VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

Žygimantas Lyva

TAISYKLĖMIS GRINDŽIAMŲ DINAMINIŲ VERSLO PROCESŲ SIMULIACIJOS TYRIMAS

RESEARCH ON THE RULE-DRIVEN DYNAMIC BUSINESS PROCESS SIMULATION

Baigiamasis magistro darbas

Informacinių sistemų programų inžinerijos programa, valstybinis kodas 621E15002
Informacinių sistemų specializacija
Informatikos inžinerijos studijų kryptis

Vilnius, 2018

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas
_____
(Parašas)
_____
(Vardas, pavardė)
_____
(Data)

Žygimantas Lyva

TAISYKLĖMIS GRINDŽIAMŲ DINAMINIŲ VERSLO PROCESŲ SIMULIACIJOS TYRIMAS

RESEARCH ON THE RULE-DRIVEN DYNAMIC BUSINESS PROCESS SIMULATION

Baigiamasis magistro darbas

Informacinių sistemų programų inžinerijos programa, valstybinis kodas 621E15002
Informacinių sistemų specializacija
Informatikos inžinerijos studijų kryptis

Vadovas dr. Kęstutis Normantas _____ _____
(Moksl. laipsnis/pedag. vardas, vardas, pavardė) (Parašas) (Data)

Lietuvių kalbos konsultantas lekt. Regina Žukienė _____ _____
(Moksl. laipsnis/pedag. vardas, vardas, pavardė) (Parašas) (Data)

Vilnius, 2018

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

Informatikos inžinerijos studijų kryptis
Informacinių sistemų programų inžinerijos studijų programa, valstybinis kodas 621E15002
Informacinių sistemų specializacija

TVIRTINU
Katedros vedėjas
_____
(Parašas)
prof. dr. Dalius Mažeika
(Vardas, pavardė)
_____
(Data)

BAIGIAMOJO MAGISTRO DARBO UŽDUOTIS

…….....Nr. .....
Vilnius

Studentui Žygimantui Lyvai
Baigiamojo darbo tema: taisyklėmis grindžiamų dinaminių verslo procesų simuliacijos tyrimas.
patvirtinta 2017 m. spalio 25 d. dekano potvarkiu Nr. 437 fm

Baigiamojo darbo užbaigimo terminas 2018 m. birželio 8 d.

BAIGIAMOJO DARBO UŽDUOTIS:

1. Atlikti susijusios literatūros bei priemonių analizę:
   a. Nustatyti pagrindinius DVP simuliacijos būdus;
   b. Išanalizuoti verslo taisyklių klasifikavimo ir specifikavimo būdus;
   c. Išanalizuoti verslo taisyklių valdymo ir/ar vykdymo sistemas.
2. Pasiūlyti būdą, kaip patobulinti DVP simuliacijos metodą ir jį palaikančią priemonę.
3. Įgyvendinti pasiūlytą būdą bei atlikti jo realizacijos testavimus.
4. Atlikti eksperimentinius tyrimus bei nustatyti pasiūlyto būdo patogumą ir efektyvumą.

Vadovas ..... dr. Kęstutis Normantas
(Parašas) (Moksl. laipsnis/pedag.vardas, vardas, pavardė)

Užduotį gavau

……..
(Parašas)

Žygimantas Lyva
(Vardas, pavardė)

……..
(Data)

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių sistemų katedra

ISBN ISSN
Egz. sk. .....
Data .....-.....-.....

Antrosios pakopos studijų **Informacinių sistemų programų inžinerijos** programos magistro baigiamasis darbas

Pavadinimas	**Taisyklėmis grindžiamų dinaminių verslo procesų simuliacijos tyrimas**
Autorius	**Žygimantas Lyva**
Vadovas	**Kęstutis Normantas**

Kalba: lietuvių

Anotacija

Šiame darbe nagrinėjama taisyklėmis grindžiamų dinaminių verslo procesų simuliacijos tema. Nustatoma, jog VGTU mokslininkų vystomas dinaminių verslo procesų simuliacijos metodas bei jį palaikanti priemonė nėra patogūs verslo taisyklėms aprašyti ir valdyti, o taisyklių vykdymas yra nepakankamai efektyvus. Todėl pasiūlyta verslo taisyklių valdymą ir vykdymą atskirti į specializuotą sistemą, taip užtikrinant efektyvesnį verslo taisyklių valdymo ir vykdymo būdą. Tyrimo metu buvo pasirinkta atviro kodo taisyklių vykdymo sistema, kuri buvo pritaikyta dinaminiam taisyklių valdymui ir vykdymui. Pasiūlytas sprendimas suteikia galimybę naudojant grafinę vartotojo sąsają kurti naujas ar keisti esamas taisykles bei jas aktyvuoti neperkraunant viso simuliacijos proceso. Be to, jis leidžia operuoti dideliais taisyklių kiekiais, užtikrinant taisyklių valdymo dinamiškumą ir vykdymo greitaveiką. Siekiant patikrinti pasiūlyto sprendimo teisingumą, patogumą bei efektyvumą, buvo atliekami eksperimentiniai tyrimai, kurių rezultatai parodė, jog darbo tikslas buvo pasiektas bei suteikė galimybę suformuoti pasiūlymus tolimesniems tyrimams.

Darbo apimtis – 83 p. teksto be priedų, 32 pav., 16 lentelių, 54 bibliografinių šaltinių. Atskirai pateikiami darbo priedai.

Prasminiai žodžiai: Dinaminiai verslo procesai, dinaminių verslo procesų simuliacija, taisyklių vykdymo ir valdymo sistemos

Vilnius Gediminas Technical University
Faculty of Fundamental Sciences
Department of Information Systems

ISBN ISSN
Copies No. .....
Date .....-.....-.....

Master Degree Studies **Information Systems Software Engineering** study programme Master Graduation Thesis

Title	**Research on The Rule-Driven Dynamic Business Process Simulation**
Author	**Žygimantas Lyva**
Academic supervisor	**Kęstutis Normantas**

Thesis language: Lithuanian

Annotation

This thesis tackles the topic of rule-driven dynamic business process simulation. During the research, the main drawbacks of the dynamic business process simulation method and supporting tool, which are developed by VGTU researchers, were identified: inconvenient definition and management of business rules and inefficient rule execution approach. Therefore, it was proposed to separate business rules management and execution from the simulation system to a separate rule management and execution system. The open source rule engine has been selected and adapted to dynamic management and execution of rules. The proposed solution enables to perform creation, modification, and activation of rules without a need to restart the entire simulation process. Moreover, it allows to operate with large amount of business rules while ensuring their dynamicity and high execution performance. Experimental evaluation has been performed to validate the propriety, convenience and efficiency of the proposed solution. The results of experimental evaluation showed that the main goal of this thesis has been achieved. Moreover, these results allowed to formulate a set of insightful recommendations for the further research.

Thesis scope – 83 p. text without appendixes, 32 ill, 16 tables, 54 bibliographic sources. Separate paper appendixes.

Keywords: Dynamic business processes, simulation of dynamic business processes, rules execution and management systems

Vilniaus Gedimino technikos universiteto egzaminų sesijų ir baigiamųjų darbų rengimo bei gynimo organizavimo tvarkos aprašo
2 priedas

(Baigiamojo darbo sąžiningumo deklaracijos forma)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Žygimantas Lyva, 20123815
(Studento vardas ir pavardė, studento pažymėjimo Nr.)
Fundamentinių mokslų fakultetas
(Fakultetas)
Informacinių sistemų programų inžinerija, ISIfm-16
(Studijų programa, akademinė grupė)

BAIGIAMOJO DARBO (PROJEKTO)

SĄŽININGUMO DEKLARACIJA

2018 m. birželio 04 d.

Patvirtinu, kad mano baigiamasis darbas tema „Taisyklėmis grindžiamų dinaminių verslo procesų simuliacijos tyrimas“ patvirtintas 2017 m. spalio 25 d. dekano potvarkiu Nr. 437fm, yra savarankiškai parašytas. Šiame darbe pateikta medžiaga nėra plagijuota. Tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos pažymėtos literatūros nuorodose.

Mano darbo vadovas dr. Kęstutis Normantas.

Kitų asmenų indėlio į parengtą baigiamąjį darbą nėra. Jokių įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs (-usi).

(Parašas) Žygimantas Lyva (Vardas ir pavardė)

TURINYS

PAVEIKSLŲ SĄRAŠAS 9
LENTELIŲ SĄRAŠAS 10
SANTRUMPOS 11
1. ĮVADAS 12
1.1. Tyrimo objektas 12
1.2. Darbo tikslas ir uždaviniai 12
1.3. Temos naujumas 13
1.4. Temos aktualumas 13
1.5. Tyrimo metodika 13
1.6. Mokslinė darbo vertė 13
1.7. Darbo struktūra 14
2. ANALITINĖ DALIS 15
2.1. Verslo procesai 15
2.1.1. Dinaminiai verslo procesai 15
2.1.2. Verslo procesų simuliacija 16
2.1.3. Įvykiais grindžiamų verslo procesų simuliacijos modeliai 18
2.2. Verslo taisyklės 20
2.2.1. Verslo taisyklės verslo procesuose 20
2.2.2. Verslo taisyklių aprašymo būdai 22
2.2.2.1. Taisyklių vaizdavimas struktūrizuota anglų kalba 22
2.2.2.2. Objektų ribojimų kalba (OCL) 24
2.2.2.3. Sprendimų lentelės 26
2.3. Verslo taisyklių valdymo ir vykdymo sistemos 28
2.3.1. Reikalavimai priemonėms skirtoms darbui su verslo taisyklėmis 28
2.3.2. Verslo taisyklių valdymo ir vykdymo sistemų savybės 30
2.3.3. Verslo taisyklių valdymo ir vykdymo sistemų architektūra 31
2.4. Taisyklių valdymo ir vykdymo sistemų analizė 32
2.5. „RETE“ algoritmas 35
2.6. Analitinės dalies apibendrinimas 37
3. PROJEKTINĖ DALIS 38
3.1. Esama situacija 38
3.2. „NRules“ sistemos analizė 40
3.2.1. „NRules“ sistemos architektūra 40

3.2.2. „NRules“ sistemos taisyklių analizė 41
3.2.3. „NRules“ sistemos veikimo analizė 46
3.3. „NRules“ sistemos dinamiškumo problemos 48
3.4. Reikalavimai siūlomo būdo įgyvendinimui 49
3.5. Siūlomo būdo koncepcija 49
3.6. Siūlomo būdo įgyvendinimas 52
3.6.1. Pakeitimai „NRules“ sistemoje 52
3.6.2. „NRules“ sistemos išplėtimas 55
3.7. Atnaujinta „NRules“ sistemos veikimo proceso schema 61
3.8. Projektinės dalies apibendrinimas 62
4. EKSPERIMENTINĖ DALIS 63
4.1. Reikalavimai eksperimentiniam tyrimui 63
4.2. „NRules“ sistemos pakeitimų dinamiškumo įvertinimas 63
4.3. „NRules“ sistemos pakeitimų efektyvumo ir teisingumo tyrimas 68
4.3.1. Dalykinės srities aprašymas 68
4.3.2. Dalykinės srities verslo taisyklių rinkinys 69
4.3.3. Dalykinės srities faktų rinkinys 71
4.3.4. Eksperimento vykdymas 71
4.3.5. Eksperimento rezultatai 73
4.4. Eksperimentinės dalies apibendrinimas 74
DARBO REZULTATAI 75
IŠVADOS 76
PASIŪLYMAI 78
DISKUSIJA 79
LITERATŪROS SĄRAŠAS 80
PRIEDAI 84

PAVEIKSLŲ SĄRAŠAS

1 pav. Verslo proceso simuliacijos gyvavimo ciklas [12] 17
2 pav. Įvykiais grindžiamų verslo procesų simuliacijos modelių įvykių tipai [16] 19
3 pav. Struktūruotą anglų kalba parašyta verslo taisyklė [24] 23
4 pav. Klasių diagrama, sudaryta remiantis OCL specifikacija [25] 25
5 pav. Taisyklių valdymo sistemos architektūra [35] 31
6 pav. „NRules“ sistemos sugeneruotas „RETE“ tinklas sudarytas iš vienos taisyklės 36
7 pav. „DBPSim“ įrankio architektūra [42] 38
8 pav. „NRules“ taisyklių valdymo ir vykdymo sistemos architektūros komponentų diagrama 40
9 pav. „NRules“ taisyklių formų sąryšių modelis 41
10 pav. Dalykinės srities klasių diagrama 43
11 pav. „Fluent DSL“ forma užrašyta taisyklė 43
12 pav. Galimos transformacijos nenaudojant kanoninio duomenų modelio 44
13 pav. Galimos transformacijos naudojant kanoninį duomenų modelį 44
14 pav. „Canonical model“ forma užrašyta taisyklė 45
15 pav. „RETE“ tinklas sudarytas iš dviejų taisyklių 46
16 pav. „NRules“ sistemos veikimo proceso schema 47
17 pav. Detalizuota sesijų valdymo (angl. Serve Session) veikimo proceso schema 48
18 pav. Siūlomo metodo aukšto lygio koncepcijos diagrama 49
19 pav. Siūlomų pakeitumų ir „NRules“ sistemos integracijos į „DBPSim“ įrankį architektūros komponentų diagrama (geltona spalva pažymėtus modulius yra siūloma sukurti arba keisti) 50
20 pav. Atnaujinto taisyklių konstruktoriaus, sukompiliuotos taisyklės, terminalo ir taisyklės mazgų klasių diagrama 52
21 pav. Atnaujinto sesijų kūrimo ir sesijos klasių diagramos 53
22 pav. Naujai sukurtų “NRules“ sistemos valdiklio, taisyklių duomenų bazės ir sesijos valdiklio klasių diagramos 55
23 pav. Naujai sukurtų „NRules“ sistemos taisyklių valdiklio, taisyklių konstruktoriaus ir „Roslyn“ taisyklių kompiliatoriaus klasių diagramos 57
24 pav. „NRules“ taisyklių valdiklio, taisyklių konstruktoriaus ir „RETE“ tinklo vizualizacijos grafinės vartotojo sąsajos 60
25 pav. Atnaujintas „NRules“ sistemos veikimo proceso schema 61
26 pav. Dinamiškumo testuose naudojamo duomenų modelio klasių diagrama 64
27 pav. Atsitiktinai sugeneruotos „C#“ programavimo kalba parašytos taisyklės sąlygos pavyzdys 64
28 pav. Taisyklės pridėjimo trukmės priklausomybė nuo taisyklių ir faktų skaičiaus, esant baziniam ir patobulintam taisyklių pridėjimo metodui 65
29 pav. Faktų užkrovimo į „RETE“ tinklą metodo įtaka taisyklės pridėjimo trukmės priklausomybei nuo taisyklių ir faktų skaičiaus, patobulintam taisyklių pridėjimo metodui 66
30 pav. Taisyklės ištrynimo trukmės priklausomybė nuo taisyklių ir faktų skaičiaus, esant baziniam ir patobulintam taisyklių pridėjimo metodui 67
31pav. „EU-Rent“ dalykinės srities duomenų modelio klasių diagrama 68
32 pav. Taisyklės pridėjimo trukmės priklausomybė nuo pridedamos taisyklės, faktų skaičiaus ir taisyklės pridėjimo metodo B. m. (Bazinis metodas) ir P. m. (Patobulintas metodas) 73

LENTELIŲ SĄRAŠAS

1 lentelė. Sprendimų lentelės dalys [27] 26
2 lentelė. Poilsio dienų skyrimo sprendimų lentelė [29] 27
3 lentelė. Taisyklių valdymo ir vykdymo sistemų palyginimas 33
4 lentelė. Atnaujinto taisyklių konstruktoriaus, sukompiliuotos taisyklės, terminalo ir taisyklės mazgų klasių diagramų metodai ir atributai 53
5 lentelė. Atnaujinto sesijų kūrimo ir sesijos klasių diagramų metodai ir atributai 54
6 lentelė. Naujai sukurtų "NRules" sistemos valdiklio, taisyklių duomenų bazės ir sesijos valdiklio klasių diagramų metodai ir atributai 56
7 lentelė. Naujai sukurtų „NRules" sistemos taisyklių valdiklio, taisyklių konstruktoriaus ir „Roslyn" taisyklių kompiliatoriaus klasių diagramų metodai ir atributai 58
8 lentelė. Atsitiktinai generuojamų taisyklių sprendimų lentelė 64
9 lentelė. Dalykinės srities verslo taisyklės 69
10 lentelė. Taisyklės pridėjimo trukmės priklausomybė nuo pridedamos taisyklės, faktų skaičiaus ir taisyklės pridėjimo metodo B. m.(Bazinis metodas) ir P. m. (Patobulintas metodas) 72
11 lentelė. Nuomininkų faktų rinkinio fragmentas 94
12 lentelė. Šalių faktų rinkinio fragmentas 94
13 lentelė. Automobilių modelių faktų rinkinio fragmentas 94
14 lentelė. Automobilių faktų rinkinio fragmentas 94
15 lentelė. Padainių faktų rinkinio fragmentas 94
16 lentelė. Automobilių nuomų faktų rinkinio fragmentas 95

SANTRUMPOS

Santrumpa	Paaiškinimas
OMG (angl. *Object Management Group*)	Tai tarptautinis, atviros narystės, ne pelno siekiantis informacinių technologijų industrijos konsorciumas.
UML (angl. *Unified Modelin Language*)	Universali kalba, skirta specifikuoti, vizualizuoti ir dokumentuoti programinės įrangos sistemas, taip pat naudojama ir verslo modeliavimui ir kitokioms ne programinėms sistemoms.
BPD (angl. *Business Process Diagram*)	Verslo proceso diagrama, naudojama BPMN notacijoje.
BPMN (*Business Process Modeling Notation*)	Grafinė verslo procesų specifikavimo notacija, kurioje naudojama vienintelio tipo diagrama, vadinama verslo proceso diagrama BPD.
SBVR (angl. *Semantics of Business Vocabulary and Rules*)	Verslo žodyno ir verslo taisyklių semantika.
OCL (angl. *Object Constraint Language*)	Objektų ribojimų aprašymo kalba.
DVP	Dinaminis verslo procesas.
XML (angl. *Extensible Markup Language*)	Bendros paskirties duomenų struktūrų bei turinio aprašomoji kalba, skirta lengvam duomenų keitimuisi tarp skirtingo tipo programų.
DSL (angl. *Domain Specific Language*)	Dalykinės srities kalba.

1. ĮVADAS

Šiuolaikinis verslas, norėdamas išlikti konkurencingas rinkoje, privalo gebėti laiku prisitaikyti prie jį supančios aplinkos pokyčių. Galimybė imituoti (simuliuoti) verslo procesus (VP), stebėti ir koreguoti jų vykdymą, pridėti naujų procesų vykdymą ar nustatyti tendencijas bei gauti prognozes tampa labai aktualu verslo atstovams, norintiems priimti savalaikius, patikimomis žiniomis grįstus sprendimus.

Tradicinės verslo procesų simuliacijos (VPS) priemonės suteikia galimybę imituoti bei stebėti VP vykdymą, tačiau dažniausiai tokiose priemonėse galimi VP vykdymo scenarijai turi būti iš anksto žinomi. Tai nėra itin aktualu verslui, kurio procesus sudėtinga aprašyti dėl daugybės scenarijų, lemiamų VP aplinkos (konteksto) pokyčių, kurių iš anksto numatyti yra pernelyg sudėtinga. Dinaminių verslo procesų (DVP) simuliacijos metodas, vystomas VGTU Informacinių sistemų mokslo laboratorijoje, siūlo taisyklėmis grindžiamą verslo procesų aprašymo ir simuliacijos sprendimą. Tačiau šiuo metu DVP sprendime taisykles aprašyti ir valdyti yra nepatogu, o taisyklių vykdymas yra nepakankamai efektyvus.

Šiame darbe nagrinėjama problema: VGTU mokslininkų vystomas dinaminių verslo procesų simuliacijos metodas bei jį palaikanti priemonė nėra patogūs verslo taisyklėms aprašyti ir valdyti, o taisyklių vykdymas yra nepakankamai efektyvus.

1.1. Tyrimo objektas

Tyrimo objektas – dinaminių verslo procesų simuliacija, taikant taisyklių valdymo ir vykdymo sistemas.

1.2. Darbo tikslas ir uždaviniai

Darbo tikslas – patobulinti dinaminių verslo procesų simuliacijos metodą pasiūlant efektyvesnį verslo taisyklių valdymo ir vykdymo būdą.

Šiam tikslui pasiekti yra formuluojami tokie uždaviniai:

1) Atlikti susijusios literatūros bei priemonių analizę:
    a) Nustatyti pagrindinius DVP simuliacijos būdus;
    b) Išanalizuoti verslo taisyklių klasifikavimo ir specifikavimo būdus;
    c) Išanalizuoti verslo taisyklių valdymo ir/ar vykdymo sistemas.
2) Pasiūlyti būdą, kaip patobulinti DVP simuliacijos metodą ir jį palaikančią priemonę.
3) Įgyvendinti pasiūlytą būdą bei atlikti jo realizacijos testavimus.
4) Atlikti eksperimentinius tyrimus bei nustatyti pasiūlyto būdo patogumą ir efektyvumą.

1.3. Temos naujumas

Tradicinės VP simuliavimo priemonės yra skirtos simuliuoti statinius verslo procesus. Šių simuliacijų metu, verslo procesų veiklos ir veiklų sekos yra griežtai apibrėžtos. Temoje keliama problema nėra nauja, tačiau verslo procesų simuliacija naudojant verslo taisyklių mašinas, kurių vykdomoji taisyklių forma yra dinamiškai valdomas „RETE“ tinklas yra vienas naujausių simuliacijos būdų, leidžiančių simuliuoti dinaminius verslo procesus. Tokios sistemos leidžia taisykles naudoti kaip autonominius vienetus ir atskirti nuo taikomosios programos kodo. Taip pat šios sistemos gali būti integruojamos su kitomis procesų simuliacijos priemonėmis.

1.4. Temos aktualumas

Dinaminių verslo procesų simuliacija šiuo metu yra aktuali tema. Ši tema pritraukia daug dėmesio jau kelerius metus, tačiau vis dar nėra pasiūlytas sprendimas, kuris užtikrintų verslo proceso dinamiškumą. Ši tema yra aktuali ne tik akademinėms bendruomenėms, bet ir versle, siekiant verslą padaryti konkurencingą. Dinaminių verslo procesų simuliacija kol kas nėra plačiai taikoma versle, todėl yra poreikis tobulinti simuliavimo įrankius, kurie padėtų optimizuoti verslo procesus, priimti teisingus sprendimus ir įvertinti galimas alternatyvas.

1.5. Tyrimo metodika

Analitinėje darbo dalyje, atliekant susijusios mokslinės literatūros analizę susijusią su dinaminių verslo procesų simuliacija ir simuliacijos dinamiškumą lemiančiais veiksniais: verslo taisyklėmis, įvykiais, yra naudojama bibliotekinio tyrimo ir lyginamosios analizės metodai. Analizuojami rinkoje esantys verslo procesų simuliacijai skirti įrankiai, efektyvūs taisyklių valdymo ir vykdymo metodai, jų galimybės, architektūra. Taip pat analizuojami verslo taisyklių aprašymo būdai. Remiantis analize yra siūlomas dinaminių verslo procesų simuliacijos būdas, leisiantis efektyviau simuliuoti dinaminius verslo procesus, naudojant efektyvius taisyklių valdymo ir vykdymo metodus, bei taisyklių aprašymo būdus. Pasiūlytas būdas įgyvendinamas ir yra atliekami jo efektyvumą nusakantis eksperimentinis tyrimas.

1.6. Mokslinė darbo vertė

Nagrinėjama tema šiuo metu yra aktuali ir iki galo neišnagrinėta, todėl šiame darbe atlikti darbai, eksperimentai bus naudingi tolimesniems darbams, susijusiems su dinaminių verslo procesų simuliacija.

1.7. Darbo struktūra

Įvade iškeliama problema, suformuojami darbo tikslai ir uždaviniai, aptariamas temos aktualumas ir naujumas. Pristatoma darbo struktūra.

Antrajame skyriuje yra atliekama su darbo tema susijusių mokslinių darbų literatūros analizė, analizuojant dalykinės srities sąvokas. Taip pat yra apžvelgiami verslo taisyklių aprašymo būdai, taisyklių valdymo ir vykdymo sistemos, joms keliami reikalavimai ir architektūra.

Trečiajame skyriuje yra atliekamas dinaminių verslo procesų simuliacijos įrankio patobulinimui pasiūlyto būdo projektavimas ir įgyvendinimas, prieš tai pateikiant reikalavimus ir siūlomo būdo koncepciją.

Ketvirtajame skyriuje yra pateikiamas siūlomo metodo efektyvumui ir teisingumui patikrinti reikalavimai eksperimentiniam tyrimui, bei atliktų eksperimentų rezultatai.

Darbo rezultatuose pateikiami darbo rezultatai

Išvadose ir pasiūlymuose pateikiamos darbo išvados bei pasiūlymai tolimesniems darbams, kurie gali būti toliau vystomi remiantis šiuo darbu.

Diskusijoje apžvelgiamos grėsmės siūlomo sprendimo validumui.

2. ANALITINĖ DALIS

Analitinės dalies tikslas yra atlikti su darbo tema susijusios literatūros analizę, susipažinti su dinaminius verslo procesus apibūdinančiomis sąvokomis. Taip pat išnagrinėti sprendžiamą problemą, susipažinti su galimais problemos sprendimų būdais ir suprasti, kokių sprendimų trūksta. Analitinėje dalyje bus nagrinėjami šaltiniai, įvairios publikacijos, moksliniai straipsniai, konferencijų darbai ir panašaus pobūdžio moksliniai leidiniai.

2.1. Verslo procesai

2.1.1. Dinaminiai verslo procesai

Verslo procesas yra rinkinys veiklų ar verslo procesų, kurių rezultatas yra organizacijos užsibrėžtas tikslas [1]. Verslo procesas yra nuoseklus veiklų išdėstymas, kuris turi pradžią ir pabaigą, aiškiai apibrėžtas įvestis ir išvestis. Pagal literatūros šaltinį [2], verslo procesas:

1. Turi tikslą;
2. Turi įvesties duomenis;
3. Turi išeigą;
4. Naudoja išteklius;
5. Turi veiklas, kurios vykdomos numatyta tvarka;
6. Gali įgyvendinti ne vieną organizacijos tikslą;
7. Sukuria vertę klientui.

Dinaminiai verslo procesai yra vienas iš verslo procesų tipų. Galima išskirti trijų tipų verslo procesus: statiniai, lankstūs ir dinaminiai verslo procesai. Literatūros šaltinyje [3], statiniai verslo procesai apibrėžiami kaip nekintančios formos procesai arba kurių pakeitimai vyksta per ilgą laiką. Pagal literatūros šaltinį [4] lankstūs verslo procesai, tai tokie procesai, kurie sugeba per priimtiną laiką ir su mažais ištekliais pasikeisti. Tokių procesų vykdymo metu sistema geba nuspręsti, kaip geriausia keisti vykdymo eigą, kad būtų pasiektas optimalus rezultatas. Remiantis literatūros šaltinyje pateikta medžiaga [3], dinaminiai verslo procesai yra sudaryti iš daug skirtingų veiklų ir procesų, kurie sunkiai gali būti iš anksto aprašyti algoritmu, nes verslo proceso eigoje priimamų sprendimų iš anksto numatyti negalima. Taigi dinaminių verslo procesų dinamiškumas priklauso nuo to, kaip greitai verslo procesai gali prisitaikyti prie nuolatos kintančios verslo aplinkos.

Gebėjimą verslo procesą adaptuoti prie kintančios aplinkos apibūdina šios sąvokos: dinamiškumas, lankstumas, judrumas ir prisitaikymas. Dažniausiai šios sąvokos apibūdinant dinaminius verslo procesus yra naudojamos kaip sinonimai. Remiantis literatūra [5], judrumas ir lankstumas apibūdina sistemos gebėjimą greitai prisitaikyti prie sistemos aplinkos pokyčių.

Labiausiai lanksčius verslo procesus, remiantis literatūros šaltiniu [6], galime laikyti tokius, kurie yra prisitaikantys. Tai reiškia, kad procesai gali kisti reaguojant į pakitusias aplinkos sąlygas, jų vykdymo metu, atsižvelgiant į įvykusių verslo procesų veiklų rezultatus, kurie gali veikti toliau vyksiančius verslo procesus. Verslo procesų dinamiškumas apibūdinamas kaip gebėjimas procesų vykdymo metu reaguoti į pasikeitusias tiek vidines, tiek išorines sistemos sąlygas, atsižvelgiant į klientų individualius poreikius [7]. Sistemos vykdomų procesų pokyčiai turi būti atliekami jų vykdymo metu taip, kad pokyčiai netūrėtų neigiamos įtakos vykdomo proceso esmei arba laukiamai proceso baigčiai.

Remiantis šaltiniais [8] dinaminiams verslo procesams gali būti suformuluoti tokie reikalavimai:

1. Dinaminiai verslo procesai turi reaguoti į bet kurį verslo procesą sudarančio komponento pokytį;
2. Veiklų sekos turi būti formuojamos vykdymo metu ir neturi būti iš anksto nustatytos;
3. Dinaminiai verslo procesai turi reaguoti į pokyčius, keičiantis proceso kontekstui;
4. Konteksto pasikeitimo trukmė turi būti daug trumpesnė nei viso proceso trukmė;
5. Proceso pokyčiai gali būti inicijuoti bet kurio proceso veikloje dalyvaujančio aktoriaus, bet kuriuo metu, su labai maža proceso atsako trukme į inicijuotus pokyčius, lyginant su viso proceso trukme.

2.1.2. Verslo procesų simuliacija

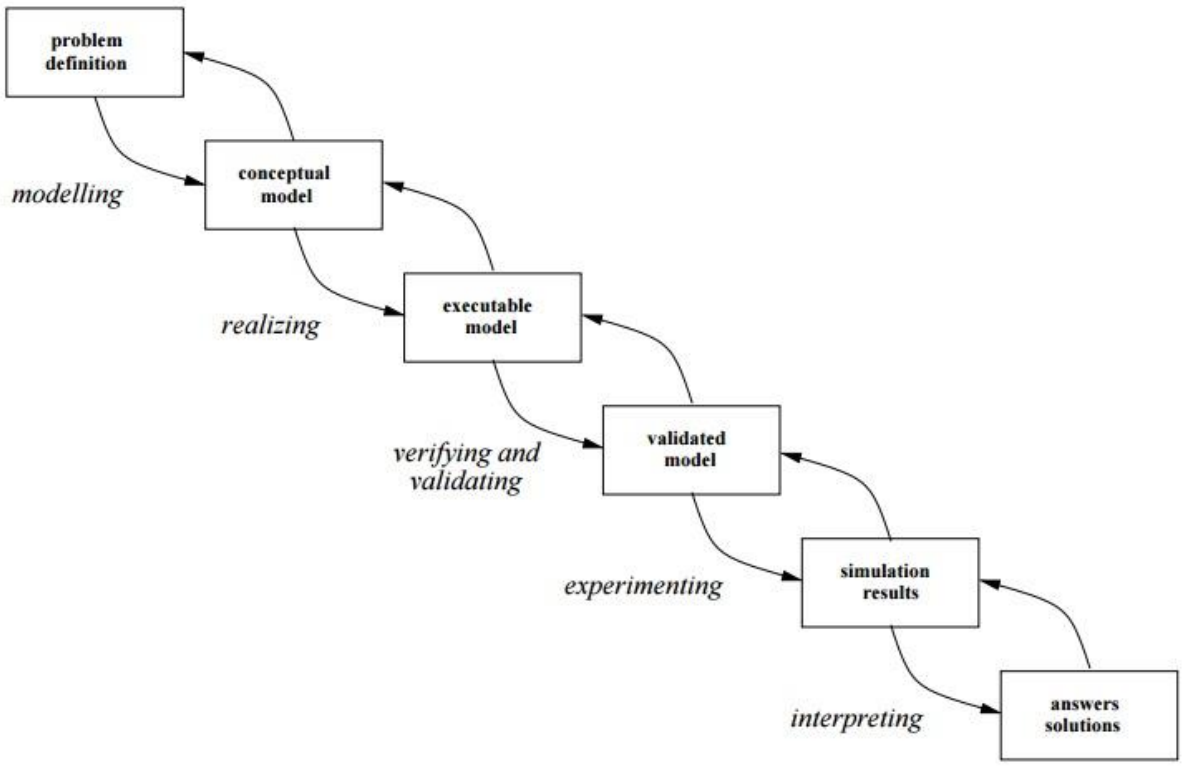
Remiantis literatūros šaltiniu [9], simuliacija – realaus gyvenimo tikros arba tikėtinos situacijos, sąlygos arba įvykio imitavimas tam, kad būtų surasta priežastis dėl ko praeityje įvyko tam tikras įvykis (nelaimės atveju) arba tam, kad būtų prognozuotos būsimos pasekmes. Verslo procesų simuliacija padeda suprasti: kaip pasiekti užsibrėžtus veiklos tikslus, kaip optimizuoti išteklių naudojimą, kokie galimi poveikiai, pakeitus veiklos funkcinius pokyčius.

Analizuojant skirtumus tarp dinaminių verslo procesų simuliacijos metodų, galima remtis verslo procesų dinamiškumo lygiais. Remiantis šaltiniu [10], yra įvardijami trys lygiai:

- Pirmajame žemiausiame lygmenyje yra naudojami sprendimų taškai, kuriuose žmogus ar automatizuota sistema nusprendžia ką daryti toliau, remiantis apibrėžtomis taisyklėmis;
- Antrajame lygmenyje yra numatytos automatinės verslo procesų konfigūracijos. Tokios kaip galimybė pasirinkti alternatyvų šabloną veiklų vykdymui arba proceso metu keisti veiklų tvarką, jeigu sąlygos yra pakitusios;
- Aukščiausiame trečiajame lygmenyje dinamiškumas yra orientuotas į verslo proceso tikslą (angl. *goal-driven*), kuris gali būti keičiamas bet kuriuo verslo proceso vykdymo metu, atsižvelgiant į naujas sąlygas ir klientų poreikius.

Daugiausia šiomis dienomis naudojamų metodų ir įrankių yra įgyvendinę žemiausią dinamiškumo lygį.

Vienas iš verslo procesų simuliacijos pritaikymo tikslų yra verslo procesų optimizavimas. Verslo procesų simuliacija gali būti vykdoma programinėmis priemonėmis. Simuliacijos metu yra atkuriama verslo proceso eiga, daromi įvairūs pakeitimai modeliuose ir taisyklėse, siekiant įvertinti jų įtaką ir pokyčių pasekmes [11]. Toks pakeitimų įvertinimo būdas yra žymiai pigesnis negu juos vertinant atliekant realius sprendimus. Norint simuliuoti verslo procesą reikia turėti ne vien gerą simuliacijai skirtą programinę įrangą, bet ir tikslų verslo proceso modelį, kuriame būtų apibrėžta verslo proceso darbo srauto logika ir visi verslo procese naudojami ištekliai. Todėl tai reikalauja tinkamos problemos analizės, modelio sukūrimo, atliktos simuliacijos ir rezultatų interpretavimo, kaip parodyta 1 pav. pateiktame simuliacijos proceso gyvavimo ciklas, kuris paremtas krioklio (angl. *waterfall*) būdu. Kiekvienas iš ciklą sudarančių žingsnių gali prasidėti ir nepasibaigus prieš jį buvusiam žingsniui [12].



