KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Jonas Geležis

Programinio kodo generavimo iš grafinių veiklos taisyklių modelių galimybių tyrimas

Magistro darbas

Darbo vadovas:
lekt. dr. K. Kapočius

Kaunas, 2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Jonas Geležis

Programinio kodo generavimo iš grafinių veiklos taisyklių modelių galimybių tyrimas

Magistro darbas

Vadovas: lekt. dr. K. Kapočius
2009-05

Recenzentas: lekt. dr. L. Motiejūnas
2009-05

Atliko: IFM-3/2 gr. stud.
J. Geležis
2009-05-22

Kaunas, 2009

Turinys

Lentelių turinys

Paveikslų turinys

Analysis of Program Code Generation from Graphical Business Rules Models

Summary

One of the reasons for a relatively slow growth of the business rules approach could be the lack of developments in the field of program code generation from the business rules models. During this work methods for code generation from IS requirements models are analysed. The focus is placed on a modified Ross method based rules modelling method aiming to create an adequate code generation methodology. To asses the results, the Microsoft Visio-based prototype would be extended to include all required code generation capabilities.

1. Įvadas

Tradiciniai sistemų projektavimo metodai didėjant operatyvaus informacijos apdorojimo poreikiui dinamiškai kintančioje aplinkoje, nėra pakankamai efektyvūs. Šių problemų sprendimui gali būti taikomi informacijos sistemų projektavimo būdai, pagrįsti veiklos taisyklių koncepcija. Veiklos taisyklių koncepcijos pranašumas tas, kad tradiciniais metodais sukurtos informacinės sistemos pakeitimo ar pritaikymo prie pasikeitusios situacijos kaštai yra pernelyg dideli. Veiklos taisyklėmis pagrįstos sistemos pakeitimus nesunkiai gali atlikti veiklos srities atstovai.

Nuo pat veiklos taisyklių koncepcijos atsiradimo buvo siūlomi įvairūs jų klasifikavimo ir struktūrizavimo modeliai. Bendrosios VT identifikavimo, klasifikavimo ir fiksavimo idėjos, pirmąkart apibendrintos 1997 m. GUIDE projekte, „evoliucionavo“ iki oficialių standartų SBVR, PRR ir kitų. Šiame darbe akcentuojamas Ronaldo Roso (Ronald Ross) pasiūlytas veiklos taisyklių klasifikavimo ir modeliavimo metodas (Ross, 1997). Ronaldo Roso metodas iš kitų išsiskiria ne tik tuo, jog pateikia veiklos taisyklių klasifikavimo ir struktūrizavimo modelį, tačiau ir veiklos taisyklių grafinio modeliavimo technikas. Lygiagrečiai su GUIDE sukurtas Roso metodas iki šiol išlieka ryškiausiu veiksmo teiginių ir išvedimo tipų taisyklių modeliavimo grafinėmis diagramomis metodų. KTU Informacijos sistemų katedroje atlikto tyrimo metu, Roso metodas buvo pritaikytas lietuvių kalbai, sukurtas metamodelis, apibrėžta taisyklių struktūrizavimo metodika. Vis tik automatizuoto VT modelių transformavimo į tam tikrą programinį kodą klausimas išliko atviras.

Problema. Vienas iš veiklos taisyklių koncepcijos plėtrą neigiamai įtakojančių veiksnių yra tas, kad programinio kodo generavimo iš veiklos taisyklių modelių sritis yra menkai ištirta. Šiame darbe nagrinėjama kodo generavimo iš grafinių IS (informacinių sistemų) reikalavimus atspindinčių modelių problema.

Darbo tikslas. Siekiama išsiaiškinti kokios yra kodo generavimo galimybės modifikuotam Roso metodui ir pamėginti sukurti adekvačią programinio kodo generavimo iš taisyklių diagramų metodiką, bei ją pritaikyti praktiškai, atliekant kodo generavimo iš grafinių veiklos taisyklių modelių eksperimentą.

Darbo uždaviniai. Atliekama veiklos taisyklių koncepcijos ir modifikuoto Roso metodo analizė kodo generavimo aspektu, kuriama programinio kodo generavimo iš taisyklių diagramų metodika. Šios metodikos pagrindu plečiamas jau ankščiau KTU Informacijos sistemų katedroje sukurtas Microsoft Visio aplinkoje veikiantis VT modeliavimo pagal Roso metodą įrankis, papildant šį įrankį kodo generavimo galimybe. Papildytas įrankis panaudojamas eksperimentui reikalingų diagramų modeliavimui ir kodo generavimui.

Analizuojami eksperimento metu gauti rezultatai ir pateikiamos išvados apie kodo generavimo galimybes iš grafinių veiklos taisyklių modelių.

Rezultatai. Atlikus modifikuoto Roso metodo analizę pateiktas pasiūlymas generuoti SQL programinį kodą iš grafinių veiklos taisyklių apibrėžiant ne tik kodo generavimo metodiką, bet ir dalį jos sėkmingai pritaikant praktiniam eksperimentui, kuris atliktas modifikavus KTU ISK pateiktą VT modeliavimo įrankį. Taip pat, nustatyta, galima kodo generatoriaus architektūra ir kad SQL kodo generavimui iš grafinių veiklos taisyklių yra tikslinga taikyti kodo generavimo šablonus.

Darbo struktūra:

Skyriuje „Veiklos taisyklių koncepcijos analizė“ pateikiama veiklos taisyklių samprata, aptariami veiklos taisyklių standartai, klasifikavimo ir struktūrizavimo metodai.

Skyriuje „Kodo generavimo iš grafinių VT modelių metodika“ aptariama kodo generavimo iš grafinių veiklos taisyklių metodika, tokią metodiką realizuojančio kodo generatoriaus architektūra.

Skyriuje „SQL kodo generavimo iš grafinių veiklos taisyklių modelių eksperimentas“ aprašomas kodo generavimo eksperimentas, kurio metu praktiškai išbandyta šiame darbe pasiūlyta kodo generavimo metodika grafiniams veiklos taisyklių modeliams paversti SQL kodu.

Kiekvieno skyriaus pabaigoje pateikiamos to skyriaus išvados, o šio darbo penktame skyriuje „Išvados“ pateikiamos visą darbą apibendrinančios išvados.

Darbo rezultatai buvo pristatyti 2009 metais vykusioje tarpuniversitetinėje magistrantų ir doktorantų mokslinėje konferencijoje „Informacinės technologijos“. Straipsnis tyrimo tematika išspausdintas konferencijos leidinyje ir pateikiamas šio darbo prieduose.

2. Veiklos taisyklių koncepcijos analizė

2.1 Veiklos taisyklių samprata

Galima pateikti keletą veiklos taisyklių (business rules) apibrėžimų:

1. Veiklos taisyklės apibrėžia operacijas, apibrėžimus ir apribojimus, kurie taikomi organizacijai siekiančiai savo tikslų.[1]
2. Veiklos taisyklė apibrėžia arba apriboja vieną aspektą veiklos, kuris užtikrina arba įtakoja veiklos elgseną.[2]
3. Veiklos taisyklė – sutarta taisyklė, kurios laikomasi verslo metu. Tokios taisyklės gali nustatyti kainų ar nuolaidų strategiją, vartotojų galimybes įvairiose situacijose, dirbančiųjų teises bei pareigas ir pan. Veiklos taisyklių aiškus formulavimas naudingas visada, tačiau jis tampa absoliučiai būtinas jei imama programuoti jų automatišką laikymąsi kompiuterio pagalba (kuriant duomenų bazių sistemas).

