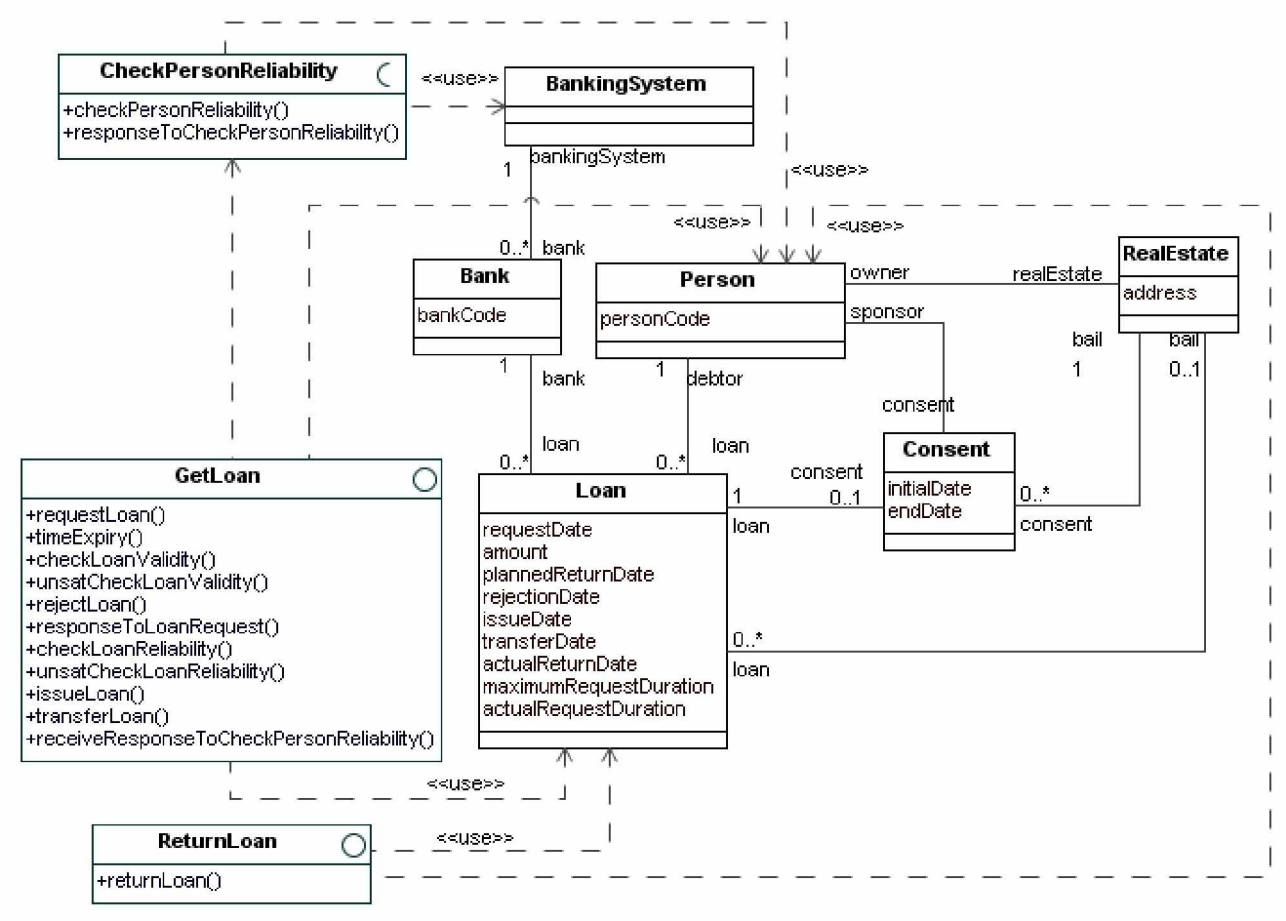


4.20 pav. Paskolos sekų diagramos pavyzdys

4.21

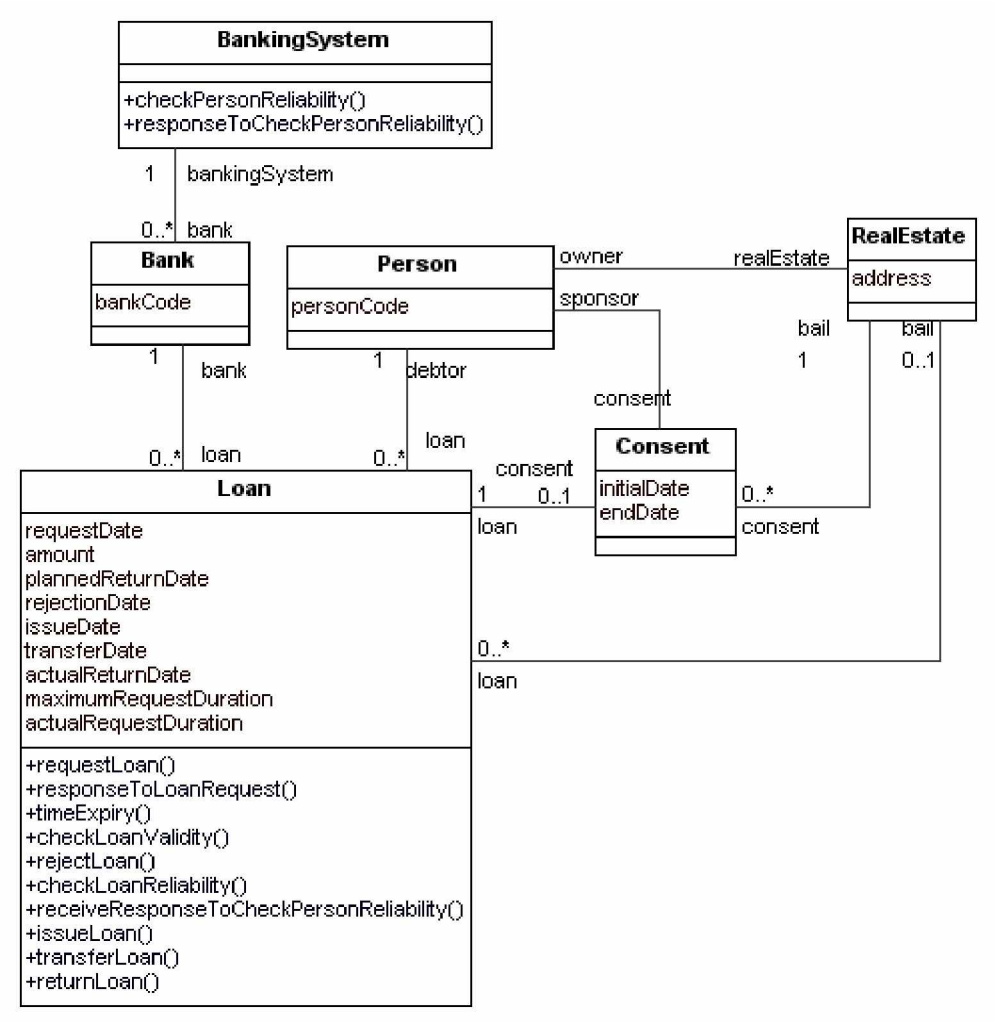
Iš veiklos žodyno gaunamas klasių modelis pateikiamas 4.16 paveiksle. Struktūrinės taisyklės realizuojamos tipų arba būsenų invariantais. Operacinės taisyklės realizuojamos operacijomis, kurios vykdo tam tikrus veiksmus. Programos kode invariantai taip pat realizuojami operacijomis, tačiau tipų ir būsenų taisyklės nėra sumaišytos su procesų taisyklėmis. Gautieji objektiniai modeliai leidžia nepriklausomai keisti vykdomus veiklos procesus (procesų arba įvykių taisykles) ir struktūrines taisykles, apibrėžiančias dalykinės srities tipus arba jų būsenas.