1 pav. Verslo proceso simuliacijos gyvavimo ciklas *[12]*

Trumpai aptarsime verslo proceso simuliacijos gyvavimo ciklą sudarančius žingsnius [12]:

1. **Problemos apibrėžimas** (angl. *Problem definition*). Apibrėžiant problemą yra sukonkretinami simuliacijos tikslai, sritis, simuliacijai skirto modelio struktūra. Apibrėžiant problemą yra iškeliamas klausimas, į kurį simuliacija turi atsakyti;

2. **Koncepcinis modelis** (angl. *Conceptual model*). Koncepciniame modelyje nurodomos objektų klasės ir sąryšiai tarp šių objektų ir objektų nustatymai. Koncepciniame modelyje yra atskiriami kiekybiniai ir kokybiniai nustatymai. Koncepciniame modelyje dažniausiai yra detalizuojama problema;
3. **Modelio vykdymas** (angl. *Executable model*). Šiame žingsnyje yra įgyvendinamas simuliacijos modelis susiejant šį modelį su koncepciniu modeliu, atsižvelgiant į tai kokia simuliacijai skirta priemonė yra pasirinkta;
4. **Modelio tikrinimas** (angl. *Validated model*). Šiame žingsnyje yra tikrinamas simuliacijos modelis. Yra tikrinamos programavimo klaidos ir modelio nustatymai. Tikrinimo metu yra atliekami testavimai, kurie yra skirti ištestuoti simuliacijos modelio kritines vietas, nepalankiomis sąlygomis. Atlikus patikrinimą, modelis tampa patvirtintu modeliu;
5. **Simuliacijos rezultatai** (angl. *Simulation results*). Šiame žingsnyje yra atliekami eksperimentinės simuliacijos su skirtingais atsitiktiniais duomenimis tam, kad patikrinti ar gauti rezultatai panašūs;
6. **Atsakymų parengimas** (angl. *Answers solutions*). Šiame žingsnyje gauti simuliacijų rezultatai yra interpretuojami ir yra gaunami atsakymai į problemos apibrėžimo metu išsikeltus klausimus.

2.1.3. Įvykiais grindžiamų verslo procesų simuliacijos modeliai

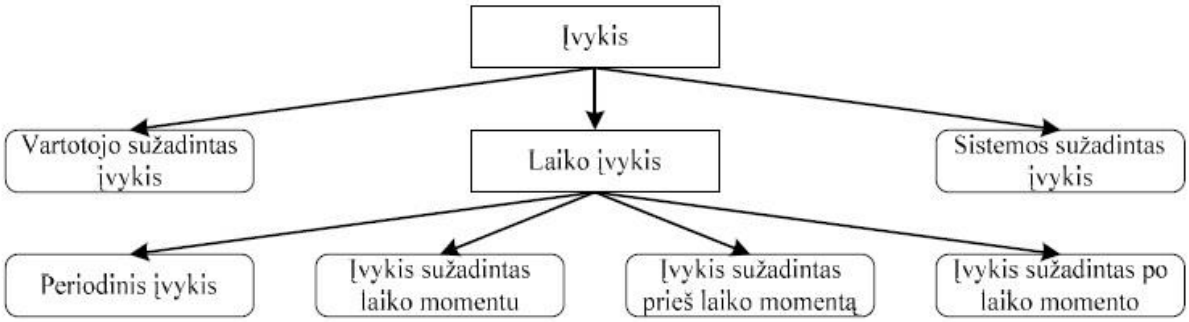
Įvykis – tai veiksmas, kuris nutinka proceso eigoje. Įvykiai nulemia vykdomų procesų eigą. Įvykiais galima laikyti veiklos pradžią ar tam tikrų sistemos vidinių ar išorinių rodiklių pokyčius. Įvykiais grindžiamų verslo procesų simuliacija leidžia verslo procesus sudarančias veiklas susieti silpnais ryšiais, taip padidinant simuliacijai skirtų modelių plečiamumo galimybes ir verslo procesų simuliacijos modelių pritaikymą prie besikeičiančių verslo poreikių. Remiantis šaltiniu [13] įvykiais pagrįstos verslo procesų simuliavimo sistemos yra tokios sistemos, kuriose pranešimų siuntėjai, reaguodami į vidinius ar išorinius įvykius, sukuria pranešimus ir siunčia juos pranešimų gavėjams, kurie tuos pranešimus apdoroja. Pranešimų siuntėjais ir gavėjais gali būti išorinės informacinės sistemos, simuliuojamuose verslo procesuose dalyvaujantys veikėjai, taip pat verslo procesą sudarančios veiklos.

Pagrindinis skirtumas tarp įvykiais grindžiamos verslo procesų simuliacijos sistemos ir įprastos, kurioje veiklos bendradarbiauja kviesdamos viena kitos metodus ir grąžindami reikšmes, yra tas, kad įvykiais grindžiamoje sistemoje veiklos nežino apie viena kitos egzistavimą [14]. Simuliuojamų verslo procesų veikloms nebūtina žinoti, kokius metodus realizuoja kiekviena iš veiklų. Pakanka nusiųsti pranešimą veiklai, kuri apdoros pranešimą ir įvykdys tam tikrus metodus [15]. Norint

realizuoti įvykiais grindžiamą verslo procesų simuliaciją, turi būti realizuotas įvykių apdorojimo mechanizmas, kuris atliktų šias pagrindines funkcijas:

1. Leistų veiklai užregistruoti įvykį;
2. Leistų kitoms veikloms matyti, kokie įvykiai yra užregistruoti;
3. Pateiktų efektyvų įvykių apdorojimo mechanizmą, t.y. nebūtina užregistruoti ir siųsti pranešimus apie įvykį, jei nėra nė vieno komponento, kuriam tas įvykis būtų aktualus

Laiko įvykis gali įvykdyti tam tikrais laiko momentais. Pavyzdžiui, verslo procesų simuliacijos sistema gali užregistruoti įvykį, kuris turi būti sužadintas tam tikru laiko momentu, prieš arba po tam tikro laiko momento. Toks įvykis yra sugeneruojamas vieną kartą. Taip pat tie patys įvykiai gali būti generuojami ciklišKai tam tikrais laiko momentais. Remiantis tyrimais [16], įvykius galima suskaidyti į keturias rūšis: įvykstantys atsitiktinai, įvykstančius prieš arba po tam tikro laiko momento ir periodiniai. Įvykių tipai pavaizduoti 2 pav.



2 pav. Įvykiais grindžiamų verslo procesų simuliacijos modelių įvykių tipai *[16]*

Verslo proceso simuliacijos modelio vartotojo sužadintas įvykis – įvykis, kurį sužadina vartotojo atliekami veiksmai su sistema. Tai gali būti mygtuko paspaudimas simuliacijos programinėje įrangoje.

Simuliacijos sistemos sužadintas įvykis – įvykis, kurį sužadina verslo procesų simuliacijos sistema, vykdydama veiklos logiką. Pavyzdžiui: įsigijus fondo akcijų už nustatytą sumą, sistema sugeneruoja įvykį.

Įvykis, sužadintas tam tikru laiko momentu – įvykis, kuris yra sugeneruojamas sistemos tam tikru laiko momentu, nepriklausomai nuo simuliacijos sistemos vartotojo vykdomų veiksmų ar sistemos vykdomos veiklos logikos.

Įvykis, sužadintas prieš laiko momentą arbo po laiko momento – įvykis, kuris yra sugeneruojamas simuliacijos sistemos prieš arbo po tam tikro laiko momento. Tokie įvykiai dažnai pasitaiko simuliacijos sistemose.

Periodinis įvykis – įvykis, kuris generuojamas ciklišKai tam tikrais laiko momentais. Pavyzdžiui: verslo proceso simuliacijos sistemoje, daug kartų per dieną sugeneruojamas įvykis, kuris suaktyvuoją veiklą – patikrinti fondų akcijų kainų pokyčius [16].

2.2. Verslo taisyklės

2.2.1. Verslo taisyklės verslo procesuose

Verslo taisyklių taikymas dinaminių verslo procesų simuliacijose yra sparčiai populiarėjantis. Nors taisyklėmis paremtos sistemos egzistuoja jau ilgą laiką, verslo procesuose pradėtos naudotis neseniai. Verslo taisyklės yra naudojamos kaip priemonė atskirti verslo logiką nuo veiklos procesų ir programų [17]. Verslo procesų simuliacijai naudojamos verslo taisyklės turi būti tokios, kad būtų suprantamos verslui ir būtų interpretuojamos taisyklių valdymo ir vykdymo sistemų. Remiantis skirtingais šaltiniais, verslo taisyklės yra apibrėžiamos kaip:

- Elgsenų arba sąlygų, kurios turi būti įvykdytos versle, paskelbimas [18];
- Verslo žinių visuma [19];
- Teiginys, kuris nusako arba apriboja kokį nors verslo aspektą, deklaruoja verslo struktūrą, kontroliuoja verslo procesus arba daro jiems kitokią įtaką [20].

Taisyklėms klasifikuoti gali būti pasitelkiami įvairiausi kriterijai: struktūros tipas, semantinės savybės, paskirtis, šaltinis, realizacijos tipas, ir kt. Šiuo metu yra sukurta gausybė įvairiausių verslo taisyklių klasifikavimo būdų. Toliau aptarsime GUIDE projekte pateikiamus verslo taisyklių tipus.

Pagal GUIDE projektą yra skiriami 3 verslo taisyklių tipai [21]:

1. **Struktūrinės taisyklė** (angl. *Structural Assertion*) – apibrėžta sąvoka ar fakto sakinys. Šiomis taisyklėmis pateikiami statiniai verslo veiklos aspektai, išreiškiantys žinomus dalykus arba kaip žinomi dalykai siejasi tarpusavyje. Struktūrinės taisyklės skirstomos į terminus, turinčius konkrečią reikšmę verslo veikloje ir faktus, apibrėžiančius ryšį tarp dviejų ar daugiau terminų:
    1.1. **Terminas** – žodis vartojamas versle, o jo apibrėžimas nusako šio žodžio vartojimo prasmę. Terminai ir jų apibrėžimai yra pagrindinis duomenų modelio komponentas pirmame Zachmano karkaso stulpelyje verslo savininko požiūrio eilutėje. Toliau jie paverčiami esybių tipais antroje architekto požiūrio eilutėje. Terminai dažnai pavirsta duomenų modelio esybių tipų, atributų vardais;
    1.2. **Faktas** – terminų susiejimas, taip sudarant prasmingus konceptus. Faktai yra tai, kas vaizduojama duomenų modelyje – tarpusavyje susiję esybių tipai, atributai priskirti esybėms tipams ir apibrėžti esybių tipų potipiai. Faktas taip pat yra pirmo Zachmano karkaso stulpelio duomenų modelyje. Faktas skirstomas į Unarinį, Binarinį ir N-narį faktus:
        1.2.1. **Unarinis** (vienanaris) fakto tipas – sąryšis nusakantis kad terminas atlieka tam tikrą rolę. Pvz.: užsakymas priimtas;

1.2.2. **Binarinis** fakto tipas – sąryšis nusakantis ryšį tarp dviejų terminų. Pvz.: vadybininkas teikia užsakymą;

1.2.3. **N-naris** fakto tipas – sąryšis nusakantis ryšį tarp daugiau nei dviejų terminų. Pvz.: vadybininkas teikia užsakymą pasirinktam tiekėjui.

2. **Veiksmo taisyklė** (angl. *Action Assertion*) – apribojimo sakinys ar sąlyga, ribojanti arba valdanti verslo veiksmus. Veiksmų taisyklėmis išreiškiami dinaminiai verslo aspektai. Šios taisyklės apibrėžia apribojimus rezultatams, kurie gali būti gaunami atlikus veiksmus. Veiksmų taisyklės taip pat skirstomos į produkcines, darnos, autorizacijos, veiksmą kontroliuojančias ir veiksmą įtakojančias taisykles:
   2.1. **Produkcinės taisyklės -** dar kitaip žinomos kaip sąlygų ir veiksmų taisyklės. Šiose taisyklėse nėra nurodomos aplinkybės, kurioms esant yra tikrinamos taisyklėje aprašytos sąlygos. Pvz.: Jeigu klientas perka prekių daugiau nei už 100 Eu , tuomet jam taikyti 5 % nuolaidą;
   2.2. **Darnos taisyklės –** tokios taisyklės, kurios visuomet galioja. Pvz.: kiekvienas projektas turi tūrėti projektų vadovą;
   2.3. **Autorizacijos taisyklės** – tokios taisyklės kurios tikrina dviejų priklausomų parametrų sąryšius. Pvz.: tik vyriausiasis vadybininkas gali suteikti klientui auksinio kliento statusą;
   2.4. **Veiksmą kontroliuojančios taisyklės** – tokios taisyklės kurios nusako kas turi būti padaryta. Pvz.: kiekvieno ketvirčio pabaigoje vadybininkas privalo pateikti ketvirtinę ataskaitą;
   2.5. **Veiksmą įtakojančios taisyklės** – tokios taisyklės nusako kas gali būti padaryta. Pvz.: kiekvienam užsakymui gali būti taikoma iki 30 % nuolaidą.
3. **Išvedimo taisyklė** (angl. *Derivation*) – algoritmas ar sakinys, skirtas žinioms apie verslo veiklą išvesti iš kitų verslo veiklos žinių. Išvedimo taisyklės skaidomos į matematinius skaičiavimus ir logines išvadas:
   3.1. **Matematinis išvedimas –** tai matematiniai skaičiavimai skirti išgauti naujoms reikšmėms, naudojant tam tikras formules ar algoritmus. Pvz.: atlyginimų skaičiavimas, nuolaidų skaičiavimas ir t.t,;
   3.2. **Išvados** – loginės išvados išvedamos remiantis esamais faktais. Išvadų reikšmės paprastai būna dviejų reikšmių (tiesa ir melas), bet gali būti ir daugiau reikšmių. Pvz.: jeigu kliento reitingas yra didesnis nei 50 %, tuomet kliento statusas yra auksinis klientas.

Atlikę veiklos ir verslo taisyklių tipų analizę, sužinojome, kad dalykinės srities terminai ir faktai apibrėžia verslo procesus, o taisyklės apriboja. Kartu šie trys aspektai: terminai, faktai ir taisyklės, apibūdina verslo taisykles.

Verslo taisyklės turi prasmę ir yra suprantamos, jeigu tenkina šiuos 8 specifinius kriterijus [22]:

- **Aktualumas/pagrįstumas** – taisyklė turi būti aktuali nagrinėjamoje srityje;
- **Atomiškumas** – taisyklė turi aprašyti vieną taisyklę, kurios jau nebeišeina suskaidyti į detalesnes verslo taisykles;
- **Deklaratyvumas** – taisyklė turi apibrėžti esybių būsenas, spendimus arba skaičiavimus, o ne procesus, kurie naudojami jiems įgyvendinti;
- **Aiškumas** – taisyklė, numatytoje jos panaudojimo srityje, turi būti vienareikšmiškai suprantama;
- **Patikimumas** – taisyklė turi būti sukurta kompetentingų dalykinės srities specialistų, remiantis dalykinės srities nuostatomis;
- **Autentiškumas** – taisyklių reikšmės skirtingose aprašymo formose turi būti vienodos;
- **Nuspėjamumas** – taisyklė turi grąžinti tas pačias išvadas nepaisant to, kas ir kaip ją inicijavo.

Tinkamai suformuluota verslo taisyklė, esant bet kokiai verslo situacijai jo dalyviams turi rodyti, ką reikia veikti kitame verslo proceso etape. Taip pat verslo taisykles modeliuoja dažniausiai verslo analitikai, naudodami natūralią kalbą arba grafines notacijas. Todėl verslo taisyklių modeliavimas neturi pareikalauti didelių programavimo žinių.

2.2.2. Verslo taisyklių aprašymo būdai

Taisyklėms apibrėžti yra nemažai būdų, metodų ir kalbų, tačiau nėra daug standartų taisyklėms aprašyti. Kokį būdą, metodą arba kalbą pasirinksime taisyklei aprašyti arba apibrėžti, priklauso nuo to, kokiame verslo sistemos abstrakcijos lygmenyje ji bus naudojama. Pagal literatūros šaltinį [23], yra siūloma skirti verslo, informacinės ir programų sistemų abstrakcijos lygmenis. Toliau nagrinėsime taisyklių aprašymo būdus informacinės sistemos abstrakcijos lygmenyje. Informacinės sistemos lygmens taisyklė – teiginys, užrašytas tam tikra taisyklių kalba, apibrėžiantis informacijos apdorojimo taisykles.

2.2.2.1. Taisyklių vaizdavimas struktūrizuota anglų kalba

SBVR specifikacija apibrėžia veiklos žodyno (angl. *business vocabulary*), veiklos faktų tipų (angl. *business fact types*) ir veiklos taisyklių (angl. *business rules*) semantiką bei XMI schemą veiklos žodynų ir veiklos taisyklių apsikeitimui tarp organizacijų ar kompiuterizuotų sistemų. SBVR

pateikia būdą užfiksuoti taisykles natūralia kalba ir išreikšti jas formalia logika taip, kad jas būtų galima apdoroti automatizuotai. SBVR numatyta išreikšti veiklos žodyną ir veiklos taisykles, ir specifikuoti veiklos reikalavimus informacinėms sistemoms natūralia kalba. Iš visų OMG modeliavimo kalbų SBVR turi didžiausią išraiškingumą. Taisyklėms apibrėžti OMG siūlo vartoti struktūrizuotą anglų kalbą [24].

SBVR standartas deklaruoja, kad veiklos taisyklės yra sudaromos faktų pagrindu, o pastarieji, atitinkamai, yra sudaromi veiklos terminų pagrindu. Vadovaujantis šiuo principu, SBVR specifikacija (modelis) yra pradedamas kurti nuo veiklos žodyno, kurį sudaro veiklos terminai ir faktai, o tuomet pereinama prie pačių veiklos taisyklių, kuriose naudojami veiklos žodyne specifikuoti konceptai.

SBVR specifikacijoje pateiktas vienas galimų anglų kalbos vartojimo būdų. Šaltinyje [24], siūloma naudoti keturis šriftų stilius, kurių vaizdavimas yra formalus:

- Terminas (angl. *term*) vartojamas konceptams – daiktavardžiams žymėti;
- Pavadinimas (angl. *name*) vartojamas individualiems konceptams – pavadinimams žymėti. Pavadinimai turi būti išreikšti tikriniais daiktavardžiais (pvz.: Vilnius). Skaitinių reikšmių pavadinimai žymimi naudojant tą patį stilių (pvz.: 25);
- *Veiksmažodis* (angl. *verb*) vartojamas faktų tipams. Faktu vadinamas ryšys tarp dviejų ir daugiau konceptų (pvz.: *yra*, *apibrėžia*, *reiškia*);
- Raktinis žodis (angl. *keyword*) žymi jungiamuosius žodžius – kvantorius (kiekybės operatorius), naudojamus sakiniams sudaryti (pvz.: kiekvienas, yra privaloma, bent vienas).

3 pav. pateikiamas verslo taisyklės pavyzdys, užrašytas struktūrizuota anglų kalba. Lietuviškai ši taisyklė skamba taip: „Privaloma, kad kiekviena nuomojama mašina priklausytų tik vienam filialui“.

3 pav. Struktūruotą anglų kalba parašyta verslo taisyklė *[24]*

SBVR standartą galima laikyti UML metamodelio plėtiniu, skirtu veiklos taisyklėms pavaizduoti. SBVR standartu galima specifikuoti labai įvairius veiklos aspektus. Juo naudojantis galima analizuoti reikalavimus sistemai, išreikštus natūraliąja kalba, išskirti pagrindinius veiklos konceptus. SBVR žodynais galima sujungti skirtingas kalbos bendruomenes, surinkti ir paaiškint įvairų žargoną, kurį naudoja tam tikros veiklos atstovai (mechanikai, informatikai, gydytojai ir pan.).

2.2.2.2. Objektų ribojimų kalba (OCL)

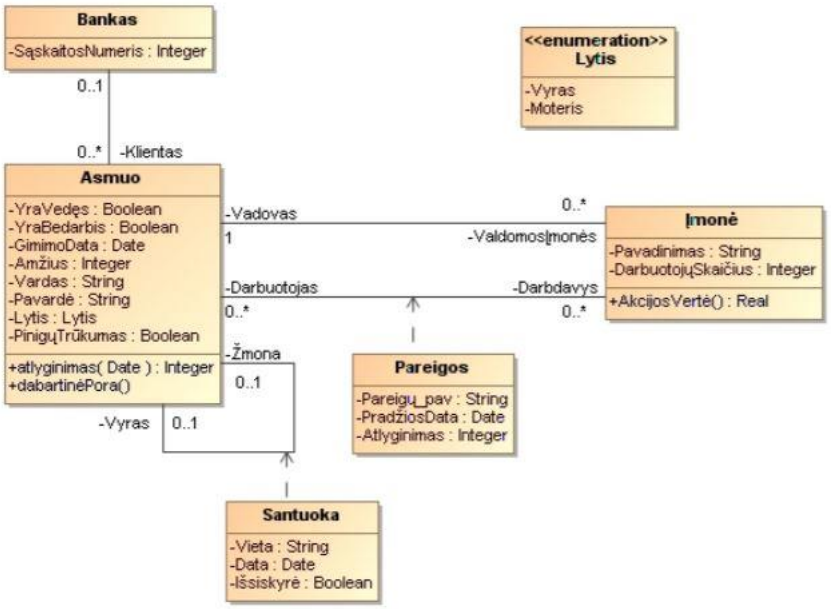
OCL (angl. *Object Constraint Language*) yra formali logikos pagrindu sukurta kalba, neturinti grafinės notacijos. Ši kalba yra skirta išreikšti tokius objektų elgsenos apribojimus, kuriuos neįmanoma išreikšti naudojant tik UML (angl. *Unified Modeling Language*) konstrukcijas. OCL yra tipų kalba, t. y. kiekvienas OCL ribojimas priklauso tam tikram tipui ir turi tenkinti kalbos tipų atitikimo taisykles, o tai reiškia, kad negalima sulyginti sveiko tipo skaičiaus su simbolių eilute. OCL išraiškos gali būti naudojamos apibrėžti operacijas arba veiksmus, kuriuos įvykdžius, pasikeis sistemos būsena. OCL specifikacija nurodo šiuos panaudojimo atvejus [25]:

- Taikyti kaip užklausų kalbą;
- Invariantų nustatymui klasėse ir kaip tipas klasių modelyje;
- Tipų invariantų nustatymui stereotipuose;
- Aprašyti prieš ir po sąlygas operacijoms ir metodams;
- Aprašyti saugikliams pereinant tarp būsenų ar veiklų;
- Pranešimų ir veiksmų užduočių apibrėžimui;
- Aprašyti operacijų apribojimus;
- Aprašyti išvedimo taisykles atributams bet kokiai išraiškai UML modelyje.

OCL kalba yra deklaratyvi, kurioje išraiška tiesiog aprašo kas, o ne kaip turi būti padaryta. Tai garantuoja, kad OCL išraiškos neturi šalutinio poveikio, t.y. OCL nekeičia sistemos būsenos. OCL pagal apibrėžimą yra tipizuota apribojimų kalba. Ja galima užrašyti logines išraiškas, kurios susiejamos su kontekstu: klasifikatoriumi, savybe ar operacija. OCL išraiškų tipai skirstomi į 4 kategorijas:

- **Inv** (invariantai) – šios išraiškos turi būti tikrinamos ir privalo būti teisingos visą objekto gyvavimo ciklą;
- **Pre** išraiškos – tikrinamos prieš vykdant metodą;
- **Post** išraiškos – tikrinamos po metodo įvykdymo;
- **Body** išraiškos – skirtos aprašyti metodo vykdomoms operacijoms ir yra naudojamos užklausoms rašyti.

OCL sintaksėje turinio (angl. *Context*) reikšminis žodis pateikia išraiškos turinį. Reikšminiai žodžiai **inv**, **pre** ir **post** nurodo ribojimų tipą, o tikroji OCL išraiška eina po dvitaškio: (**context** TipoVardas **inv**:). Toliau naudosime Paveiksle 4 pateiktą klasių diagramos pavyzdį, aprašydami pagrindines OCL kalbos savybes.



4 pav. Klasių diagrama, sudaryta remiantis OCL specifikacija *[25]*

OCL išraiškoje žodis pats (angl. *self*) naudojamas nurodyti konteksto egzempliorių. Pvz.: jeigu kontekstas yra asmuo, tuomet **self** nurodo asmens egzempliorių. Visos OCL išraiškos, kurios išreškia invariantus, turi būti loginio (angl. *boolean*) tipo. Pvz.: įmonės tipo kontekste, apibrėžiame invariantą, kad darbuotojų skaičius visada turi būti didesnis negu 50:

context Įmonė **inv**:

self.DarbuotojųSkaičius > 50

OCL išraiška gali būti prieš ir po sąlygų dalis. Yra pridedama **pre** ir **post** prieš tam tikrą prieš ir po sąlygą. Šio tipo OCL ribojimo struktūra atrodo taip:

context TipoVardas::OperacijosVardas(parametras1:tipas1,…):GrąžinamasTipas

pre: parametras1 > ...

post: **result** = …

Žodis rezultatas (angl. *result*) nurodo operacijos rezultatą. Parametrų vardai pvz.: parametras1, gali būti naudojami OCL išraiškoje. Nagrinėjamai klasių diagramai galime užrašyti tokį ribojimą:

context Asmuo::atlyginimas(D: Date):Integer

post: **result** = 5000

OCL išraiška gali būti naudojama pradinei arba išvedamai atributo reikšmei žymėti. Tai gali būti atlikta naudojant tokią sintaksę:

context TipoVardas::AtributoVardas:Tipas

init: --tam tikra išraiška, nurodanti pradinę reikšmę

context TipoVardas::AsociacijosVardas:Tipas

derive: --tam tikra išraiška, nurodanti išvedimo reikšmę

Pradinė ir išvesta išraiška gali būti naudojamos kartu viename kontekste. Pavyzdžiui:

context Asmuo::atlyginimas:Integer

init: Tėvai.atlyginimas->sum()*1%

derive: if Asmuo.Age < 18

then Tėvai.atlyginimas->sum()*1%

else Pareigos.Atlyginimas

endif

UML, kartu su OCL, tenkina raiškos, tikslumo ir viena reikšmiškumo reikalavimus, ši OCL kalba turi ir trūkumų. Pagrindinis jų yra tas, kad OCL neturi grafinės notacijos, todėl užrašyti taisykles nėra paprasta – reikia būti pakankamai susipažinusiam su OCL sintakse ir naudojimo taisyklėmis.

2.2.2.3. Sprendimų lentelės

Sprendimų lentelės – struktūros, leidžiančios vizualiai apjungti taisykles, kurių sąlygos tikrinamos naudojant tuos pačius atributus, o taisyklių veiksmų dalyje atliekami tie patys veiksmai [26]. Lentelių metodas problemoms spręsti buvo ištobulintas XX a. 7-ajame dešimtmetyje General Electric ir kt. organizacijų. Oficialūs sprendimų lentelių komponentai ir terminologija buvo standartizuoti bendromis daugelio organizacijų, įskaitant CODASYL (angl. *Conference on Data Systems Languages*), pastangomis 1962 m. Nuo tada bazinis sprendimo lentelių formatas iš esmės nepakito [27].

Sprendimų lentelės sudaromos taip, kad skirtingos situacijos stulpeliuose būtų pateikiamos vieną kartą. Toks sprendimų lentelių sudarymo būdas užtikrina nagrinėjamų situacijų pilnumą ir unikalumą: nei viena sąlygų kombinacija lentelėje nėra pateikiama daugiau negu vieną kartą. Lentelėje 1 pateikta sprendimų lentelė, kurią sudaro 4 dalys:

- Sąlygos, apie kurias turima informacija yra naudojama sprendimo priėmimui (pateikiamos viršutinėje kairėje lentelės dalyje);
- Sąlygų galimų reikšmių aibės (sąlygų kombinacijos) (pateikiamos viršutinėje dešinėje lentelės dalyje);
- Veiksmai, apibrėžiantys sprendimo priėmimo proceso rezultatą (pateikiami apatinėje kairėje lentelės dalyje);
- Veiksmų galimų reikšmių aibės (sąlygų kombinacijas atitinkančios veiksmų kombinacijos) (pateikiamos apatinėje dešinėje lentelės dalyje).

1 lentelė. Sprendimų lentelės dalys *[27]*

Sąlygos	Galimų sąlygų reikšmių aibių kombinacijos
Veiksmai	Galimų veiksmų reikšmių aibių kombinacijos

Kiekvienas lentelės stulpelis (sprendimo stulpelis) nurodo veiksmą, kuris turi/neturi būti atliekamas konkrečiai sąlygų reikšmių kombinacijai. Remiantis [28] šaltiniu, sprendimų lentelės gali būti naudojamos, kai yra tenkinamos šios sąlygos:

- Veiklos taisyklės yra panašios, t. y. jos nurodomos tam pačiam subjektui, turi visiškai vienodas vertinamas sąlygas bei turi lygiavertį (bet ne identišką) poveikį;
- Kiekviena vertinama sąlyga turi baigtinį reikšmių ar reikšmių intervalų skaičių;
- Pateikiant skirtingas reikšmes ar reikšmių intervalus, rezultatas negali būti apskaičiuojamas pagal formulę. (Jei rezultatas gali būti apskaičiuojamas, naudojant taisyklę ar taisyklių aibę unifikuotai formulei gauti, tuomet toks metodas yra geresnis.)

Žemiau pateiksime sprendimų lentelės, naudojamos poilsio dienų apskaičiavimui, pavyzdį [29].

2 lentelė. Poilsio dienų skyrimo sprendimų lentelė *[29]*

Amžius	<18	18<=ir<45		45<=ir<60		>=60
Darbo stažas	-	<25	>=25	<40	>=40	-
Skiriamos 22 poilsio dienos	x	x	x	x	x	x
Skiriamos 5 papildomos poilsio dienos	x	-	-	-	-	-
Skiriamos 2 papildomos poilsio dienos	-	-	x	x	x	x
Skiriamos 3 papildomos poilsio dienos	-	-	-	-	x	x

Lentelėje naudojami žymėjimai: (x) – nurodytas veiksmas yra vykdomas, (-) – nurodytas veiksmas nėra vykdomas.

Lentelėje 2 poilsio dienos skiriamos remiantis šiomis taisyklėmis:

1. Poilsio dienų skaičius priklauso nuo darbuotojo amžiaus ir darbo stažo;
2. Darbuotojas gauna mažiausiai 22 poilsio dienas;
3. Papildomos poilsio dienos skiriamos pagal pateiktus kriterijus:
   3.1. Darbuotojai jaunesni nei 18 metų gauna 5 dienas;
   3.2. Darbuotojams turintiems nemažiau nei 25 metų darbo stažą, skiriamos 2 dienos;
   3.3. Darbuotojams, kurių amžius yra lygus arba didesnis už 45 metus, yra skiriamos 2 dienos nepriklausomai nuo darbo stažo;
   3.4. Darbuotojams, turintiems mažiausiai 40 metų darbo stažą arba 60 metų amžiaus ir vyresniems, skiriamos 3 papildomos dienos.

Sudarant sprendimų lenteles, veiklos taisyklių sistemose išvengiama šių anomalijų [30]: nesuderinamumo, kuris atsiranda tuomet, kai egzistuoja tomis pačiomis sąlygomis paremtos, bet skirtingus rezultatus pateikiančios taisyklės; pertekliškumo (angl. *Redundancy*), kuris sistemoje dažniausiai nesukelia klaidų, tačiau kenkia sistemos efektyvumui; nepilnumo (angl. *Incompleteness*), kuris sistemoje atsiranda dėl nenaudojamų atributų reikšmių ar kombinacijų, negalimų veiksmų, trūkstamų žinių.

2.3. Verslo taisyklių valdymo ir vykdymo sistemos

Verslo taisyklės yra tam tikro tipo konkrečios dalykinės srities žinios. Versle dalykinės srities žinių savininkai yra verslininkai, kurie formuluoja verslo taisykles. Verslo taisyklės gali būti vykdomos automatizuotai arba darbuotojų rankiniu būdu. Tačiau norint žinias panaudoti programų sistemose, reikia jas perkelti iš verslo sistemos lygmens į programų sistemos lygį. Verslo taisyklių perkėlimas į programų sistemos lygį, padeda šiuolaikiniam verslui egzistuoti nuolat besikeičiančioje verslo aplinkoje, kurioje keičiasi ir verslo taisyklės. Iš to iškyla problema, kaip verslo taisykles, suformuluotas verslo sistemos lygmenyje, kuo greičiau ir efektyviau perkelti į programų sistemų lygmenį. Šiai problemai spręsti yra kuriamos darbui su verslo taisyklėmis skirtos verslo taisyklių valdymo sistemos. Šių sistemų naudojimas padeda spręsti toliau išvardintų problemų scenarijus [31]:

- Verslo taisyklės gali būti gaunamos ne iš vienos vietos. Galimas atvejis, kai verslo taisyklės skleidžiamos per kelias geografiškai nutolusias vietas arba per kelis verslo politikos dokumentus;
- Jei verslo taisyklės giliai paslėptos taikomojoje programoje, nukenčia verslo taisyklių matomumas;
- IT personalui dažnai trūksta verslo srities žinių verslo taisyklių pakeitimų prasmei suprasti. Verslininkai, kurie turi šių žinių, neturi pakankamos kompetencijos, kad darytu informacinėje sistemoje tokius pakeitimus. Atsiranda vėlavimu tarp verslo politikos sukūrimo ir įgyvendinimo organizacijos informacinėje sistemoje. Tokiu vėlavimų kaštai yra dideli, jei įmonė veikia dinamiškoje rinkoje.