Pagrindiniai reikalavimai verslo taisyklėms yra šie:

- **Deklaratyvumas**: Pasakoma, vien *kokia* tai yra taisyklė, nesigilinant į tai, kokiu būdu jos priverčiama laikytis.
- **Tikslumas**: Turi būti galima tik viena taisyklės interpretacija, dėl kurios visi turi sutarti.
- **Nedalomumas**: Taisyklė turi nurodyti tik vieną normą ar dėsnį. Jei jų yra daugiau, taisyklė suskaidoma į kelias.
- **Išbaigtumas**: Taisyklėje neturi būti jai pačiai prieštaraujančių teiginių.
- **Minimalizmas**: Dvi taisyklės neturi aiškinti iš esmės to paties. [3]

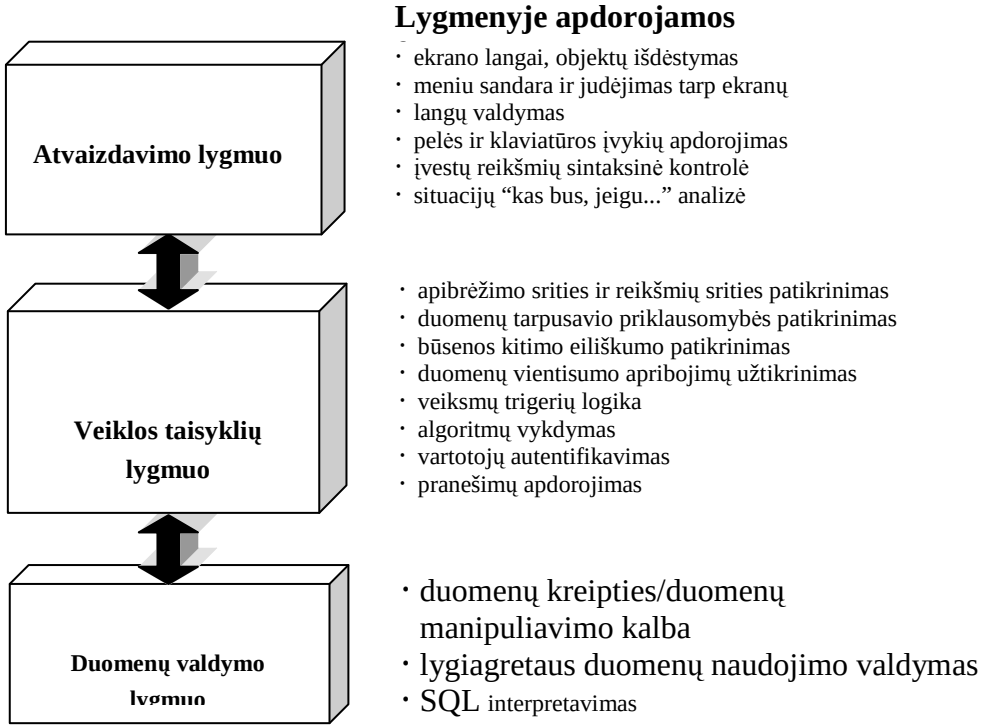
Tradiciniai informacijos sistemų projektavimo metodai, didėjant operatyvaus informacijos apdorojimo poreikiui dinamiškai kintančioje aplinkoje, nėra pakankamai efektyvūs. Šių problemų sprendimui taikomas informacijos sistemų projektavimo būdas, pagrįstas veiklos taisyklių koncepcija. Veiklos taisyklių koncepcijos pranašumą, palyginti su tradiciniu požiūriu, lemia tokie veiksniai:

- Tradiciniais metodais sukurtos informacinės sistemos pakeitimo ar pritaikymo pasikeitusiai situacija yra pernelyg didelės. Veiklos taisyklėmis pagrįstos sistemos pakeitimus nesunkiai gali atlikti veiklos srities atstovai.
- Egzistuojantys informacinių sistemų specifikavimo ir projektavimo modeliai blogai išreiškia dalykinėje srityje galiojančias taisykles, jos yra padrikai paskirstytos po visą

sistemą. Taikant veiklos taisyklių nepriklausomo sluoksnio koncepciją, galima išspręsti šią problemą, nes taisyklių identifikavimas ir automatizavimas yra išreiškiamas formaliais metodais, o pačios taisyklės saugomos atskirai nuo kitų sistemos komponentų.

Sistemoje, pagrįstoje VT principais, taisyklės yra saugomos autonomiškoje veiklos taisyklių saugykloje, kuri kartu su specialia veiklos taisyklių interpretavimo sistema užtikrina informacinės sistemos adaptyvumą kintančios veikos sąlygomis.

Tradicinėje kliento-serverio architektūroje VT sudaro atskirą sluoksnį, esantį tarp duomenų valdymo ir atvaizdavimo sluoksnių (žr. 1 pav.). Iš šių sluoksnių VT sistema perima ir daugumą duomenų tikrinimo, algoritmų logikos, apsaugos bei kitų funkcijų. Reikia pabrėžti, kad kiekvienas iš trijų sluoksnių funkcionuoja visiškai nepriklausomai. Atskyrus VT nuo kitų architektūros sluoksnių, sistemos modifikavimui, pritaikant ją prie pasikeitusių veiklos sąlygų, nereikia atlikti sudėtingų struktūros pakeitimų ar perrašyti programos modulių. Taisyklių pasikeitimus gali fiksuoti patys sistemos vartotojai, naudodamiesi paprastomis priemonėmis. Visi sistemos moduliai, kurių veikimui daro įtaką pasikeitusi taisyklė, įvertina šiuos pasikeitimus, perimdami pakitusius funkcionalumo apribojimus iš jos specifikacijos. [4,5]



1 pav. Veiklos taisyklių vieta ir funkcijos kliento-serverio architektūroje

Pažymėtina, jog VT turi būti traktuojamos ne kaip veiklos objektai, duomenų bazės trigeriai ar apribojimai, pranešimai, procedūros, veiksmai. Taisykles reikia nagrinėti kaip visiškai

savarankišką modeliavimo objektą, kuris savo ruožtu yra susijęs su 1 pav. pateiktais sistemos veikimo aspektais.[6]

2.2 Veiklos taisyklių klasifikavimo ir struktūrizavimo modeliai

2.2.1 GUIDE veiklos taisyklių projektas

GUIDE veiklos taisyklių projektas pateikė ir apibendrino bendrąsias VT identifikavimo, klasifikavimo ir fiksavimo idėjas. GUIDE veiklos taisyklių projekto turėjo 4 pagrindinius tikslus [15]:

- Apibrėžti ir apibūdinti veiklos taisykles ir susijusias sąvokas tokiu būdu nustatant kas yra ir kas nėra veiklos taisyklė.
- Apibrėžti veiklos taisyklių konceptualųjį modelį tam, kad išreikštų (terminais suprantamais informacinių technologijų profesionalams) kas yra veiklos taisyklė ir kaip ji siejasi su informacinėmis sistemomis.
- Numatyti tikslų pagrindą veiklos taisyklių atvirkštinei inžinerijai iš egzistuojančių sistemų.
- Numatyti tikslų pagrindą naujų sistemų inžinerijai remiantis formaliais veiklos taisyklių apibrėžimais.

GUIDE projektas pateikė veiklos taisyklių modelį, kuriame išskirti trys pagrindiniai taisyklių tipai, t.y. kiekviena veiklos taisyklė turi būti viena iš paminėtų [15]:

- Struktūrinis teiginys – apibrėžta sąvoka arba fakto konstatavimas, kuris išreiškia kokį nors veiklos struktūros aspektą. Tai apima tiek sąvokas, tiek faktus surinktus iš sąvokų.
- Veiksmo teiginys – tai konstatavimas apribojimo arba sąlygos, kuris riboja arba valdo veiklos veiksmus.
- Išvedimas – žinios išvestos iš kitų veiklos žinių.

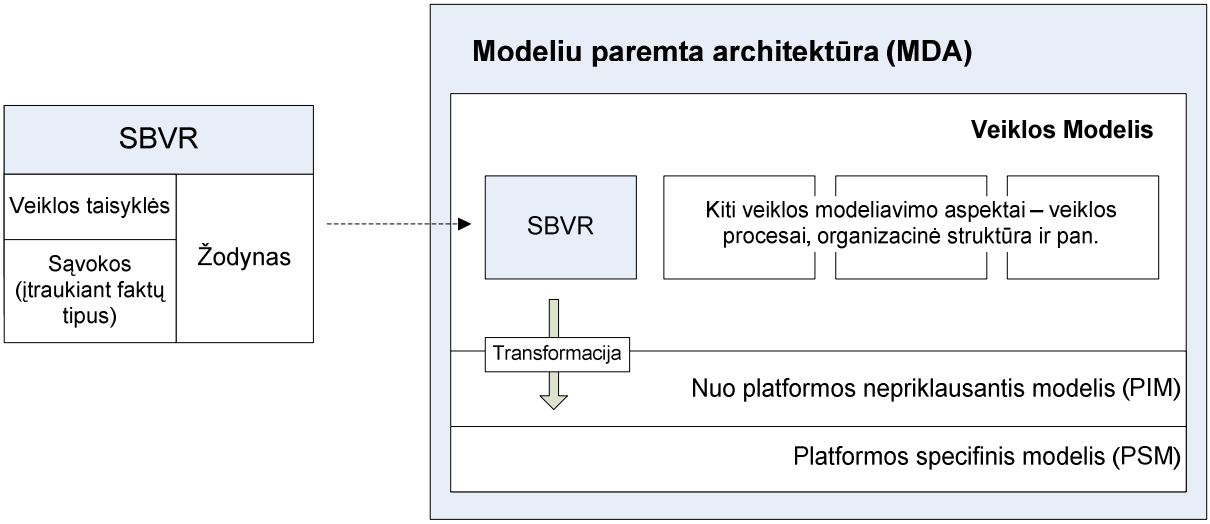
2.2.2 SBVR

SBVR (Semantics of Business Vocabulary and Business Rules) yra OMG (Object Management Group) priimtas standartas, kurį sukūrė Business Rule Team konsorciumas [7]. SBVR apibrėžia žodyną ir taisykles dokumentavimui veiklos žodynų, veiklos faktų ir veiklos taisyklių, taip pat apibrėžia XMI schemą veiklos žodynų ir veiklos taisyklių apsikeitimui tarp organizacijų ir programinės įrangos įrankių.