Iš veiklos žodyno ir veiklos taisyklių gauta paskolų paslaugų klasių diagrama (PIM) pavaizduota 4.21 paveiksle. Ši diagrama skirta paskolų paslaugų sistemai sukurti. Iš paskolų paslaugų interfeisų `GetLoan`, `ReturnLoan` ir bankų sistemos interfeiso `CheckPersonReliability` galima sugeneruoti paskolų paslaugų WSDL (angl. *Web Service Definition Language*) aprašus, o iš operacijų specifikacijų generuoti paslaugų metodus.

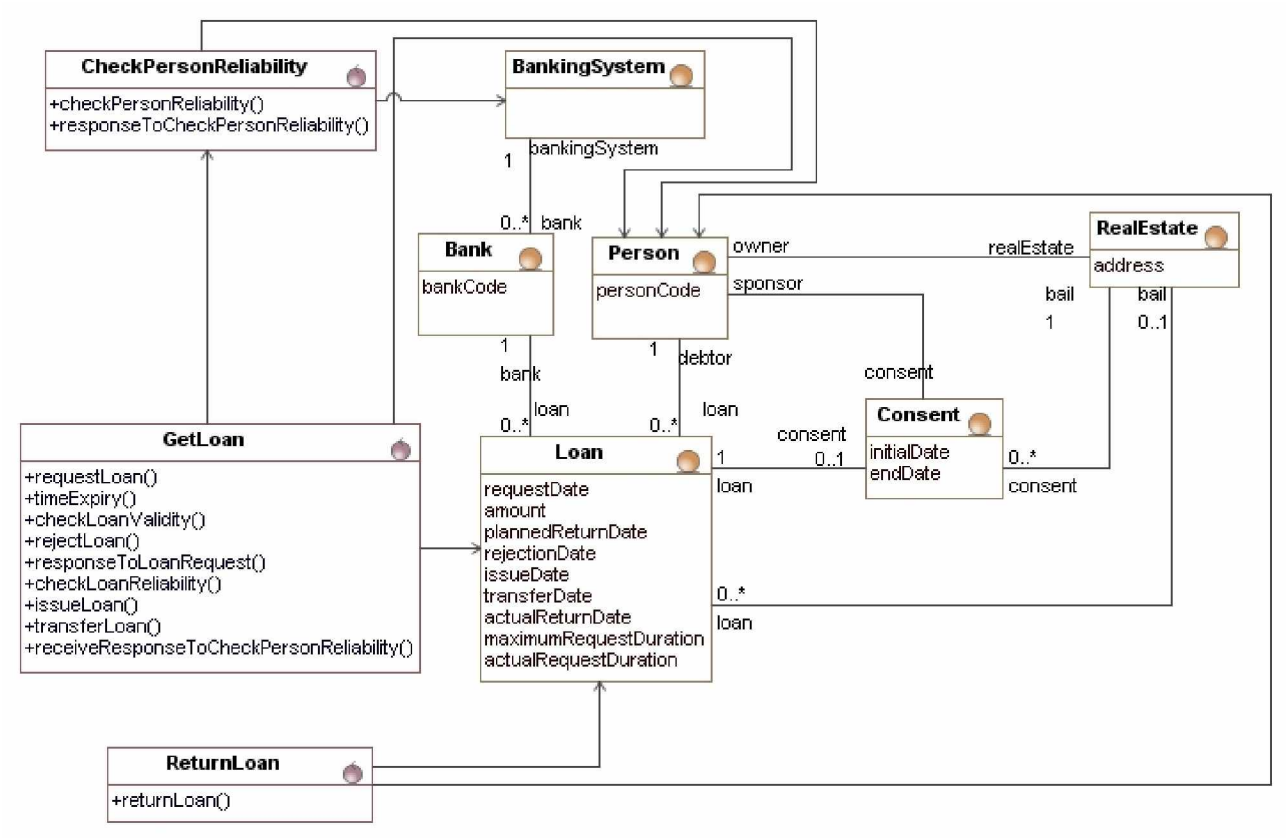


4.21 pav. Iš veiklos žodyno, veiklos taisyklių ir reikalavimų gautas paslaugų sistemos modelis (PIM)