2.3.1. Reikalavimai priemonėms skirtoms darbui su verslo taisyklėmis

1998 metais ECOOP (*European Conference on Object-Oriented Programming*) konferencijoje vykusio seminaro metu buvo suformuluoti reikalavimai, kuriuos turi tenkinti darbui su verslo taisyklėmis skirtos priemonės ir sistemos. Toliau pateiksime seminaro metu suformuluotus reikalavimus [32]:

1. **Centralizuota saugykla**. Verslo taisyklės turi būti realizuotos vienoje centralizuotoje saugykloje. Saugykla gali būti centralizuota ne fiziniame lygmenyje, bet loginiame – jį centralizuoti būtina. Jei jį atitinka reikalavimą, vienoje saugykloje yra saugomas pilnas taisyklių rinkinys;
2. **Prisitaikymo galimybė**. Esamas taisyklių rinkinys turi būti lengvai keičiamas – taisyklės modifikuojamos, trinamos ar įrašomos naujos;
3. **Konfliktų aptikimas.** Turi būti galimybė aptikti konfliktuojančias taisykles;

4. **Kodo derinimo vykdymo metu** (angl. *debugging*) **palaikymas.** Galimybė surasti vykdymo metu atsirandančias klaidas;
5. **Deklaratyvioji kalba.** Verslo taisyklės turi būti išreiškiamos deklaratyviąja kalba;
6. **Specialūs verslo taisyklių peržiūros įrankiai.** Tai verslo taisyklių peržiūrai skirti įrankiai, kurie taip pat gali riboti priėjimą prie verslo taisyklių priklausomai nuo jų tipo bei prisijungusio vartotojo tipo;
7. **Efektyvumas.** Tai sunkiai pamatuojamas rodiklis, bet seminaro dalyviai sutarė, kad efektyvumas turėtų neišeiti už „sveiko proto ribų“;
8. **Aiški verslo taisyklių išraiška.** Šis reikalavimas reiškia, kad verslo taisykles palaikanti sistema ar priemonė verslo taisykles modeliuotų, saugotų, vykdytų ir įvardintų aiškiai kaip verslo taisykles, o ne funkcinius ar duomenų reikalavimus;
9. **Formalus pagrindas.** Verslo taisyklės turi būti vaizduojamos formaliai. Šis reikalavimas gali būti taikomas ir dviem etapais: pirminiame verslo taisyklių modeliavimo etape gali būti naudojamas ir neformalus būdas, bet vėliau verslo taisyklės turi būti išreikštos aiškia, formalia kalba. Taip pat pastebėta, kad geriau yra rinktis kurią nors standartinę kalbą – KIF (angl. *Knowledge Interchange Format*), OCL (angl. *Object Constraint Language*), UML (angl. *Unified Modeling Language*) [33] [34] ir pan;
10. **Verslo taisyklių identifikavimas ir ištraukimas iš realaus pasaulio atvaizdavimų.** Žinių ištraukimas (angl. *elicitation*) yra sudėtingas procesas, kurio metu neaiškiai išreikštos žinios, sukauptos kaip žmonių patirtis bei saugomos įvairiais pavidalais (kaip dokumentai, duomenys ir pan.) yra identifikuojamos, įvardijamos ir vaizduojamos išreiškiant aiškiai. Vis dėlto visiškas šio proceso automatizavimas yra labai gražus, bet kol kas nepasiekiamas tikslas;
11. **Vientisumo ir nuoseklumo palaikymas.** Norint išsaugoti verslo taisyklių rinkinio vientisumą ir nuoseklumą, būtina užtikrinti, kad tik tam skirtomis priemonėmis būtų galima rinkinį peržiūrėti ir modifikuoti. Taip pat reikia užtikrinti, kad verslo taisyklių rinkinį pasiektų tik tokias teises turintys vartotojai;
12. **Vaizdavimas.** Verslo taisyklės gali būti vaizduojamos įvairiai. Pageidautina, kad vaizdavimo būdas priklausytų nuo vartotojo tipo, lygio ar pasirinkimo. Vaizdavimas nebūtinai turi būti formalus. Pavyzdžiui, galima taisykles vaizduoti lentelėmis, grafais ar medžio tipo struktūromis;
13. **Atvirumas.** Turi būti galimybė įrašyti naujas taisykles bei modifikuoti esamas;
14. **Samprotavimo mechanizmas.** Turi būti galimybė ne tik saugoti taisykles, bet ir gauti naujus faktus ar taisykles;

15. **Skirtingų lygių taisyklių palaikymas.** Verslo taisyklės gali būti kelių lygių – priklausomos nuo dalykinės srities, nuo konkrečios informacinės ar programų sistemos ir pan. Tokiam reikalavimui tenkinti turi būti palaikoma verslo taisyklių klasifikavimo ar grupavimo galimybė;
16. **Integravimas su jau egzistuojančiomis sistemomis.** Svarbu integruoti verslo taisyklėms skirtą priemonę su jau egzistuojančiomis sistemomis, jų kūrimo metodais, technologijomis ir notacijomis;

2.3.2. Verslo taisyklių valdymo ir vykdymo sistemų savybės

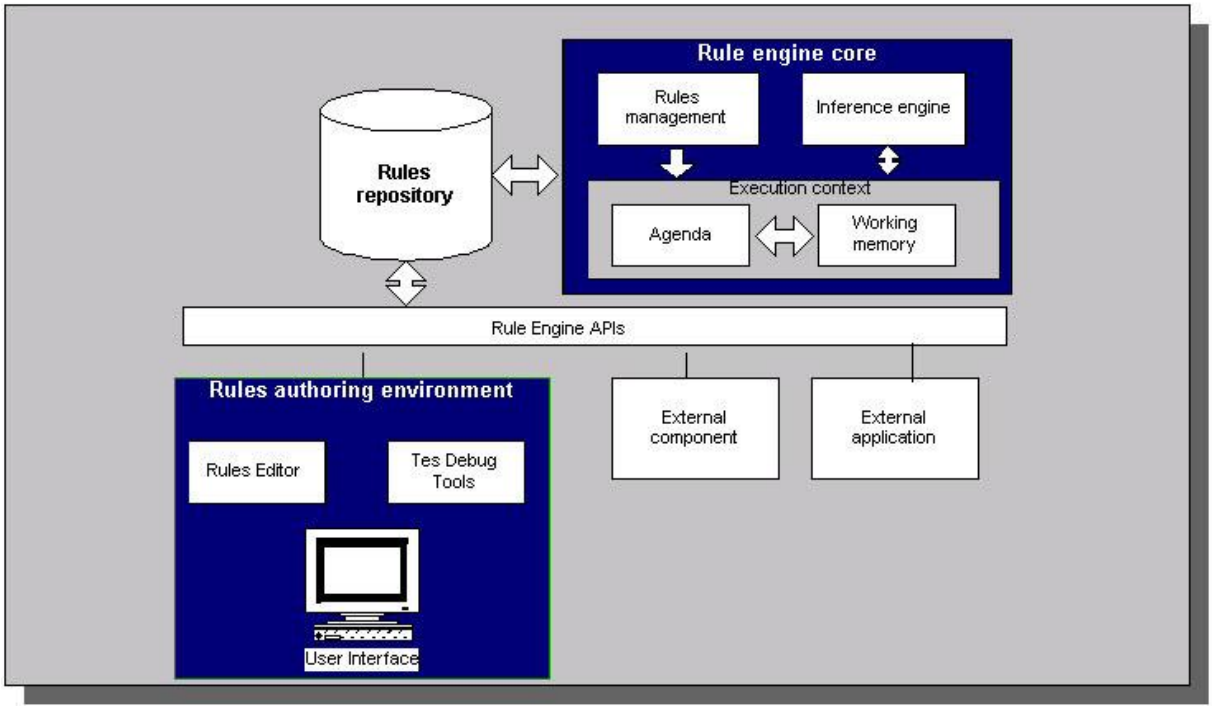
Vienas iš didžiausių verslo taisyklių valdymo sistemų naudojimo privalumų yra verslo taisyklių atskyrimas nuo taikomosios programos kodo. Verslo logikos atskyrimas nuo taikomosios programos logikos suteikia galimybes verslo žmonėms keisti verslo logiką be IT personalo pagalbos. Verslo taisyklių valdymo sistemos leidžia organizacijai pateikti taisykles panašiu į natūralią kalbą formatu užuot naudojus sudėtingą taikomosios programos kodą.

Gerai verslo taisyklių valdymo sistemai būdingos šios savybės [31]:

- optimizuota greitaveika bei minimalus atminties resursai;
- galimybė atvaizduoti verslo taisykles natūralia kalba, siekiant, kad visų tipų vartotojai jas geriau suprastu;
- galimybė atskirti taisykles nuo taikomosios programos logikos ir saugoti jas centralizuotoje taisyklių saugykloje;
- galimybė kurti bei keisti taisykles nepriklausomai nuo IT personalo;
- taisyklių versijos kontrolė, skirta taisyklių pakeitimams laike kontroliuoti;
- prieigos kontrolė, leidžianti verslo politika keisti tik sankcionuotai;
- galimybė pasiekti taisykles nuotoliniu būdu, kad jos būtų keičiamos realiu laiku;
- paprasta integracija į egzistuojančią organizacijos IT struktūra;
- paprastai naudojama, plečiama ir lanksti API.
- Verslo taisyklių valdymo sistemų naudojimas suteikia šiuos privalumus:
- vartotojai gali aprašyti taisykles panašia į natūralią kalbą, tuo padidindami taisyklių skaitomumą;
- vartotojai gali kurti, testuoti, keisti ir integruoti verslo taisykles su informacinėmis sistemomis;
- verslo taisykles galima administruoti, saugoti, valdyti atskirai nuo informacinės sistemos;
- verslo taisyklės gali būti taikomos tuo pačiu metu keliose informacinėse sistemose;
- verslo taisykles galima greitai pakeisti ir pakeitimus įdiegti į informacinę sistemą.

2.3.3. Verslo taisyklių valdymo ir vykdymo sistemų architektūra

Paveikslas 1 pateikiame taisyklių valdymo sistemos architektūrą.



5 pav. Taisyklių valdymo sistemos architektūra *[35]*

Iš 5 pav. matome, kad sistemą sudaro trys pagrindiniai elementai: taisyklių saugykla, taisyklių variklio branduolys, taisyklių kūrimo aplinka [35].

Taisyklių saugykloje yra saugomos verslo veiklos žinios – verslo taisyklės, kurios yra susiejamos su atitinkamais verslo procesais. Taisyklių saugykla naudojasi likusios sistemos sudedamosios dalys: taisyklių variklio branduolys ir taisyklių kūrimo aplinka.

Sekantis sistemos elementas yra taisyklių variklio branduolys, kurį sudaro:

- **Taisyklių valdymo modulis** (angl. *Rules management module*). Šis modulis turi prieiga prie taisyklių saugyklos, iš kurios paima taisykles ir perduoda taisyklių vykdymo tvarkyklei;
- **Taisyklių parinkimo valdiklis** (angl. *Agenda*). Šis modulis atsakingas už taisyklių prioretizavimą ir perdavimą išvadų varikliui;
- **Darbinė atmintis** (angl. *Working memory*). Šis modulis saugo faktų reikšmės, kurie yra naudojami atitinkamų taisyklių;
- **Konteksto vykdymo modulis** (angl. *Execution context module*). Šis modulis užtikrina išvadų varikliui kontekstą, kurį sudaro taisyklių parinkimo valdiklio parinktos vykdymui taisyklės ir darbinėje atmintyje saugomų faktų reikšmės. Šiame modulyje gali būti saugomi keli kontekstai;

- **Išvadų variklis** (angl. *Inference engine*). Šis modulis yra atsakingas už taisyklių vykdymą. Taisyklės yra vykdomos tol, kol bus įvykdytos visos taisyklės, kurios yra susijusios su darbinėje atmintyje saugomais faktais.

Trečiasis sistemos elementas yra taisyklių kūrimo aplinka. Tai aplinka su klientui skirta grafine sąsaja, kurioje klientas gali į taisyklių saugyklą įtraukti naujas taisykles. Taisyklės dažniausiai yra įvedinėjamos aukšto lygio taisyklių kalba, kuri yra panaši į natūralią kalbą. Taisyklių įvedimui palengvinti yra integruoti taisyklių tvarkyklės, kurios tikrina įvedamų taisyklių sintaksę. Taip pat yra galimybė įvestas taisykles simuliuoti.

Aprašytą taisyklių valdymo sistemos architektūrą naudoja tokie komerciniai produktai kaip: Blaze Advisor, ILOG JRules, Jess.

2.4. Taisyklių valdymo ir vykdymo sistemų analizė

Verslo taisyklių valdymo ir vykdymo sistemų paskirtis yra užtikrinti greitesnį ir efektyvesni taisyklių kūrimo, modifikavimo bei vykdymo procesą. Verslo taisyklių valdymo sistemos leidžia kiekvieną verslo taisyklę užrašyti kaip nepriklausomą teiginį, kuris aprašo tam tikrą verslo aspektą. Taip aprašytos taisyklės veikia kaip autonominiai vienetai, atskirtos nuo taikomosios programos ar verslo procesų simuliacijos įrankių. Toliau atliksime taisyklių valdymo ir vykdymo sistemų analizę, remiantis šaltinyje [31] aprašytais kriterijais.

Verslo taisyklių išraiška. Skirtingos verslo taisyklių valdymo sistemos turi skirtingus verslo taisyklių vaizdavimo metodus. Kai kurios sistemos naudoja natūralią kalbą, kai taisyklės yra užrašomos, naudojant supaprastintą anglų kalbą (pvz.: IF – THEN taisykles). Kitos sistemos verslo taisyklėms užrašyti naudoja ir tokius taisyklių aprašymo metodus kaip:

- sprendimų lentelės;
- sprendimų medžiai;
- taisyklių šablonai.

Grafinė taisyklių kūrimo aplinka. Grafinė taisyklių kūrimo aplinka suteikia galimybę taisykles rašyti ne tik programuotojams bet ir verslo atstovams. Kūrimo aplinka gali būti kaip atskira sistemos paprogramė arba serverio tipo programa. Ne visos taisyklių valdymo ir vykdymo sistemos turi grafinę taisyklių kūrimo aplinką, bet dažniausiai turi taisyklių redagavimui skirtus tekstinius redaktorius.

Apsauga ir versijų kontrolė. Apsauga ir versijų kontrolė priklauso nuo to, kaip sistemoje organizuota saugykla ir versijų valdymas. Kai kurie įrankiai palaiko sudėtingą prieigos teisių administravimą su įeigos bei išeigos kontrole, versijų bei registracijos žurnalų funkcijas. Bendruoju atveju sistemos leidžia nustatyti vartotojų vaidmenis bei prieigos prie taisyklių teises, priklausomai nuo vartotojo vaidmens arba grupės sistemoje.

Taisyklių testavimas ir koregavimas. Taisyklių testavimas ir koregavimas leidžia kūrėjams tikrinti potencialius verslo taisyklių konfliktus bei jų tarpusavio priklausomybes. Skirtingos taisyklių valdymo ir vykdymo sistemos siūlo skirtingus sprendimus šiam funkcionalumui užtikrinti – nuo standartinių taisyklių redagavimo langų – tekstinių redaktorių iki specialių grafinių taisyklių rinkinių atvaizdavimo priemonių.

Kūrimo aplinka. Kūrimo aplinka priklauso nuo taisyklių valdymo ir vykdymo sistemos programavimo kalbos. Dažniausiai yra naudojamos kelios programavimo kalbos skirtingiems sistemos moduliams programuoti: „C“, „Java“ ir t. t. Be to kai kurios sistemos palaiko atvirus standartus (pvz.: J2EE), kurie yra paremti moduliniais komponentais.

Integracija su kitomis sistemomis. Vienas iš svarbiausių taisyklių valdymo ir vykdymo sistemų privalumų yra galimybė integruotis su kitomis sistemomis pvz.: WEB tarnybomis, duomenų bazių valdymo sistemomis.

3 lentelė. Taisyklių valdymo ir vykdymo sistemų palyginimas

Kriterijus / Sistema	Atviro kodo?	Verslo taisyklių išraiška	Apsauga ir versijų kontrolė	Grafinė taisyklių kūrimo aplinka	Testavimo ir derinimo įrankiai	Kūrimo aplinka	Integracija su kitomis sistemomis
ILOG Rules	Ne	*BAL (angl. *Business Action Language*) * IF-THEN-ELSE *TRL (angl. *Technical Rule Language*)	Yra	Yra	Yra	*C++ *Java	*COM *COBRA *IBM MQSeries
Blaze Advisor	Ne	*SRL (angl. *Structured Rule Language*) *Sprendimų lentelės *Sprendimų medžiai	Yra	Yra	Yra	*Java	*XML *COM *COBRA *MTS *J2EE
Open Rules	Taip	*Sprendimų lentelės	Nėra	Yra	Yra	*Java	*MS Exel *Eclipse IDE
Drools	Taip/Ne	*WHEN-THEN *Sprendimų lentelės *Sprenfimų medžiai	Yra	Yra	Yra	*Java	*Eclipse IDE
Nrules	Taip	*WHEN-THEN *DSL (angl. Domain Specific Language)	Nėra	Nėra	Yra	*.Net	Nėra

„ILOG Rules“ turi keletą produktų. „ILOG Riles“ naudoja „C++” klasių bibliotekas, o „ILOG Jrules“ „Java“ klasių bibliotekas. „ILOG“ taisyklės yra „ECA“ (angl. *Event – Condition – Action*) tipo taisyklės. „ILOG“ taisyklės sudarytos iš trijų dalių: antraštės, sąlygos ir veiksmo. Taisyklės antraštė nurodo taisyklės pavadinimą, prioritetą ir paketo pavadinimą. Sąlygos prasideda reikšminiu žodžiu „WHEN“, po jo užrašomos sąlygos atskirtos kabliataškiais. Veiksmo dalys prasideda reikšminiu žodžiu „THEN“, po jo aprašomas veiksmas, kuris yra vykdomas, kai visos sąlygos yra patenkintos. Taisyklės prioritetas kontroliuoja taisyklių paleidimo eilės tvarką. Paketo pavadinimas susieja taisyklę su taisyklių paketu. Naudojant paketus susietos taisyklės yra grupuojamos [31].

„Blaze Advisor“ yra taisyklių valdymo ir vykdymo sistema, kuri turi daug įrankių, skirtų kurti sistemas, kuriose proceso veiklos metu sprendimai yra priimami vadovaujantis aprašytomis taisyklėmis. Pagrindinis naudojamas verslo taisyklių vaizdavimo metodas – struktūrinė taisyklių kalba „SRL“ (angl. *Structured Rule Language*), kuri nėra tinkama verslo žmonėms. „Blaze Advisor“ leidžia naudoti sprendimų lenteles, kurios yra naudojamos aprašyti sąlygas ir veiksmus verslo taisyklėse. „Blaze Advisor“ turi integruotą aplinką, skirta lengvam veiklos taisyklių kūrimui. Ši aplinka yra tiesiogiai orientuota į vartotojus ir aprūpina žemo lygio programavimo bei grafine aplinka, skirta taisyklėms rašyti, redaguoti, peržiūrėti ir testuoti [36].

„Open Rules“ tai viena iš mūsų nagrinėjamų taisyklių valdymo ir vykdymo sistemų, kuri yra atviro kodo. Ši sistema leidžia verslo analitikams įkelti ir koreguoti verslo taisykles iš „Microsoft Excel“, „Microsoft Word“, „OpenOffice“ arba „Google Docs“ dokumentų. Šios sistemos taisyklės yra rašomos ir koreguojamos „Java“ programavimo kalba. Taip pat yra galimybė rašyti taisykles naudojant sprendimų lenteles. Yra galimybė „Open Rules“ kartu naudoti su „Eclipse IDE“ atviro kodo integruota kūrimo aplinka. Ši aplinka leidžia tvarkyti taisykles. „Eclipse“ turi „Java“ kodui gerai išvystytą redaktorių. „Open Rules“ palaiko šiuos taisyklių tipus: loginio įvedimo, ryšių įvedimo, apribojimų, veiksmų, tikrinimo [37].

„Drools“ yra projektas, sukurtas „JBoos“ ir „Red Hat“ kompanijų. Šis projektas apima verslo taisyklių vykdymo variklį ir siūlo keletą įrankių skirtų koreguoti ir valdyti verslo taisykles. Dools taisyklių variklis apdoroja taisykles, kurios yra formuojamos „WHEN-THEN“ formatu, kurios yra išreikštos „DRL“ (angl. *Drools Language*) kalba, kuri yra labai panaši į „Java“ programavimo kalbą. „Drools“ yra viena iš taisyklių mašinų, kuri turi grafinį redaktorių, kuris suteikia galimybę grafiškai modeliuoti taisykles [38].

„NRules“ tai atviro kodo, nemokama taisyklių valdymo ir vykdymo sistema, kuri šiuo metu yra palaikoma ir taisoma jos kūrėjų. Ši sistema leidžia vykdyti „WHEN-THEN“ formato produkcines taisykles. Ši sistema yra pritaikyta „.NET“ platformai. Deja ši sistema neturi apsaugos ir versijų kontrolės, taip pat grafinės taisyklių kūrimo ir valdymo aplinkos. Tačiau ši sistema turi testavimui skirtą „Debug“ režimą.

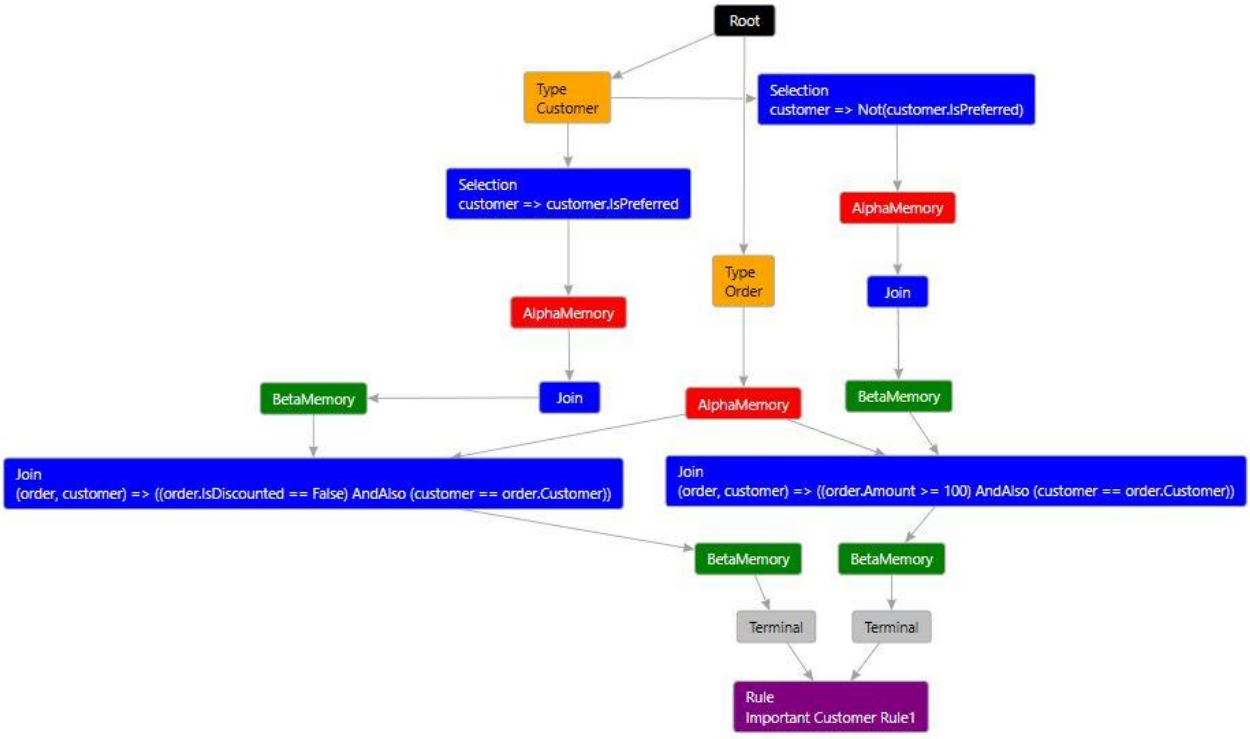
2.5. „RETE“ algoritmas

„RETE“ algoritmas – efektyvus taisyklių šablonų aptikimo algoritmas, naudojamas produkcinių taisyklių sistemose. Šį algortimą sukūrė dr. Charlis Forgis ir pirmą kartą 1978 m. aprašė savo daktaro disertacijoje [39]. Šodis „RETE“ išvertus iš lotynų kalbos reiškia tinklas. Šis algoritmas yra naudojamas taisyklių vykdymo ir valdymo sistemose, kurios priklauso tokioms kompanijoms kaip „IBM“ ir „Red Hat“. Šios kompanijos yra išleidusios skirtingas šio algoritmo versijas: „IBM“ kompanijai priklauso „RetePlus“, o „Red Hat“ priklauso „ReteOO“ algoritmas. Toliau pateiksime pagrindines „RETE“ algoritmo versijas:

- **„RETE II“** – 1980 m. pristatyta oficiali algoritmo versija, kuri pakeitė pirminę „RETE“ algoritmo versiją. Ši versija pasižymėjo geresne greitaveika kompleksinėse sistemose. Taip pat į šį algoritmą buvo įtrauktas atvirkštinės grandinės (angl. *Backward chaining*) metodas. Šis metodas tai yra vienas iš produkcinių taisyklių sistemų vykdymo metodų, kuris yra orientuotas į tikslą, kai iš anksto yra žinoma tikslas, kurį taisyklių vykdymo sistema stengiasi pasiekti;
- **„RETE III“** – 2000 m. pristatyta nauja algoritmo versija, kuri yra 3 kartus greitesnė už prieš tai buvusią versiją. Ši algortimo versija yra „Blaze Advisor“ sistemos dalis;
- **„RETE NT“** – 2010 m. pristatyta versija, kuri yra 500 kartų greitesnė už pirminę algoritmo versiją ir 10 kartų greitesnė nei „RETE II“ versija. Šiuo metu ši algoritmo versija priklauso „Sparkling Logic“ kompanijai.

Remiantis šaltiniu [40], „RETE“ algoritmas gali būti suskaidytas į dvi dalis: taisyklių kompiliavimas ir jų vykdymas. Taisyklių kompiliavimo algoritmas aprašo kaip taisyklės esančios produkcinėje atmintyje (angl. *production memory*) yra keičiamos į produkcinį tinklą. Šis tinklas dar kitaip yra vadinamas diskriminaciniu tinklu (angl. *descrimination network*), kurį sudaro skirtingo tipo mazgai, kurie veikia kaip filtrai – praleidžia arba nepraleidžia darbinėje atmintyje (angl. *working memory*) esančius faktus. Remiantis „RETE“ tinklą sudaro: šaknies mazgas (angl. *Root node*), tipų mazgai (angl. *Type nodes*), alfa tinklas (angl. *Alpha network*), sujungimo mazgai (angl. *Join nodes*), beta tinklas (angl. *Beta network*), terminalų mazgai (angl. *Terminal nodes*) ir taisyklių mazgai (angl. *Rule node*). Alfa tinklas aprašo apribojimus skirtingų tipų faktų atributams, o beta tinklas aprašo sąlygas tarp skirtingų šablonų. „RETE“ tinklas prasideda vienu šakniniu mazgu, kuris yra sujungtas su visais tipų mzgais. Tipų mazgai yra atsakingi už skirtingų faktų tipų filtravimą, t.y. kiekvienas tipo mazgas praleidžia tik tam tikro tipo faktus. Alfa tinklas sudarytas iš alfa atrankos (angl. *Selection node*) ir alfa atminties (angl. *Aplha memory node*) mazgų. Alfa atrankos mazgas atsakingas už faktų atributų filtravimą. Kiekvienas alfa mazgas skirtas aprašyti sąlygai, skirtai vienam fakto atributui. Alfa atrankos mazgų sąlygas tenkinantys faktai patenka į alfa atminties mazgą, kuriame yra saugomi. Beta tinklą sudaro sujungimo (angl. *Join node*) ir beta atminties (angl. *Beta memory node*) mazgai.

Sujungimo mazgai turi du įėjimus, iš dešinės pusės alfa atminties mazgą, kuris saugo pavienius faktus, o iš kairės beta atminties mazgą, kuriame yra saugomi faktų rinkiniai. Faktų rinkiniai gali būti sudaryti iš dviejų skirtingų ir to pačio tipo faktų, kai taisyklės sąlyga sudaryta iš daugiau nei vienos sąlygos, naudojant loginius „ir/arba“ elementus. Taip beta atmintyje gali būti saugomi ne tik faktų rinkiniai, bet ir pavieniai faktai, kai taisyklės sąlyga sudaryta iš vieno alfa atrankos mazgo. Terminalo mazguose yra saugomos taisyklės sąlygą tenkinantys faktai. Taisyklės, kurios gali būti aktyvuojamos tenkinant vieną iš taisyklės sąlygą sudarančių sąlygų, kurių sąlygos sujungiamos „arba“ loginiais elementais, turi kelis terminalo mazgus. Į terminalo mazgą patekę faktai, su terminalo mazgu susietą taisyklę patalpina į aktyvacijos laukiančių taisyklių eilę (angl. *Agenda*). Taisyklių aktyvavimo eigai nustatyti yra naudojamos taisyklių vykdymo konfliktų sprendimų strategijos. Prie terminalo mazgo yra prijungta po vieną taisyklės mazgą, kuriame yra saugoma informacija apie taisyklėje aprašytus veiksmus ir taisyklės vykdymu susijusi informacija. Toliau 6 pav., pateiksime „NRules“ sistemos sugeneruotą „RETE“ tinklą, sudarytą iš vienos taisyklės, kuri skamba taip: „Jeigu klientas yra privilegijuotas ir turi užsakymą, kuriam nepritaikyta akcija arba klientas yra neprivilegijuotas ir turi užsakymą, kurio suma yra didesnė arba lygi 100, jo užsakymui yra taikoma akcija“.



6 pav. „NRules“ sistemos sugeneruotas „RETE“ tinklas sudarytas iš vienos taisyklės

Paveiksle pavaizduotą tinklą sudaro: kliento (angl. *Customer*) ir užsakymo (angl. *Order*) mazgai, nes taisyklės sąlygoje yra naudojami dviejų skirtingų tipų faktai. Taip pat tinklą sudaro du alfa tinklo atrankos mazgai, nes taisyklės sąlygoje yra naudojamos vieno iš kliento fakto atributų dvi

galimos vertės. Kadangi taisyklės sąlygą sudaro dvi sąlygos sujungtos „arba“ loginiu operatoriumi, kada esant vienai iš jų teisingai yra įvykdoma taisyklėje aprašyti veiksmai, yra naudojami du beta tinko sujungimo mazgai, kurie tikrina ar beta ir alfa atmintyse esantys faktai tenkina jose aprašytas sąlygas. Sėkmės atveju faktų rinkinys yra įrašomas į naują betą atminties mazgą, kuriame esantys faktai patenka į atitinkamą terminalo mazgą, kuris patalpina su juo susijusią taisyklę į aktyvacijos laukiančių taisyklių eilę.

Tinklo viršuje esančiuose tinklo mazguose faktai turi daug atitikimų, o leidžiantis tinklu į apačią, mazguose faktų atitikimų skaičius mažėja. Taip yra todėl, kad faktai, esantys darbinėje atmintyje, keliauja tinklu iš viršaus į apačią nuo vieno mazgo prie kito tik tada, kai tenkina mazguose esančias sąlygas. „RETE“ tinklas pasižymi tinklo mazgų dalijimosi savybe, kurios dėka skirtingos taisyklės, kurių sąlygos, ar dalis sąlygų naudoja tuos pačius tinklo mazgus, nes taisyklių sąlygas sudarantys faktų tipai ir faktų tipams ar jų atributams keliamos sąlygos sutampa. Tokiu būdu išvengiama besidubliuojančių tinklo mazgų.

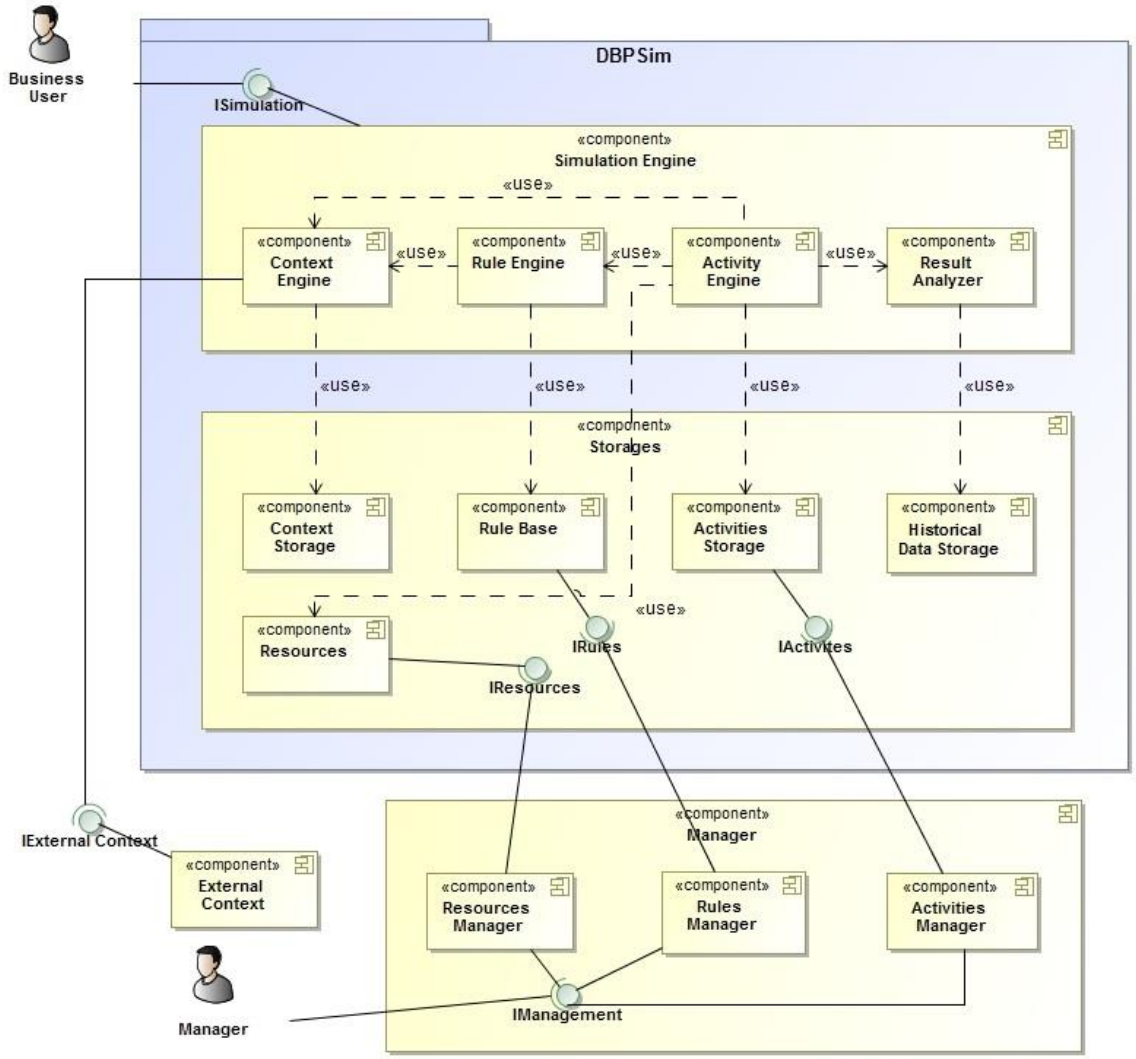
2.6. Analitinės dalies apibendrinimas

Verslo procesai yra natūraliai dinamiški ir yra nuolatos paveikiami dinamiškai kintančios aplinkos. Tam, kad būtų galima tinkamai šiuos procesus simuliuoti, vienas iš būdų yra naudoti verslo procesų simuliacijos būdus, kuriuose būtų naudojami taisyklių valdymo ir vykdymo mechanizmai. Todėl šiame skyriuje buvo išnagrinėtos su tema susijusios sąvokos, susipažinta su taisyklių aprašymo būdais, taisyklių valdymo ir vykdymo sistemoms keliamais reikalavimais, šių sistemų privalumais ir architektūra bei šiose sistemose naudojamu „RETE“ algoritmu. Taip pat remiantis pasirinktais kriterijais literatūriškai išanalizuotos 5-ios taisyklių valdymo ir vykdymo sistemos. Svarbiausi kriterijai: sistema atviro kodo, palaikomos produkcinės taisyklės ir kūrimo aplinka „.NET“. Šiuos kriterijus atitiko vienintelė „Nrules“ sistema. Todėl tolimesniame darbe šiai taisyklių valdymo ir vykdymo sistemai skirsime daugiausiai dėmesio.