SBVR yra pirmoji OMG specifikacija įtraukianti natūralios kalbos formalų panaudojimą modeliavime ir pirmoji suteikianti detalų formalios logikos modelį. SBVR

pateikia būdą užfiksuoti specifikacijas natūralia kalba ir išreikšti juos formalia logika taip, kad juos būtų galima apdoroti automatizuotai. Paremta lingvistika ir formalia logika SBVR suteikia būdą atvaizduoti sakinius valdomomis natūraliomis kalbomis kaip logines struktūras vadinamas semantinėmis formuluotėmis. SBVR numatyta išreikšti veiklos žodyną ir veiklos taisykles ir specifikuoti veiklos reikalavimus informacinėms sistemoms natūralia kalba. SBVR modeliai yra deklaratyvūs, o ne imperatyviniai ar procedūriniai. Iš visų OMG modeliavimo kalbų SBVR turi didžiausią išraiškingumą.

MDA architektūroje SBVR yra veiklos modelio sluoksnyje, kaip tai parodyta 2 pav.



2 pav. SBVR padėtis MDA architektūroje

Toks SBVR pozicionavimas turi dvi išvadas:

- SBVR yra orientuojamas į veiklos taisykles ir veiklos žodynus, apimant tuos, kurie tinka kartu naudoti su tomis taisyklėmis. Kiti veiklos taisyklių modelių aspektai taip pat turi būti išvystyti, įskaitant ir veiklos procesus ir organizacinę struktūrą, tačiau šie dalykai turės būti aptarti kitose OMG iniciatyvose.
- Veikos modeliai, įskaitant modelius, kuriuos palaiko SBVR aprašo veiklas, o ne IT sistemas, kurios jas palaiko.

MDA architektūroje, IT sistemos yra apibrėžiamos naudojant nuo platformos nepriklausančius modelius PIM ir platformos specifinius modelius PSM. Veiklos modelių transformacijoms į PIM modelius reikalingos gairės. Šių gairių aptarimas išeina už SBVR ribų. Tikimasi, jog OMG užtikrins, kad metamodeliai skirti skirtingiems veiklų modeliavimo aspektams sudarys rišlią visumą ir paskatins gairių kūrimą kaip vykdyti transformacijas iš veiklos modelių į PIM modelius.

2.2.3 PRR

PRR (Production Rule Representation) standartas pasiūlytas OMG skirtas patenkinti poreikį turėti bendrą produkcinių taisyklių atvaizdavimą, naudojamą įvairių gamintojų taisyklių varikliuose.

PRR išpildo eilę reikalavimų susijusių su veiklos taisyklėmis programinės įrangos sistemomis, OMG standartais ir kitais taisyklių standartais:

- Pateikia standartinį produkcinių taisyklių atvaizdavimą, kuris yra suderinamas su veiklos variklių gamintojų produkcinių taisyklių apibrėžimais. Jis gali būti naudojamas tarpusavio apsikeitimui veiklos taisyklėmis tarp taisyklių modeliavimo įrankių (ir kitų įrankių palaikančių taisyklių modeliavimą kaip funkciją kokioje nors kitoje užduotyje).
- Pateikia standartinį produkcinių taisyklių atvaizdavimą, kuris lengvai susietinas su veiklos taisyklėmis kaip nurodyta veiklos taisyklių valdymo įrankių gamintojų.
- Pateikia standartinį produkcinių taisyklių apibrėžimą, kuris palaiko ir skatina sistemų pardavėjus palaikyti produkcinių taisyklių vykdymą.
- Pateikia OMG MDA architektūros PIM modelį su didele palaikymo PSM lygmenyje tikimybe iš taisyklių variklių gamintojų ir gali būti įtraukiamas į kitus OMG metamodelius, papildant juos produkcinių taisyklių galimybėmis.
- Pateikia pavyzdžius kaip OMG UML gali būti panaudotas produkcinių taisyklių palaikymui standartizuotu ir naudingu būdu.
- Pateikia standartinį produkcinių taisyklių atvaizdavimą, kuris gali būti naudojamas kaip pagrindas, pavyzdžiui W3C konsorciumo RIF formate (Rule Interchange Format) ir produkcinių taisyklių RuleML versijai.

Produkcinės taisyklės. PRR-Core ir PRR-OCL.

PRR apibrėžia su MOF2 suderinamas metamodelis, kuris pateikia:

- Produkcinių taisyklių apibrėžimą tiesioginio susiejimo išvadoms ir procedūriniam apdorojimui.
- Nenormatyvinį tarpusavio apsikeitimo išraiškų kalbos (PRR OCL) apibrėžimą skirtos taisyklių sąlygų ir veiksmų išraiškoms, taip kad būtų galima išraiškas pakeisti alternatyviais atvaizdavimais naudojamais tam tikro gamintojo arba kituose standartuose.
- Apibrėžia taisyklių rinkinius (angl. rulesets), kaip taisyklių kolekcijas su konkrečiais vykdymo režimais (nuosekliu arba išvestiniu).

Metamodelis yra sudarytas iš:

- Pagrindinės struktūros vadinamos PRR Core (PRR branduolys)
- Nenormatyvinės abstrakčios OCL pagrįstos sintaksės PRR išraiškoms, apibrėžiamos išplėstu PRR Core metamodeliu vadinamu PRR OCL.

Būsimi PRR plėtiniai gali apimti:

- taisyklių metamodelius kitoms taisyklių klasėms, tokioms kaip Įvykio-Sąlygos-Veiksmo (angl. ECA – Event-Condition-Action), atgalinio susiejimo ir apribojimams
- grafinėms notacijoms būdingus taisyklių atvaizdavimus, tokius kaip sprendimų lentelės ir sprendimų medžiai
- atvaizdavimą taisyklių rinkinių sekų didesniuose sprendimuose
- transformacijas tarp PRR ir kitų MDA modelių, tokių kaip SBVR.

Kitos konkrečios sintaksės gali būti pritaikytos PRR Core ateityje, tuo tikslu PRR yra suprojektuotas taip, kad būtų praplečiamas.

Produkcinės taisyklės apibrėžimas. PRR standartas apibrėžia, kad produkcinė taisyklė yra programinės logikos sakinys, nusakantis vykdymą vieno ar daugiau veiksmų tuo atveju, jeigu sąlygos yra patenkinamos. Produkcinės taisyklės turi operacinę semantiką. Produkcinių taisyklių vykdymo rezultatas gali priklausyti nuo taisyklių tvarkos, neatsižvelgiant į tai ar tokia tvarka yra apibrėžta taisyklės vykdymo mechanizmo ar taisyklių atvaizdavimo tam tikra tvarka.

Produkcinė taisyklė paprastai yra atvaizduojama šitaip:

if [sąlyga] then [veiksmų sąrašas]

Kai kurios realizacijos išplečia šį apibrėžimą įtraukiant “else“ konstrukciją:

if [sąlyga] then [veiksmų sąrašas] else [alternatyvių veiksmų sąrašas]

nors ši forma nėra taikoma PRR, kadangi visos taisyklės turinčios “else“ sakinį gali būti redukuotos į pirmąją formą, be “else“ ir semantika kada “else“ veiksmai yra vykdomi gali būti sudėtinga kai kuriose išvestinėse schemose. Šiuo atveju tai reiškia, kad konvertavimas iš PSM į PIM gali būti pilnas, bet neapgręžiamas. Taisyklės su “else“ sakiniais PSM modelyje išsireikštų į keletą PIM taisyklių, kurios negalėtų būti sutransliuotos atgal į originalias taisykles, tačiau naujos taisyklės būtų ekvivalenčios savo funkcionalumu.