Reikalavimų ir veiklos taisyklių transformavimas į paslaugų klasių modelį nėra vienintelis galimas realizavimo būdas. Pavyzdžiui, galimas grynai objektinis vaizdavimas, kai operacijos priskiriamos dalykinės srities klasėms (4.22 pav.) arba valdikliams (4.23 pav.).



4.22 pav. Iš veiklos žodyno, veiklos taisyklių ir reikalavimų gautas dalykinės srities klasių modelis (PIM)



4.23 pav. Iš veiklos žodyno, veiklos taisyklių ir reikalavimų gautas klasių modelis, kai operacinės veiklos taisyklės vaizduojamos valdiklių operacijomis (PIM)

4.3. Įskiepio naudingumo tyrimas

Palyginti TBE įskiepyje naudojamas šablonizuotas veiklos taisykles su OCL reiškiniais buvo panaudoti 3 skirtingo tipo reiškiniai. Tokiu būdų buvo palygintas reiškinių sudėtingumas ir analitiko įgūdžiai, reikalingi jiems sudaryti. Lyginami OCL reiškiniai ir TBExpression kalbos pagrindu sukurti reiškiniai pateikiami 4.4 lentelėje.

4.4 lentelė. OCL ir TBE reiškinių palyginimas

OCL reiškinys	TBE reiškinys
self.oclInState(ReliablePerson) implies self.loan->forAll(rl|rl.oclInState(ReturnedLoan) and rl.returnDate<=rl.plannedReturnDate	{([Loan.debtor.state] = [PersonState.reliablePerson]) IF {(for all ([Person.loan.state]=[LoanState.returnedLoan])) and (Person.loan.actualReturnDate)<=[Person.plannedReturnDate]}}
loan.oclInState(TransferredLoan)	([Loan.state] = [LoanState.transferedLoan])
self.oclInState(Returned) implies	{([Loan.actualReturnDate] is defined) IF

not self.actualReturnDate.oclIsUndefined()	([Loan.state] = [LoanState.returnedLoan])}

OCL kalbos žinių neturinčiam vartotojui OCL reiškiniai yra sudėtingi, o natūraliai kalbai artimi reiškiniai yra lengviau suprantami. Suformuoti reiškiniai turi labai panašią kalbos struktūrą, todėl tai leistų ateityje reiškinius transformuoti.

Įskiepio naudingumo įvertinimas:

- TBE įskiepio naudojimas suteikia veiklos taisyklėms lankstumo. Įskiepis naudodamas TBE Profile modulį, leidžia kurti, redaguoti natūraliai kalbai artimus šablonus. Tai suteikia veiklos taisyklėms didelį pranašumą ir lankstumą lyginant su standartiniu rankiniu būdu įvedamom veiklos taisyklėmis.
- TBE įskiepis sumažina klaidų tikimybę įvedinėjant veiklos taisykles. Automatizuotas reiškinių įvedimas sumažina klaidų tikimybę, kadangi vartotojas suveda reiškinių veiksnius (modelių elementus), iš pasiūlyto sąrašo.
- TBE įskiepis yra suprojektuotas taip, kad būtų lengvai praplečiamas. TBE įskiepis sukurtas JAVA programavimo kalba, objektinio programavimo principais. Tai leidžia praplėsti funkcionalumą be didesnių pastangų.

4.4. Tyrimo išvados

1. Įvertinus TBE įskiepio programos kodą, nustatyta kad jis suprogramuotas objektiniu principu. Tai leidžia lengvai jį plėsti ateityje.
2. TBE šablonų panaudojimas suteikia veiklos taisyklių specifikavimui lankstumo, juos aprašant įvairiomis kalbomis.
3. Atlikę eksperimentinį IS kūrimo uždavinį taikant įskiepį, paaiškėjo kad įskiepis geba įvesti ribojimus į 5 UML diagramas. Tų diagramų užtenka aprašyti visą IS kūrimo procesą.
4. Interaktyvus automatizuotas reiškinių specifikavimas sumažina korektiškumo klaidas, todėl sukurtas modelis yra teisingesnis.
5. Palyginę 3 skirtingus OCL bei TBE reiškinius, matome kad be tam tikrų OCL kalbos įgūdžių vartotojui sunku perprasti pačius reiškinius. Tuo pačiu kalbos struktūra išlieka mažai pakitus, kas suteikia ateityje atlikti transformacijas iš TBE reiškinio į OCL.

4.5. Sistemos ateities tobulinimo darbai

Veiklos taisyklių šablonų atpažinimas pagal stereotipus yra įprogramuotas į sistemą. Naujų algoritmų kūrimą, arba esamų algoritmų modifikavimą gali atlikti tik programuotojas. Tam reikalingas sistemos perkompiliavimas ir pakartotinas įdiegimas. Šiai problemai spręsti ateityje turi būti sukurtas komponentas galintis atlikti dinaminį papildomo programos kodo

užkrovimą sistemos veikimo metų.