3. PROJEKTINĖ DALIS

3.1. Esama situacija

„DBPSim“ yra VGTU Informacinių sistemos katedros vystomas nemokamas simuliacijos įrankis. Šis įrankis yra skirtas modeliuoti dinaminius procesus ir vėliau juos simuliuoti. Šiuo įrankiu galima rankiniu būdu kurti dinaminių VP modelius, kuriuose nurodomos veiklos, jų vykdymo sąlygos (taisyklės) bei atlikimo turinys, pavyzdžiui, konteksto keitimas, apdorojimas ir kai kurių išteklių tipų valdymas. Šis įrankis leidžia apsirašyti „ECA“ taisykles, kurios yra atskirtos nuo programinio kodo, todėl tai leidžia bet kuriuo simuliacijos metu pakeisti taisykles. Simuliacijos rezultatas gali būti pateikiamas tekstiniu bei grafiniu pavidalu. Naudotis šiuo įrankių nemokant programuoti yra sudėtinga [41]. Toliau remiantis literatūros šaltinyje [42] pateikta informacija, pateiksime „DBPSim“ įrankio architektūrą 7 pav.



7 pav. „DBPSim“ įrankio architektūra *[42]*

Toliau paaiškinsime įrankio architektūros sudedamąsias dalis [42]:

1. **Simuliacijos sąsaja** (angl. *Simulation interface*) – ši sąsaja yra atsakinga už duomenų ir valdymo komandų apsikeitimą tarp vartotojo ir simuliacijos variklio;
2. **Simuliacijos variklis** (angl. *Simulation engine*) – atsakingas už simuliacijos vykdymą. Simuliacijos variklį sudaro šie komponentai:
    a. **Veiklos variklis** (angl. *Activity engine*) – atsakingas už pasirinktos veiklos vykdymą;
    b. **Taisyklių variklis** (angl. *Rule engine*) – atsakingas už pasirinktų taisyklių tikrinimą, kurios atsakingos už esamą kontekstą. Veiklos variklis taisyklių variklį naudoja nustatant kurios veiklos bus toliau vykdomos;
    c. **Konteksto variklis** (angl. *Context engine*) – atsakingas už konteksto analizę. Taisyklių variklis naudoja konteksto variklį, norėdamas nustatyti, kurios taisyklės atitinka esamą kontekstą ir kurios tūrėtų būti naudojamos vykdant tolimesnę veiklą;
    d. **Rezultatų analizatorius** (angl. *Result analyzer*) – naudojamas analizuoti įvykdytų verslo procesų istorinius duomenis. Veiklos variklis naudoja rezultatų analizatorių, norėdamas patikrinti, ar pasirinkta vykdymui veikla nesukelia nepriimtinų veiksmų seką.
3. **Saugykla** (angl. *Storage*) – sudaro **taisyklių bazė** (angl. *Rule base*), **veiklų saugykla** (angl. *Activities storage*), **konteksto saugykla** (angl. *Context storage*), **istorinių duomenų saugykla** (angl. *Historical data storage*), **Resursai** (angl. *Resources*), kurios yra atsakingos už naudojamų taisyklių, veiklos, konteksto, istorinių duomenų ir išteklių naudojamų simuliacijos vykdymui saugojimą ir rezultatų analizę. Didelis dėmesys skiriamas istorinių duomenų saugyklai, kurioje saugomi duomenys yra naudingi norint atskirti sėkmingus ir nesėkmingus verslo procesų simuliacijos atvejus, kurie gali būti panaudoti kaip žinios naujoms simuliacijoms;
4. **Valdiklis** (angl. *Manager*) – sudarytas iš **resursų valdiklio** (angl. *Resources Manager*), **taisyklių valdiklio** (angl. *Rule Manager*), **veiklų valdiklio** (angl. *Activities Manager*), yra atsakingas už valdymą, sukuriant, ištrinant ir teikiant reikalingą informaciją pagal užklausas per valdymo sąsają;
5. **Išorinis kontekstas** (angl. *External context*) – jutiklis, naudojamas konteksto saugykloje, norint nustatyti išorinį kontekstą proceso simuliacijai;
6. **Resursai** (angl. *Resources*) – verslo procesų vykdymui naudojami ištekliai. Šie resursai naudojami vidinei konteksto būsenai nustatyti ir išsaugoti konteksto saugykloje. Taip pat išteklius naudoja aktyvumo variklis norint apdoroti pasirinktas veiklas.

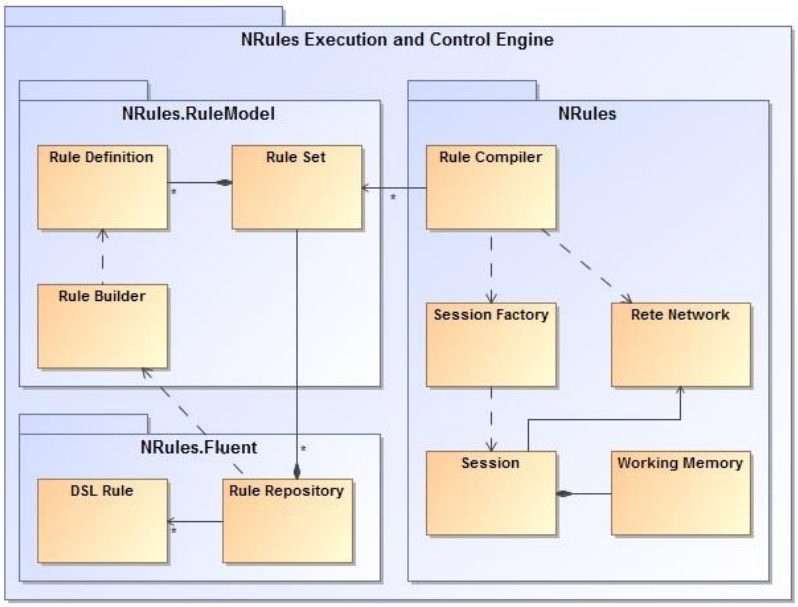
Šiame skyriuje yra siūloma taisyklių variklį ir taisyklių bazę iškelti už sistemos ribų ir pakeisti šiuos komponentus „NRules“ taisyklių valdymo ir vykdymo sistema. 3.5 skyriuje pasiūlysime būdą, kaip integruoti „NRules“ sistemą ir pateiksime metodą kaip išspręsti „NRules“ sistemos dinamiškumo problemas. Prieš tai sekančiame skyriuje atliksime „NRules“ sistemos analizę.

3.2. „NRules“ sistemos analizė

Viena iš pagrindinių siūlomo metodo architektūros sudedamųjų dalių yra „NRules“ taisyklių variklis, kuris turi užtikrinti dinaminį taisyklių vykdymą ir valdymą. Toliau nagrinėsime „NRules“ sistemos architektūrą, taisyklių variklio apdorojamų taisyklių savybes, taisyklių ir faktų užkrovimą, jų valdymą ir taisyklių vykdymą. Šios analizės tikslas – identifikuoti „NRules“ sistemos dinamiškumo problemas.

3.2.1. „NRules“ sistemos architektūra

Toliau pateiksime detalizuotą „NRules“ sistemos architektūros komponentų diagramą ir detalizuosime sistemą sudarančias dalis.



8 pav. „NRules“ taisyklių valdymo ir vykdymo sistemos architektūros komponentų diagrama

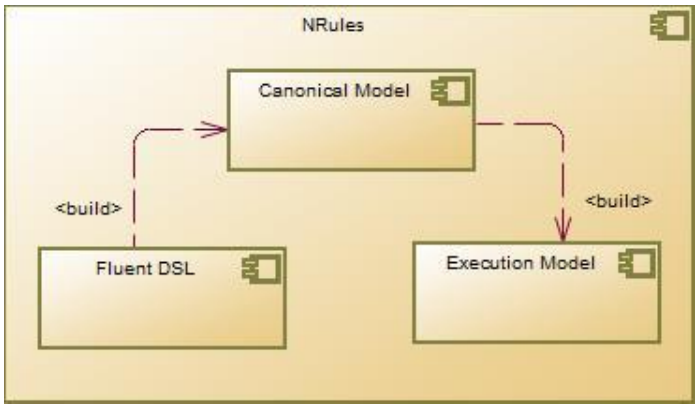
Paveiksle 8 pateiktą „NRules“ taisyklių valdymo ir vykdymo sistemos architektūrą sudaro šios dalys:

1. **Specifinės srities kalba** (angl. DSL – *Domain Specific Language*) – kalba, kuria užrašytos taisyklės, naudojamos „NRules“ sistemoje. Ši kalba yra naudojama SQL užklausoms reliacinėse duomenų bazėse formuoti ir kt.;

2. **Taisyklių saugykla** (angl. *Rule Repository*) – DSL kalba užrašytų taisyklių saugykla, kurioje yra saugomos simuliacijos metu naudojamos ir „RETE“ tinklą sudarančios taisyklės.
3. **Taisyklės konstruktorius** (angl. *Rule Builder*) – atsakingas už taisyklių konvertavimą iš vartotojo užrašytos taisyklės į kanoninę taisyklės formą;
4. **Taisyklės apibrėžtis** (angl. *Rule Definition*) – kanonine taisyklės forma užrašyta taisyklė;
5. **Taisyklių rinkinys** (angl. *Rule Set*) – taisyklių rinkinys, paruoštas kompiliavimui į vykdomąją taisyklių formą;
6. **Taisyklės „kompiliatorius“** (angl. *Rule Compiler*) – atsakingas už taisyklės konvertavimą į vykdomąją taisyklės formą „RETE“ tinklą;
7. **Sesijos fabrikas** (angl. *Session Factory*) – atsakingas už sesijų sukūrimą;
8. **„RETE“ tinklas** (angl. *RETE Network*) – vykdomoji taisyklių forma;
9. **Sesija** (angl. *Session*) – taisyklių vykdymo sesija, kurią sudaro faktų rinkinys ir „RETE“ tinklas sudarytas iš taisyklių saugykloje esančių taisyklių;
10. **Darbinė atmintis** (angl. *Working Memory*) – atmintis kurioje yra saugomi sesijose esantys faktai ir „RETE“ tinklas.

3.2.2. „NRules“ sistemos taisyklių analizė

Nagrinėjant taisyklių variklius, pagrindinė jų sudedamoji dalis yra taisyklės. Taisyklės gali egzistuoti skirtingose formose. „NRules“ sistemoje taisyklės gali būti šių formų: „Fluent DSL“, „Canonical Model“, „Execution Model“. Toliau pateiksime šių taisyklių formų sąryšių modelį 9 pav.



9 pav. „NRules“ taisyklių formų sąryšių modelis

„Fluent DSL“ – tai tokia taisyklių forma, kai taisyklės yra užrašomos naudojant DSL (angl. *Domain Specific Language*) kalbą. Šia kalba užrašytos taisyklės yra sudarytos iš dviejų dalių, t.y. iš sąlygos, kuri atitinka „*When()*“ metodą ir veiksmo – „*Then()*“ metodo. Sąlyga aprašo reikalavimus,

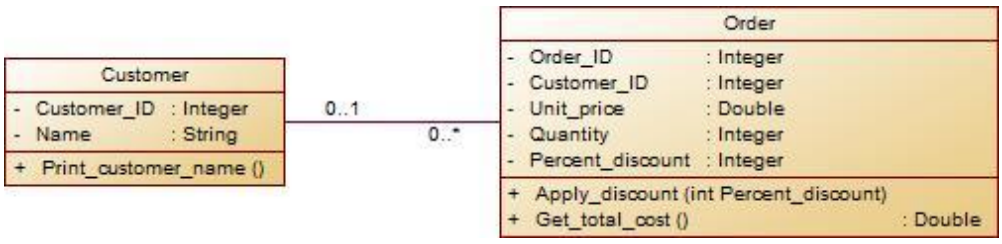
kuriuos turi tenkinti taisyklių variklio darbinėje atmintyje esantys faktai. Dažniausiai taisyklių sąlygas tenkina ne vienas o keli faktai, todėl ta pati taisyklė yra įvykdoma keletą kartų. Taisyklių sąlygoms užrašyti gali būti naudojamos „LINQ“ užklausos, kurių rezultatas yra sąlygą tenkinantys faktai. Išskiriami šie „LINQ“ užklausų operatoriai:

- **„*Match*“** – užklausos atlikimo pradžios operatorius;
- **„*Where*“** – faktų filtravimo, remiantis aprašytomis sąlygomis, operatorius;
- **„*GroupBy*“** – faktų grupavimo operatorius;
- **„*Collect*“** – faktų rinkinio sudarymo operatorius;
- **„*Select*“** – faktų išrinkimo į atskirus sąrašus operatorius;
- **„*SelectMany*“** – faktų išrinkimo į vieną sąrašą operatorius.

Veiksmai aprašo operacijas, kurios yra vykdomos su prieš tai sąlygas tenkinančiais faktais. Faktai objektinėje programavimo kalboje – objektai. Objektai – tai vienareikšmiškai identifikuojamos esybės, apie kurias saugoma informacija, pavyzdžiui, „Užsakymas“, „Klientas“ – jos pagal prasmę niekuo nesiskiria nuo esybių reliaciniame modelyje. Objekto atributas, kaip ir reliaciniame modelyje, yra išmatuojama jo savybė. To paties objekto egzemplioriai skiriasi atributų reikšmėmis. Kiekvienas objektas turi priklausyti kuriai nors klasei. Operacijų metu yra vykdomos operacijos, kurias aprašo klasės metodai. Metodai – funkcijos kurias objektas moka vykdyti. Visos operacijos su objektais atliekamos inicijuojant jų metodus. Taip pat objektai pasižymi inkapsuliacija – niekas išskyrus patį objektą, negali keisti jo vidinės struktūros. Objektai gali būti keičiami tik per iš anksto užprogramuotus jų metodus, o keičiant struktūrą reikia sukurti naują objektą [43]. Kartu su taisyklėmis taip pat yra saugomi papildoma informacija apie taisykles – metaduomenys:

1. **Vardas** (angl. *Name*) – taisyklės pavadinimas;
2. **Apibrėžimas** (angl. *Description*) – taisyklės apibrėžimas, paaiškinimas;
3. **Žymė** (angl. *Tag*) – su taisykle susijusi žymė ar kelios žymės, naudojamos taisyklių grupavimui ar filtravimui;
4. **Prioritetas** (angl. *Priority*) – taisyklės prioritetas, skirtas nustatyti taisyklių vykdymo seką, kai vienu metu yra aktyvuotos kelios taisyklės.
5. **Pakartojamumas** (angl. *Repeatability*) – parametras skirtas kontroliuoti pakartotiną taisyklių vykdymą, esant tiems patiems faktams.

Toliau pateiksime „Fluent DSL“ forma užrašytą taisyklės pavyzdį 11 pav., prieš tai pateikę nagrinėjamos dalykinės srities koncepcinį modelį „Domain model“, 10 pav.



10 pav. Dalykinės srities klasių diagrama

```
[Name("TestRule1"), Description("Test rule that apply discount")]
[Tag("Test"), Tag("Discount")]
[Priority(1)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RuleWithMetadata : Rule
{
    public override void Define()
    {
        Customer customer = null;
        Order order = null;

        When()
            .Match<Customer>(() => customer, c => c.Customer_ID == 122)
            .Match<Order>(() => order, o => o.Customer_ID == customer.Customer_ID, o => o.Quantity > 100);
        Then()
            .Do(ctx => customer.Print_customer_name())
            .Do(ctx => order.Apply_discount(20))
            .Do(ctx => Console.WriteLine("Totaly order cost: {0} eu", order.Get_total_cost()));
    }
}
```

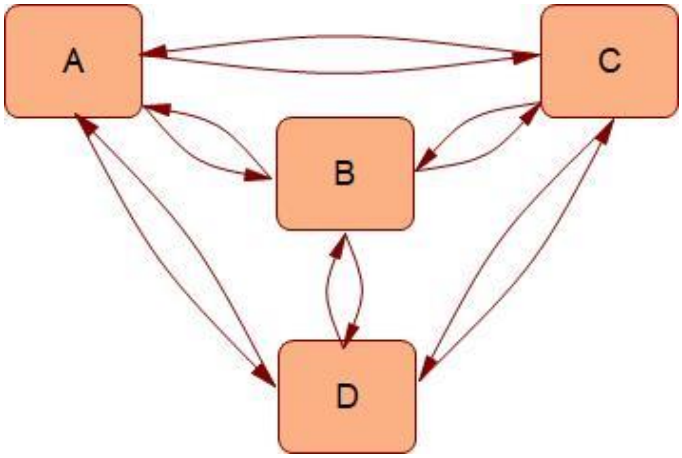
11 pav. „Fluent DSL“ forma užrašyta taisyklė

Apibendrinant galima išskirti „Fluent DSL“ taisyklių užrašymo formos privalumus:

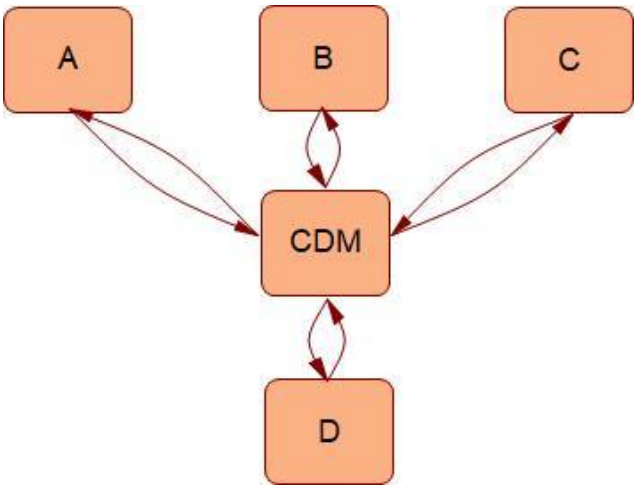
1. Taisyklių užrašymui naudojamos išraiškos yra artimos natūraliai kalbai, tai leidžia taisykles lengviau suprasti ir naudoti dalykinės srities atstovams;
2. Taisyklėms aprašyti naudojamas „Fluent interface“ suteikia galimybe taisyklių išraiškas užrašyti vienoje eilutėje nenaudojant papildomų kintamųjų tarpiniams rezultatams saugoti;
3. Naudojant „LINQ“ užklausas, galima užrašyti taisykles, kurios galiotų ne individualiems faktams, o faktų rinkiniams.

„Canonical model“ – kanoninis taisyklių modelis, kuris gali būti transformuojamas į vykdomąjį modelį. Kanoninio taisyklių modelio naudojami tipai nėra griežtai apibrėžti, todėl savarankiškai reikia užtikrinti taisyklėms apibrėžti naudojamų tipų teisingumą. „NRules“ sistema jos vykdymo metu leidžia kurti naujas taisykles, naudojant kanoninį taisyklių modelį. Viena iš pagrindinių kanoninio taisyklių modelio tikslų – užtikrinti ryšį tarp taisyklių vykdymo ir valdymo sistemos ir skirtingomis formomis užrašytų taisyklių. Atsiradus papildomoms taisyklių užrašymo formoms (pvz. sprendimų lentelės), jos būtų transformuojamos į tą patį bendrą kanoninį taisyklių modelį. Toliau pateiksime pavyzdį, įrodantį kanoninio taisyklių modelio naudojimo efektyvumą. Tarkime, kad nagrinėjama sistema yra sudaryta iš 3 skirtingų taisyklių užrašymo formų ir taisyklių vykdymo elemento. Tuomet bendras galimų transformacijų skaičius yra lygus 12, 12 pav. Jeigu sistemą papildytume kanoninių duomenų modeliu maksimalus transformacijų skaičius sumažėtų iki 8-ių, 13

pav. Naudodami tą patį dalykinės srities koncepcinį modelį, 14 pav., pateiksime „NRules“ sistemos kanonine taisyklių forma užrašytą taisyklę.



12 pav. Galimos transformacijos nenaudojant kanoninio duomenų modelio



13 pav. Galimos transformacijos naudojant kanoninį duomenų modelį

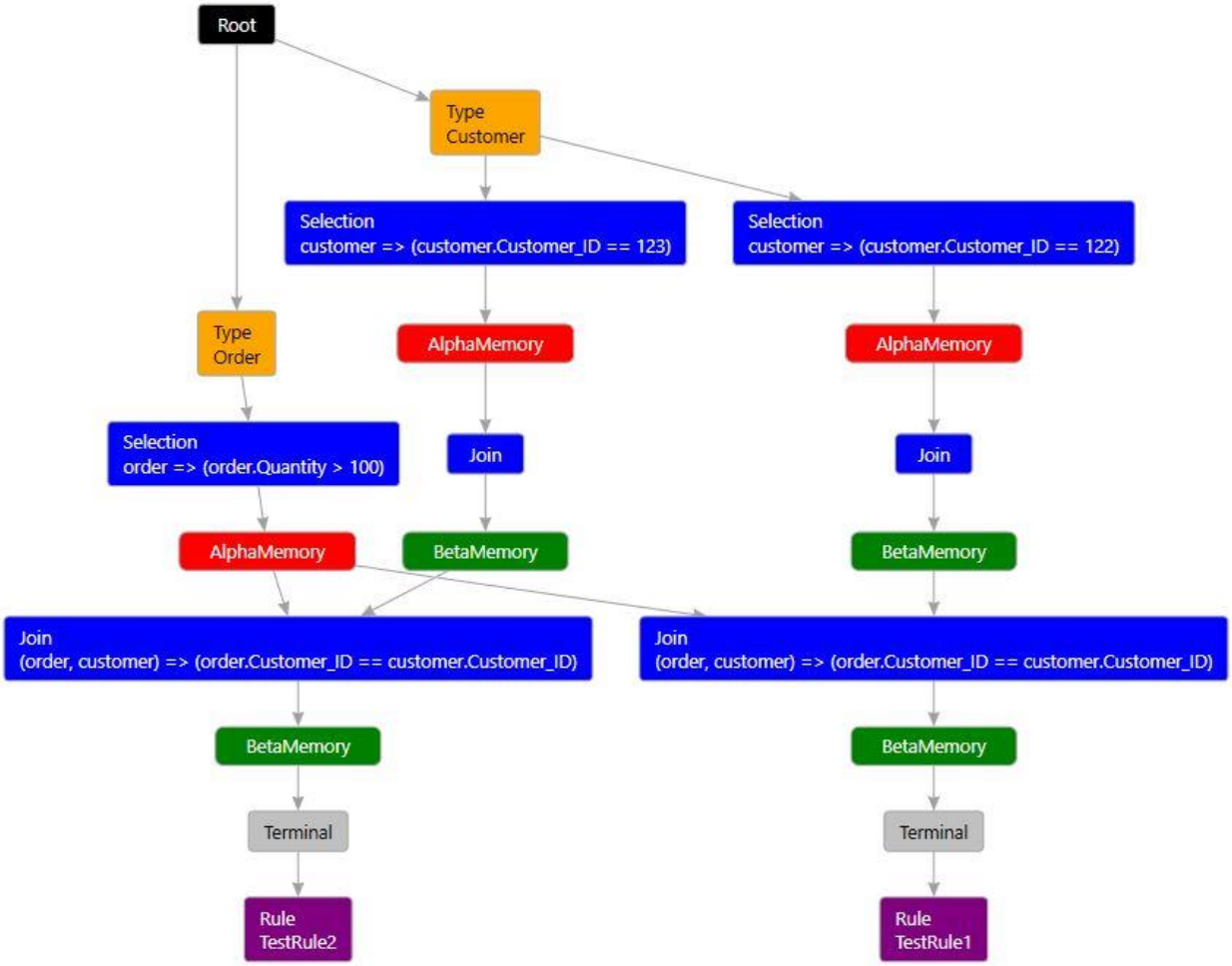
```
public IRuleDefinition BuildRule2()
{
    //Create rule builder
    var builder = new RuleBuilder();
    builder.Name("TestRule2");
    builder.Description("Test rule that apply discount");
    builder.Tag("Test");
    builder.Tag("Discount");
    builder.Priority(1);
    builder.Repeatability(RuleRepeatability.Repeatable);

    //Build conditions
    PatternBuilder customerPattern = builder.LeftHandSide().Pattern(typeof(Customer), "customer");
    Expression<Func<Customer, bool>> customerCondition =
        customer => customer.Customer_ID == 123;
    customerPattern.Condition(customerCondition);

    PatternBuilder orderPattern = builder.LeftHandSide().Pattern(typeof(Order), "order");
    Expression<Func<Order, Customer, bool>> orderCondition1 =
        (order, customer) => order.Customer_ID == customer.Customer_ID;
    Expression<Func<Order, bool>> orderCondition2 =
        order => order.Quantity > 100;
    orderPattern.Condition(orderCondition1);
    orderPattern.Condition(orderCondition2);

    //Build actions
    Expression<Action<IContext, Customer>> action1 =
        (ctx, customer) => customer.Print_customer_name();
    Expression<Action<IContext, Order>> action2 =
        (ctx, order) => order.Apply_discount(20);
    Expression<Action<IContext, Order>> action3 =
        (ctx, order) => Console.WriteLine("Totaly order cost: {0} eu", order.Get_total_cost());

    builder.RightHandSide().Action(action1);
    builder.RightHandSide().Action(action2);
    builder.RightHandSide().Action(action3);

    //Build rule model
    return builder.Build();
}
```

14 pav. „Canonical model“ forma užrašyta taisyklė

Apibendrinant galima išskirti „Canonical model“ taisyklių užrašymo formos privalumus:

1. Naudojant kanoninę taisyklių užrašymo formą, galima kurti naujas taisykles, „NRules“ sistemos veikimo metu;
2. Kanoninis taisyklių modelio naudojimas užtikrina efektyvų transformacijų skaičių tarp skirtingų taisyklių užrašymo formų ir taisykles apdorojančių sistemos komponentų.

„Execution model“ – vykdomasis taisyklių modelis. Šia forma užrašytos taisyklės gali sukurti naujus faktus ir juos patalpinti į taisyklių variklio darbinę atmintį, taip pat pakeisti jau esamus faktus. Šie pokyčiai gali aktyvuoti naujas taisykles. „NRules“ yra produkcinių taisyklių sistema, kurios vykdomoji forma yra „RETE“ tinklas. „RETE“ tinklo naudojimas leidžia efektyviai atrasti taisykles, kurių sąlygas tenkina faktai. Pateiksime 15 pav. „RETE“ tinklą, kuris buvo sudaromas, prieš tai dviem skirtingomis formomis užrašytas taisykles, konvertavus į vykdomąsias taisykles.



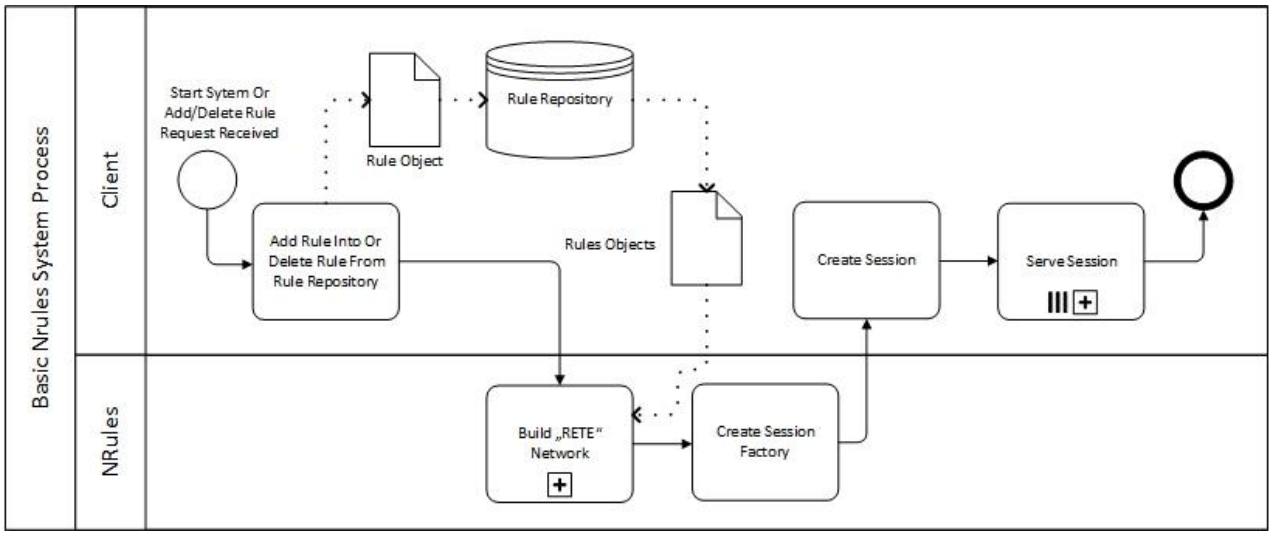
15 pav. „RETE“ tinklas sudarytas iš dviejų taisyklių

3.2.3. „NRules“ sistemos veikimo analizė

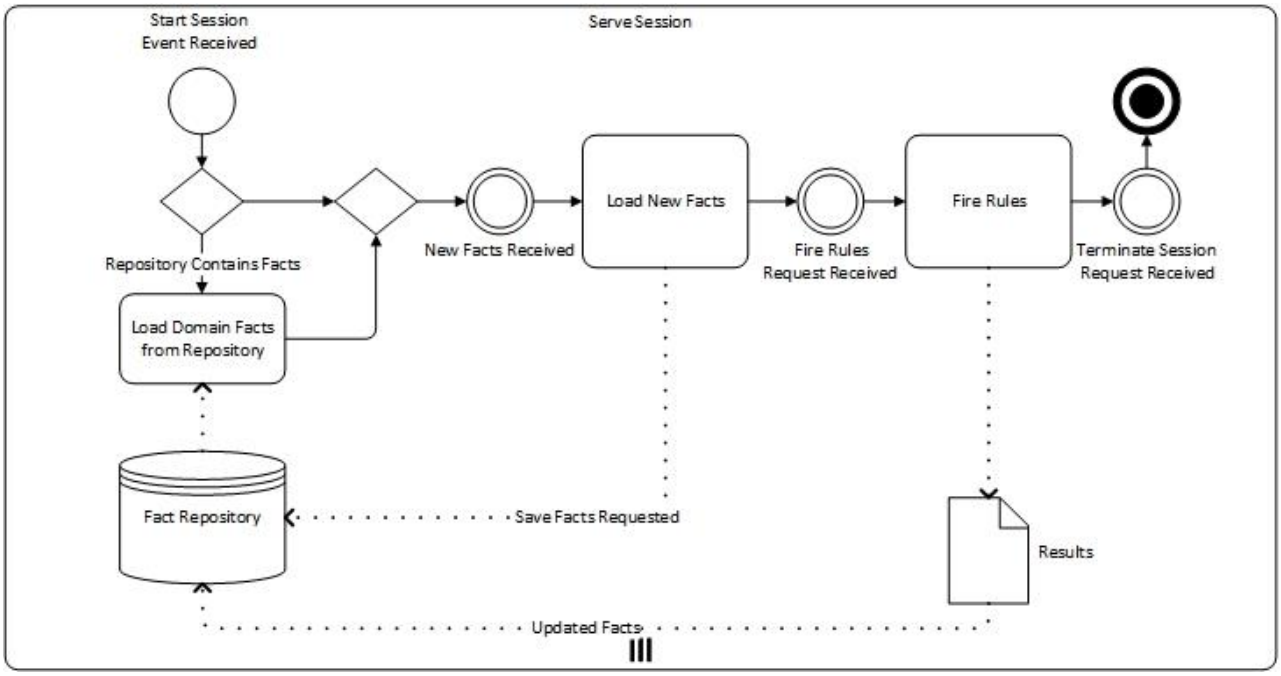
Atlikdami „NRules“ sistemos dinamiškumo analizę, dėmesys bus skiriamas naujų taisyklių ir faktų kūrimo, esamų atnaujinimo ir ištrynimo galimybių analizei, kai sistema, kurioje yra naudojamos šios taisyklės ir faktai yra aktyvios būsenos.

Nuosekliai apžvelgsime žingsnius, kuriuos reikia įvykdyti, kad „NRules“ sistema būtų paruošta darbui. Iš pradžių yra aprašomos taisyklės. Taisyklės gali būti aprašomos naudojant „Fluent DSL“ arba „Canonical Model“ taisyklių formą. Parašytos taisyklės yra jungiamos į taisyklių rinkinius. Taisyklių rinkinių gali būti ne vienas. Tos pačios taisyklės gali būti skirtinguose taisyklių rinkiniuose. Kiekvienas iš rinkinių turi savo pavadinimą. Suformuoti taisyklių rinkiniai yra patalpinami į taisyklių saugyklą. Taisyklių saugykloje esančių taisyklių koreguoti negalima, galima tik pridėti naujus taisyklių rinkinius. Norint atnaujinti ar ištrinti taisyklę ar taisykles yra galimybė gauti visas taisykles esančias taisyklių saugykloje, tada jas pakoreguoti ar ištrinti ir sukurti naują taisyklių saugyklą su atnaujintomis ar naujomis taisyklėmis.