Produkcinių taisyklių rinkinio apibrėžimas. Produkcinių taisyklių konteineris laikomas produkcinių taisyklių rinkiniu (ruleset). Produkcinių taisyklių rinkinys suteikia:

- priemonę surinkti taisykles susijusias su kokiu nors verslo procesu ar veikla kaip funkcinį vienetą,

- vykdymo vienetą taisyklių variklyje kartu su sąsaja (interfeisu) taisyklių sužadinimui.

Taisyklių rinkinyje esančios taisyklės dirba su objektų aibe (duomenų šaltiniu), kur objektai pateikiami pagal taisyklių rinkinio:

- parametrus
- kontekstą iškvietimo metu.

Vykdymo pabaigoje pasikeitusios reikšmės atspindi taisyklių rinkinio iškvietimo rezultatą

Taisyklės kintamojo apibrėžimas. Sąlygos ir veiksmo sąrašai turi išraiškas, kurios nurodo į du skirtingus kintamųjų tipus (standartiniai kintamieji ir taisyklės kintamieji). Paskelbimo metu:

- standartinis kintamasis turi tipą ir nebūtiną pradinę reikšmę. Kai kuriose sistemose, galimas apribojimo taikymas kintamajam, tokie atvejai nenagrinėjami PRR. Standartiniai kintamieji apibrėžiami taisyklių rinkinių lygyje.
- taisyklės kintamasis turi tipą ir sritį nusakomą filtro pritaikyto duomenų šaltiniui. Be filtro taisyklės kintamojo sritis pagal nutylėjimai yra visi objektai atitinkantys tipą ir esantys duomenų šaltinio ribose. Taisyklių kintamieji gali būti apibrėžti taisyklių lygmenyje, arba taisyklių rinkinių lygmenyje, pastaruoju atveju taisyklių kintamųjų reikšmės yra prieinamos visoms taisyklėms taisyklių rinkinyje.

Taisyklių kintamųjų semantika. Vykdymo metu:

- standartiniai kintamieji yra susiejami su viena reikšme (kuri gali būti kolekcija) iš jų srities. Reikšmė gali būti priskirta pradiniu priskyrimu, priskirta, arba pakartotinai perskirta taisyklės veiksme.
- taisyklių kintamieji yra susiejami su aibe reikšmių iš reikšmių srities pagal tipą ir filtrą. Kiekviena reikšmių kombinacija susieta su taisyklių kiekvienu taisyklės kintamuoju konkrečiai taisyklei yra vadinama sietimi (binding). Ji susieja kiekvieną taisyklės kintamąjį su reikšme (objektu arba kolekcija) duomenų šaltinyje. Šie susiejimai yra vykdymo sąvokos: jie nėra modeliuojami tiesiogiai, bet yra taisyklių kintamųjų nurodymo taisyklių apibrėžimuose rezultatas.

Tai reiškia, kad produkcinė taisyklė yra įvertinama lyginant su visomis taisyklės kintamųjų reikšmėmis. Taisyklės kintamojo panaudojimas, kad produkcinės taisyklės paskelbimas yra:

for [taisyklės kintamieji] if [sąlyga] then [veiksmų sąrašas]

Produkcinių taisyklių semantika. Produkcinių taisyklių operatyvinė semantika bendru atveju tiesiogiai susiejamoms taisyklėms (per produkcinių taisyklių variklį) yra tokios:

1. Suderinimo: taisyklės yra kuriamos remiantis taisyklių sąlygų apibrėžimu ir duomenų šaltinio esama būsena.
2. Konfliktų sprendimo: parinkti vykdomus taisyklių egzempliorius, pagal strategiją
3. Veikimo: pakeisti duomenų šaltinio būseną vykdant pasirinktos taisyklės egzemplioriaus veiksmus.

Ten, kur netaikomi taisyklių varikliai ir vykdomas paprastesnis nuoseklus taisyklių apdorojimas, nelieka konfliktų sprendimo ir vyrauja paprastesnė taisyklių vykdymo strategija.

PRR Core yra klasių rinkinys, kuris leidžia produkcines taisykles ir taisyklių rinkinius apibrėžti visiškai nuo platformos nepriklausančiu būdu be poreikio specifikuoti OCL sąlygų ir veiksmų atvaizdavimui. Visos sąlygos ir veiksmai yra “nepermatomi“ ir yra tiesiog eilutės. Nors tai riboja galimybes transformuoti taisykles iš vienos produkcinės aplinkos (PSM) į kitą, tai leistų dalinti taisyklėmis tarp visų įrankių, suprantančių bazinę produkcinių taisyklių struktūrą.

PRR OCL pasinaudoja OCL metamodeliu, kad atvaizduotų išraiškas siejamas su produkcinių taisyklių dalimis (RuleVariable, Condition ir Action). OCL metaklasių rinkinys naudojamas PRR yra paimtas iš BasicOCL. Metaklasės iš pilno OCL nėra naudojamos.

PRR OCL sudarytas iš:

- klasių rinkinio iš BasicOCL paketo ir rinkinio specifinių apribojimų, kurie apibrėžia OCL klasių panaudojimą PRR OCL kontekste
- PRRActionOCL paketo, kuris praplečia BasicOCL paketą ir pateikia klases produkcinių taisyklių veiksmo daliai atvaizduoti.
- PRR OCL Standard Library standartinės bibliotekos paremtos OCL Standard Library bibliotekos, kuri nurodo iš anksto nustatytus tipus ir operacijas, kurias bet kuri PRR OCL realizacija privalo palaikyti.[8]

2.2.4 Grafinis veiklos taisyklių modelis. Modifikuotas Roso metodas

Veiklos taisyklių klasifikacija. Pagal Roso metodą, taisyklės gali būti atominės ir išvestinės. Išskiriami 32 atominių (nedalomų) taisyklių tipai, kurie grupuojami į 7 šeimas [17]. Išvestinė taisyklė - tai taisyklė, kuri išreiškiama kitų taisyklių aprašais. Išvestinės taisyklės nėra atominės ir gali būti sudarytos iš keleto atominių taisyklių arba kitų išvestinių taisyklių. Išskiriami 58 išvestinių taisyklių tipai, kurie pagal naudojimo sritį grupuojami į 12 šeimų. Visos veiklos taisyklių šeimos pateikiamos 1 lentelėje. Būtina pažymėti, kad ypač

daug dėmesio R. Rosas skiria taisyklių tipų aprašymui, todėl šis modelis klasifikavimo atžvilgiu ypač tikslus. Kiekvienam iš veiklos taisyklių (VT) tipų yra numatyta unikali santrumpa. Toks sprendimas labai pagerina modelio pritaikomumą ir galimų produktų suderinamumą.

1 lentelė. Veiklos taisyklių tipų šeimos pagal Roso metodą[17]

Atominių taisyklių šeimos	***Išvestinių taisyklių šeimos***	
I. Egzempliorių patvirtintojai II. Tipo patvirtintojai III. Pozicijos patvirtintojai IV. Funkciniai patvirtintojai V. Lginamieji įvertintojai VI. Matematiniai įvertintojai VII. Projekcijos valdikliai	A. Egzempliorių testavimo C. Atributų modifikavimo E. Sekos specifikavimo G. Sąlyginio laiko įvertinimo I. Veiklos koordinavimo K. Egzempliorių kopijavimo	B. Pozicijos patikrinimo D. Sekos valdymo F. Kompozicijos struktūrų testavimo H. Atnaujinimų įvertinimo J. Egzempliorių įgalinimo L. Egzempliorių iškvietimo

Taisyklių modeliavimo (formalizavimo) metodas. Be detalios taisyklių klasifikavimo sistemos, R.Rosas pasiūlė ir unikalią VT modeliavimo metodiką, užrašant jas grafine forma. Taisyklės teiginio pateikimas formaliu pavidalu susideda iš penkių pagrindinių etapų. Galutinis rezultatas - specialia grafine notacija užrašyta taisyklė. Analitikas turi atsakyti į šiuos klausimus:

1. Ar taisyklė yra apribojimas, ar sąlyga? (kitaip tariant, ar taisyklė turi būti vykdoma, ar yra išreikšta tik kaip patikrinimas).
2. Kas yra taisyklės bazė? (t. y. kuriam tipui duomenų modelyje reikėtų "priskirti" nagrinėjamą "taisyklę").
3. Kas yra taisyklės korespondentas(-ai)? (t. y. kokie tipai duomenų modelyje būtini norint patikrinti taisyklę).
4. Koks taisyklės tipas? (t. y. kokius testus pritaiko taisyklė).
5. Koks taisyklės bazės ir korespondento ryšys? (t. y. kaip tipai, reikalingi taisyklei patvirtinti, susiję tarpusavyje).