Dabartinė sistema yra pritaikyta dirbti su klasių, būsenų mašinų, protokolų būsenų mašinų, veiklos bei sekų diagramomis. Prie kokių taškų prijungiami šablonizuoti ribojimai kiekvienam atvejui yra įprogramuoti į sistemą, todėl ateityje keičiantis UML specifikacijai, bei MagicDraw UML API funkcionalumui taip pat tektų perkompiliuoti įskiepį.

Suformuoti bendra šablonų struktūra leidžiačią pagal tam tikrus algoritmus transformuoti reiškinius į OCL.

Patobulinti veiklos taisyklių įvedimo mechanizmą, taip kad jis truktų kuo trumpiau. Taip pat vartotojui, turinčiam didelį elementų modelį, suteikti galimybę pačiam nusistatyti išrenkamus elementus.

Patobulinti reiškinio formavimo kalbą. Būtų naudinga įvesti papildomus žodžius, trumpinius, bei specifinius raktažodžius, kuriuos vartotojas taip pat turėtų galimybę įvesti į sistemą. Tai palengvintų reiškinių skaitymą ir sumažintų vaizduojamo teksto ilgį.

5. IŠVADOS

1. Literatūros šaltinių analizė rodo, kad veiklos taisyklių automatizavimas šiuo metu yra aktualus uždavinys. Tačiau veiklos taisyklės dažniausiai automatizuojamos naudojant specializuotą jų vykdymo programinę įrangą, o universalūs CASE įrankiai neturi patogių priemonių veiklos taisyklėms specifikuoti.
2. Veiklos taisyklių klasifikacijų ir metamodelių analizė leido daryti išvadą, kad visos veiklos taisyklių klasifikacijos yra panašios. Jos apima pagrindinius taisyklių tipus – struktūrines ir dinamines arba veiksmų taisykles. Norint šias taisykles taikyti modeliais grindžiamame objektiniame projektavime, tikslinga jas sieti su UML modeliais: struktūrines taisykles vaizduoti klasių modelio invariantais, o dinamines – prieš ir po sąlygomis.
3. Taikant taisykles informacinių sistemų kūrimo procese, taisykles tikslinga vaizduoti ir elgsenos modeliuose – būsenų mašinose, veiklos ar sekų diagramose.
4. Objektines taisykles galima specifikuoti OCL kalba, tačiau ji mažai paplitusi dėl to, kad ji per sudėtinga ne tik veiklos dalyviams, bet ir daugeliui analitikų ir projektuotojų. Todėl tikslinga papildyti CASE įrankius galimybėmis įvesti veiklos taisykles paprastesne ribota natūralia kalba.
5. Atliktos UML metamodelio, MagicDraw UML API, kitų CASE įrankių ir analitikų poreikių analizės pagrindu buvo nustatyti pagrindiniai struktūrizuotų ribojimų įvedimo reikalavimai: taisyklių įvedimas į įvairius UML modelius, modelio elementų pasirinkimas taisyklių įvedimo metu, modelio ir taisyklių atitikimo tikrinimas, naujų šablonų kūrimas.
6. Pagal šią metodiką Java kalba buvo sukurtas įskiepio prototipas, kuris leidžia kurti ir redaguoti ribojimus visose UML diagramose, suformuojant juos iš dalykinės srities modelio elementų, operatorių ir funkcijų. Galima pasirinkti modelio elementus pagal navigavimo kelius priklausomai nuo šių elementų ryšių, taip pat klasių ir jų supertipų savybes (atributus ir operacijas), operatorius. Leidžiama laisvai įvesti papildomas funkcijas, operatorius ir konstantas.
7. Ribojimams įvesti sukurti šablonai, kuriuos galima naudoti rekursiniu būdu, įdedant vieną į kitą. Kadangi tai bandomasis prototipas, į jį įtrauktos dažniausiai naudojamos funkcijos, operatoriai ir ribojimų šablonai, ateityje jų sąrašą reikėtų papildyti.
8. Įskiepiui kurti buvo taikomi objektinio programavimo principai ir naudojamas IntelliJ IDEA 6.0 paketas, dėl kurio programos kodo kūrimo procesas buvo lengvai valdomas ir analizuojamas.

9. Įskiepis išbandytas trim dalykinėms sritims: automobilių nuomai, kur buvo aprašinėjamos veiklos taisyklės; gamybinei sistemai, kur buvo aprašomi vientisumo ribojimai; taip pat ištisam veiklos taisyklėmis grindžiamos informacinės sistemos kūrimo procesui. Šie bandymai patvirtino, kad CASE įrankiams galima sukurti struktūrizuotų veiklos taisyklių įvedimo priemones ir taikyti jas projektuojant informacines sistemas.
10. Įskiepio taikymo tyrimas parodė, kad veiklos taisyklių aprašymo efektyvumas priklauso nuo kuriamos informacines sistemos sudėtingumo. Kuo sistema sudėtingesnė, tuo sudėtingiau aprašyti taisykles.
11. Remiantis įskiepio taikymo tyrimu, galima tvirtinti, kad dalinai automatizuotas ribojimų įvedimas padidina jų išsamumą ir teisingumą, o programos atliekamas elementų pasirinkimo srities susiaurinimas žymiai palengvina analitiko darbą.
12. Šio tyrimo pagrindu buvo išspausdintas straipsnis ir pristatytas pranešimas konferencijoje „Informacinės technologijos 2008“

6. LITERATŪRA

[1] **Editorial Staff of BRCommunity.com.** A Brief History of the Business Rule Approach , 2006, Prieiga per internetą *BRCommunity.com*