Patalpinus taisykles į taisyklių saugyklą, taisyklių saugykla, naudojant taisyklių kompiliatorių (angl. *Rule compiler*) yra sukompiliuojama ir sudaromas „RETE“ tinklas. Sukompiliuotos taisyklės ir suformuotas „RETE“ tinklas yra saugomos sesijų fabrike (angl. *Session factory*) „fabrike“. Suformuotu „RETE“ tinklu gali naudotis ne vienas klientas. Klientais gali būti sistemos posistemiai, kurie naudojasi taisyklių valdymo ir vykdymo sistema. Kiekvienam klientui sesijų fabrikas sukuria po sesiją, kurios darbinėje atmintyje (angl. *Working memory*) yra saugomi toje sesijoje patalpinti dalykinės srities faktai. Kiekviena iš sesijų turi po vienodą „RETE“ tinklo kopiją. Skirtingų sesijų „RETE“ tinklai vienas nuo kito skiriasi užkrautais faktais. Kiekvienoje iš sesijų galima pridėti naujus ir ištrinti ar atnaujinti esamus faktus. Pridėjus ar atnaujinus kiekvienos sesijos faktus, gali būti aktyvuojamos naujos taisyklės. Vykdant taisyklės gali būti atnaujinami esami faktai ir pridedami nauji faktai. Tokiu būdu pačios taisyklės gali keisti sesijų kontekstą. Toliau, 16, 17 paveiksluose, pateiksime apibendrintą „NRules“ sistemos veikimo proceso schemą.



16 pav. „NRules“ sistemos veikimo proceso schema



17 pav. Detalizuota sesijų valdymo (angl. *Serve Session*) veikimo proceso schema

Apibendrinant galime teigti, kad „NRules“ taisyklių vykdymo ir valdymo sistema geba vykdyti iš anksto sudarytus taisyklių rinkinius, dinamiškai pridėti naujus ar ištrinti ir atnaujinti esamus faktus, neperkraunant visos sistemos. Tačiau nėra galimybės pridėti naujų taisyklių ar ištrinti esamas taisykles. Norint atlikti pakeitimus su taisyklėmis yra būtina iš naujo sukompiliuoti visas taisykles, tokiu būdu atnaujinant „RETE“ tinklą. Po kiekvieno „RETE“ tinklo atnaujinimo yra būtina pakartoti 15 pav. pateiktą proceso schemą. Tai reiškia, kad reikia iš naujo sukurti visų klientų sesijas ir jas užpildyti faktais, o tam yra būtina saugoti kiekvienoje iš sesijų naudojamus faktus. Esant dideliems taisyklių ir faktų kiekiams klientų sesijose, šių sesijų atnaujinimas užimtų nemažai laiko. Taip pat „NRules“ sistemos taisyklių valdymo modulis neturi grafinės vartotojo sąsajos, kas apsunkina naudojimąsi šia sistema.

3.3. „NRules“ sistemos dinamiškumo problemos

Atlikus „NRules“ sistemos analizę, buvo identifikuotos šios dinamiškumo problemos:

1. Nėra numatytos galimybės pridėti naujas ar ištrinti ar atnaujinti esamas taisykles;
2. Keičiant taisykles yra būtina sukompiliuoti visas produkcinėje atmintyje esančias taisykles, tam, kad būtų atnaujintas „RETE“ vykdomasis tinklas;
3. Nėra centralizuotos sistemos leidžiančios pakeistą „RETE“ tinklą atnaujinti visose klientų sesijose;
4. Nėra galimybės atnaujinti klientų sesijų. Norint atlikti pakeitimus klientų sesijose yra būtina sukurta naujas sesijas ir užkrauti buvusius faktus;
5. Taisyklių valdymo modulis neturi grafinės vartotojo sąsajos.

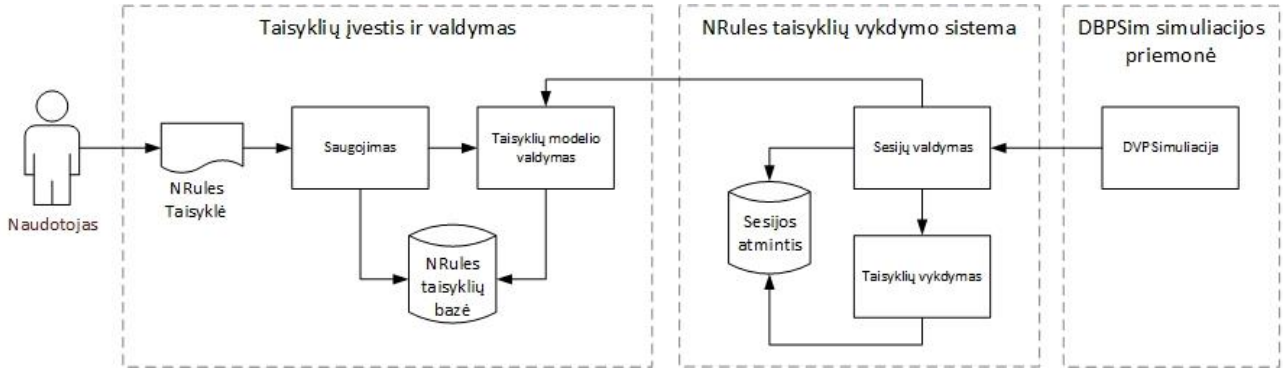
3.4. Reikalavimai siūlomo būdo įgyvendinimui

Atlikdami literatūros analizę išsiaiškinome, kad taisyklių valdymo ir vykdymo sistemos leidžia taisykles aprašyti kaip autonominius vienetus ir vykdyti ir tvarkyti nepriklausomai nuo verslo procesų simuliacijos įrankių. Tai užtikrina greitesnį ir efektyvesnį taisyklių kūrimo, keitimo bei vykdymo procesą. Atliekant „NRules“ sistemos analizę buvo nustatyti trūkumai susiję su sistemos dinamiškumu. Todėl siūlomas būdas yra susijęs su „NRules“ sistemos patobulinimu ir paruošimu integracijai su dinaminių verslo procesų simuliacijos įrankiu, siekiant patobulinti DVP simuliaciją. Yra siūlomas būdas, kaip išspręsti „NRules“ sistemos dinamiškumo problemas, kokius pakeitimus joje atlikti ir kaip išplėsti ją papildomais komponentais: „NRules“ sistemos valdymo moduliu ir taisyklių valdymo moduliu su grafine vartotojo sąsaja. Toliau suformuluosime taisyklių vykdymo ir valdymo sistemai keliamus reikalavimus:

1. Galimybė vykdyti, kurti naujus ar keisti esamus faktus ir taisykles neperkraunant viso simuliacijos proceso.
2. Taisyklių vykdymo ir valdymo sistema tūrėtų tūrėti grafinę vartotojo sąsają, skirtą verslo žmonėms kurti naujas ar koreguoti jau esamas taisykles.
3. Taisyklių vykdymo ir valdymo sistema turi gebėti operuoti dideliais taisyklių kiekiais ir užtikrinti taisyklių valdymo dinamiškumą ir vykdymo greitaveiką.
4. Taisyklių vykdymo ir valdymo sistema turi tūrėti galimybę būti integruota su kitomis sistemomis.

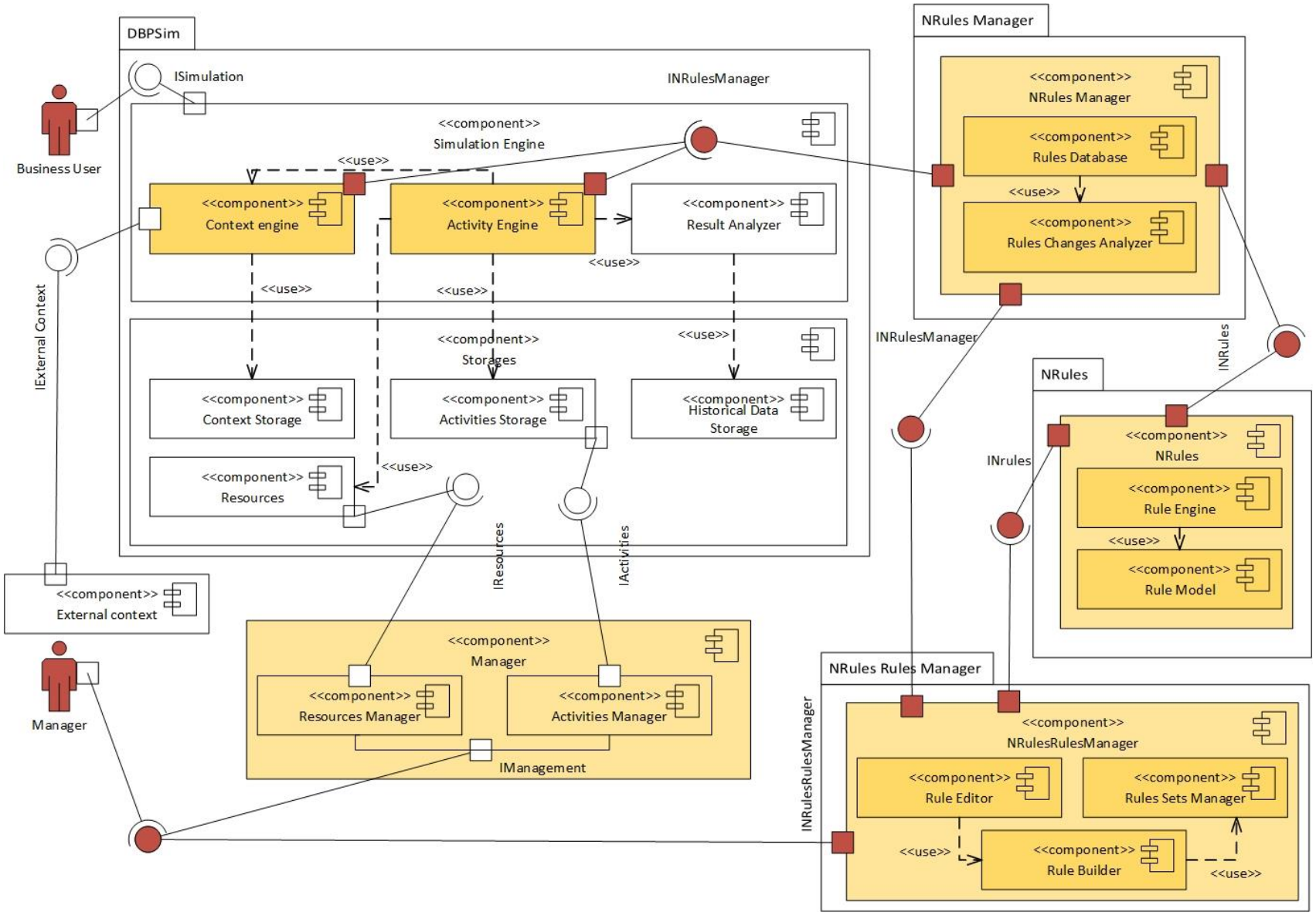
3.5. Siūlomo būdo koncepcija

Remiantis keliamais reikalavimais sudaryta siūlomo metodo aukšto lygio koncepcijos diagrama, kuri pateikta 18 pav.



18 pav. Siūlomo metodo aukšto lygio koncepcijos diagrama

Toliau detalizuosime 19 pav. pateiktą siūlomą „NRules“ ir „DBPSim“ sistemų architektūros komponentų diagramą.



19 pav. Siūlomų pakeitumų ir „NRules“ sistemos integracijos į „DBPSim“ įrankį architektūros komponentų diagrama (geltona spalva pažymėtus modulius yra siūloma sukurti arba keisti)

Lyginant pateiktą architektūros komponentų diagramą su buvusia „DBPSim“ (8 pav.) sistemos architektūra yra atlikti tokie pakeitimai:

1. „DBPSim“ taisyklių variklis ir taisyklių bazė pakeista nauju „NRules“ valdymo moduliu (angl. *NRules manager*).
2. Pridėta „NRules“ sistema ir naujas „NRules“ taisyklių valdiklis (angl. *NRules Rules Manager*).

Toliau aptarsime naujų komponentų sudedamąsias dalis ir paskirtį:

1. **„NRules“ valdymo modulis** (angl. *NRules manager*) – šis modulis yra atsakingas už duomenų ir valdymo perdavimą tarp taisyklių valdiklio modulio, „NRules“ sistemos ir DVP simuliacijos įrankio. Šio modulio dėka galima dinamiškai keisti taisykles klientų sesijose. Taip pat šis modulis užtikrina centralizuotą taisyklių vykdymo ir valdymo sistemą ir padeda atskirti „NRules“ sistemą nuo likusios sistemos. Šį modulį sudarantys komponentai yra šie:
   1.1. **Taisyklių bazė** (angl. *Rule database*) – atsakinga už taisyklių ir taisyklių rinkinių saugojimą, pridėjimą ir ištrynimą;
   1.2. **Taisyklių pasikeitimo analizatorius** (angl. *Rules changes analyzer*) – atsakingas už taisyklių pasikeitimų analizę, kurios metu yra analizuojama kokios taisyklės buvo atnaujintos, pridėtos ar ištrintos.
2. **„NRules“ taisyklių valdymo modulis** (angl. *NRules rules manager*) – atsakingas už naujų taisyklių rinkinių sukūrimą ir taisyklių atnaujinimą jose. Taip pat šis modulis leidžia vartotojams simuliacijos metu pridėti, atnaujinti ar ištrinti taisykles. Yra galimybė taisyklių bazėje saugomus taisyklių rinkinius su taisyklėmis išsaugoti į taisyklių biblioteką (.dll failą) ir taip pat iš jos užkrauti. Šis modulis turi vartotojo sąsają, kuri bus detalizuota tolimesniuose skyriuose. Šį modulį sudarantys komponentai yra šie:
   2.1. **Taisyklės redaktorius** (angl. *Rule Editor*) – suteikia galimybę, naudojant tekstinį redaktorių redaguoti „NRules“ kanonines taisyklių forma užrašytas taisykles. Baigus redagavimą, taisyklė yra sukompiliuojama ir užkraunamą į taisyklių bazę ir atnaujinami „RETE“ tinklai klientų sesijose;
   2.2. **Taisyklės konstruktorius** (angl. *Rule builder*) – atsakinags už pridedamų naujų taisyklių kompiliavimą į „NRules“ kanoninę taisyklių formą. Taip pat suteikia galimybę taisyklių rinkinius esančius taisyklių bazėje išsaugoti į taisyklių biblioteką ir taip pat iš jos užkrauti. Tai leidžią sudarytus taisyklių rinkinius naudoti skirtingose simuliacijose;
   2.3. **Taisyklių rinkinių valdiklis** (angl. *Rule sets manager*) – atsakingas už taisyklių rinkinių valdymą, pridedant, atnaujinant ar ištrinant taisykles.

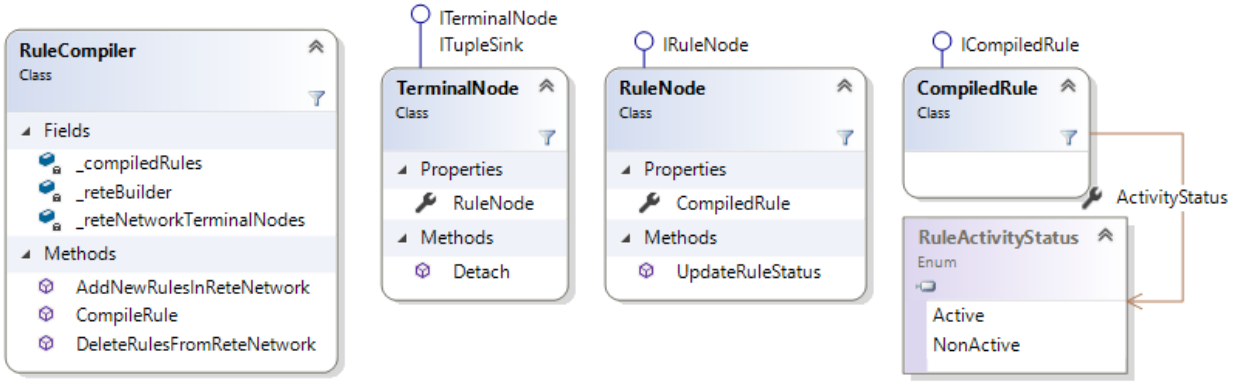
Sekančiame skyriuje aprašysime siūlomų „NRules“ sistemos pakeitimų įgyvendinimą.

3.6. Siūlomo būdo įgyvendinimas

Šiame skyriuje aprašysime išplėstos „NRules“ sistemos komponentus ir atliktus pakeitimus su pačia „NRules“ sistema. Atliktus pakeitimus detalizuosime sistemą sudarančių komponentų klasių diagramomis.

3.6.1. Pakeitimai „NRules“ sistemoje

20 pav. yra pateikti „NRules“ taisyklių konstruktoriaus (angl. *Rule builder*), sukompiliuotos taisyklės (angl. *Compiled rule*), terminalo (angl. *Terminal node*) ir taisyklės (angl. *Rule node*) mazgų klasių diagramos.



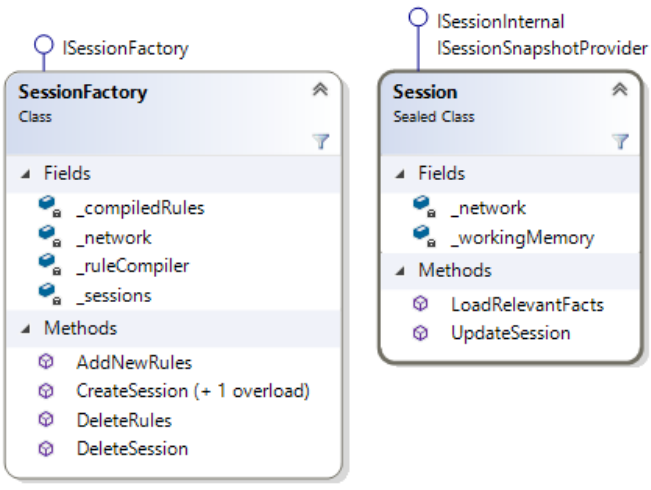
20 pav. Atnaujinto taisyklių konstruktoriaus, sukompiliuotos taisyklės, terminalo ir taisyklės mazgų klasių diagrama

Taisyklių konstruktoriaus yra atsakingas už taisyklių transformavimą į „RETE“ tinklą. Šiame modulyje pridėti papildomi metodai: „AddNewRuleInReteNetwork“, „DeleteRulesFromReteNetwork“. Šie metodai atnaujina „RETE“ tinklą pridedant naujas ar ištrinant esamas taisykles. Taisyklės pridėjimo metu, naudojant išsaugotą „RETE“ tinklo konstruktorių „_reteBuilder“, kuris yra sukuriamas pirminio „RETE“ tinklo sukūrimo metu, „RETE“ tinklas yra papildomas nauja taisykle, sudarant naujus tinklo mazgus. Jeigu su pridedama taisykle susiję mazgai jau yra sukurti, sukuriamas tik papildomas terminalo mazgas, kuris yra sujungiamas su pridedamos taisyklės mazgu. Sukurti nauji terminalo mazgai papildo rinkinį „_ReteNetworkTerminalNodes“ sudarytą iš taisyklių ir terminalų, kuriame yra saugoma kiekviena pridėta taisyklė ir taisyklei priklausantys terminalo mazgai. Atnaujintas „RETE“ tinklo konstruktorius taip pat yra išsaugojamas. Ištrinant taisyklę yra ieškoma ištrinamos taisyklės terminalo mazgų. Surastuose taisyklių terminalų mazguose, naudojant „UpdateRuleStatus“ metodą, prie terminalo prijungtos taisyklės statusas „RuleActivityStatus“ yra pakeičiamas į neaktyvią „NonActive“. Tokiu būdu nereikia atlikti pakeitimų su tinklo struktūra. Tinkle likę ištrintos taisyklės mazgai turi privalumų ir trūkumų: pridedant naujas taisykles, likę mazgai gali būti pakartotinai panaudojami, tačiau jeigu ištrinamos taisyklės yra pavienės, likę mazgai tinkle yra pertekliniai, kurie lėtina taisyklių perrinkimą.

4 lentelė. Atnaujinto taisyklių konstruktoriaus, sukompiliuotos taisyklės, terminalo ir taisyklės mazgų klasių diagramų metodai ir atributai

Komponento pavadinimas	Komponento tipas	Komponento aprašymas
RuleCompiler. AddNewRuleInReteNetwork	Metodas	Atnaujina „RETE“ tinklą pridėdamas su taisyklėmis susijusius naujus tinklo mazgus.
RuleCompiler. CompileRule	Metodas	Transformuoja kanonine taisyklės formą į vykdomąją taisyklės formą – „RETE“ tinklą. Išsaugo su taisykle susijusius terminalo mazgus ir atnaujina „RETE“ tinklo konstruktorių.
RuleCompiler. DeleteRulesFromRuleTerminalNodes	Metodas	Suranda su taisykle susijusius terminalo mazgus ir pakeičia su terminalo mazgu sujungtos taisyklės statusą į neaktyvią.
TerminalNode. Detach	Metodas	Atnaujina prie terminalo mazgo prijungtos taisyklės mazgo statusą.
RuleNode. UpdateRuleStatus	Metodas	Pakeičia taisyklės aktyvumo parametro vertę.
RuleCompiler. _compiledRules	Atributas	Sukompiliuotų ir „RETE“ tinklą sudarančių taisyklių rinkinys.
RuleCompiler. _reteBuilder	Atributas	„RETE“ tinklo konstruktorius, kuriame yra saugoma tinklą sudarantys mazgai.
RuleCompiler. _reteNetworkTerminalNodes	Atributas	Rinkinis sudarytas iš „RETE“ tinkle esančių taisyklių ir taisyklių terminalų mazgų. Šis rinkinys naudojamas nustatyti su ištrinama taisykle susijusius terminalo mazgus.

Toliau aptasrime pakeitimus atliktus su „NRules“ sistemos sesijų kūrimo (angl. *Session Factory*) ir sesijos (angl. *Session*) komponentais. Šių komponentų klasių diagramos pateiktos 21 pav.



21 pav. Atnaujinto sesijų kūrimo ir sesijos klasių diagramos

Sesijų kūrimo modulis yra atsakingas už naujų sesijų sukūrimą ir jų atnaujinimą. Sesijos sukūrimo metu naudojant prieš tai aptartą taisyklių konstruktorių yra sukuriamas „RETE“ tinklas, kuris yra saugomas sesijų kūrimo modulyje, o jo kopija yra saugoma naujai sukurtoje sesijoje. Norimos ištrinti ar pridėti taisyklės yra perduodamos sesijų kūrimų moduliui, kuriame atsižvelgiant į taisyklių pakeitimus iš pradžių yra deaktyvuojamos „RETE“ tinkle norimos ištrinti taisyklės, o po to pridedamos į „RETE“ tinklą naujos taisyklės. Tokiu būdu yra atnaujinamas bazinis „RETE“ tinklas „_network“ ir tinklą sudarančios taisyklės „compiledRules“, kurie yra saugomi sesijų kūrimo modulyje ir yra naudojami sukuriant naujas sesijas. Atnaujinus bazinį tinklą, taip pat yra atnaujinami tinklai ir sesijose „_sessions“, kurios taip pat yra saugomos sesijų kūrimo modulyje. Atnaujinus tinklus sesijose, taip pat yra užkraunami su naujomis pridėtomis taisyklėmis susiję faktai, kurie yra saugomi kiekvienos sesijos darbinėje atmintyje „_workingMemory“. Skirtingų sesijų „RETE“ tinklų struktūra yra vienoda, skiriasi tik užkrautų faktų tipais ir kiekiu. Pridėta galimybė ištrinti sesijas, ištrinant kartu su sesija aktualią informaciją. Sesijų kūrimo modulis veikia kaip centralizuota sistema, kuri saugo informaciją apie kiekvieną iš sesijų ir atlieka atnaujinimus jose.

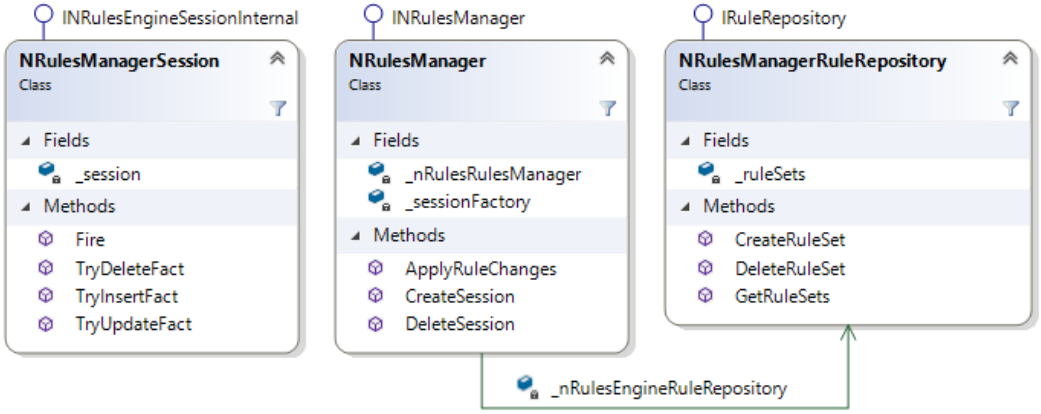
5 lentelė. Atnaujinto sesijų kūrimo ir sesijos klasių diagramų metodai ir atributai

Komponento pavadinimas	**Komponento tipas**	**Komponento aprašymas**
SessionFactory. AddNewRules	Metodas	Išplečia bazinį sesijų kūrimo modulio „RETE“ tinklą ir sesijose esančius tinklus, remiantis naujai pridėtų taisyklių rinkiniu. Sesijų kūrimo modulyje saugomas „RETE“ tinklą sudarančių taisyklių rinkinys papildomas naujomis taisyklėmis.
SessionFactory. CreateSession	Metodas	Sukuria naują sesiją su baziniu „RETE“ tinklu ir nauja darbine atmintimi. Papildo sesijų kūrimo modulyje saugomų sesijų rinkinį naujai sukurta sesija.
SessionFactory. DeleteRules	Metodas	Ištrina iš sesijų kūrimo modulyje saugomo „RETE“ tinklą sudarančių taisyklių rinkinio nereikalingas taisykles. Atnaujina sesijų „RETE“ tinklus, pakeisdamas ištrinamų taisyklių statusus „RETE“ tinkle į neaktyvias.
SessionFactory. DeleteSession	Metodas	Ištrina iš sesijų kūrimo modulyje saugomų sesijų rinkinio norima ištrinti sesiją ir kartu su ja susijusią informaciją.
SessionFactory. _compiledRules	Atributas	Sesijų kūrimo modulyje saugomas bazinį „RETE“ tinklą sudarančių taisyklių rinkinys.
SessionFactory. _network	Atributas	Sesijų kūrimo modulyje saugomas bazinis „RETE“ tinklas.
SessionFactory. _ruleCompiler	Atributas	Taisyklių konstruktoriaus, kuriame yra saugoma informacija apie konstruojamą bazinį „RETE“ tinklą.
SessionFactory. _sessions	Atributas	Sesijų kūrimo modulio sukurtų sesijų rinkinys.

Session. LoaRelevantFacts	Metodas	Užkrauna į sesijos „RETE“ tinklą sesijos darbinėje atmintyje esančius faktus, kurie yra susiję su naujomis pridedamomis taisyklėmis.
Session. UpdateSession	Metodas	Pakeičia sesijoje saugomą „RETE“ tinklą atnaujintu tinklu.
Session. _network	Atributas	Sesijoje saugomas individualus sesijos „RETE“ tinklas.
Session. _workingMemory	Atributas	Sesijos darbinė atmintis, kurioje yra saugomi į sesiją užkrauti faktai ir „RETE“ tinklą sudarantys alfa ir beta atminties mazgai.

3.6.2. „NRules“ sistemos išplėtimas

„NRules“ sistemos išplėtimui buvo sukurtas „NRules“ sistemos valdiklis (angl. *NRules manager*). „NRules“ sistemos valdiklio ir jo naudojamų papildomų komponentų klasių diagramos pateiktos 22 pav.



22 pav. Naujai sukurtų “NRules“ sistemos valdiklio, taisyklių duomenų bazės ir sesijos valdiklio klasių diagramos

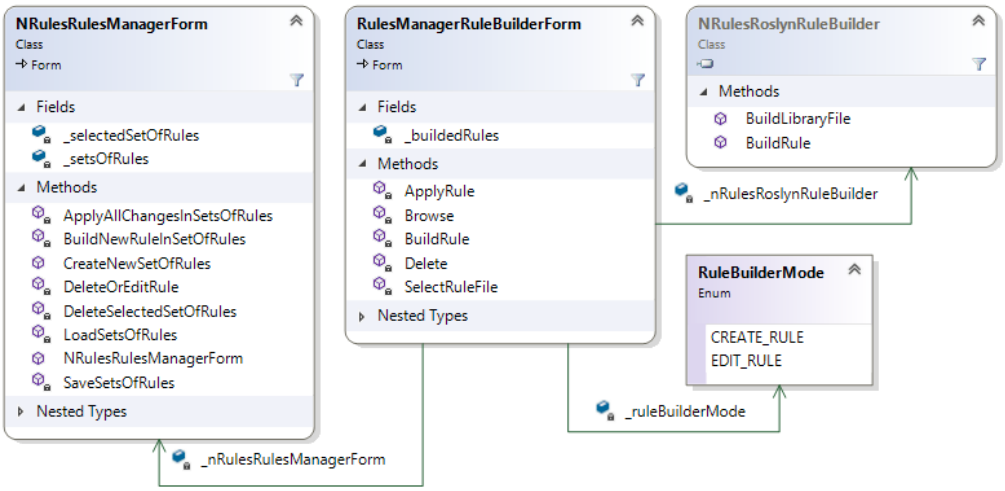
Pagrindinis „NRules“ valdiklio paskirtis (angl. *NRules manager*) – užtikrinti duomenų mainus tarp išorinės sistemos – DVP simuliacijos įrankio, „NRules“ sistemos ir taisyklių valdymo modulio (angl. *NRules rules manager*). Šiame valdiklyje taip pat yra taisyklių duomenų bazė (angl. *NRules manager rule repository*), kuri yra tvarkoma taisyklių valdymo modulio. Atlikus pakeitimus taisyklių duomenų bazėje, „NRules“ sistemos valdiklis išanalizuoja atliktus pakeitimus ir naudodamasis patobulintais sesijų kūrimo modulio metodais dinamiškai atnaujina klientų sesijas. Tam, kad „NRules“ sistema būtų izoliuota nuo išorinės sistemos, papildomai yra sukurtas „NRules“ valdiklio sesijų valdiklis, kuris tiesiogiai perduoda operacijas į „NRules“ sistemos sesijų valdiklį.

6 lentelė. Naujai sukurtų “NRules“ sistemos valdiklio, taisyklių duomenų bazės ir sesijos valdiklio klasių diagramų metodai ir atributai

Komponento pavadinimas	Komponento tipas	Komponento aprašymas
NRulesManager. ApplyRuleChanges	Metodas	Analizuoja taisyklių bazėje atliktus pakeitimus ir nustato kurios taisyklės buvo ištrintos ir pridėtos. Jeigu tarp ištrinamų taisyklių yra pridedamų taisyklių, tai iš ištrinamų ir pridedamų taisyklių rinkinių šios taisyklės pašalinamos. Paruošti taisyklių rinkiniai yra perduodami „NRules“ sistemos sesijų konstruktoriui, kuriame yra atnaujinami kiekvienoje iš sesijų „RETE“ tinklas.
NRulesManager. CreateSession	Metodas	Sukuria naują „NRules“ sesiją ir ją užregistruoja sesijų konstruktoriaus modulyje. Sesijos valdymas yra perduodamas išorinei sistemai.
NRulesManager. DeleteSession	Metodas	Ištrina sesiją ir su ja susijusią informaciją, bei išregistruoja iš sesijų konstruktoriaus modulio.
NRulesManager. _nRulesRulesManager	Atributas	Taisyklių valdymo modulis, kuris valdo taisyklių duomenų bazę ir informuoja „NRules“ sistemos valdiklį apie įvykdytus pakeitimus.
NRulesManager. _sessionFactory	Atributas	Sesijų konstruktoriaus modulis, kuris atsakingas už sesijų valdymą ir pakeitimus jose.
NRulesManagerSession. Fire	Metodas	Perduoda iš išorinės sistemos komandą „NRules“ sistemos sesijai, kad reikia įvykdyti aktyvuotas taisykles.
NRulesManagerSession. TryDeleteFact	Metodas	Perduoda iš išorinės sistemos komandą ir informaciją apie norimą ištrinti faktą „NRules“ sistemos sesijai, kurioje iš darbinės atminties ir „RETE“ tinklo ištrinamas faktas.
NRulesManagerSession. TryInsertFact	Metodas	Perduoda iš išorinės sistemos komandą ir informaciją apie norimą pridėti faktą „NRules“ sistemos sesijai, į kurios darbinę atmintį ir „RETE“ tinklą užkraunamas faktas.
NRulesManagerSession. TryUpdateFact	Metodas	Perduoda iš išorinės sistemos komandą ir informaciją apie norimą atnaujinti faktą „NRules“ sistemos sesijai, į kurios darbinės atminties ir „RETE“ tinklo yra pašalinamas faktas ir užkraunamas atnaujintas faktas.
NRulesManagerSession. _session	Atributas	„NRules“ sistemos sesijų konstruktoriuje užregistruota sesija.

NRulesManagerRuleRepository. CreateRuleSet	Metodas	Sukuriamas taisyklių duomenų bazėje naujas taisyklių rinkinys, į kurį taisyklių valdiklis talpina taisykles.
NRulesManagerRuleRepository. DeleteRuleSet	Metodas	Ištrinamas taisyklių rinkinys ir jame esančios taisyklės iš taisyklių duomenų bazės.
NRulesManagerRuleRepository. GetRuleSets	Metodas	Gražina visus taisyklių duomenų bazėje esančius taisyklių rinkinius.
NRulesManagerRuleRepository. _ruleSets	Atributas	Taisyklių duomenų bazėje saugomi taisyklių rinkiniai su juose esančiomis taisyklėmis. Ta pati taisyklė negali būti skirtinguose taisyklių rinkiniuose.

Taisyklių įvedimui ir taisyklių duomenų bazės rinkinių sudarymui ir valdymui buvo sukurtas „NRules“ sistemos taisyklių valdiklis (angl. *NRules rules manager*). Šis valdiklis taip pat naudoja papildomai sukurtus naujus modulius: taisyklių konstruktorių (angl. *Rules manager rule builder*) ir specialiai „NRules“ sistemos taisyklėms pritaikyta „Roslyn“ kompiliatorių (angl. *NRules Roslyn rule builder*). „NRules“ sistemos valdiklis ir taisyklių konstruktorius turi grafines vartotojo sąsajas, kurios pateiktos 24 pav., o jas sudarančių komponentų atliekamos operacijos yra detalizuotos 7-oje lentelėje. Toliau 23 pav. pateiksime aptartų modulių klasių diagramas.