Akivaizdu, kad norint formaliai užrašyti taisykles reikia naudotis dalykinės srities duomenų modeliu. Koks būtent modelis turėtų būti naudojamas, R. Rosas nenurodo. Tačiau yra apibrėžta, kokie duomenų tipai turi būti žinomi. Pagrindiniai jų yra esybės, atributai, asociacijos, sąryšiai, potipiai.

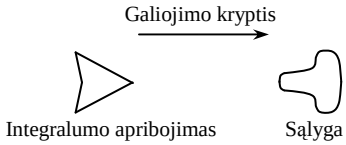
Toliau pateikiamas trumpas kiekvieno iš penkių VT modeliavimo etapų aprašas.

Integralumo apribojimas ar sąlyga? Visos taisyklės iš esmės gali būti tik dviejų rūšių - arba apribojimas, arba sąlyga.

Integralumo apribojimas - tai taisyklė, kurios rezultatas visada privalo būti loginis vienetas (loginė tiesa). Kadangi tokios taisyklės neturi duoti neigiamo rezultato, todėl privalo būti vykdomos.

Sąlyga - tai taisyklė, kurios rezultatas gali būti arba loginis vienetas (loginė tiesa), arba loginis nulis (loginis melas). Rezultatas gali būti ir nežinomas. Šios taisyklės naudingos, kai reikia nustatyti, kokias taisykles vykdyti arba kokius testavimus atlikti (t. y. kokias kitas taisykles iškviesti) atsižvelgiant į tam tikrą situaciją. Tokias taisykles galima išdėstyti nuosekliai, nurodant, kas turi vykti, jeigu aukštesnio lygio taisyklė yra tenkinama.

Modeliuojant taisykles apribojimams ir sąlygoms atskirti, naudojami specialūs žymėjimai, pavaizduoti 3 pav. Simboliai gali būti sukami į bet kurią pusę, tačiau būtina nepamiršti jų galiojimo krypties.



3 pav. Taisyklės vykdymo ir taisyklės tikrinimo žymėjimas[5]

Bazės nustatymas. Kiekviena taisyklė visuomet apibrėžia tam tikrą į duomenų modelį įeinančio tipo egzempliorių. Šis tipas vadinamas *baze*.

Dažniausiai bazė - tai duomenų objektas, atributo tipas ir pan., tačiau tai gali būti ir kita VT ar jos išeigos reikšmė. Taisyklė gali būti traktuojama kaip jos bazės savybė.

Kiekviena taisyklė privalo turėti bazę. Bazė yra svarbiausias elementas interpretuojant taisyklę, be to, bazės egzemplioriai gali būti įtraukti į taisyklės atliekamą patikrinimą. Šis patikrinimas daromas su kiekvienu bazės egzemplioriumi.

Korespondento (-ų) nustatymas. Visos taisyklės, be bazės, visuomet apima dar bent vieną duomenų modelio tipo egzempliorių. Tokie tipai vadinami *korespondentais* ir jie yra būtini kiekvienoje taisyklėje. Korespondentai visuomet yra svarbūs atliekant taisyklės tipo apibrėžtą testą. Korespondentu gali būti ne tik duomenų tipas (pvz., duomenų objektas, atributo tipas ir pan.), bet ir kita taisyklė, jos išeigos reikšmė arba veiksmas.

Kiekviena taisyklė turi vieną ar daugiau korespondentų. Galima išskirti šias ypatybes:

- Nedalomos taisyklės, priklausančios *tipo patvirtintojų* šeimai, privalo turėti du arba daugiau korespondentų, nes veikia kaip loginiai operatoriai IR ar ARBA.
- Nedalomos taisyklės, priklausančios *matematinių įvertintojų* šeimai, gali turėti daugiau negu vieną korespondentą.
- Kitoms šeimoms priklausančios nedalomos (atominės) taisyklės gali turėti tik vieną ir ne daugiau korespondentų.

Tiek išeinantis iš bazės ryšys, tiek ir einantys į korespondentus ryšiai visada vaizduojami punktyrine rodykle.

Taisyklių tipų parinkimas. Parinkti taisyklės tipą yra būtina, nes tik esant nurodytam tipui galima teisingai interpretuoti taisyklę. Taisyklė gali priklausyti tiek atominiam tipui, tiek išvestiniam. Visi taisyklių tipai tinka ir integralumo apribojimams, ir sąlygoms. Taisyklės tipo santrumpa įrašoma apribojimo arba sąlygos simbolio viduje.

Bazės ir korespondento (-ų) susiejimas. Be pagrindinių taisyklės elementų, dažnai svarbu išskirti ir teisingą bazės ir korespondento sąryšį. Šis sąryšis privalo plaukti iš paties duomenų modelio.

Galima tarti, jog duomenų modelis yra sudarytas iš faktų apie probleminę sritį, kurie gali būti šių kategorijų:
- sąryšio tipai, pvz.: "klientas pateikia užsakymą"
- potipių sąryšiai, pvz.: "automobilis yra prekė"
- atributų sąryšiai, pvz.: "automobilis turi registracijos numerį".

Svarbu pabrėžti, jog VT niekuomet negali įvesti naujo fakto (sąryšio). Visi svarbūs faktai (sąryšiai) turi būti išreikšti duomenų modelyje. Taigi kiekvienas taisyklės bazės ir korespondento (-ų) sąryšis turi būti paimtas iš duomenų modelio. Būtina išskirti ir kitą prielaidą: kiekvienas pavaizduotas taisyklės bazės ir korespondento (-ų) sąryšis (fakto tipas) yra laikomas svarbiu taisyklei interpretuoti. Taigi nesvarbūs sąryšiai neturi būti įtraukiami į diagramą.

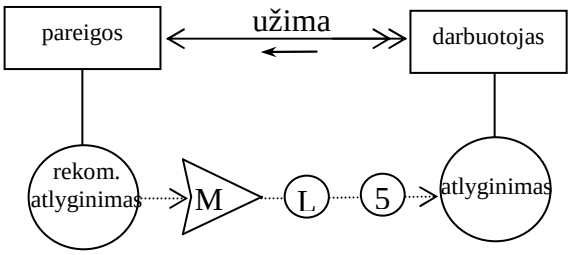
Papildomos VT sintaksės ypatybės. Roso notacijoje yra numatyta galimybė naudoti papildomus simbolius, galinčius pakeisti standartinę tam tikro tipo taisyklės interpretaciją. Šie simboliai skirstomi į dvi pagrindines grupes:

1. Interpretatoriai - tai simboliai, nurodantys, jog taisyklę jos bazės atžvilgiu reikia interpretuoti ne taip, kaip reikėtų remiantis tik jos tipu. Šie simboliai žymimi ant bazės ryšio rodyklės.

2. Apribojimai - tai simboliai, nurodantys, jog taisyklei jos korespondento (-ų) egzempliorių atžvilgiu turi būti taikomi specialūs apribojimai.

Be specialiųjų simbolių, dar gali būti naudojamos konstantos, taisyklių išeigos reikšmės ir kitos papildomos priemonės (Ross, 1997). Paprastai taisyklės išeigos reikšmė yra "nematoma" ir naudojama taisyklės viduje tam tikram testavimui atlikti arba apribojimui pritaikyti. Tačiau kai kurios taisyklės turi tiesiogiai tikrinti kitų taisyklių išeigos reikšmes (tai labiau būdinga sąlygos tipo taisyklėms). Tuomet VT išeigos reikšmė atsispindės ir taisyklės, naudojančios šią reikšmę, diagramoje. Kaip jau buvo minėta, taisyklės išeigos reikšmė gali būti kitos taisyklės bazė arba korespondentas. [5,6,9]

Paprastos taisyklės pavyzdys pateikiamas 4 pav. Ji apibrėžia tokį apribojimą: Rekomenduojamas pareigų atlyginimas turi būti mažesnis už bent penkių darbuotojų, užimančių šias pareigas, atlyginimą. Šios taisyklės bazė yra duomenų modelio esybės pareigos atributas rekom.atlyginimas (rekomenduojamas atlyginimas). Taisyklės korespondentas - duomenų modelio esybės darbuotojas atributas atlyginimas. Mus domina tik tos pareigos, kurias užima darbuotojai, todėl į diagramą įtrauktas ir šis duomenų modelio sąryšis (faktas). Taisyklės tipas yra Mažiau už (kodas MU). Diagramoje taip pat naudojami ir du apribojimai: žemesniojo slenksčio apribojimas (simbolizuoja rutuliukas su raide L) ir numeratorius (rutuliukas su skaičiumi 5).