[2] **Ronald G. Ross and Gladys S. W. Lam Principals, Business Rule Solutions, LLC** RuleSpeakTM Sentence Templates *Developing Rule Statements Using Sentence Patterns, 2001*, Prieiga per internetą www.BRSolutions.com

[3] **T.A. Halpin.** "Business Rule Modality," *Proc. CAiSE'06 Workshops*. eds. T. Latour & M. Petit. Namur University Press (2006), pp. 383-394. Prieiga per internetą : http://www.orm.net/pdf/RuleModality.pdf

[4] **T.A. Halpin.** "Verbalizing Business Rules (Part 12)," *Business Rules Journal*, Vol. 6, No. 10 (October 2005). Prieiga per internetą http://www.BRCommunity.com/a2005/b252.html

[5] **David Hay, Keri Anderson Healy and others**. GUIDE Business Rules Project Report (October 1997). Prieiga per internetą http://www.guide.org/ap/apbrules.pdf

[6] **Semantics of Business Vocabulary and Business Rules Specification**, 2006, psl.: 1-390.

[7] **Gerd Wagner, Sergey Lukichev, Norbert E. Fuchs, and Silvie Spreeuwenberg** First-Version Controlled English Rule Language, 2005, psl. 2-47.

[8] **OMG** Object Constraint Language OMG Available Specification Version 2.0, 2006. psl.: 19-185.

[9] **J.Martin, M.Odell**. *Object-Oriented Methods: a Foundation*. *Prentice-Hall*, 1995, 412 p.

[10] The External Rule Language. TEMPORA Manual. *Esprit Project (P2469)*, 1993.

[11] **Fair Isaac Corporation ILOG SA.** Production Rule Representation, 2007

[12] **Riškus A**. Programavimas Java: vadovėlis. – K.: Technologija, 2004. – 150 p.

[13] **MagicDraw UML OpenAPI** dokumentacija [interaktyvus] Prieiga per internetą: http://www.magicdraw.com/files/manuals/12.5/MagicDraw%20OpenAPI%20UserGuide.pdf?NMSESSID=469bba9a3c2fadebc22b3d0490befba6

[14] **MOF 2.0 / XMI Mapping Specification**, v2.1 [interaktyvus]. [žiūrėta 2007-03-07]. Prieiga per internetą: http://www.omg.org/technology/documents/formal/xmi.htm

[15] Programos pagalbos kūrimas [interaktyvus]. [žiūrėta 2007-05-10]. Prieiga per internetą: http://www.devcity.net/Articles/114/makingofhelp.aspx

[16] Programos pagalbos paleidimas iš programos [interaktyvus]. [žiūrėta 2007-05-12].

Prieiga per internetą: http://www.javacoffeebreak.com/faq/faq0030.html

[17] Programos sąsajos kūrimo metodika [interaktyvus]. [žiūrėta 2007-03-22]. Prieiga per internetą: http://www.java2s.com/Code/Java/Swing-JFC/StyledText.htm

[18] Strelka ribojimų įvedimo programos pavyzdys [interaktyvus]. [žiūrėta 2007-03-01]. Prieiga per internetą: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=Strelka

[19] Leap SE ribojimų įvedimo programos pavyzdys [interaktyvus]. [žiūrėta 2008-12-15]. Prieiga per internetą: http://www.leapse.com/index.htm

[20] BROOD požiūris [interaktyvus]. [žiūrėta 2008-10-15]. Prieiga per internetą: http://books.google.lt/books?id=ULvgXaCULkUC&printsec=frontcover#PPA133,M1

7. PRIEDAI

7.1. Konferencijos straipsnis

UML CASE ĮRANKIO VEIKLOS TAISYKLIŲ ĮSKIEPIS

Justinas Bisikirskas, Audrius Bartkus

Kauno Technologijos Universitetas

Straipsnis trumpai apžvelgia struktūrizuotų ribojimų informacinių sistemų modeliams įvedimą. Išbandyti struktūrizuotus ribojimus buvo pasirinkta *MagicDraw* sistema, kuriai sukurtas veiklos taisyklių įvedimo įskiepis. Įskiepis remiasi pagrindiniais šablonizuoto įvedimo principais ir dalinai užtikrina įvedamų duomenų teisingumą, kadangi aprašant ribojimus galima pasirinkti modelio elementus pagal jų navigavimo ryšius. Sukurtas prototipas skirtas įvesti veiklos taisykles į klasių diagramas.

1. ĮVADAS

Kuriant informacinių sistemų programinę įrangą, labai svarbu išsamiai apibrėžti vartotojų poreikius ir sukurti efektyvų sistemos modelį, kurį būtų galima automatiškai apdoroti. Veiklos taisyklių modeliai vystomi jau daug metų [1, 2, 3, 4, 11], tačiau pastaruoju metu į šią veiklą įsitraukė ir objektinių standartų organizacija OMG [5]. UML CASE įrankių automatizuoto kūrimo galimybes gali padidinti OCL (*Object Constraint Language*) [9] ar kitos formalios kalbos, skirtos papildyti grafinius modelius veiklos taisyklių aprašais. Tačiau OCL ir kitos formalios kalbos yra per mažai naudojamos, kadangi programinės įrangos kūrime dalyvaujantys specialistai dažnai nemoka sudėtingų modeliavimo kalbų.