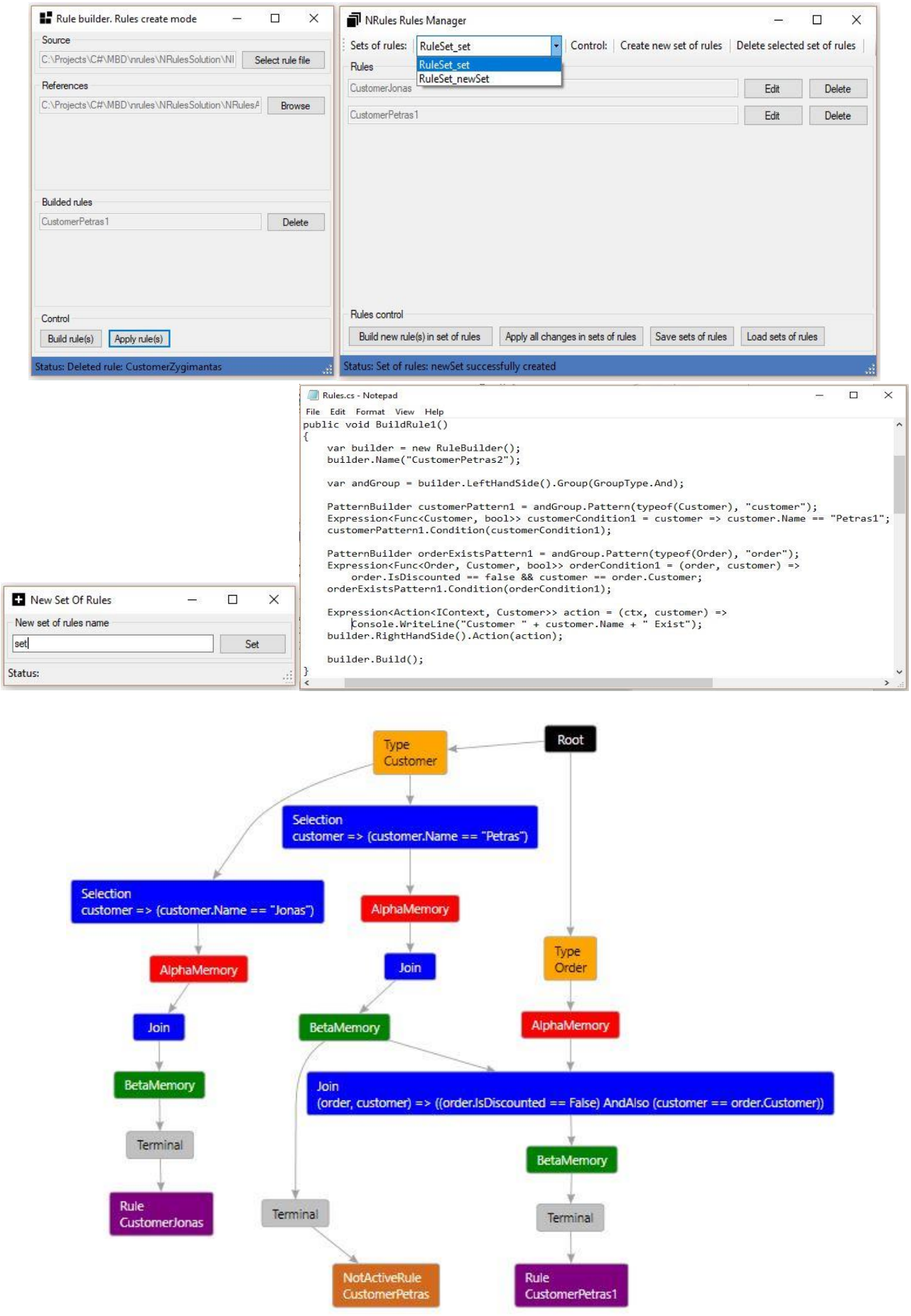


23 pav. Naujai sukurtų „NRules“ sistemos taisyklių valdiklio, taisyklių konstruktoriaus ir „Roslyn“ taisyklių kompiliatoriaus klasių diagramos

7 lentelė. Naujai sukurtų „NRules“ sistemos taisyklių valdiklio, taisyklių konstruktoriaus ir „Roslyn“ taisyklių kompiliatoriaus klasių diagramų metodai ir atributai

Komponento pavadinimas	Komponento tipas	Komponento aprašymas
NRulesRulesManagerForm. ApplyAllChangesInSetsOfRules	Metodas	Informuoja „NRules“ sistemos valdiklį apie atliktus pakeitimus taisyklių duomenų bazėje.
NRulesRulesManagerForm. BuildNewRuleInSetOfRules	Metodas	Aktyvuoja taisyklių konstruktoriaus programos langą.
NRulesRulesManagerForm. CreateNewSetOfRules	Metodas	Sukuria naują taisyklių rinkinį taisyklių duomenų bazėje, prieš tai nurodant taisyklių rinkinio pavadinimą.
NRulesRulesManagerForm. DeleteOrEditRule	Metodas	Pagal tai ką pasirenka vartotojas yra ištrinama taisyklė iš pasirinkto taisyklių rinkinio arba redaguojama taisyklė naudojant operacinės sistemos tekstinį redaktorių.
NRulesRulesManagerForm. DeleteSelectedSetOfRules	Metodas	Ištrina pasirinktą taisyklių rinkinį ir jame esančias taisykles iš taisyklių duomenų bazės.
NRulesRulesManagerForm. LoadSetsOfRules	Metodas	Naudojant „NRules“ sistemos taisyklėms pritaikytą „Roslyn“ taisyklių kompiliatorių, užkrauna iš pasirinktos taisyklių bibliotekos taisyklių rinkinius ir juose esančias taisykles.
NRulesRulesManagerForm. SaveSetsOfRules	Metodas	Taisyklių duomenų bazėje esančius taisyklių rinkinius ir juose esančias taisykles išsaugo į taisyklių biblioteką, naudojant pritaikytą „Roslyn“ taisyklių kompiliatorių.
NRulesRulesManagerForm. _selectedSetOfRules	Atributas	Pasirinktas taisyklių rinkinys, kuriame yra atliekami pakeitimai.
NRulesRulesManagerForm. _setsOfRules	Atributas	Taisyklių duomenų bazėje esantys taisyklių rinkiniai su kuriais yra atliekami pakeitimai.
RulesManagerRuleBuilderForm. ApplyRule	Metodas	Sukompiliuotų taisyklių rinkinys perduodamas taisyklių valdymo moduliui, kuris patikrina ar esamuose taisyklių rinkiniuose ir naujai sukompiliuotose taisyklėse nėra pasikartojančių taisyklių ir prideda tik naujas taisykles į pasirinktą taisyklių rinkinį.
RulesManagerRuleBuilderForm. Browse	Metodas	Užkrauna taisyklės kompiliavimui reikalingą komponentą (.dll) bibliotekos failą.
RulesManagerRuleBuilderForm. BuildRule	Metodas	Užkrauto taisyklių failo turinys ir visi su taisyklėmis susiję užkrauti komponentai: taisyklėse naudojamas taisyklių modelį ir kiti išorinius komponentai (.dll) bibliotekų failai, perduodamos „Roslyn“ taisyklių kompiliatoriui.

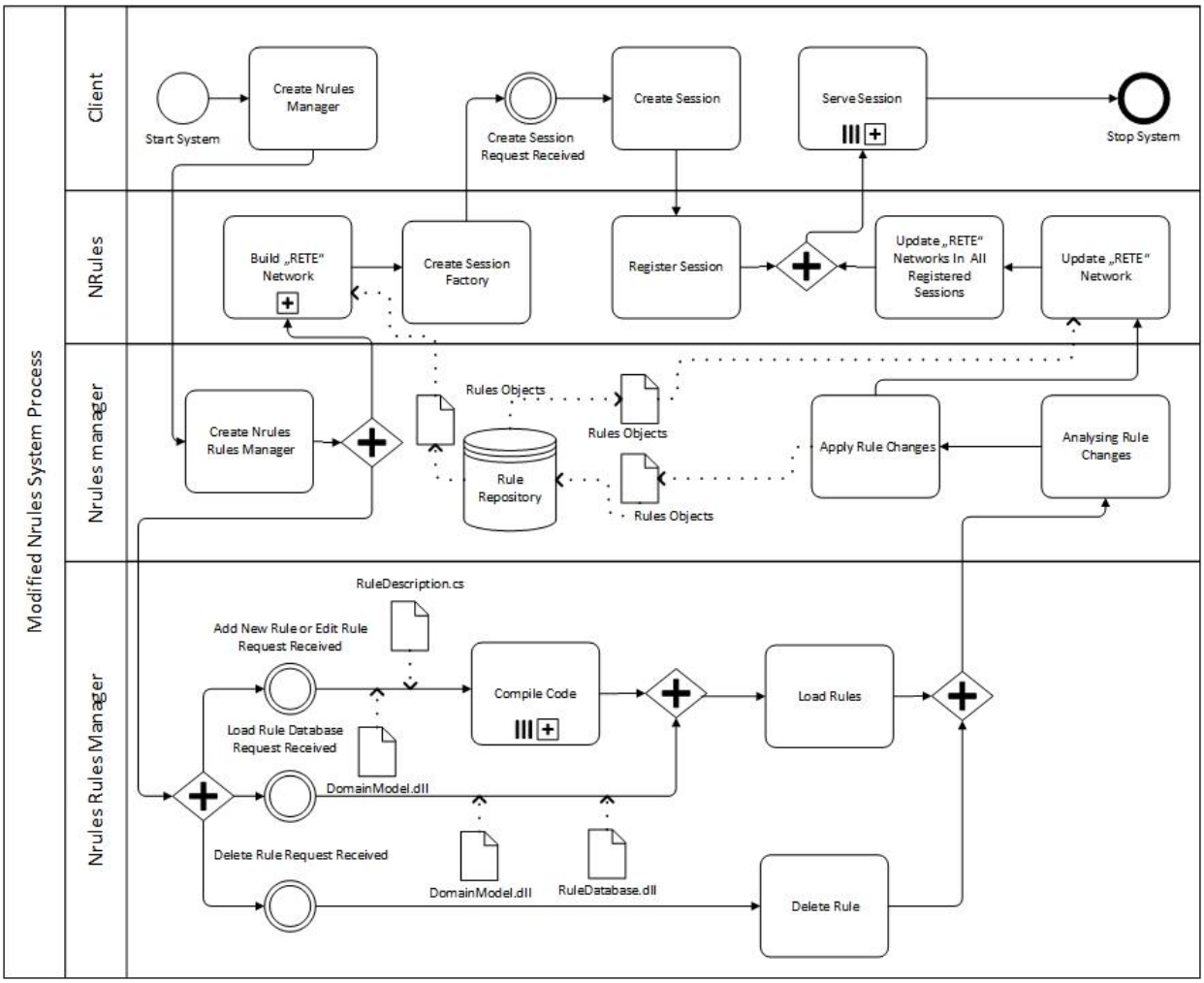
RulesManagerRuleBuilderForm.Delete	Metodas	Pasirinktame kompiliavimui taisyklės faile gali būti daugiau nei viena taisyklė, todėl nereikalingas taisykles galima ištrinti.
RulesManagerRuleBuilderForm.SelectRuleFile	Metodas	Pasirenkamas norimų sukompiliuoti taisyklių failas.
RulesManagerRuleBuilderForm._buildedRules	Atributas	Sukompiliuotų taisyklių rinkinys.
RulesManagerRuleBuilderForm._nRulesRuleManagerForm	Atributas	Taisyklių valdymo modulis, su kuriuo yra apsikeičiama informacija ir komandomis.
RulesManagerRuleBuilderForm._ruleBuilderMode	Atributas	Taisyklių konstruktoriaus režimas. Taisyklių konstruktorius gali būti naudojamas pridėti naujoms taisyklėms arba redaguoti esamoms taisyklėms. Atlikus taisyklės redagavimą, taisyklė yra sukompiliuojama ir atnaujinama pasirinktame taisyklių rinkinyje.
RulesManagerRuleBuilderForm._nRulesRoslynRuleBuilder	Atributas	„NRules“ sistemos taisyklėms pritaikytas „Roslyn“ kompiliatorius.
NRulesRoslynRuleBuilder.BuildLibraryFile	Metodas	Sukompiliuojamas taisyklių rinkinių ir juose esančių taisyklių bibliotekos (.dll) failas.
NRulesRoslynRuleBuilder.BuildRule	Metodas	Naudojant užkrautą taisyklių failo turinį ir taisyklėms aprašyti naudojamus komponentus yra sukompiliuojamos pavienės taisyklės į kanoninę taisyklių formą.



24 pav. „NRules“ taisyklių valdiklio, taisyklių konstruktoriaus ir „RETE“ tinklo vizualizacijos grafinės vartotojo sąsajos

3.7. Atnaujinta „NRules“ sistemos veikimo proceso schema

Remiantis 3.6 skyriuje pristatytais ir įgyvendintais „NRules“ sistemos pakeitimais, sudaryta atnaujinta „NRules“ sistemos veikimo proceso schema, kuri pateikta 25 pav.



25 pav. Atnaujintas „NRules“ sistemos veikimo proceso schema

Lyginant atnaujintą „NRules“ sistemos veikimo proceso schemą su 16 pav. pateikta schema, galime išskirti šiuos esminius pakeitimus:

1. Atnaujinta sistema geba simuliacijos metu, nesukuriant iš naujo sesijų, dinamiškai pridėti naujas ar ištrinti, atnaujinti esamas taisykles.
2. Atsirado galimybė eksportuoti taisyklių bazėje saugomus taisyklių rinkinius į taisyklių bibliotekos (.dll) failą. Taip pat iš sukurtos bibliotekos importuoti išsaugotus taisyklių rinkinius.
3. Sukurtas „NRules“ sistemos taisyklių valdymo modulis su grafine vartotojo sąsaja, kas palengvina verslo žmonėms atlikti pakeitimus su taisyklėmis.

4. Sukurtas „NRules“ sistemos valdymo modulis, leidžia atskirti „NRules“ sistemą nuo DVP simuliacijos įrankių, kas suteikia sistemai moduliarumo ir palengvina integracijas su kitomis sistemomis.
5. Atnaujinant taisykles yra atnaujinama „RETE“ tinklo struktūra, kurią galima pavaizduoti grafiškai.

3.8. Projektinės dalies apibendrinimas

Remiantis suformuluotais reikalavimais siūlomo būdo įgyvendinimui ir sukurta siūlomo būdo koncepcine ir architektūrine komponentų diagramomis, buvo atlikti „NRules“ sistemos pakeitimai ir sukurti nauji „NRules“ sistemos moduliai: „NRules“ sistemos ir taisyklių valdymo moduliai. Siūlomų pakeitimų realizacija parodė, kad dinaminis „RETE“ tinklo valdymas suteikia galimybę efektyviai pridėti naujas ir atnaujinti ar ištrinti esamas taisykles, neperkraunant viso simuliacijos proceso, taip užtikrinant efektyvesnį verslo taisyklių valdymo ir vykdymo būdą.

4. EKSPERIMENTINĖ DALIS

4.1. Reikalavimai eksperimentiniam tyrimui

Atsižvelgiant į projektinėje dalyje pasiūlytam metodui iškeltus reikalavimus, eksperimentiniam tyrimui keliami šie reikalavimai:

1. Turi būti ištirtas įgyvendintų „NRules“ taisyklių vykdymo ir valdymo sistemos pakeitimų dinamiškumas, įvertinant naujų taisyklių pridėjimo ir ištrynimo greitaveiką ir ją palyginti su buvusia sistemos greitaveika, užtikrinant tas pačias sąlygas: vienodi taisyklių ir faktų rinkiniai. Eksperimentams atlikti turi būti parašyti funkciniai testai (angl. *Unit tests*), naudojant „Microsoft Visual Studio“ programos platformą. Greitaveikos eksperimentų rezultatai turi būti pateikti grafiškai.
2. Turi būti įvertintas įgyvendintų „NRules“ sistemos pakeitimų efektyvumas pasirenkant konkrečią dalykinę sritį. Specifikuoti pasirinktos dalykinės srities duomenų bei taisyklių modelį, sudarant dalykinės srities duomenų modeliui klasių diagramą, o taisykles užrašant „NRule“ sistemos „Fluent DSL“ taisyklių forma. Remiantis literatūros analize, sudaromos taisyklės turi būti šių tipų: darnos, išvedimo ir perspėjimo (veiksmą įtakojančios). Eksperimento metu suformuoti faktų rinkinį specifikuotai dalykinės srities modeliui, kuriame būtų tam tikras skaičius taisykles tenkinančių ir netenkinančių faktų. Eksperimento metu operuoti taisyklėmis ir inicijuoti jų vykdymą. Eksperimentui atlikti turi būti parašytas funkcinis testas, o rezultatai turi būti pateikti lentelės ir grafiko pavidalu.

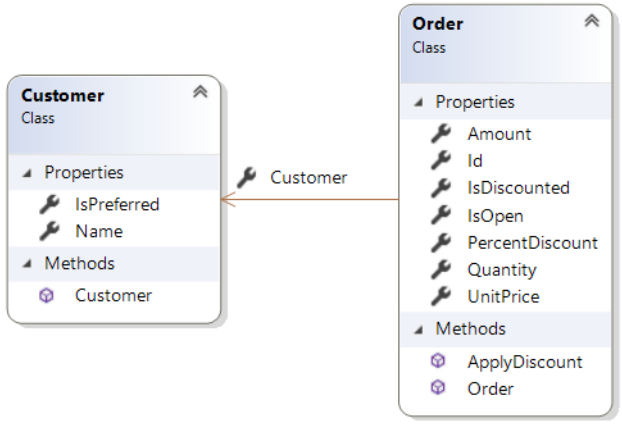
4.2. „NRules“ sistemos pakeitimų dinamiškumo įvertinimas

„NRules“ taisyklių vykdymo ir valdymo sistemos pakeitimų dinamiškumo įvertinimui, buvo parašyti šie funkciniai testai, kurių programinis kodas pateiktas 1 ir 2 prieduose:

1. **Taisyklių pridėjimo testas.** Šio testo metu yra pridedama po vieną taisyklę, kol taisyklių skaičius produkcinėje atmintyje pasiekia 100. Taisyklių užkrovimas yra kartojamas esant skirtingam faktų skaičiui darbinėje atmintyje, pradedant nuo 0 ir didinant kas 100 faktų, kol faktų skaičius pasiekia 1000. Visas testas kartojamas du kartus, vieną kartą naudojant bazines „NRules“ sistemos metodus, o kitą kartą naudojant patobulintus metodus. Abejais atvejais yra naudojami tie patys taisyklių ir faktų rinkiniai. Testų metu yra matuojamas laikas, per kurį taisyklė yra pridedama į „NRules“ sistemos „RETE“ tinklą ir produkcinę sistemos atmintį, bei yra užkraunami kiekvienoje iš sesijų faktai į atnaujintą „RETE“ tinklą.
2. **Taisyklių ištrynimo testas.** Šio testo metu yra atimama po vieną taisyklę, pradedant taisyklių skaičiumi produkcinėje atmintyje lygiu 100 ir baigiant, kai taisyklių skaičius tampa lygus 0. Taisyklių atėmimas yra kartojamas esant skirtingam faktų skaičiui,

pradedant nuo 0 ir baigiant 1000. Faktų kitimo žingsnis yra lygus 100. Testas kartojamas du kartus, naudojant bazinius ir patobulintus metodus. Testų metu yra matuojamas laikas, per kurį taisyklė yra ištrinama iš „NRules“ sistemos „RETE“ tinklo ir produkcinės sistemos atminties. Taisyklių ištrynimo testas, skirtingai nuo taisyklių pridėjimo testo, nereikalauja faktų užkrovimo į atnaujintą „RETE“ tinklą, nes taisyklių ištrynimo metu neatsiranda naujų aktyvuotų taisyklių.

Testuose naudojamos taisyklės ir faktai yra generuojami atsitiktinai. Taisyklėse ir faktuose naudojamo duomenų modelio klasių diagrama pateikta 26 pav., o generuojamų taisyklių sprendimų lentelė pateikta 8 lentelėje.



26 pav. Dinamiškumo testuose naudojamo duomenų modelio klasių diagrama

8 lentelė. Atsitiktinai generuojamų taisyklių sprendimų lentelė

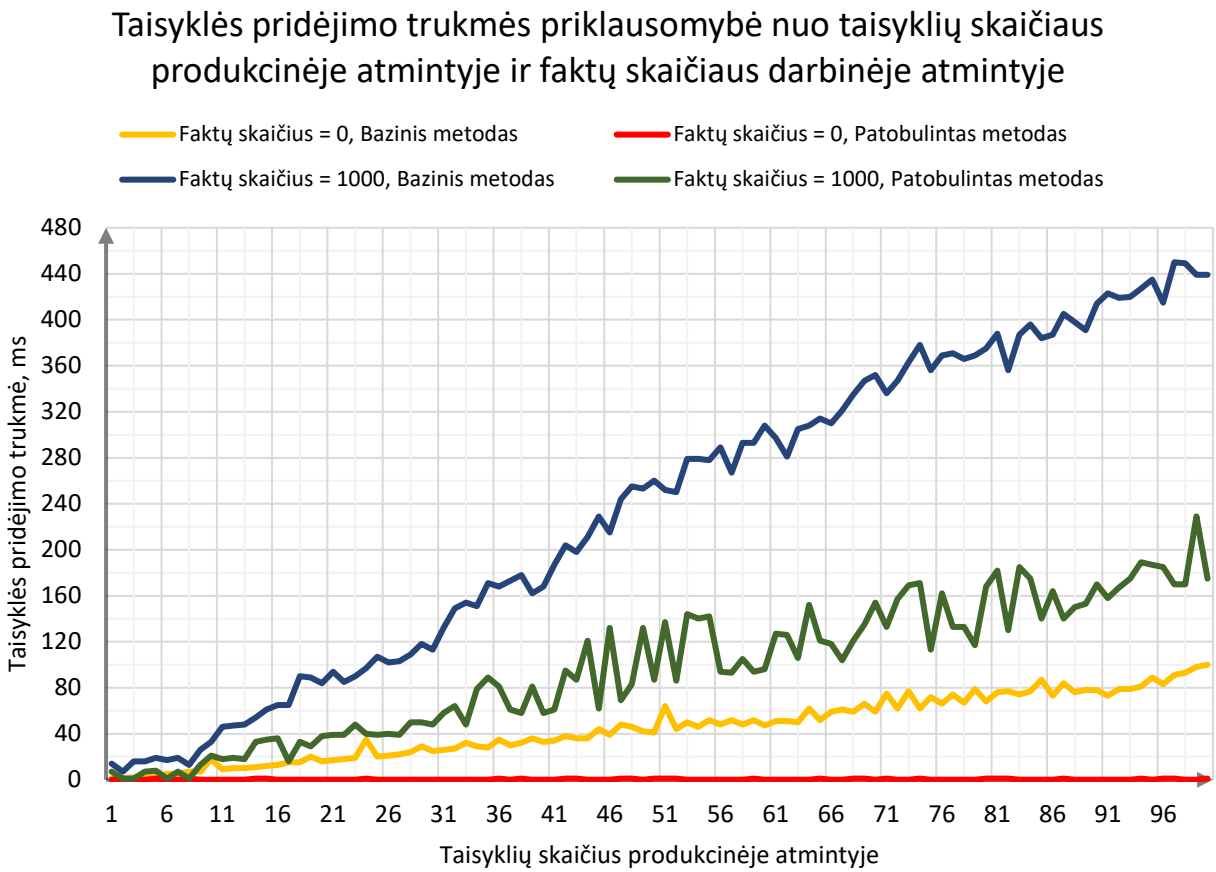
Kliento (angl. *Customer*) atributas (a)	Kliento atributo vertė (b)
Užsakymo (angl. *Order*) atributas (a)	Užsakymo atributo vertė (b)
Užsakymo užsakovas (a)	Klientas
Taikyti nuolaida užsakymui	x

Lentelėje pateiktose sąlygose (a) žymi vieną iš galimų klasės atributų, o (b) atitinkamo atributo vieną iš galimų verčių. Kadangi esant bet kuriai taisyklės sąlygai yra įvykdomas tas pats veiksmas (taikoma nuolaida užsakymui), todėl toliau 27 pav. pateikiame atsitiktinai sugeneruotos ir „C#“ programavimo kalba parašytos taisyklės sąlygos pavyzdį, kurios žodinė išraiška pateikta 2.5 darbo skyriuje.

```
if(((Customer.IsPreferred == true) && (Order.IsDiscounted == false) && (Customer == Order.Customer)) || ((Customer.IsPreferred == false) && (Order.Amount >= 100) && (Customer == Order.Customer)))
```

27 pav. Atsitiktinai sugeneruotos „C#“ programavimo kalba parašytos taisyklės sąlygos pavyzdys

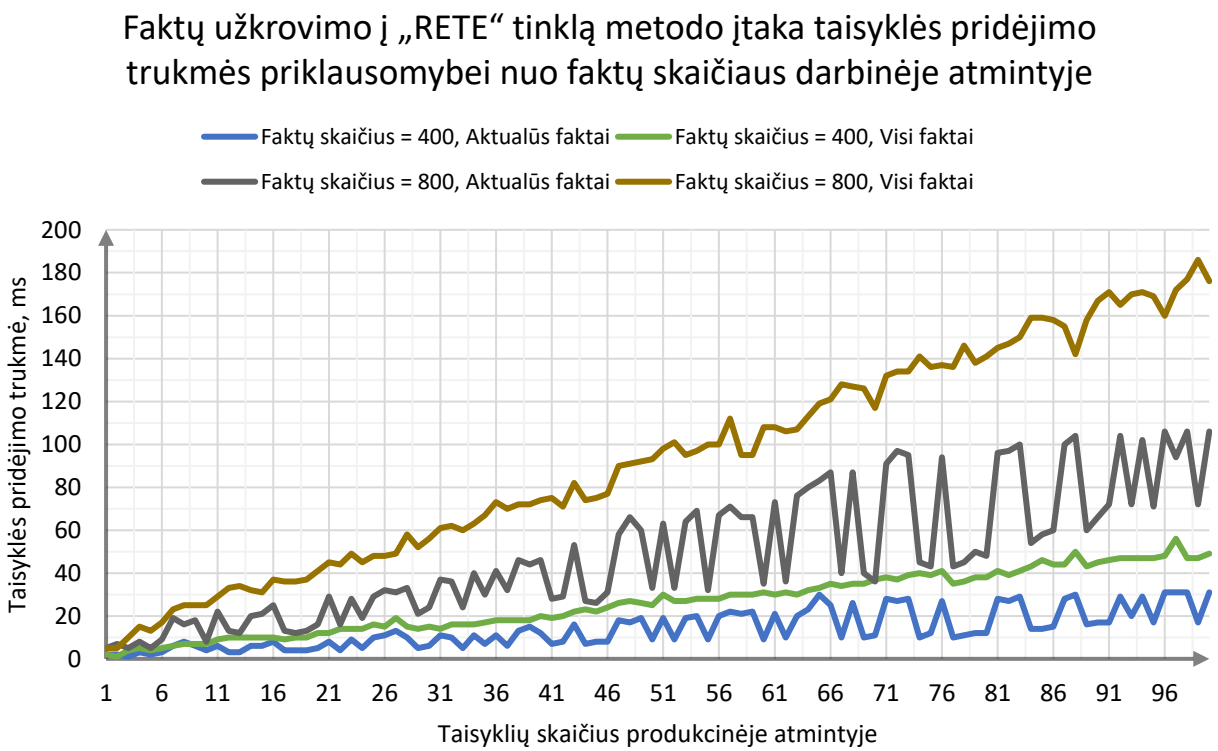
Testuojant tiek bazinius tiek patobulintus taisyklių pridėjimo ir ištrynimo metodus yra naudojamas vienodas taisyklių ir faktų rinkinys. Toliau grafiškai pateiksime ir aptarsime gautus taisyklių pridėjimo ir ištrynimo testų rezultatus.



28 pav. Taisyklės pridėjimo trukmės priklausomybė nuo taisyklių ir faktų skaičiaus, esant baziniam ir patobulintam taisyklių pridėjimo metodui

Pateiktame taisyklių pridėjimo testo rezultatų grafike 28 pav., yra pateikta taisyklės pridėjimo trukmės priklausomybė nuo taisyklių skaičiaus, kai faktų skaičius yra lygus ribinėms testo vertėms: 0 ir 1000. Iš pateikto grafiko matome, kad esant faktų skaičiui lygiam 0, naudojant taisyklių pridėjimui patobulintą metodą, taisyklės pridėjimo trukmė svyruoja nuo 0 iki 2 ms. Tai reiškia, kad taisyklės pridėjimo trukmė nuo taisyklių skaičiaus produkcinėje atmintyje nepriklauso. Naudojant bazinį metodą, taisyklės pridėjimo trukmė didėja, didėjant taisyklių skaičiui produkcinėje atmintyje. Taisyklių skaičiui esant lygiam 99, taisyklės pridėjimo trukmė baziniu metodo atveju yra lygi 100 ms, o patobulinto metodo atveju yra lygi 1 ms. Tai reiškia, kad esant prieš tai išvardintoms sąlygoms, patobulinto metodo atveju, taisyklė yra 100 kartų greičiau pridedama, nei bazinio metodo atveju. Padidinus faktų skaičių iki 1000, tiek bazinio tiek patobulinto taisyklių pridėjimo metodo atveju, atsirado tiesioginė taisyklės pridėjimo trukmės priklausomybė nuo taisyklių skaičiaus produkcinėje atmintyje. Bazinio metodo atveju ši priklausomybė yra didesnė ir esant faktų skaičiui lygiam 1000, o

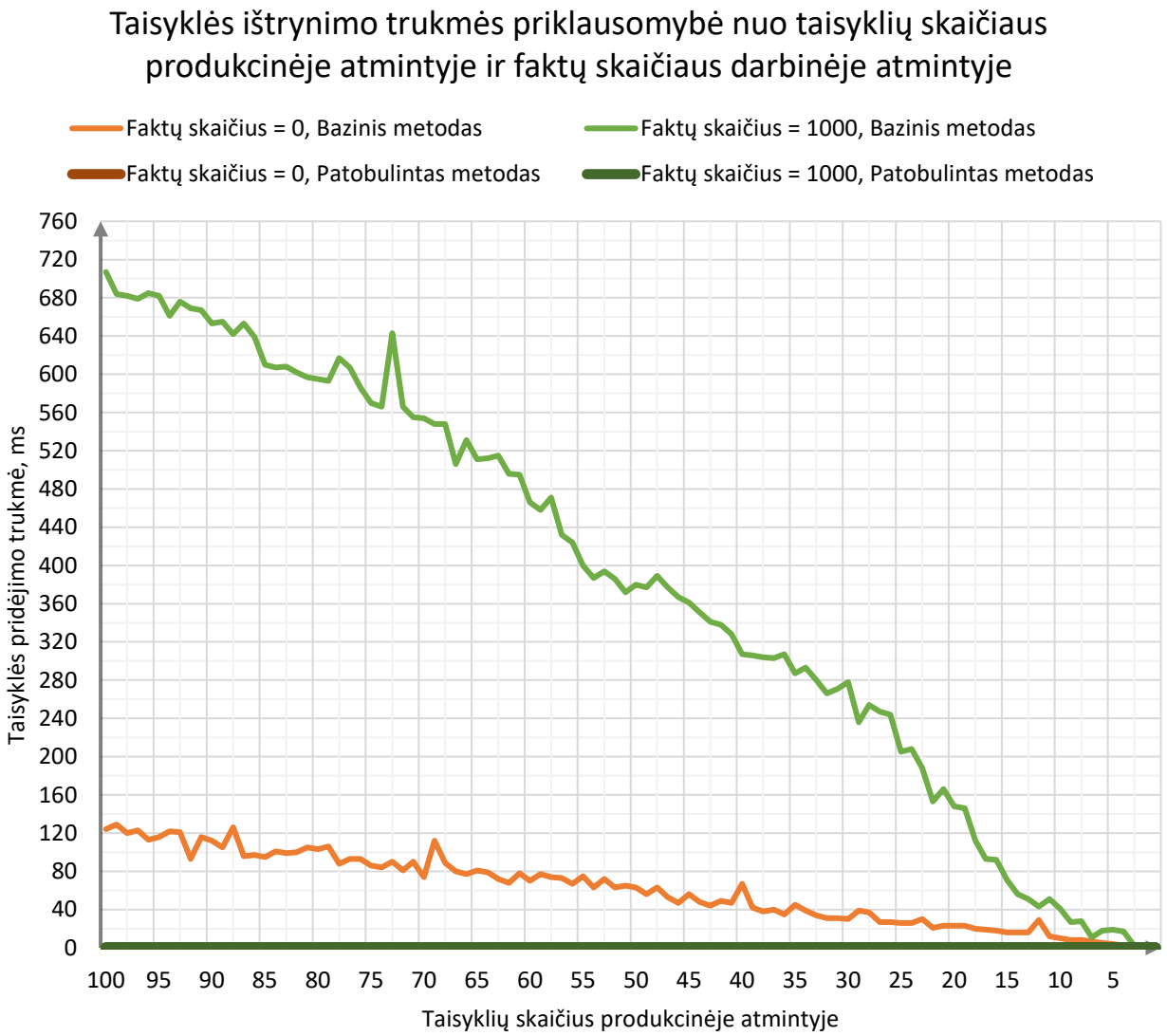
taisyklių skaičiui 99, taisyklės pridėjimo trukmė yra lygi 439 ms, kai patobulinto metodo atveju siekia 175 ms. Tai yra 2,5 karto greičiau nei bazinio metodo atveju. Patobulinto metodo priklausomybės kreivė yra netolygiai didėjanti, taip yra todėl, kad pridedant naują taisyklę į atnaujintą „RETE“ tinklą yra užkraunami iš darbinės atminties tik su taisykle susiję faktai. Skirtingo tipo faktų skaičius darbinėje atmintyje yra nevienodas, todėl pridėti naujas taisykles su kuriomis yra susiję mažiau faktų, užtrunka trumpiau. Šiai priklausomybei įvertinti, atliktas papildomas testas, kurio rezultatų grafikas pateiktas 29 pav.



29 pav. Faktų užkrovimo į „RETE“ tinklą metodo įtaka taisyklės pridėjimo trukmės priklausomybei nuo taisyklių ir faktų skaičiaus, patobulintam taisyklių pridėjimo metodui

Pateiktame grafike pavaizduoti testo rezultatai, kuris buvo atliktas naudojant patobulintą taisyklių pridėjimo metodą. Šio testo metu buvo matuojamas laikas per kurį nauja taisyklė pridedama esant skirtingam faktų skaičiui. Testas buvo atliekamas naudojant du skirtingus faktų užkrovimo metodus. Vienas iš metodų užkraudavo į „RETE“ tinklą visus darbinėje atmintyje esančius faktus, o kitas metodas tik tuos faktus kurie buvo aktualūs naujai pridėtai taisyklei. Iš gautų rezultatų matome, kad esant tam pačiam faktų skaičiui ir užkraunant tik aktualius faktus, naujos taisyklės pridėjimas užtrunka mažiau, lyginant su visų faktų užkrovimu. Vidutiniškai vienos taisyklės pridėjimas užkraunant visus faktus trunka 62 ms laiko, kai tuo tarpu užkraunant tik aktualius faktus užtrunka 27 ms laiko, o tai yra 2.3 karto greičiau.