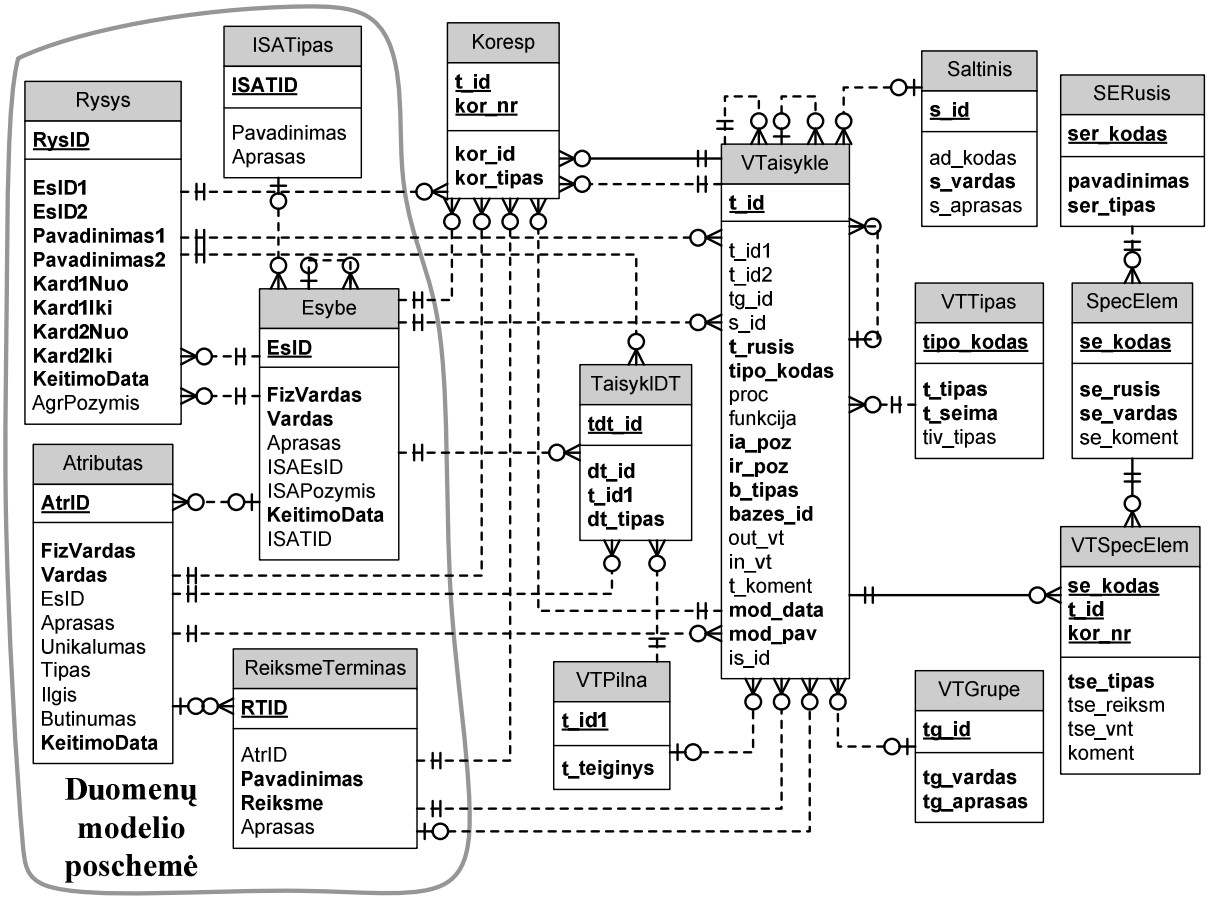
4 pav. Veiklos taisyklės grafinio žymėjimo iliustracija[5]

Modifikuotas Roso metodas pateikia VT saugyklos modelį (metamodelį). Saugykloje turi būti saugoma tik VT diagramoje atsispindinti informacija. 5 pav. pateiktas VT saugyklos metamodelis. Jame matyti, kad yra išskiriama duomenų modelio poschemė. Be duomenų modelio elementų išsaugojama ir reikšmių lentelė ReiksmeTerminas, kurioje bus saugomos VT diagramose naudojamos konstantos.

Kiekviena logiškai vientisa VT, išreiškiama viena diagrama, VTPilna gali susidėti iš keleto veiklos taisyklių VTaisykle, priklausančių vienam iš Roso metode numatytų tipų VTTipas. Kiekviena pilna taisyklė gali būti susijusi su daugeliu dalykinę sritį charakterizuojančių duomenų modelio elementų iš lentelių Esybe, Rysys, Atributas, ReiksmeTerminas. Kiekviena veiklos taisyklė VTaisykle, turi bazę ir vieną ar daugiau

korespondentų Korespondentas. Korespondentu, kaip ir baze, gali būti duomenų modelio elementas, konstanta kita VT VTaisykle arba kitos VT išeigos reikšmė. Veiklos taisyklės VTaisykle diagramoje gali būti naudojami vienas ar daugiau specialiųjų simbolių SpecElem, kurie sugrupuoti pagal rūšį SERusis.

Be tiesiogiai VT diagramoje atsispindinčios informacijos yra saugomi ir papildomi duomenys apie taisyklės kilmės šaltinį Saltinis, be to, taisyklės gali būti grupuojamos į įvairaus pobūdžio grupes VTGrupe.[6,9]



5 pav. Modifikuotu Roso metodu formalizuotų VT saugyklos loginės struktūros modelis[17]

2.3 Kodo generatoriai

Nėra žinoma apie programinio kodo generavimo iš grafinių veiklos taisyklių modelių įrankių egzistavimą, tačiau kitose srityse programinio kodo generavimui įrankių yra išties nemažai. Čia paminėti atskirų įrankių ko gero nevertėtų, tiesiog galima pažymėti, kad egzistuoja skirtingų tipų kodo generatoriai:

- kompiliatoriai – kodo generatoriai, kurie paprastai tam tikros kalbos programinį kodą verčia kito tipo kodu (pvz. mašininiu kodu), kurį gali vykdyti kita mašina (dažnai kompiuteris).[10]

- aukšto lygio programavimo kalbų kodo generatoriai – tai programos automatiškai generuojančios kodą tokioms kalboms, kaip C++, C#, Java, Perl, Python, Ruby ir daugelis kitų. Šie įrankiai skiriasi savo galimybėmis priklausomai nuo sudėtingumo.

Kodo generatoriai dar gali būti skirstomi priklausomai nuo savo modelio į:

- aktyvius
- pasyvius

Aktyvūs kodo generatorių modeliai – tai generatoriai sukuriantys kodą, kurį palaiko ilgą laiką. Pasyvūs kodo generatorių modeliai – tai generatoriai, vieną kartą sukuriantys kodą ir palaikymo problemas paliekantys programuotojui, tai taip vadinami meistrai (wizard).

Naudojant kodo generatorių suteikiami šie privalumai sugeneruotam kodui:

- Itin nuoseklus (generuotas mašinos) programinis kodas.
- Keičia programuotojų darbo pobūdį: kodas tampa švaresnis ir paprastesnis, kadangi jam tereikia daryti taip ko reikia konkrečiu momentu. Reikalavimams vėliau pasikeitus, tiesiog modifikuojamas generatorius ir greitai gaunama nauja kodo versija.
- Stabilus ir neturintis klaidų, dirba iš pirmo karto (programos derinimas vykdomas rašant patį kodo generatorių, taigi programos derinimo darbas jau būna įvykdytas prieš projekto pradžią)
- Programinis kodas sukuriamas labai greitai
- Generatoriai yra tinkinami (turint programinį kodo generatoriaus kodą, galima lengvai tinkinti kodą, kuris bus generuojamas)
- Programuotojui suteikiama laisvė susitelkti ties tomis kūrimo sritimis, kurioms reikia daug mąstymo ir taip padidinamas programuotojo produktyvumas
- Atlieka patarėjo vaidmenį (generuotas kodas yra 100% nuoseklus, todėl programuotojai gali perimti programinio kodo stilių iš generuoto kodo, bei iš jo pasimokyti. Iš to seka, kad visas programinis kodas tampa labiau nuoseklus ir aiškus, nepaisant to fakto, kad yra kuriamas labai didele sparta).