Pastaruoju metu intensyviai kuriamos veiklos taisyklių kalbos, kurias būtų galima naudoti informacinių sistemų projektavime [12], o dar geriau – kad jas būtu galima aprašyti natūraliai kalbai artima kalba [8]. Šio darbo tikslas – palengvinti veiklos taisyklių specifikavimą ir įvedimą į informacinių sistemų modelius, tam sukuriant reikalavimų lygio taisyklių vaizdavimo modelį, skirtą IS analitikams ir projektuotojams, bei jo realizaciją UML CASE įrankyje MagicDraw UML.

2. SUSIJUSIŲ DARBŲ APŽVALGA

Kalba, naudojama bendravimui tarp sistemos analitiko ir tam tikros srities eksperto, tam, kad išanalizuoti ir dokumentuoti sistemos reikalavimus, neturėtų būti „techninė“. Ji turėtų būti vizuali ir/arba sudaroma natūralios kalbos ar taisyklių rei6kiniais, kuriuos ekspertas supranta be papildomo techninio pasiruošimo.

UML mums siūlo vaizdavimo kalbą. Integravimo taisyklės ir išvedimo taisyklės gali būti vaizduojamos UML modeliuose tekstinėmis anotacijomis OCL kalba.

Tam, kad leistų vizualiai modeliuoti taisykles, REWERSE darbo grupė sukūrė UML

paremtą taisyklių modeliavimo kalbą (URML), kuri paveldi UML klasių modelius ir papildo juos taisyklėmis, ir „Strelka“ – įrankį, kuris sudaro grafinius URML modelius [7].

Mūsų manymu, grafinis taisyklių vaizdavimas, kai modelis nedidelis, yra puikus reikalavimų specifikavimo būdas. Bet jei modelis didelis ir taisyklių daug, grafinis taisyklių vaizdavimas labai išplės modelį ir jį bus sunkiau analizuoti.

Veiklos taisyklių metamodelio analizė pagal RuleML [6] trijuose skirtinguose abstrakcijos lygiuose:

4. Veiklos srities lygmenyje taisyklės yra formuluotės, kurios išreiškia veiklą (pvz. apibrėžia veiklos srities pagrindinius terminus, ribojimus, operacijas ir kt.) deklaratyviai, dažniausiai natūralia kalba ar vizualiai. Pavyzdžiui:

(T1) „Vairuotojas, kuris nuomojasi automobilį, turi būti vyresnis nei 25 metų“.

(T2) „Aukso pirkėjas privalo turėti daugiau nei vieną milijoną dolerių depozite“.

(T3) „Kai akcijos kaina krinta daugiau nei 5% ir investicijoms netaikomas pelno mokestis, tai akcijas parduoti“.

T1 yra vientisumo taisyklė, T2 diferencijavimo taisyklė, T3 reakcijos taisyklė. Tai pagrindinės semantinės veiklos taisyklių kategorijos. Faktiškai dauguma veiklos taisyklių yra reakcijos taisyklės, kurios nusako veiklos strategijas.

5. Nuo platformos nepriklausančiame lygyje, taisyklės yra formalios, išreikštos formalizmu ar skaičiavimo paradigma. Tai tarsi visos veiklos srities abstrakcija. Taisyklių kalbos naudojamos šiame lygmenyje yra SQL:1999, OCL 2.0 ir ISO Prolog.
6. Nuo platformos priklausančiame lygmenyje, taisyklės nusakomos specifinėmis vykdomųjų programų kalbomis: Oracle 10g, Jess 3.4, XSB 2.6 Prolog ar kt.

Bendrai, taisyklės talpina žinias, kuriomis aprašomi samportavimai. Jos gali specifikuoti:

- Statinius ar dinaminius vientisumo ribojimus;
- Išvedimą
- Reakcijas

Panagrinėkime pagrindines modelių taisykles [10]:

Integralumo (vientisumo) taisyklės.

Integralumo taisyklės, taip pat žinomos kaip integralumo ribojimai, susideda iš loginių sakinių. Taisyklė T1 (aprašyta anksčiau) yra statinio ribojimo pavyzdys. Dinaminio ribojimo pavyzdys: ,,Nuomos rezervavimo patvirtinimas turi būti susijęs su automobilio iš tam tikros automobilių grupės priskyrimu atsižvelgiant į pareikalavimo datą ir laikant ją prioritetu nuomojant automobilį.” Gerai žinomos kalbos ribojimams išreikšti yra SQL ir OCL.

Išvedimo taisyklės (*Derivation Rules*).

Išvedimo taisyklės susideda iš vienos ar daugiau sąlygų ir vienos ar daugiau išvadų, kurios abi išreiškiamos loginėmis formulėmis (angl. *LogicalFormula*).

Reakcijos taisyklės (*Reaction rules*).

Reakcijos taisyklių tipas yra laikomos svarbiausiu veiklos taisyklių tipu. Jos susideda iš privalomo sužadinimo įvykio, nebūtinos sąlygos, ir vykdomo įvykio, kurie yra įvykių

(*EventTerm)*, loginių formulių (*LogicalFormula)* ir veiksmų *(ActionTerm)* tipų. Reakcijos taisyklės gali turėti veiksmų seką ir susietą taisyklę, aprašomą kaip *else* sąlyga, kuri taip pat yra loginė formulė.

3. VEIKLOS TAISYKLIŲ ĮVEDIMO Į CASE ĮRANKĮ PROTOTIPAS

Sukurtas veiklos taisyklių arba ribojimų (UML kalboje visos veiklos taisyklės traktuojamos kaip ribojimai) įvedimo įskiepis skirtas sistemų analitikams, nagrinėjantiems kuriamų sistemų reikalavimus bei specifikuojantiems juos *MagicDraw* įrankiu.