Atliekant taisyklių ištrynimo testą, kurio rezultatų grafikas pateiktas 30 paveiksle, buvo užkrautas maksimalus taisyklių skaičius ir trinama po vieną taisyklę, kol taisyklių skaičius pasiekia 0, esant ribinėms faktų vertėms: 0 ir 1000 faktų. Iš gautos priklausomybės grafiko, pastebėta, kad patobulinto taisyklių ištrynimo metodo atveju, taisyklės ištrynimo trukmė nepriklauso nuo taisyklių skaičiaus produkcinėje atmintyje ir faktų skaičiaus darbinėje atmintyje. Kai tuo tarpu bazinio metodo atveju, taisyklės ištrynimo trukmė tiesiogiai priklauso nuo taisyklių ir faktų skaičiaus, Kuo didesnis taisyklių ir faktų skaičius, tuo didesnė taisyklės ištrynimo trukmė.



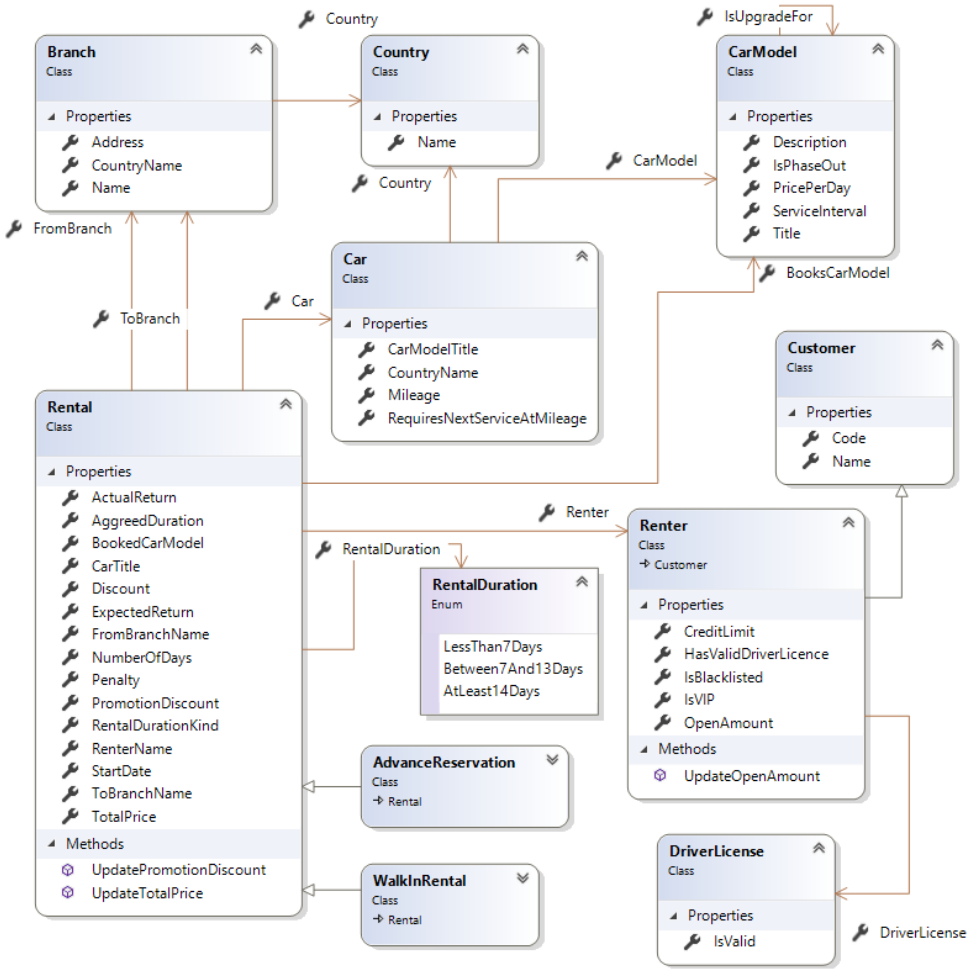
30 pav. Taisyklės ištrynimo trukmės priklausomybė nuo taisyklių ir faktų skaičiaus, esant baziniam ir patobulintam taisyklių pridėjimo metodui

4.3. „NRules“ sistemos pakeitimų efektyvumo ir teisingumo tyrimas

„NRules“ taisyklių vykdymo ir valdymo sistemos pakeitimų efektyvumo ir teisingumo įvertinimui, bus atliekamas eksperimentas, kuriam atlikti bus pasirinkta reali dalykinė sritis, sudaromos šiai dalykinei sričiai aktualios taisyklės ir faktų rinkiniai. Remiantis prieš tai atliktų eksperimentų rezultatais ir išsiaiškinus, kad atlikus pakeitimus „NRules“ sistemoje, taisyklių ištrynimo trukmė nepriklauso nuo taisyklių ir faktų skaičiaus, nuspręsta, kad šio eksperimento metu bus nagrinėjama tik taisyklių ir faktų skaičiaus įtaka, taisyklių pridėjimo trukmei.

4.3.1. Dalykinės srities aprašymas

„NRules“ taisyklių vykdymo ir valdymo sistemos pakeitimų efektyvumo ir teisingumo įvertinimui, bus atliekamas eksperimentas, kurio metu pasirinktas mokslinėje literatūroje plačiai nagrinėjamas automobilių nuomos dalykinės srities pavyzdys „EU-Rent“ [44]. Remiantis šiuo pavyzdžiu buvo sudarytas dalykinės srities duomenų modelio klasių diagrama, kuri pateikta 31 pav.



31pav. „EU-Rent“ dalykinės srities duomenų modelio klasių diagrama

Pasirinkta dalykinė sritis – tai automobilių nuomos verslas, turintis savo veiklos filialų (angl. *branch*) įvairiose Europos Sąjungos šalyse. Dalykinėje srityje nagrinėjamas pagrindinis įvykis: automobilio modelio rezervacija, kurio pagrindu sudaromi įvairūs faktai tolimesniam eksperimentui.

4.3.2. Dalykinės srities verslo taisyklių rinkinys

Automobilių nuomos sąlygos apibrėžiamos verslo taisyklių rinkiniu. Šiame eksperimente buvo panaudotos minėtame pavyzdyje aptariamos taisyklės. Taisyklės buvo pritaikytos, prie 31 pav. pateikto, dalykinės srities duomenų modelio įgyvendinimo. Taisyklės aprašymas pateiktas 9 lentelėje anglų kalba, siekiant neiškreipti analizuojamos dalykinės srities supratimo, o „Fluent DSL“ taisyklių forma parašytų taisyklių programinis kodas pateiktas 3 priede.

9 lentelė. Dalykinės srities verslo taisyklės

Nr.	Taisyklės pavadinimas	Taisyklės tipas	Prioritetas	Aprašymas
1	Rental Uses At Most One Car	Darnos	10	It is necessary that each rental uses at most one car.
2	Rental Has Exactly One Signee	Darnos	10	It is necessary that each rental is signed by exactly one renter.
3	Rental Books Exactly One Car Model	Darnos	10	It is necessary that each rental books exactly one car model.
4	Renter Open Amount Is Sum Of Total Price Of Each Signed Rental	Išvedimo	10	It is necessary that the open amount of each renter is sum of the total price of each rental that is signed by the renter.
5	Rental Total Preliminary Price Computation Rule	Išvedimo	5	It is necessary that the total price of each rental is the number of days * price per day of the car model that is booked by that rental * (100 - discount of that rental) / 100 - promotion discount of that rental.
6	Rental Total Delayed Price Computation Rule	Išvedimo	5	It is necessary that the total price of each rental is increased by a penalty of 20 * (actual return - expected return), if actual return is after expected return.

7	Rental Total Early Price Computation Rule	Išvedimo	7	It is necessary that the total price of each rental is decreased by a penalty of 5 * (expected return - actual return), if actual return is before than expected return.
8	Rental Promotion Discount For Audi	Išvedimo	7	If the car model of the rental is Audi and the start date of the rental is within year, then set the promotion discount of the rental to the agreed duration of the rental * 5 and display the message: The rental is eligible for the Audi promotion.
9	Rental Promotion Discount For Bmw	Išvedimo	10	If the car model of the rental is BMW and the start date of the rental is within this year, then set the promotion discount of the rental to the agreed duration of the rental * 7 and display the message: The rental is eligible for BMW promotion.
10	Rental Has Minimum Mileage	Darnos	10	It is obligatory that a rental uses a car that is available and that rental books a car model and that car is an instance of that car model and that car has mileage that is minimum of mileage of each car that is available.
11	Rental Car	Darnos	10	If rental uses car and car is in current location and car belongs to branch and branch is in owning country and current location is same as owning country, then rental to destination branch and destination branch is in owning country.
12	Rental Renter Is Blacklisted	Darnos	10	It is prohibited that a renter has signed a rental, if that renter is blacklisted.
13	Rental Car Model Is Phased Out	Darnos	10	It is prohibited that a rental books a car model, if that car model is phased-out.
14	Renter Open Amount Vs Credit Limit	Darnos	10	It is prohibited that a renter has an open amount that is larger than the credit limit of that renter.
15	Renter Has Valid Driver Licence	Darnos	10	It is obligatory that a renter has a driver license that is valid.

16	Renter Is VIP Client	Perspėjimo	10	It is recommended that a renter who is VIP should receive company gifts.
17	Rental Is At Least 14 Days	Perspėjimo	10	It is recommended that a rental which is at least 14 days should receive proper attention.

4.3.3. Dalykinės srities faktų rinkinys

Dalykinės srities faktų rinkinys buvo sudarytas remiantis toliau pateiktais faktų sudarymo reikalavimais:

1. Turi būti sudaryta skirtingų automobilių modelių aibė sudaryta nemažiau nei iš 18-os skirtingų automobilių modelių;
2. Turi būti sudaryta aibė padalinių visose Europos Sąjungos šalyse;
3. Turi būti kiekviename iš filialų ne mažiau nei 50 automobilių, bei būtų panaudoti visi automobilių modeliai;
4. Turi būti sudaryta aibė nuomininkų, kad būtų tenkinami šie reikalavimai:
    a. Nuomininkai sudaryti iš skirtingų šių savybių kombinacijų: „*IsBlacklisted*“, „*IsVip*“, „*DriverLicense.IsValid*“;
    b. Nuomininkų skaičius artimas automobilių skaičiui.
5. Turi būti sudaryta aibė automobilių nuomos faktų.

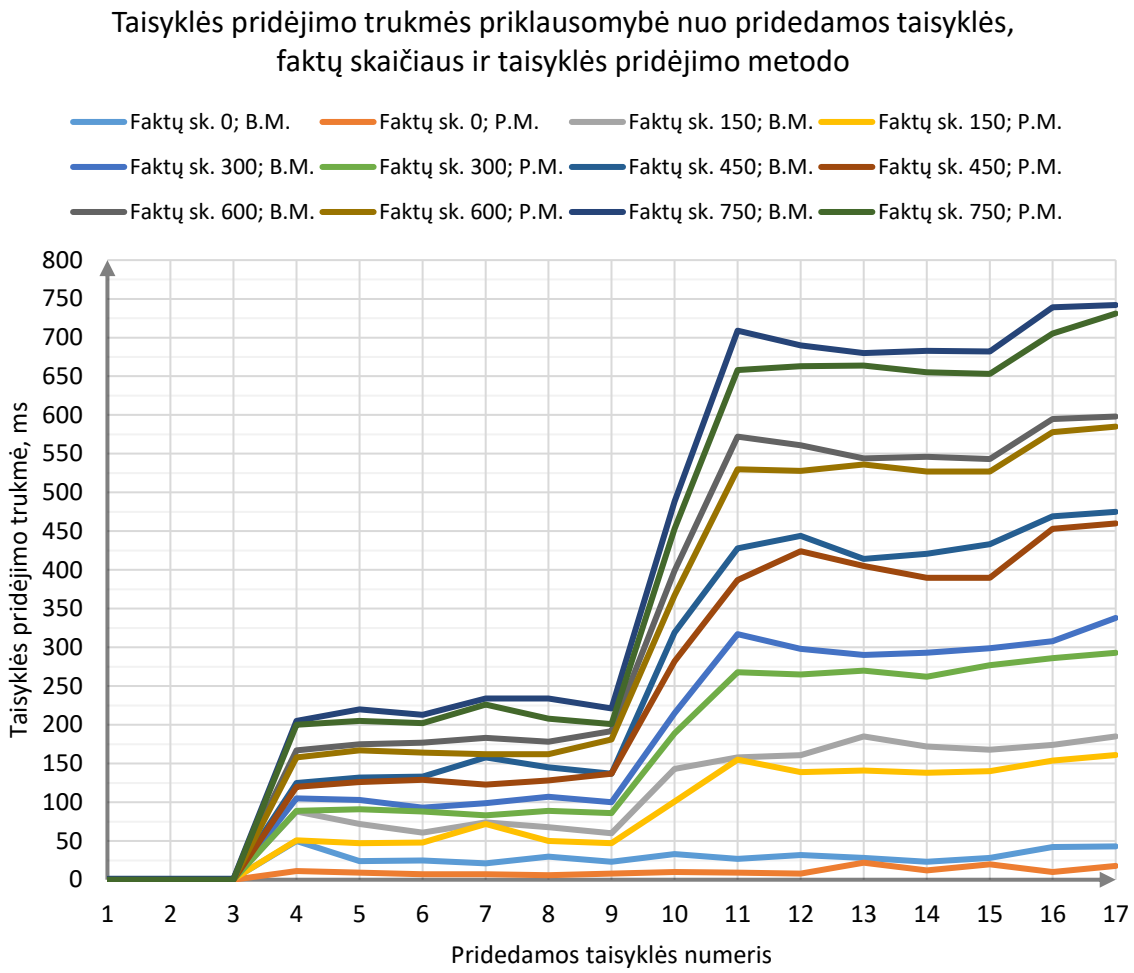
Remiantis suformuluotais reikalavimais, buvo sudaryti faktų rinkiniai, kurių fragmentai pateikti 4 priede.

4.3.4. Eksperimento vykdymas

Eksperimentui atlikti buvo parašytas funkcinis testas. Eksperimento metu buvo užkrauti faktai: aibė automobilių modelių (18 automobilių modelių), aibė šalių (28 šalys), aibė padalinių (112 padalinių), aibė nuomininkų (1000 nuomininkų) ir automobilių aibė (951 automobiliai). Automobilių nuomų faktai užkraunami po 150 faktų, pradedant nuo 0 faktų ir didinant faktų skaičių tol, kol faktų skaičius atmintyje pasiekia 750 faktų. Esant skirtingiems automobilių nuomos faktų skaičiams, buvo pridedama po vieną 9 lentelėje pateiktą taisyklę ir matuojamas taisyklės pridėjimo laikas, naudojant patobulintą ir bazinį „NRules“ taisyklių pridėjimo metodus. Taip pat viso eksperimento metu buvo stebima ar skirtingų metodų atveju, aktyvuotų ir įvykdytų taisyklių skaičius ir faktų skaičius darbinėje atmintyje, po taisyklių įvykdymo, buvo vienodas. Netenkinant vienos iš šių sąlygų, laikoma, kad eksperimentas įvykdytas nesėkmingai. Eksperimento rezultatai pateikti 10 lentelėje ir 32 pav.

10 lentelė. Taisyklės pridėjimo trukmės priklausomybė nuo pridedamos taisyklės, faktų skaičiaus ir taisyklės pridėjimo metodo B. m.(Bazinis metodas) ir P. m. (Patobulintas metodas)

	Faktų skaičius	**0**		**150**		**300**		**450**		**600**		**750**	
	Taisyklės pridėjimo metodas	**B. m.**	**P. m.**	**B. m.**	**P. m.**	**B. m.**	**P. m.**	**B. m.**	**P. m.**	**B. m.**	**P. m.**	**B. m.**	**P. m.**
		Taisyklės pridėjimo trukmė, ms											
Pridedamos taisyklės numeris	**1**	1	0	1	0	1	0	1	0	1	0	1	0
	2	1	0	1	0	1	0	1	0	1	0	1	0
	3	1	0	1	1	1	0	1	0	1	0	1	0
	4	50	11	88	51	105	89	125	120	167	158	205	200
	5	24	9	72	47	103	91	132	126	175	167	220	205
	6	25	7	61	48	93	88	133	129	177	164	213	202
	7	21	7	74	72	99	83	158	123	183	162	234	226
	8	30	6	68	50	107	89	145	128	178	162	234	208
	9	23	8	60	47	100	86	137	137	192	181	221	201
	10	33	10	143	101	215	189	319	282	399	367	488	453
	11	27	9	158	155	317	268	428	387	572	530	709	658
	12	32	8	161	139	298	265	444	424	561	528	690	663
	13	28	22	185	141	290	270	414	405	544	536	680	664
	14	23	12	172	138	293	262	421	390	546	527	683	655
	15	28	20	168	140	299	277	433	390	543	527	682	653
	16	42	10	174	154	308	286	469	453	595	578	739	705
	17	43	18	185	161	338	293	475	460	598	585	742	731



32 pav. Taisyklės pridėjimo trukmės priklausomybė nuo pridedamos taisyklės, faktų skaičiaus ir taisyklės pridėjimo metodo B. m. (Bazinis metodas) ir P. m. (Patobulintas metodas)

4.3.5. Eksperimento rezultatai

Remiantis gautais eksperimento rezultatais, 10 lentelė ir 32 pav., galime teigti, kad taisyklės pridėjimo rezultatai, naudojant patobulintą taisyklės pridėjimo metodą dinamiškai plečiant „RETE“ tinklą, yra nežymiai geresni, nei naudojant bazinį taisyklių pridėjimo metodą. Efektyvumas išaugo 1,4 karto, lyginant su pirmame eksperimente gautais rezultatais, kai efektyvumas siekė 2,5 karto. Taip yra todėl, kad pridedamų naujų taisyklių skaičius nėra didelis, o patobulinto taisyklių pridėjimo metodas yra skirtas efektyviai pridėti didelį kiekį taisyklių. Taip pat atliekant eksperimentą pastebėta, kad pridedant tokias taisykles, kurių sąlygoms sudaryti yra naudojami skirtingų faktų tipų, siejasi su didžiąja dalimi faktų, esančių darbinėje atmintyje. O tai reiškia, kad visus susijusius faktus reikia atnaujinti – užkrauti juos į „RETE“ tinklą. Fakto atnaujinimo operaciją sudaro du veiksmai: fakto ištrynimas ir užkrovimas į „RETE“ tinklą. Todėl esant tokioms taisyklėms efektyviau yra ištrinti „RETE“ tinklo alfa ir beta atminties mazguose esančius faktus ir užkrauti visus faktus iš darbinės

faktų atminties. Pateiktame rezultatų grafike, pastebimi taisyklių pridėjimo trukmės padidėjimai, esant 10 ir 11 taisyklėms. Taip yra todėl, kad taisyklės siejasi su didžiąja dalimi faktų, kuriuos užkrauti į „RETE“ tinklą užtrunka laiko. Tai akivaizdžiai matosi 9 lentelėje pateiktame taisyklių aprašyme. Esant faktų skaičiui lygiam 0, šių aptartų taisyklių užkrovimo trukmės neišsiskiria nuo kitų taisyklių pridėjimo trukmės. Tai patvirtina su taisyklę susijusių faktų skaičiaus įtaką, taisyklių pridėjimo trukmei.

4.4. Eksperimentinės dalies apibendrinimas

Atlikti eksperimentai parodė pasiūlytų „NRules“ sistemos pakeitimų efektyvumą, teisingumą, patogumą ir pritaikymą dinaminių verslo procesų simuliacijoje. Buvo įrodyta, kad dinaminis „RETE“ tinklo valdymas pagreitina naujų taisyklių pridėjimą ir visiškai eliminuoja laiko resursų naudojimą taisyklėms ištrinti. Atliekant „NRules“ sistemos pakeitimų dinamiškumą įvertinančius eksperimentus, buvo gauti rezultatai, kurie parodė, kad taisyklės pridėjimo trukmę vidutiniškai pavyko sumažinti 2,5 karto. Atliekant „NRules“ sistemos efektyvumą ir teisingumą įvertinančius eksperimentus, buvo pasirinkta reali dalykinė sritis, sudarytas dalykinės srities verslo taisyklių rinkinys ir atsitiktinai sugeneruoti faktai. Eksperimentų rezultatai įrodė atnaujintos „NRules“ sistemos efektyvumą ir teisingumą. Įgyvendinti „NRules“ sistemos pakeitimai, leidžia kurti naujas ar keisti esamas taisykles bei jas aktyvuoti neperkraunant viso simuliacijos proceso.

DARBO REZULTATAI

1. Atlikus dalykinės srities sąvokų analizę susipažinta su dinaminiais verslo procesais, jų tipais. Nustatyti dinaminiams verslo procesams keliami reikalavimai. Susipažinta su verslo procesų simuliacija, verslo procesų dinamiškumo lygiais, simuliacijos gyvavimo ciklu. Nustatyti pagrindiniai DVP simuliacijos būdai. Be to, susipažinta su simuliacijose naudojamais įvykių ir taisyklių tipais.
2. Atlikus literatūros analizę, susijusią su verslo taisyklėmis, susipažinta su „GUIDE“ projekte pateikiamais verslo taisyklių tipais. Taip pat išsiaiškinta, kokius kriterijus turi tenkinti verslo taisyklės, kad būtų suprantamos ir turėtų prasmę. Susipažinta su taisyklių aprašymo būdais: „SBVR“, „OCL“ kalbomis ir sprendimų lentelėmis.
3. Atlikus taisyklių valdymo ir vykdymo sistemų analizę, susipažinta su šių sistemų privalumais ir savybėmis, šioms sistemoms keliamais reikalavimais. Išanalizuota šių sistemų architektūros bendriausias atvejis. Taip pat pasirinktais kriterijais atlikta 5-ių sistemų lyginamoji analizė.
4. Projektinėje dalyje buvo išanalizuota DVP simuliacijos įrankio ir „NRules“ sistemos esama situacija. Identifikuotos „NRules“ sistemos dinamiškumo problemos. Remiantis suformuotais reikalavimais taisyklių vykdymo ir valdymo sistemai, buvo pasiūlytas būdas „NRules“ sistemos dinamiškumo problemoms spręsti. Taip pat pasiūlyta „NRules“ sistemos integracija į DVP simuliacijos įrankį, pateikiant „NRules“ sistemos pakeitimų ir integracijos į „DBPSim“ įrankį architektūros komponentų diagramą. Atlikti „NRules“ sistemos pakeitimai, kurie tenkina suformuluotus reikalavimus taisyklių vykdymo ir valdymo sistemai.
5. Eksperimentinėje dalyje suformuoti reikalavimai eksperimentiniam tyrimui. Remiantis suformuotais reikalavimais, atlikti „NRules“ sistemos pakeitimų dinamiškumą patikrinantys ir įrodantys eksperimentai. Taip pat pasirinkus realią dalykinę sritį, buvo sudarytas dalykinės srities verslo taisyklių rinkinys ir atsitiktinai sugeneruoti faktai, kurie buvo panaudoti „NRules“ sistemos efektyvumą ir teisingumą patikrinančiame ir įrodančiame eksperimente.

IŠVADOS

1. Atlikus su dinaminiais verslo procesais ir jų simuliacija susijusios literatūros analizę, nustatyta, kad daugiausia verslo procesų simuliacijos įrankių yra įgyvendinę žemiausią dinamiškumo lygį, kai yra naudojami sprendimų taškai, kuriuose žmogus ar automatizuota sistema nusprendžia ką daryti toliau, remiantis apibrėžtomis taisyklėmis. Aukščiausiame lygmenyje dinamiškumas yra orientuotas į verslo proceso tikslą, kuris gali būti keičiamas bet kuriuo verslo proceso simuliacijos metu.
2. Išanalizavus ir nustačius pagrindinius DVP simuliacijos būdus, nustatyta, kad įvykiais grindžiamas DVP simuliacijos būdas leidžia verslo procesus sudarančias veiklas susieti silpnais ryšiais, taip padidinat simuliacijai skirtų modelių plečiamumo galimybes. Taisyklėmis grindžiamų DVP simuliacijos būdas leidžia atskirti dalykinės srities žinias nuo veiklos procesų ir programų.
3. Išanalizavus verslo taisyklių tipus, nustatyta, kad veiksmo taisyklės labiausiai tinka dinaminiams verslo aspektams išreikšti.
4. Susipažinus su verslo taisyklių specifikavimo būdais, pastebėta, kad iš visų išanalizuotų taisyklių specifikavimo būdų, SBVR turi didžiausią išraiškingumą ir pateikia būdą kaip aprašyti taisykles struktūrizuota anglų kalba ir išreikšti jas formalia logika taip, kad jas būtų galima apdoroti automatizuotai.
5. Susipažinus su taisyklių valdymo ir vykdymo sistemomis, joms keliamais reikalavimais ir architektūra, pasirinktais kriterijais: sistema atviro kodo, palaikomos produkcinės taisyklės, kūrimo aplinka „.NET“, buvo atlikta pasirinktų sistemų lyginamoji analizė. Pastebėta, kad visos aptartos sistemos gali aprašyti ir vykdyti produkcines taisykles, tačiau dauguma verslo taisyklių valdymo sistemų naudoja savo sukurtas verslo taisyklių rašymo kalbas, kas labai apsunkiną skirtingų sistemų suderinimą tarpusavyje. Atlikus lyginamąją analizę, nustatyta, kad pasirinktus kriterijus atitiko vienintelė „NRules“ sistema.
6. Atskyrus „NRules“ sistemos taisyklių valdymą ir vykdymą, taisyklių valdymui buvo sukurta grafinė vartotojo sąsaja, kuri leidžia patogiai valdyti taisyklių rinkinius, juos sudarant iš naujų pridedamų taisyklių arba juos išsaugant į taisyklių biblioteką, kas leidžia šias taisyklių bibliotekas pakartotinai panaudoti. Atnaujinta „RETE“ tinklo vizualizacijos priemonė leidžia atvaizduoti ne tik esamas taisykles, bet ir ištrintas.
7. Pasiūlius ir įgyvendinus „NRules“ sistemos pakeitimus, skirtus dinamiškumo problemoms išspręsti, pastebėta, kad dinaminis „RETE“ tinklo valdymas suteikia galimybę efektyviai pridėti naujas ir atnaujinti ar ištrinti esamas taisykles, neperkraunant viso simuliacijos proceso.

8. Eksperimentiniai tyrimai parodė įgyvendintų „NRules“ sistemos pakeitimų efektyvumą ir teisingumą operuojant dideliais taisyklių kiekiais. Pasiūlyti ir įgyvendinti pakeitimai leido žymiai pagreitinti naujų taisyklių pridėjimą ir esamų atnaujinimą, o taisyklių ištrynimo trukmę sumažinti iki nykstamai mažos vertės.
9. Eksperimentinių tyrimų metu pastebėta, kad taisyklės pridėjimo trukmė tiesiogiai priklauso nuo su taisyklę susijusių faktų skaičiaus darbinėje atmintyje. Kuo susijusių faktų yra daugiau, tuo taisyklės pridėjimas užtrunka ilgiau.
10. Pridedant ar atnaujinant taisykles yra atnaujinami „RETE“ tinkle esantys faktai užkraunant su pridedamomis ar atnaujinamomis taisyklėmis susijusius faktus. Šis susijusių faktų užkrovimas yra efektyvus tik tuo atveju, jeigu su taisyklėmis susijusių faktų skaičius yra mažas, palyginus šį skaičių su faktų skaičiumi esančiu darbinėje atmintyje. Jeigu didžioji dalis faktų yra susiję su taisyklėmis, tai efektyviau yra užkrauti visus darbinėje atmintyje esančius faktus į atnaujintą „RETE“ tinklą, prieš tai išvalius tinklo alfa ir beta atminties mazgų atmintis.
11. Naudojantis esamais „NRules“ sistemos funkciniais testais (angl. *Unit tests*), nustatyta, kad atlikti pakeitimai neigiamos įtakos likusiai sistemai netūrėjo.

PASIŪLYMAI

Esamoje situacijoje „NRules“ sistemoje pridėjus naują ar atnaujinus esamą, nėra saugoma informacija kokie nauji „RETE“ tinklo mazgai buvo sukurti ir su kuriais jau esamais mazgais nauja taisyklė siejasi, todėl su taisykle susiję faktai keliauja per visus su faktais susijusius mazgus, o tai užtrunka laiko. Todėl norint užkrauti faktus tik į tuos mazgus, su kuriais taisyklė siejasi yra reikalinga susijusių mazgų paieška. Šiai paieškai pagreitinti, siūloma saugoti kiekvienos taisyklės individualų „RETE“ tinklą, kuriuos sujungus būtų gaunamas bendras visų taisyklių „RETE“ tinklas. Tokiu būdu būtu galima lengvai pridėti naujas taisykles ir užkrauti su jomis susijusius faktus. Taip pat lengvai ištrinti taisykles ne tik pakeičiant jų statusą į neaktyvias, bet ir ištrinant tik su jomis susijusius mazgus. Šių pasiūlymų įgyvendinimas leistų paspartinti taisyklių pridėjimą, bei sutaupytų sistemos resursų, ištrinant taisykles ir su jomis susijusią informacija „RETE“ tinkle.

Sistemos mobilumui ir greitaveikai pagerinti yra siūloma taisyklių saugojimui naudoti duomenų bazę, kurioje būtu saugomos ne tik esamu laiku sistemos naudojamos taisyklės, bet ir naudotos ir žadamos naudoti taisyklės. Įgyvendintame taisyklių valdymo modulyje, yra galimybė sistemos išjungimo metu aktualias taisykles išsaugoti į taisyklių bibliotekas ir taip pat sistemos paleidimo metu iš šių bibliotekų užkrauti. Esant dideliems taisyklių kiekiams šios operacijos užtrunka daug laiko. Duomenų bazė taip pat galėtų saugoti atsargines aktualių taisyklių kopijas, tokiu būdu apsisaugant nuo duomenų praradimo.

Taisyklių įvedimo galimybėms praplėsti siūloma įgyvendinti naujų taisyklių įvedimą naudojant sprendimų lenteles. Taip pat yra reikalingas naujų taisyklių patikros reikalavimų praplėtimas, nes šiuo metu yra atliekamas tik taisyklės pavadinimo patikrinimas.

DISKUSIJA

Šiame skyriuje, nepaisant to, kad atliktų eksperimentų rezultatai įrodo „NRules“ sistemos pakeitimų teisingumą ir efektyvumą, žemiau apžvelgsime grėsmes siūlomo sprendimo validumui (angl. *Threats to validity*):

1. Kas jeigu „NRules“ pakeitimai netinkamai atlikti?
    a. Idėja ir projektas buvo aprobuoti vadovo bei kitų VGTU mokslininkų;
    b. „NRules“ pakeitimai buvo ištestuoti atliekant „NRules“ sistemos autorių sukurtus funkcinius testus (angl. *Unit tests*), kurių rezultatai parodė, kad atlikti pakeitimai neigiamos įtakos likusiai sistemai netūrėjo;
    c. Taip pat „NRules“ sistemos pakeitimai buvo ištestuoti, naudojant pakeitimams įvertinti parašytus funkcinius testus.
2. Kas jeigu eksperimentai suplanuoti ir įvykdyti netinkamai?
    a. Eksperimentai suplanuoti kartu su vadovu, remiantis metodų efektyvumo matavimo gairėmis;
    b. Atliktais eksperimentais buvo siekiama patikrinti hipotezę, kad dinaminis „RETE“ tinklo valdymas „NRules“ taisyklių variklio sesijose (t.y. išlaikant tą pačią atmintį skirtingose sesijose) yra efektyvesnis nei sesijų sukūrimas iš naujo, keičiantis taisyklėms;
    c. Atlikti eksperimentai apima tiek atsitiktinai sukurtas taisykles, tiek suplanuotas taisykles pasirinktai dalykinei sričiai;
    d. Pasirinkta dalykinė sritis iš visuotinai aptariamų pavyzdžių, todėl eksperimentai gali būti atkartoti kitose aplinkose.
3. Ar pasiūlyti ir įgyvendinti „NRules“ sistemos pakeitimai nėra specifinis tik nagrinėjamiems atvejams?
    a. Nėra, nes tai priklauso nuo dalykinės srities ir yra įrodyta atliktais eksperimentais;
    b. Eksperimento metu išbandyti trys dažniausiai pasitaikančių taisyklių tipai: darnos, išvedimo ir perspėjimo, siekiant išbandyti skirtingus atvejus.

LITERATŪROS SĄRAŠAS

[1] Hlupic, V., Robinson, S. (1998, December). Business process modelling and analysis using discrete-event simulation. In Proceedings of the 30th conference on Winter simulation (pp. 1363-1370). IEEE Computer Society Press.

[2] Eriksson, H., Penker, M., 1999. Business Modeling with UML: Business Patterns at work. Canada: John Wiley & Sons.

[3] Szelagowski, M, 2014. Static and Dynamic Processes. Dynamic BPM [žiūrėta 2016-12-05]. Prieiga per internetą: <http://www.bpmleader.com/2014/08/28/static-and-dynamicprocesses/>.

[4] Bhat, Jyoti M., Nivedita Deshmukh, 2005. Methods for modeling flexibility in business processes. Workshop on Business Process Modeling, Design and Support (BPMDS05), Proceedings of CAiSE05 Workshops, 1-8.

[5] Gong, Y., Janssen, M., 2012. From policy implementation to business process man- agement: principles for creating flexibility and agility. Gov. Inf. Q. 29 (1), S61– S71. doi:10.1016/j.giq.2011.08.004.

[6] Pucher, M.J. 2010. Agile-, adhoc-, dynamic-, social-, or adaptive BPM. (January, 2016): <http://isismjpucher.wordpress.com/2010/03/30/dynamic- vs- adaptive- bpm/>.

[7] Adams, M., et al., 2010. Dynamic workflow. In: Hofstede, et al. (Eds.), Modern Business Process Automation. Springer-Verlag Berlin Heidelberg, pp. 123–145. doi:10.1007/978- 3-642- 03121- 2_4.

[8] Rusinaite, T.; Savickas, T; Vasilecas, O. 2015a. Analysis of dynamic business processes and proposed implementation, in: 18-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“ teminė konferencija, 20–23.

[9] BusinessDictionary, 2015. Simulation [žiūrėta 2016-12-01]. Prieiga per internetą: <http://www.businessdictionary.com/definition/simulation.html>.

[10] Zur Muehlen, M., 2004. Workflow-Based Process Controlling: Foundation, Design, and Application of Workflow-Driven Process Information Systems. Logos, Berlin.

[11] Ryan, J., Heavey C., 2006. Process Modeling for simulation. School of Hospitality Management and Tourism. 57(5): 437-450..

[12] Van Der Aalst, Voorhoeve, M., 2016. Business Process Simulation. Lecture notes 2II75:6-8.

[13] Szabolcs Rozsnyai, Josef Schiefer, Alexander Schatten. Concepts and Models for Typing Events for Event-Based Systems, 2006..