Kodo generatorių trūkumai:

- Pirmiausia reikia parašyti kodo generatorių
- Pritaikomi tik esant tam tikroms sąlygoms
- Visada yra toks kodas, kuris privalo būti parašytas programuotojo. Šio kodo kiekis priklauso nuo konkretaus projekto. Paprastai, generuotas kodas yra

daugiau pagalbinis programuotojo rašomam kodui, pvz. API (angl. application programming interface – programų kūrimo sąsaja)

- Siekiant naudoti duomenų bazėms, jos privalo būti gerai suformuotos (teisingai suprojektuotos ir normalizuotos). Paprastai kodo generatoriams iškyla problemų juos naudojant su duomenų bazėmis, kurios turi ypatingų ar unikalių projektavimo savybių.

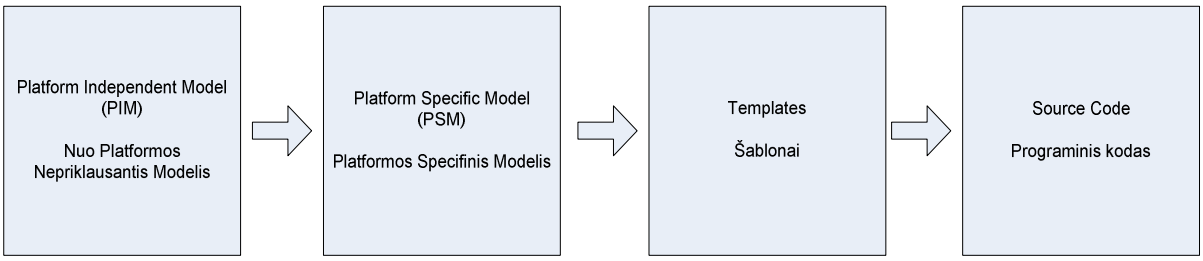
2.3.1 MDA – modeliu paremta architektūra

Modeliu paremta architektūra MDA (Model-driven architecture) yra rinkinys standartų, kuriuos nustatė OMG (Object Management Group). Standartų sąraše yra:

- UML
- XMI
- MOF
- OCL
- CWM

Šiais standartais remiasi generatoriai naudojami UML modeliams paversti į veikiantį Java, C++, C# programinį kodą. Idealiu atveju MDA generatoriai generuoja pilnai veikiančias programas, sukonstruotas iš aibės sujungtų komponentų, kurie gaunami iš vieno ar kelių įvesties modelių. Vienas iš svarbiausių MDA principų yra skirtingų lygių modeliai. Aukščiausiame lygmenyje yra nuo platformos nepriklausantis modelis PIM (Platform Independent Model). Platforma, šiame kontekste reiškia kažką panašaus į .NET ar J2EE karkasus. Taigi nuo platformos nepriklausantis modelis, žymį modelį, kuris nepriklauso nuo jokios sistemos specifikos. PIM specifikuoja ir susieja esybes labiausiai abstrakčioje formoje, kurioje dar išpildomi visi reikalavimai. Sekantis, žemesnis lygmuo yra platformos specifinis modelis PSM (Platform Specific Model). Šis modelis yra papildyta PIM versija, kurioje įtraukiamos visos reikalingos struktūros reikalingos įgyvendinti PIM, bet vis dar modeliuojamas UML formoje. Generatoriai naudoja PSM, kad sukurtų išvesties kodą. Bendras architektūros vaizdas gali būti pateikiamas 6 pav. pavaizduota schema. Rodyklės tarp kvadratų (žr. 6 pav.) žymi transformacijas, kurios yra apibrėžiamos transformacijos taisyklių. MDA nustato taisyklių egzistavimą, bet leidžia apibrėžti šias taisykles.

PIM ir PSM modelių išskyrimas MDA architektūroje yra esminis kodo generavimui. Lygmuo taro modelio ir šablonų pasirūpina susiejimu tarp aukšto lygio modelio ir įgyvendinimo specifikos konkrečioje sistemoje. Tai reiškia, kad šablonai gali būti orientuoti vien vertimą į kodą vietoje to, kad tvarkyti tiek vertimą į kodą, tiek modelio interpretavimą.



6 pav. MDA architektūros schema (adaptuota iš [11])

Atskyrimas į atskiras sritis MDA modelyje gali būti palygintas į architektūrinį 3-jų pakopų web serverio modelį. Nepriklausomo modelio dalykai yra laikomi atskirai nuo platformos specifinio modelio, o verslo modelio dalykai laikomi atskirai nuo vartotojo sąsajos. Šis daugiapakopis požiūris programų inžinerijoje yra laikomas įprastu skaidymu. Tai palengvina verslo modelio atsiejimą ir įgalina kurti aiškesnį kodą. Visų transformacijų panaudojimas viename lygmenyje įveda daugiau sistemos informacijos į abstraktų modelį ir gaunamas išvesties kodas yra prastesnės kokybės.

Tam, kad padėtų kurti PSM modelius konkrečiai platformai, MDA standartai pateikia rinkinį profilių, kurie naudojami platformos specifiniame modelyje (PSM) iškviesti specifines technologijas (beans, assemblies).

Vertėtų atkreipti dėmesį į XMI ir OCL panaudojimą MDA modelyje. XMI (XML Metadata Interchange) yra standartas metaduomenų informacijos apsikeitimui panaudojant XML. XMI – tai savotiškas UML modelių eksportavimo standartas. Jis panaudojamas kaip įvestis MDA generatoriui. Bendru atveju nuo platformos nepriklausantis UML modelis yra palaikomas MOF (Meta Object Facility) saugykloje ir per XMI eksportuojamas generatoriui. Generatorius tada perskaito XMI ir pritaiko jam transformacijas tam, kad sukurtų platformos specifinį modelį PSM. Šis modelis tada panaudojamas kaip įvestis tam tikram šablonų rinkiniui, kuris sukuria išvesties programinį kodą. Tuo tarpu OCL (Object Constraint Language) yra panaudojama kartu su UML, kad būtų sukurtas pilnas modelis (PIM arba PSM). OCL naudojama modelio detalizavimui išgaunant papildomą apribojimų detalumo lygmenį. [11]

2.3.2 Kodo generatorių tipai

- **Kodą apdorojantis generatorius.** Kodą apdorojantis generatorius (angl. code munger), kuris kaip pradinius duomenis gauna tam tikrą programinį kodą, kuriame išrenkamos tam tikros svarbios savybės ir sugeneruojamas išvesties failas. Toks generatorius turi gana platų pritaikymą, pvz. galima panaudoti dokumentacijos generavimui arba konstantų ar funkcijos prototipų nuskaitymui iš failo.

- **Eilutėmis kodą plečiantis generatorius.** Eilutėmis kodą plečiantis generatorius (angl. inline-code expander) taip pat kaip įvestį priima programinį kodą ir sukuria generuotą kodą išvestyje. Įvesties failas turi specialų žymėjimą, kurį generatorius pakeičia generuotuoju kodu, kurdamas galutinį rezultatą. Eilutėmis kodą plečiantys generatoriai dažnai naudojami įterpti SQL į programinio kodo failą. Programuotojai pažymi SQL kodą atitinkamomis žymėmis ir generatoriui perskaičius tokią žymę yra įterpiamas kodas, kuris realizuoja SQL užklausą ar komandą. Esminė idėja yra išlaikyti kodą nepriklausomą nuo infrastruktūros, kurios reikia užklausos valdymui.
- **Mišraus kodo generatorius.** Mišraus kodo generatorius (angl. mixed-code generator) nuskaito programinio kodo failą ir pakeičia kodą tam tikrose vietose. Skirtumas nuo eilutėmis kodą plečiančio generatoriaus yra tas, kad mišraus kodo generatorius išvesties kodą įveda į pradinį įvesties failą. Šio tipo aktyvus generatorius ieško specialiai suformuotų komentarų ir juos radęs papildo tam tikru reikalingu kodu. Mišraus kodo generavimas turi daug panaudojimo galimybių. Vienas iš dažno panaudojimo atvejų yra generuoti išrikiavimo kodą, kuris keičia informaciją tarp dialogo valdiklių ir jų atitinkamų kintamųjų duomenų struktūroje. Komentarai kode nurodo atvaizdavimą tarp duomenų elementų ir valdiklių, ir mišraus kodo generatorius pridedą realizaciją, kuri atitinkamai parenkama pagal komentaro specifikaciją.
- **Dalinis klasės generatorius.** Dalinis klasės generatorius (angl. partial class generator) nuskaito abstraktų apibrėžiamąjį failą, kuris turi pakankamai informacijos tam, kad būtų sukurtas klasių rinkinys. Sekantis etapas yra panaudoti šablonus išvesties bazinių klasių bibliotekų sukūrimui. Šios klasės po to kompiliuojamos su klasėmis sukurtomis programuotojų ir taip užbaigiamas klasių rinkinio kūrimas.
- **Pakopos arba lygmens generatorius.** Šiame modelyje generatorius tenka generuoti pilną lygmenį iš n lygmenų sistemos. Kaip vieną iš pavyzdžių šio tipo generavimo galima įvardinti jau aptartą MDA, kur UML kuriančioji programinė įranga naudojama kartu su generatoriumi ir dažnai XML pobūdžio apibrėžiančiuoju failu tam, kad sukurtų vieną ar daugiau sistemos lygmenų arba pakopų. Esminis skirtumas tarp lygmens generatoriaus ir dalinio klasės generatoriaus yra tas, kad lygmens generatorius sukuria visą kodą lygmeniui, kai tuo tarpu dalinis klasės generatorius sukuria bazines klases, bet būtina dar parašyti išvestines klases, kad būtų užbaigtas visas lygmuo. Dalinio klasės generatoriaus privalumas yra realizacijos sparta. Didžiausi sunkumai kuriant pakopos arba lygmens generatorius yra ypatingų atvejų projektavimas. [12]