Mūsų sukurtame įskiepyje struktūrizuota kalba realizuota kartu su interaktyvia logine pagalba. Vartotojas įveda taisykles pagal šablonus, kuriuos pateikia sistema. Be to, ji leidžia pasirinkti konteksto elementus (klases ir atributus), kadangi veiklos taisyklės specifikuojamos naudojant konteksto sąvokas. Vartotojui tereikia spragtelti pelės dešiniu klavišu ant ribojimo eilutėje esančio teksto ir sistema automatiškai atpažįsta elementus pagal specialius simbolius: ar tai ribojimo frazė, sąlyga, ar argumentas. Tai padarius, sistema į atitinkamus ribojimų argumentų ir šablonizuotų formų laukus pateikia visus galimus elementus, kuriuos vartotojas gali pasirinkti. Tokiu būdu vartotojui nesunku įvesti ribojimą, o duomenų teisingumą iš dalies padeda užtikrinti sistema.

Prototipo funkcionalumas

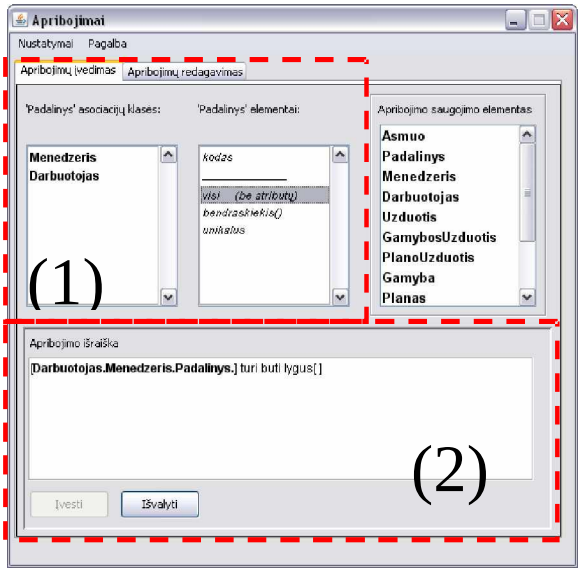
Naudojant šį ribojimų įvedimo įskiepį galima:

- Įvesti ribojimus *MagicDraw* klasių diagramos elementams;
- Redaguoti įvestus *MagicDraw* klasių diagramos ribojimus;
- Šalinti ribojimus.

Ribojimo įvedimas susideda iš kelių etapų:

§ ***Ribojimo reiškinio įvedimas:***

Ribojimo reiškinį galima vesti ranka (2) arba naudojant ribojimų frazių ruošinius (1) [2, 11] (1 pav.).



1 pav. Įskiepio ribojimų įvedimo langas

(1) variantas yra patogesnis, tačiau ribotų galimybių, kadangi norint įvesti specifinį ribojimą su specifine fraze, kurios nėra ruošiniuose, reikia įvesti tą frazę ranka, o tai jau yra (2) variantas.

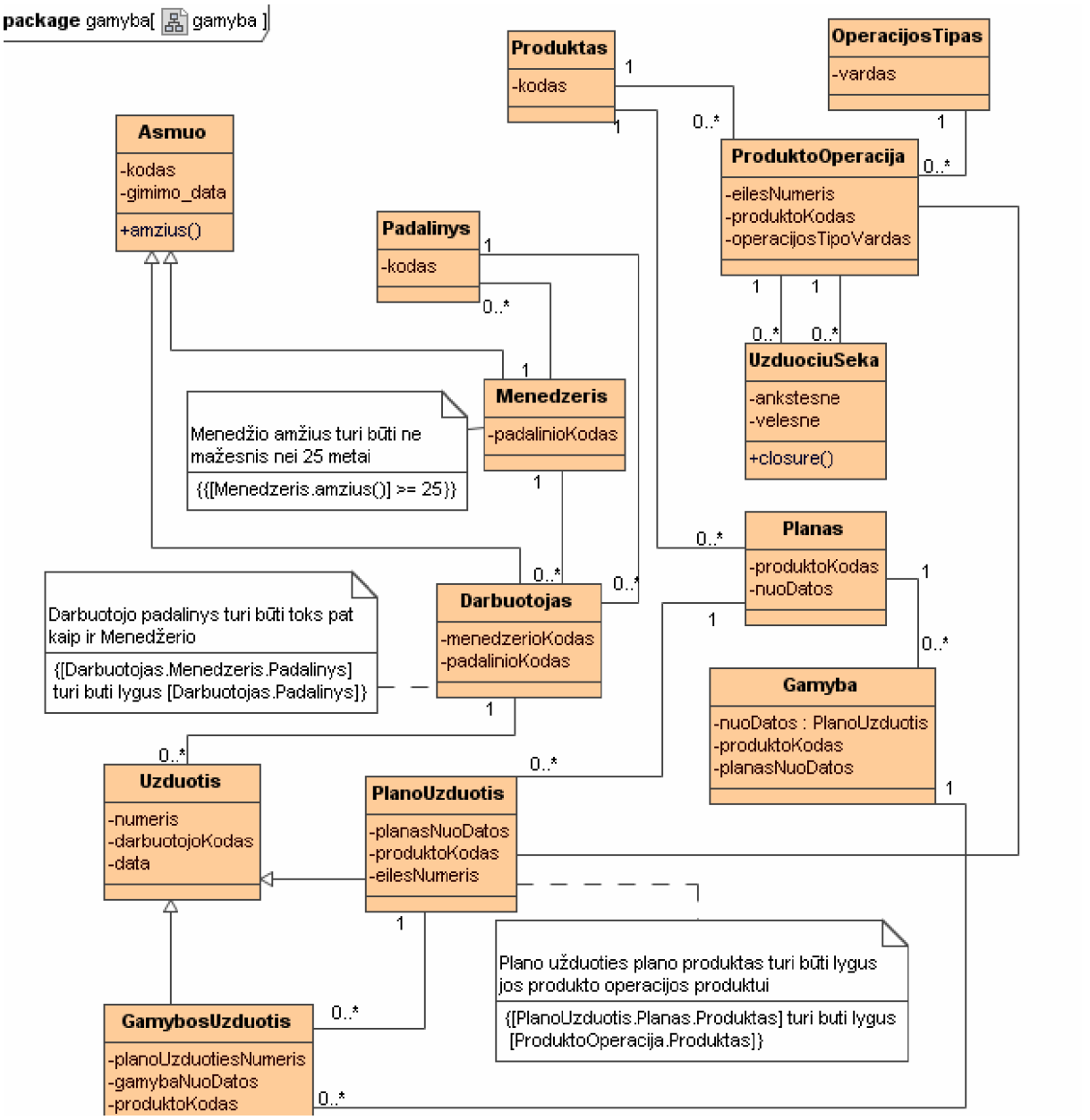
Ribojimo frazės rašymas antruoju (2) variantu yra laisvesnio pobūdžio – vartotojas gali vesti savo žodžius, bet pagrindinius elementus – sąlygas, argumentus ir frazes privalo išskirti jiems būdingais simboliais:

- frazę – { frazės tekstas };
- sąlygą – (sąlygos tekstas);
- argumentą – [argumentas].

Laikantis šių taisyklių, ribojimo įvedimas į modelį yra greitas ir aiškus.

Šiuo metu realizuotas ribojimų įvedimas tik klasių diagramoms. Galutiniame variante ribojimai bus įvedami ir į kitas diagramas. Bus galima susikurti savo ribojimų šablonus Tokiu būdu vartotojas galės lanksčiai aprašyti jam reikalingus ribojimus.

Ribojimų įvedimo pavyzdys, naudojant įskiepį, pateikiamas 2 paveiksle.



2 pav. Vientisumo ribojimų įvedimas, naudojant įskiepį

4. IŠVADOS

Pasiūlytas prototipas, skirtas įvesti UML klasių ribojimus, leidžia dalinai užtikrinti duomenų teisingumą, o naudojant šablonizuotus ribojimus, sudarytus iš natūraliai kalbai artimų sąvokų, leidžia nesunkiai juos perprasti ir naudotis.

Įskiepio prototipas leidžia iš modelio pasirinkti elementus pagal navigavimo kelius priklausomai nuo elementų ryšių, taip pat klasių ir jų supertipų savybes (atributus ir operacijas), operatorius. Leidžiama ir laisvai įvesti papildomas funkcijas, operatorius ir konstantas.

Ribojimų įvedimas kalba, artima natūraliai, yra sudėtingas iteracinis procesas. Prototipo kūrimo metu buvo išnagrinėta daugelis specifikacijų, kurios yra daugiau teorinės nei praktiškai pagrįstos. Sukurtas įskiepis neturi analogų.

Įskiepis ateityje galėtų turėti išplėtimą: transformuoti ribojimus iš ir į OCL kalbą.

5. LITERATŪROS SĄRAŠAS

[1] **Editorial Staff of BRCommunity.com.** A Brief History of the Business Rule Approach , 2006, Prieiga per internetą *BRCommunity.com*

[2] **Ross, R. G., Lam. G.S.W.** Principles of Business Rule Solutions, LLC RuleSpeakTM Sentence Templates *Developing Rule Statements Using Sentence Patterns, 2001*, Prieiga per internetą www.BRSolutions.com

[3] **Halpin, T.A.** Business Rule Modality. *Proc. CAiSE'06 Workshops*. eds. T. Latour & M. Petit. Namur University Press (2006), 383-394. Prieiga per internetą : http://www.orm.net/pdf/RuleModality.pdf

[4] **Halpin, T.A.** Verbalizing Business Rules (Part 12), *Business Rules Journal*, Vol. 6, No. 10 (October 2005). Prieiga per internetą http://www.BRCommunity.com/a2005/b252.html

[5] Semantics of Business Vocabulary and Business Rules Specification, 2006, 1-390.

[6] **Wagner, G., Tabet, S., Boley, H.** *MOF-RuleML: The Abstract Syntax of RuleML as a MOF model*", Integrate 2003, OMG Meeting, Boston, October 2003.

[7] Lukichev, S., Wagner, G. Visual Rules Modeling. In proceedings of Sixth International Andrei Ershov Memorial Conference Perspectives of System Informatics, Novosibirsk, Russia, June 2006, Springer LNCS, 2005.

[8] **Wagner, G., Lukichev, S., Fuchs, N.E., Spreeuwenberg, S.** First-Version Controlled English Rule Language, 2005, 2-47.

[9] **OMG** Object Constraint Language OMG Available Specification Version 2.0, 2006, 19-185.

[10] **Martin, J., Odell**, M. *Object-Oriented Methods: a Foundation*. *Prentice-Hall*, 1995, 412 p.

[11] The External Rule Language. TEMPORA Manual. *Esprit Project (P2469)*, 1993.

[12] **Isaac, F.** Production Rule Representation, **Corporation ILOG SA,** 2007

6. BUSINESS RULES PLUG-IN FOR UML CASE TOOL

This paper shortly reveals template-based business rules for information system models. To enter formatted business rules into models we have developed plug-in for MagicDraw UML. This plug-in is based on business rule templates, which partially ensure correctness of UML class models. Developed plug–in enables user to enter business rules to class diagrams, however we are planning to extend it for other UML diagrams..