[14] Tarak Modi. Event-Driven Architecture: Achieving Architectural Agility, 2005..

[15] Rene Meier, Vinny Cahill. Taxonomy of Distributed Event-Based Programming Systems, 2004.

[16] Rudaitis, G. 2008. Įvykiais grindžiamų informacinių sistemų modeliavimo ir realizavimo metodika. Daktaro disertacija. Kauno technologijos universitetas. Kaunas. 95 p.

[17] Milanovic, M., Gasevic, D., Rocha, L. (2011, August). Modeling flexible business processes with business rule patterns. In Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International (pp. 65-74). IEEE.

[18] Object Management Group, 2002. Business Rules in Models, Request For Information. [žiūrėta 2017-05-10]. Prieiga per internetą: <http://cgi.omg.org/cgi-bin/doc?ad/2002-9-13s>.

[19] Odell, James J. Advanced object-oriented analysis and design using UML. Vol. 12. Cambridge University Press, 1998..

[20] Hay, David, and Keri Anderson Healy. GUIDE Business Rules Project, Final Report-revision 1.2. GUIDE International Corporation, Chicago (1997)..

[21] Defining Business Rules What Are They Really? The Business Rules Group, formerly, known as the GUIDE Business Rules Project, Final Report. Revision 1.3, July, 2000, 1-77. [žiūrėta 2017-05-11]. Prieiga per internetą: <http://www.docstoc.com/docs/11646109.

[22] Von Halle, Barbara. Business rules applied: building better systems using the business rules approach. Wiley Publishing, 2001.

[23] Caplinskas, Albertas, Audrone Lupeikiene, and Olegas Vasilecas. "Shared conceptualisation of business systems, information systems and supporting software." Databases and Information systems II. Springer Netherlands, 2002. 109-120..

[24] Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 OMG Available Specification, OMG Document No. formal/2008-01-02, 2008..

[25] OCL, OMG. Object Constraint Language (OCL) Version 2.2. (2006).

[26] Boyer, J., Mili, H.: Agile Business Rule Development Process, Architecture, and JRules Examples. Springer-Verlag Berlin Heidelberg (2011).

[27] Cunneyworth, W. Table driven design a development strategy for minimal maintenance information systems, 1994..

[28] Ross, R.G. Decision Tables, Part 1 ~ The Route to Consolidated Business Logic. Business Rules Journal [interaktyvus]. 2005, July (Vol. 6, No. 7) [žiūrėta 2017-05-05]. Prieiga per internetą: <http://www.BRCommunity.com/a2005/b240.html>..

[29] Van Eijndhoven, T., Iacob, M. E., Ponisio, M. L. (2008, September). Achieving business process flexibility with business rules. In Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE (pp. 95-104). IEEE.

[30] Vanthienen, J. PROcedural LOGic Analyzer 5.2: User's manual. K. U. Leuven, Department of Applied Economic Sciences, 2003. 119 p..

[31] Avdejenkov V. Verslo taisyklių valdymo sistemų taikymo įmonėse tyrimas. Daktaro disertacija. 2009, Vilniaus Gedimino technikos universitetas..

[32] Mens, K.; Wuyts, R.; Bontridder, D.; Grijseels, A. ECOOP'98 Workshop Report: Tools and Environments for Business Rules, ECOOP'98 Workshop Reader, Springer, 1998.

[33] Booch, G.; Rumbaugh, J.; Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[34] Object Management Group. Unified Modeling Language Revision Task Force. Unified Modeling Language Specification. Version 1.3, June 1999.

[35] Abulsorour A., Visveswaran S. (2002) Implement business rule engines in a J2EE enterprise, [interaktyvus], [žiūrėta 2017.05.20]. Prieiga per internetą: http://www.javaworld.com/article/2074551/core-java/business-process-automation-made-easy-with-java.

[36] Eager A., September 2009. FICO Blaze Advisor 6.7, [žiūrėta 2016-12-09]. Prieiga per internetą: <http://www.fico.com/en/FIResourcesLibrary/ButlerGroupTechnologyAudit.pdf>.

[37] OpenRules, 2011. Open Rules, [žiūrėta 2016-12-09]. Prieiga per internetą: <http://openrules.com/>.

[38] JBoss Community, 2011. Drools - The Business Logic integration Platform, [žiūrėta 2016-12-09]. Prieiga per internetą: <http://www.jboss.org/drools/>.

[39] Forgy C., 1979. On the efficient implementation of production systems. Daktaro disertacija. Carnegie-Mellon universitetas. 178 p. [žiūrėta 2018-05-28]. Prieiga per internetą: <http://reports-archive.adm.cs.cmu.edu/anon/scan/CMU-CS-79-forgy.pdf>.

[40] Proctor M., Neale M. Drools - Drools Documentation. JBoss Community. [žiūrėta 2018-05-28]. Prieiga per internetą: <http://www.jbug.jp/trans/jboss-rules3.0.2/ja/html_single/>.

[41] Rima, A. 2016. Taisyklėmis grindžiamų veiklos procesų su bendrai naudojamais ištekliais modeliavimas ir simuliacijos tyrimas. Daktaro disertacija. Vilniaus Gedimino technikos universitetas. Vilnius, Technika. 114 p.

[42] Vasilecas, O.; Kalibatiene, D.; Lavbič, D. 2016. Rule- and context-based dynamic business process modelling and simulation, Journal of systems and software122: 1-15.

[43] Beconytė, G. Duomenų bazių projektavimas. Mokomoji knyga geomokslų specialybės studentams. Vilniaus universitetas, 2012.

[44] Know Gravity, 2008. Mini EU-Rent Business Model, [žiūrėta 2018-06-04]. Prieiga per internetą: <https://www.knowgravity.com/pdf-e/Mini%20EU-Rent%20BU.pdf>.

[45] Di Bona, D., Re, G. L., Aiello, G., Tamburo, A., Alessi, M. (2011, November). A methodology for graphical modeling of business rules. In Computer Modeling and Simulation (EMS), 2011 Fifth UKSim European Symposium on (pp. 102-106). IEEE.

[46] Lukichev, S., Wagner, G. (2006). UML-Based Rule Modeling with Fujaba. Proceedings of the 4th International Fujaba Days 2006, University of Bayreuth, Germany. Pages: 31 – 35.

[47] Microsoft, 2011. BizTalk Server Overview, [žiūrėta 2016-12-09]. Prieiga per internetą: <http://www.microsoft.com/biztalk/en/us/overview.aspx>.

[48] IBM, 2011. WebSphere ILOG JRules, [žiūrėta 2016-12-09]. Prieiga per internetą: <http://www-01.ibm.com/software/integration/business-rule-management/jrules/>.

[49] Object Management Group, 2014. Business Process Model and Notation, [žiūrėta 2016-12-10]. Prieiga per internetą: < http://www.bpmn.org/ >..

[50] Caplinskas, Albertas, Audrone Lupeikiene, and Olegas Vasilecas. Shared conceptualisation of business systems, information systems and supporting software. Databases and Information systems II. Springer Netherlands, 2002. 109-120.

[51] Vanthienen, Jan. Quality by Design: Using Decision Tables in Business Rules. Business Rules Journal. Prieiga per internetą: <https://www.brcommunity.com/articles.php?id=n008>.

[52] Vanthienen, J. PROcedural LOGic Analyzer 5.2: User's manual. K. U. Leuven, Department of Applied Economic Sciences, 2003. 119 p..

[53] Abulsorour A., Visveswaran S. (2002) Implement business rule engines in a J2EE enterprise, [interaktyvus], [žiūrėta 2017.05.20]. Prieiga per internetą: http://www.javaworld.com/article/2074551/core-java/business-process-automation-made-easy-with-java--par.

[54] Doorenbos R., 1995. Production Matching for Large Learning Systems. Daktaro disertacija. Carnegie-Mellon universitetas. 208 p. [žiūrėta 2018-05-28]. Prieiga per internetą: <http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf>.

PRIEDAI

1. Taisyklės pridėjimo trukmės priklausomybės nuo taisyklių ir faktų skaičiaus, esant baziniam ir patobulintam taisyklių pridėjimo metodui, funkcinio testo programinis kodas:

```
/// <summary>
/// Taisyklės pridėjimo trukmės priklausomybės nuo taisyklių ir faktų skaičiaus,
/// esant baziniam ir patobulintam taisyklių pridėjimo metodui, testas
/// </summary>
[TestMethod]
public void TaisyklėsPridėjimoTestas()
{
    //Sukuriamas rinkinys atsitiktinai sugeneruotu taisyklių
    List<IRuleDefinition> randomGeneratedRules = new List<IRuleDefinition>();
    //Sukuriamas rinkinys atsitiktinai sugeneruotu faktų
    List<object> randomGeneratedFacts = new List<object>();
    //Sukuriamas laikmatis, taisyklės pridėjimo trukmei matuoti
    Stopwatch timer = new Stopwatch();
    //Išmatuotos taisyklių pridėjimo trukmės, naudojant skirtingus metodus
    long elapsedTime = 0;
    long elapsedTime1 = 0;
    //Maksimalus taisyklių skaičius produkcinėje atmintyje
    int ruleCnt = 100;
    //Testo kartojimų skaičius, esant skirtingiems faktų kiekiams
    int repeatCnt = 10;
    //Sukuriami taisyklių rikiniai, naudojami skirtinguose taisyklių pridėjimo
    metoduose
    IRuleSet ruleset = _ruleRepository.CreateRuleSet("RuleSet4");
    IRuleSet ruleset1 = _ruleRepository1.CreateRuleSet("RuleSet4");
    //Sukompiliuojamos taisyklės taisyklių bazėse ir sukuriami sesijų fabrikai
    ISessionFactory sessionFactory = _ruleRepository.Compile();
    ISessionFactory sessionFactory1 = _ruleRepository1.Compile();
    //Sukuriamos sesijos, naudojamos skirtinguose taisyklių pridėjimo metoduose
    ISession session = sessionFactory.CreateSession();
    ISession session1 = sessionFactory1.CreateSession();
    //Atsitiktinai sugeneruojamos taisyklės
     for (int ruleGenerationIndex = 0; ruleGenerationIndex < ruleCnt;
     ruleGenerationIndex++)
    {
        randomGeneratedRules.Add(_rulesDescription.RandomGenerateRule());
    }
    //Vykdomas testas
    for (int repeatIndex = 0; repeatIndex < repeatCnt; repeatIndex++)
    {
        randomGeneratedFacts.Clear();

        //Atsitiktinai sugeneruojami faktai
       for (int factGenerationIndex = 0; factGenerationIndex < (repeatIndex * 100);
       factGenerationIndex++)
        {
            randomGeneratedFacts.Add(_rulesDescription.GenerateRandomFact());
        }
        //Atlaisvinami taisyklių rinkiniai
        ruleset.ClearRuleSet();
        ruleset1.ClearRuleSet();
        //Sukuriamas patobulinto metodo sesijų fabrikas
        sessionFactory = _ruleRepository.Compile();
        //Sukuriama patobulinto metodo sesija
        session = sessionFactory.CreateSession();
        //Užkraunami į patobulinto metodo sesiją atsitiktinai sugeneruoti faktai
        session.InsertAll(randomGeneratedFacts);
        //Pridedama po vieną taisyklę, kol taisyklių skaičius pasiekia prieš tai
        nurodytą maksimalų taisyklių skaičių
```

```
            for (int ruleTestIndex = 0; ruleTestIndex < (ruleCnt); ruleTestIndex++)
            {
                //Pridedama taisyklė į patobulinto metodo taisyklių rinkinį
                ruleset.Add(new List<IRuleDefinition> {
                randomGeneratedRules.ElementAt(ruleTestIndex) });
                //Pradedamas matuoti taisyklės pridėjimo laikas
                timer.Start();
                 //Naudojant patobulintą metodą yra pridedama nauja taisyklė   ir
                 atnaujinamas patobulinto metodo sesijos "RETE" tinklas.
                sessionFactory.AddNewRulesInSessionFactory(new List<IRuleDefinition> {
                randomGeneratedRules.ElementAt(ruleTestIndex) }, true);
                //Baigiamas matuoti taisyklės pridėjimo laikas
                timer.Stop();
                //Išsaugomas taisyklės pridėjimo laikas
                elapsedTime = timer.ElapsedMilliseconds;
                //Nustatomas laikmatis naujiems matavimams
                timer.Reset();
                //Pridedama taisyklė į bazinio metodo taisyklių rinkinį
                ruleset1.Add(new List<IRuleDefinition> {
                 randomGeneratedRules.ElementAt(ruleTestIndex) });
                //Pradedamas matuoti taisyklės pridėjimo laikas
                timer.Start();
                 //Sukompiliuojamos visos taisyklės esančios bazinio metodo taisyklių
                bazėje, tokiu būdu atnaujinant sesijų fabriką
                sessionFactory1 = _ruleRepository1.Compile();
                //Sukuriamas nauja bazinio metodo sesija su atnaujintu "RETE" tinklu
                session1 = sessionFactory1.CreateSession();
                //Užkraunami į bazinio metodo sesiją atsitiktinai sugeneruoti faktai
                session1.InsertAll(randomGeneratedFacts);
                //Baigiamas matuoti taisyklės pridėjimo laikas
                timer.Stop();
                //Išsaugomas taisyklės pridėjimo laikas
                elapsedTime1 = timer.ElapsedMilliseconds;
                //Nustatomas laikmatis naujiems matavimams
                timer.Reset();
                //Išvedami išmatuoti taisyklių pridėjimo laikai
                Console.WriteLine(string.Format("{0}:{1};{2}", (ruleTestIndex + 1),
                elapsedTime, elapsedTime1));
            }
        }
    }
```

2. Taisyklės ištrynimo trukmės priklausomybė nuo taisyklių ir faktų skaičiaus, esant baziniam ir patobulintam taisyklių pridėjimo metodui, funkcinio testo programinis kodas:

```
/// <summary>
/// Taisyklės ištrynimo trukmės priklausomybės nuo taisyklių ir faktų skaičiaus,
/// esant baziniam ir patobulintam taisyklių pridėjimo metodui, testas
/// </summary>
[TestMethod]
public void TaisykliuIstrynimoTestas()
{
    //Sukuriamas rinkinys atsitiktinai sugeneruotu taisyklių
    List<IRuleDefinition> randomGeneratedRules = new List<IRuleDefinition>();
    //Sukuriamas rinkinys atsitiktinai sugeneruotu faktų
    List<object> randomGeneratedFacts = new List<object>();
    //Sukuriamas laikmatis, taisyklės pridėjimo trukmei matuoti
    Stopwatch timer = new Stopwatch();
    //Išmatuotos taisyklių pridėjimo trukmės, naudojant skirtingus metodus
    long elapsedTime = 0;
    long elapsedTime1 = 0;
    //Maksimalus taisyklių skaičius produkcinėje atmintyje
    int ruleCnt = 100;
```

```
//Testo kartojimų skaičius, esant skirtingiems faktų kiekiams
int repeatCnt = 10;
//Sukuriami taisyklių rikiniai, naudojami skirtinguose taisyklių pridėjimo
//metoduose
IRuleSet ruleset = _ruleRepository.CreateRuleSet("RuleSet");
IRuleSet ruleset1 = _ruleRepository1.CreateRuleSet("RuleSet1");
//Sukompiliuojamos taisyklės taisyklių bazėse ir sukuriami sesijų fabrikai
ISessionFactory sessionFactory = _ruleRepository.Compile();
ISessionFactory sessionFactory1 = _ruleRepository1.Compile();
//Sukuriamos sesijos, naudojamos skirtinguose taisyklių pridėjimo metoduose
ISession session = sessionFactory.CreateSession();
ISession session1 = sessionFactory1.CreateSession();
//Atsitiktinai sugeneruojamos taisyklės
for (int ruleGenerationIndex = 0; ruleGenerationIndex < (ruleCnt + 1);
 ruleGenerationIndex++)
{
    randomGeneratedRules.Add(_rulesDescription.RandomGenerateRule());
}
//Vykdomas testas
for (int repeatIndex = 0; repeatIndex < repeatCnt + 1; repeatIndex++)
{
    randomGeneratedFacts.Clear();
    //Atsitiktinai sugeneruojami faktai
    for (int factGenerationIndex = 0; factGenerationIndex < (repeatIndex * 100);
             factGenerationIndex++)
    {
        randomGeneratedFacts.Add(_rulesDescription.GenerateRandomFact());
    }

    Console.WriteLine("Generated fact cnt:{0}", (repeatIndex * 100));
    Debug.WriteLine("Generated fact cnt:{0}", (repeatIndex * 100));
    //Atlaisvinami taisyklių rinkiniai
    ruleset.ClearRuleSet();
    ruleset1.ClearRuleSet();
    //Sukuriamas patobulinto metodo sesijų fabrikas
    ruleset.Add(randomGeneratedRules);
    ruleset1.Add(randomGeneratedRules);
    //Sukuriamas patobulinto metodo sesijų fabrikas
    sessionFactory = _ruleRepository.Compile();
    //Sukuriamas bazinio metodo sesijų fabrikas
    sessionFactory1 = _ruleRepository1.Compile();
    //Sukuriama patobulinto metodo sesija
    session = sessionFactory.CreateSession();
    //Sukuriama bazinio metodo sesija
    session1 = sessionFactory1.CreateSession();
    //Užkraunami į patobulinto ir bazinio metodų sesijas atsitiktinai
    //sugeneruotus faktus
    session.InsertAll(randomGeneratedFacts);
    session1.InsertAll(randomGeneratedFacts);

    //Atimama po vieną taisyklę, kol taisyklių skaičius pasiekia 0
    for (int ruleTestIndex = ruleCnt; ruleTestIndex >= 0; ruleTestIndex--)
    {
        //Ištrinama taisyklė iš patobulinto metodo taisyklių rinkinio
        ruleset.Delete(new List<string> {
        randomGeneratedRules.ElementAt(ruleTestIndex).Name });
        //Pradedamas matuoti taisyklės ištrynimo laikas
        timer.Start();
        //Naudojant patobulintą metodą yra ištrinama taisyklė ir
        //Atnaujinamas patobulinto metodo sesijos "RETE" tinklas.
        sessionFactory.DeleteRules(new List<IRuleDefinition> {
         randomGeneratedRules.ElementAt(ruleTestIndex) });
        //Baigiamas matuoti taisyklės ištrynimo laikas
        timer.Stop();
        //Išsaugomas taisyklės ištrynimo laikas
```

```
            elapsedTime = timer.ElapsedMilliseconds;
            //Nustatomas laikmatis naujiems matavimams
            timer.Reset();
            //Ištrynama taisyklė iš bazinio metodo taisyklių rinkinio
            ruleset1.Delete(new List<string> {
             randomGeneratedRules.ElementAt(ruleTestIndex).Name });
            //Pradedamas matuoti taisyklės ištrynimo laikas
            timer.Start();
            //Sukompiliuojamos visos taisyklės esančios bazinio metodo taisyklių
            //bazėje, tokiu būdu atnaujinant sesijų fabriką
            sessionFactory1 = _ruleRepository1.Compile();
            //Sukuriamas nauja bazinio metodo sesija su atnaujintu "RETE" tinklu
            session1 = sessionFactory1.CreateSession();
            //Užkraunami į bazinio metodo sesiją atsitiktinai sugeneruoti faktai
            session1.InsertAll(randomGeneratedFacts);
            //Baigiamas matuoti taisyklės ištrynimo laikas
            timer.Stop();
            //Išsaugomas taisyklės ištrynimo laikas
            elapsedTime1 = timer.ElapsedMilliseconds;
            //Nustatomas laikmatis naujiems matavimams
            timer.Reset();
            //Išvedami išmatuoti taisyklių ištrynimo laikai
            Console.WriteLine(string.Format("{0}:{1};{2}", (ruleTestIndex + 1),
             elapsedTime, elapsedTime1));
        }
    }
}
```

3. Dalykinės srities verslo taisyklių rinkinio, parašyto „Fluent DSL“ taisyklių forma, programinis kodas:

```
[Name("RentalUsesAtMostOneCar")]
[Description("It is necessary that each rental uses at most one car.")]
[Tag("Validation")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalUsesAtMostOneCar : Rule
{
    public override void Define()
    {
        When()
            .Match<Rental>(r => r.Car == null);
        Then()
            .Do(ctx => ctx.HandleRetractFactsAction());
    }
}
```

```
[Name("RentalHasExactlyOneSignee")]
[Description("It is necessary that each rental is signed by exactly one renter.")]
[Tag("Validation")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalHasExactlyOneSignee : Rule
{
    public override void Define()
    {
        When()
            .Match<Rental>(r => r.Renter == null);
        Then()
            .Do(ctx => ctx.HandleRetractFactsAction());
    }
}
```

```
[Name("RentalBooksExactlyOneCarModel")]
```

```
[Description("It is necessary that each rental books exactly one car model.")]
[Tag("Validation")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalBooksExactlyOneCarModel : Rule
{
    public override void Define()
    {
        When()
            .Match<Rental>(r => r.BooksCarModel == null);
        Then()
            .Do(ctx => ctx.HandleRetractFactsAction());
    }
 }
```

```
[Name("RenterOpenAmountIsSumOfTotalPriceOfEachSignedRental")]
[Description(@"It is necessary that the open amount of each renter is sum of the
 total price of each rental that is signed by the renter.")]
[Tag("Derivation")]
[Priority(5)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RenterOpenAmountIsSumOfTotalPriceOfEachSignedRental : Rule
{
    public override void Define()
    {
        Renter renter = null;
        IEnumerable<Rental> rentals = null;
        decimal total = 0;

        When()
            .Match<Renter>(() => renter)
            .Query(() => rentals, x => x
                .Match<Rental>(r => r.Renter == renter)
                .Collect())
            .Let(() => total, () => rentals.Sum(r => r.TotalPrice))
            .Having(() => renter.OpenAmount != total);

        Then()
            .Do(ctx => renter.UpdateOpenAmount(total))
            .Do(ctx => ctx.Update(renter));
    }
}
```

```
[Name("RentalTotalPreliminaryPriceComputationRule")]
[Description(@"It is necessary that the total price of each rental is the number
of days * price per day of the car model that is booked by that rental * (100
discount of that rental) / 100 - promotion discount of that rental.")]
[Tag("Derivation")]
[Priority(5)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalTotalPreliminaryPriceComputationRule : Rule
{
    public override void Define()
    {
        Rental rental = null;
        decimal total = 0;
        var msg = "The total price of each rental is the number of days * price
        per day of the car model that is booked by that rental * (100 – discount
        of that rental) / 100 - promotion discount of that rental.";

        When()
            .Match<Rental>(() => rental, r => r.ActualReturn == null ||
            r.ActualReturn == r.ExpectedReturn)
            .Let(() => total, () =>
                rental.NumberOfDays *
                rental.BooksCarModel.PricePerDay *
```

```
                (100 - rental.Discount) / (100 - rental.PromotionDiscount))
            .Having(() => rental.TotalPrice != total);

        Then()
            .Do(ctx => rental.UpdateTotalPrice(total))
            .Do(ctx => ctx.Update(rental))
            .Do(ctx => ctx.DisplayMessage(msg));
    }
 }
[Name("RentalTotalDelayedPriceComputationRule")]
[Description(@"It is necessary that the total price of each rental is increased by
a penalty of 20 * (actual return - expected return), if actual return is after
expected return.")]
[Tag("Derivation")]
[Priority(5)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalTotalDelayedPriceComputationRule : Rule
{
    public override void Define()
    {
        Rental rental = null;
        decimal total = 0;
        var msg = "The total price of each rental is increased by a penalty of 20
        * (actual return - expected return), if actual return is after expected
        return.";

        When()
            .Match<Rental>(() => rental, r => r.ActualReturn > r.ExpectedReturn)
            .Let(() => total, () =>
                rental.TotalPrice +
                (20m * (decimal)((DateTime)rental.ActualReturn -
             rental.ExpectedReturn).TotalDays))
            .Having(() => rental.TotalPrice != total);

        Then()
            .Do(ctx => rental.UpdateTotalPrice(total))
            .Do(ctx => ctx.Update(rental))
            .Do(ctx => ctx.DisplayMessage(msg));
    }
}
[Name("RentalTotalEarlyPriceComputationRule")]
[Description(@"It is necessary that the total price of each rental is decreased by
a penalty of 5 * (expected return - actual return), if actual return is before
than expected return.")]
[Tag("Derivation")]
[Priority(7)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalTotalEarlyPriceComputationRule : Rule
{
    public override void Define()
    {
        Rental rental = null;
        decimal total = 0;
        var msg = "The total price of each rental is decreased by a penalty of 5 *
        (expected return - actual return), if actual return is before than
        expected return.";
        When()
            .Match<Rental>(() => rental, r => r.ActualReturn < r.ExpectedReturn)
            .Let(() => total, () =>
                rental.TotalPrice -
                (5m * (decimal)(rental.ExpectedReturn -
                (DateTime)rental.ActualReturn).TotalDays))
            .Having(() => rental.TotalPrice != total);
```

```
            Then()
                .Do(ctx => rental.UpdateTotalPrice(total))
                .Do(ctx => ctx.Update(rental))
                .Do(ctx=> ctx.DisplayMessage(msg));
    }
}
```

```
[Name("RentalPromotionDiscountForAudi")]
[Description(@"If the car model of the rental is Audi and the start date of the
rental is within year, then set the promotion discount of the rental to the agreed
duration of the rental * 5 and display the message: The rental is eligible for the
Audi promotion.")]
[Tag("Derivation")]
[Priority(7)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalPromotionDiscountForAudi : Rule
{
    public override void Define()
    {
        Rental rental = null;
        int discount = 0;
        string msg = "The rental is eligible for the Audi promotion.";

        When()
            .Match<Rental>(() => rental, r =>
        r.BooksCarModel.Title.ToLower().Contains("audi"), r => r.StartDate.Year ==
        DateTime.Now.Year)
            .Let(() => discount, () =>
                rental.AggreedDuration <= 10 ? rental.AggreedDuration * 5 : 50)
            .Having(() => rental.PromotionDiscount != discount);

        Then()
            .Do(ctx => rental.UpdatePromotionDiscount(discount))
            .Do(ctx => ctx.Update(rental))
            .Do(ctx => ctx.DisplayMessage(msg));
    }
}
```

```
[Name("RentalPromotionDiscountForBmw")]
[Description(@"If the car model of the rental is BMW and the start date of the
rental is within this year, then set the promotion discount of the rental to the
agreed duration of the rental * 7 and display the message: The rental is eligible
for BMW promotion.")]
[Tag("Derivation")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalPromotionDiscountForBmw : Rule
{
    public override void Define()
    {
        Rental rental = null;
        int discount = 0;
        string msg = "The rental is eligible for BMW promotion.";

        When()
            .Match<Rental>(() => rental, r =>
        r.BooksCarModel.Title.ToLower().Contains("bmw"), r => r.StartDate.Year ==
        DateTime.Now.Year)
            .Let(() => discount, () =>
                rental.AggreedDuration <= 10 ? rental.AggreedDuration * 7 : 70)
             //slight improvement to avoid divide by zero: up to ten days OR 70
            .Having(() => rental.PromotionDiscount != discount);

        Then()
            .Do(ctx => rental.UpdatePromotionDiscount(discount))
            .Do(ctx => ctx.DisplayMessage(msg));
```

```
    }
}
```

```
[Name("RentalHasMinimumMileage")]
[Description(@"It is obligatory that a rental uses a car that is available and
that rental books a car model and that car is an instance of that car model and
that car has mileage that is minimum of mileage of each car that is available.")]
[Tag("Validation")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalHasMinimumMileage : Rule
{
    public override void Define()
    {
        Rental rental = null;
        Car car = null;

        IEnumerable<Rental> allActiveRentals = null;
        IEnumerable<Car> availableCars = null;
        int minMileage = 0;

        When()
            .Query(() => allActiveRentals, x => x
                .Match<Rental>(r => r.ActualReturn == null)
                .Collect())

            .Query(() => availableCars, x => x
                .Match<Car>(c => allActiveRentals.Any(r => r.Car != c))
             //collect all cars that are not in all active rentals
                .Collect())

            .Match<Rental>(() => rental, r => availableCars.Any(c => c == r.Car))
            .Match<Car>(() => car, c => rental.Car == c, c => rental.BooksCarModel
             == c.CarModel)

            .Let(() => minMileage, () =>
                availableCars.Min(c => c.Mileage))

            .Having(() => car.Mileage == minMileage);

        Then()
            .Do(ctx => ctx.HandleRetractFactsAction());
    }
}
```

```
[Name("RentalCar")]
[Description(@"If rental uses car and car is in current location and car belongs
to branch and branch is in owning country and current location is same as owning
country, then rental to destination branch and destination branch is in owning
country.")]
[Tag("Validation")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalCar : Rule
{
    public override void Define()
    {
        Rental rental = null;
        Car car = null;
        Branch branch = null;

        When()
            .Match<Rental>(() => rental, r => r.Car != null)
            .Match<Car>(() => car, c => rental.Car == c)
            .Match<Branch>(() => branch, b => rental.FromBranch == b)
            .Having(() => car.Country != branch.Country);
```

```
            Then()
                .Do(ctx => ctx.HandleRetractFactsAction());
        }
    }

    [Name("RentalRenterIsBlacklisted")]
    [Description(@"It is prohibited that a renter has signed a rental, if that renter
    is blacklisted.")]
    [Tag("Validation")]
    [Priority(10)]
    [Repeatability(RuleRepeatability.Repeatable)]
    public class RentalRenterIsBlacklisted : Rule
    {
        public override void Define()
        {
            Rental rental = null;
            When()
                .Match<Rental>(() => rental, r => r.Renter != null)
                .Match<Renter>(r => rental.Renter == r, r => r.IsBlacklisted);
            Then()
                .Do(ctx => ctx.HandleRetractFactsAction());
        }
    }

    [Name("RentalCarModelIsPhasedOut")]
    [Description(@"It is prohibited that a rental books a car model, if that car model
    is phased-out.")]
    [Tag("Validation")]
    [Priority(10)]
    [Repeatability(RuleRepeatability.Repeatable)]
    public class RentalCarModelIsPhasedOut : Rule
    {
        public override void Define()
        {
            Rental rental = null;
            When()
                .Match<Rental>(() => rental)
                .Match<CarModel>(c => rental.BooksCarModel == c, c => c.IsPhaseOut);
            Then()
                .Do(ctx => ctx.HandleRetractFactsAction());
        }
    }

    [Name("RenterOpenAmountVsCreditLimit")]
    [Description(@"It is prohibited that a renter has an open amount that is larger
    than the credit limit of that renter.")]
    [Tag("Validation")]
    [Priority(10)]
    [Repeatability(RuleRepeatability.Repeatable)]
    public class RenterOpenAmountVsCreditLimit : Rule
    {
        public override void Define()
        {
            When()
                .Match<Renter>(r => r.OpenAmount > r.CreditLimit);
            Then()
                .Do(ctx => ctx.HandleRetractFactsAction());
        }
    }

    [Name("RenterHasValidDriverLicence")]
    [Description(@"It is obligatory that a renter has a driver license that is
    valid.")]
    [Tag("Validation")]
    [Priority(10)]
    [Repeatability(RuleRepeatability.Repeatable)]
```

```
public class RenterHasValidDriverLicence : Rule
{
    public override void Define()
    {
        Rental rental = null;
        Renter renter = null;

        When()
            .Match<Rental>(() => rental)
            .Match<Renter>(() => renter, r => rental.Renter == r)
            .Having(() => !renter.DriverLicense.IsValid);

        Then()
            .Do(ctx => ctx.HandleRetractFactsAction());
    }
}
```

```
[Name("RenterIsVIPClient")]
[Description(@"It is recommended that a renter who is VIP should receive company
gifts.")]
[Tag("Notification")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class VIPRenterShouldReceiveProperAttention : Rule
{
    public override void Define()
    {
        var msg = "VIP Clients should receive company gifts.";
        Renter renter = null;
        Rental rental = null;

        When()
        .Match<Rental>(() => rental)
        .Match<Renter>(() => renter, r => rental.Renter == r, r => r.IsVIP);

        Then()
            .Do(ctx => ctx.DisplayMessage(msg));
    }
}
```

```
[Name("RentalIsAtLeast14Days")]
[Description(@"It is recommended that a rental which is at least 14 days should
receive proper attention.")]
[Tag("Notification")]
[Priority(10)]
[Repeatability(RuleRepeatability.Repeatable)]
public class RentalAtLeast14Days : Rule
{
    public override void Define()
    {
        var msg = "Rental of at least 14 days should receive company gifts.";
        When()
            .Match<Rental>(r => r.RentalDuration == RentalDuration.AtLeast14Days);

        Then()
            .Do(ctx => ctx.DisplayMessage(msg));
    }
}
```

4. Dalykinės srities faktų rinkinių fragmentai:

11 lentelė. Nuomininkų faktų rinkinio fragmentas

Fakto nr.	IsVip	IsBlacklisted	OpenAmount	CreditLimit	Name	Code
1	True	False	0	1700	Renter #0001	0001
2	False	False	0	3400	Renter #0002	0002
3	False	False	0	600	Renter #0003	0003

12 lentelė. Šalių faktų rinkinio fragmentas

Fakto nr.	Name
1	Austria
2	Belgium
3	Bulgaria

13 lentelė. Automobilių modelių faktų rinkinio fragmentas

Fakto nr.	Title	Description	PricePerDay	IsPhaseOut	ServiceInterval
1	Chevy Aveo	Economy	30	False	Renter #0001
2	Nissan Verso	Compact	55	False	Renter #0002
3	Toyota Auris	Compact	55	False	Renter #0003

14 lentelė. Automobilių faktų rinkinio fragmentas

Fakto nr.	Mileage	RequiresNextServiceAtMileage	CarModelTitle	CountryName
1	21001	29578	Audi A4	Austria
2	31006	49766	Chevy Express	Austria
3	61002	69878	Audi A4	Austria

15 lentelė. Padainių faktų rinkinio fragmentas

Fakto nr.	CountryName	Name	Address
1	Austria	Austria #1	Address of @Austria #1
2	Austria	Austria #2	Address of @Austria #2
3	Austria	Austria #3	Address of @Austria #3

16 lentelė. Automobilių nuomų faktų rinkinio fragmentas

Fakto nr.	Discount	PromotionDiscount	Total Price	NumberOfDays	Car Title	FromBranchName	ToBranchName	RenterName	Start Date	ExpectedReturn	RentalDurationKind	Aggreed Duration	Penalty	BookedCarModel
1	0	0	200	0	BMW X1	Austria #1	Austria #1	Renter #0476	2018-06-07	2018-06-10	Between7And13Days	4	0	BMW X1
2	0	0	1755	0	BMW X3	Austria #1	Austria #1	Renter #0005	2018-06-30	2018-07-16	AtLeast14Days	27	0	BMW X3
3	0	0	1425	0	Toyota Avensis	Austria #1	Austria #2	Renter #0209	2018-06-22	2018-07-02	Between7And13Days	19	0	Toyota Avensis