2.4 Veiklos taisyklių koncepcijos analizės išvados

1. Apžvelgta veiklos taisyklių koncepcija, veiklos taisyklių klasifikavimo ir struktūrizavimo modeliai ir iš jų išskirtas modifikuotas Roso metodas kaip ryškiausias grafinis VT klasifikavimo ir struktūrizavimo modelis.
2. Išnagrinėti kodo generatorių pavyzdžiai, įgalina pritaikyti metodikas, kuriant kodo generatorių darbui su grafiniais veiklos taisyklių modeliais.
3. Nepavyko surasti egzistuojančios metodikos, kaip grafinius veiklos taisyklių modelius paversti programiniu kodu.

3. Kodo generavimo iš grafinių VT modelių metodika

3.1 Kodo generavimo koncepcija

Siekiant generuoti programinį kodą, reikalingas kodo generatorius. Apžvelgus prieinamas atvirojo kodo alternatyvas, nuspręsta generatorių realizuoti Ruby programavimo kalba [13]. Ši kalba puikiai pritaikoma tiek mažoms, tiek didelėms objektiškai orientuotoms programoms kurti.

Kodo generavimui iš modelių naudojami šablonai, o darbui su jais labai svarbios šablonų sistemos. Be jų generuojamas kodas išsidėsto viename lygmenyje su kitas funkcijas atliekančiu generatoriaus kodu, pvz. generuojant Java kodą Java programavimo kalba parašytu generatoriumi, tampa sudėtinga atskirti kur aprašomas generuojamas programos kodas, o kur generatoriaus vykdomasis kodas. Nagrinėjamu atveju buvo nuspręsta naudoti Ruby kalbai pritaikytą ERB tekstinių šablonų sistemą [14]. ERB pasižymi tuo, jog yra kompaktiškas, paprastas, patikimas ir portatyvus. Naudojant ERB, išsprendžiama vykdomojo ir generuojamo programinio kodo atskyrimo problema, nes skirtingos paskirties kodas patalpinamas atskiruose failuose.

Aukščiau minėtų elementų tarpusavio sąveika apibendrinta 1 paveikslėlyje pateiktoje koncepcinėje siūlomo kodo generatoriaus architektūros schemoje. Nagrinėjamu atveju generatorius tiesiogiai komunikuoja su VT saugykla, kurioje saugoma informacija apie VT modelius, kurie turės būti transformuojami į programinį kodą.

3.1.1 ERB ir kodo generavimo šablonų metodika

ERB panaudojamas darbui su šablonais tokiu principu: atidaromas šablono failas, sukuriamas ERB objektas su nurodytu šablonu ir ERB objekto pagalba šablonas įvykdomas, pavyzdžiui:

```
File.open( sablonoFailas ) { |sablonas|
  erb = ERb.new( sablonas.read )
  erb.result( binding )
}
```

Reikia pažymėti, kad *result* metodo viduje iškviečiamas metodas *binding* grąžina susiejamus kintamuosius esamame kontekste. Šiuo atveju tai reiškia, kad šablonas turės priėjimą prie *erb* ir *sablonas* lokalių kintamųjų. Tokiu principu ERB šablonui perduodami argumentai.

Paprasčiausias ERB šablonas gali būti tiesiog paprastas tekstas:

```
SELECT * FROM
```

Tokio šablono vykdymo rezultatas būtų:

```
SELECT * FROM
```

Norint pasiekti kažką sudėtingesnio naudojamos dvi ERB konstrukcijos. Pirmoji yra aprašoma šitaip: <% ... %> ir naudojama įterpti Ruby kalbos kodui į šabloną. Kodas yra įvykdomas, tačiau vykdymo rezultatas ignoruojamas ir neišvedamas, pavyzdžiui:

```
<% sqlTable = “Manager” %>
SELECT * FROM
```

Rezultatas lyginant su ankstesniu pavyzdžiu išlieka nepakitęs:

```
SELECT * FROM
```

Niekas nepasikeičia, kadangi priskyrimo operacija, nors ir įvykdoma, jos rezultatas yra ignoruojamas. Šiai situacijai pakeisti taikoma antroji ERB konstrukcija: <%= ... %>. Ši konstrukcija bloko išraiškos rezultatą perduoda į išvesties srautą. Pritaikius abi konstrukcijas, galime gauti tokį šabloną:

```
<% sqlTable = “Manager” %>
SELECT * FROM <%= sqlTable %>
```

Rezultatas:

```
SELECT * FROM Manager
```

Pirmosios konstrukcijos metu atliekamas priskyrimas, kurio reikšmė antros konstrukcijos dėka yra išvedamas, taip sudarant SQL užklausą. Palyginimui pateikiama kaip sudaromas šablonas Open PROMOL kalboje [16]:

```
$
"Define table: " {Manager, Department, Library} cn := Manager;
$
SELECT * FROM @sub[cn]
```

Rezultatas:

```
SELECT * FROM Manager
```

```
SELECT * FROM Department

SELECT * FROM Library
```

Kadangi ERB šablonuose galima naudoti Ruby kalbą, tai panaudojant jos konstrukcijas, sudaromos sudėtingesnės SQL užklausos. Pavyzdžiui, turime masyvą aprašantį duomenų bazės lenteles:

```
sqlTables = []

sqlTables.push("Employee")
sqlTables.push("Manager")
sqlTables.push("Department")
```

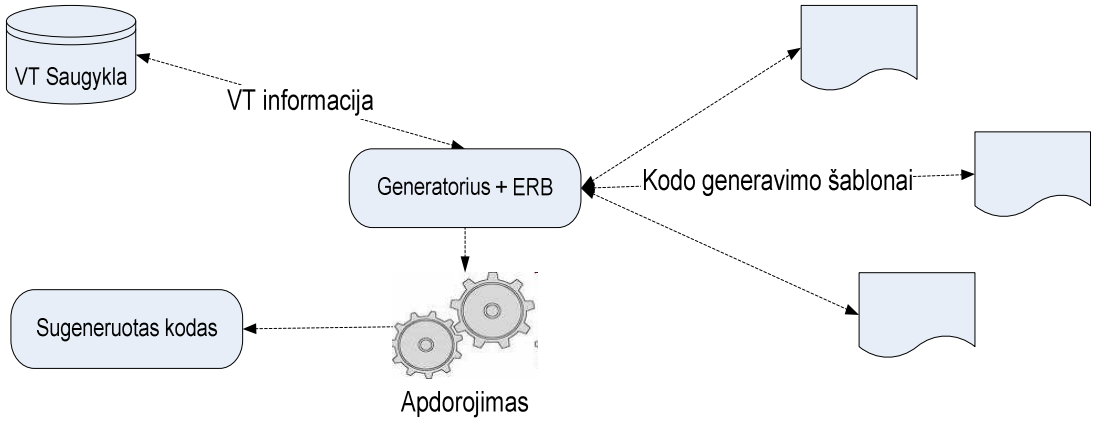
Taip pat turime tokį užklausos generavimo šabloną:

```
<% rez = "" %>
SELECT * FROM <% sqlTables.each { |table|
  rez = rez + table + ", "
}%><%= rez.chomp(", ")%>
```

Perdavus masyvą pateiktam šablonui ir jį įvykdžius gaunamas rezultatas:

```
SELECT * FROM Employee, Manager, Department
```

Tokiu būdu sudaroma dinamiškai kintanti SQL kodo dalis.



7 pav. Kodo generatoriaus architektūra

3.1.2 Kodo generavimo iš VT modelių metodika

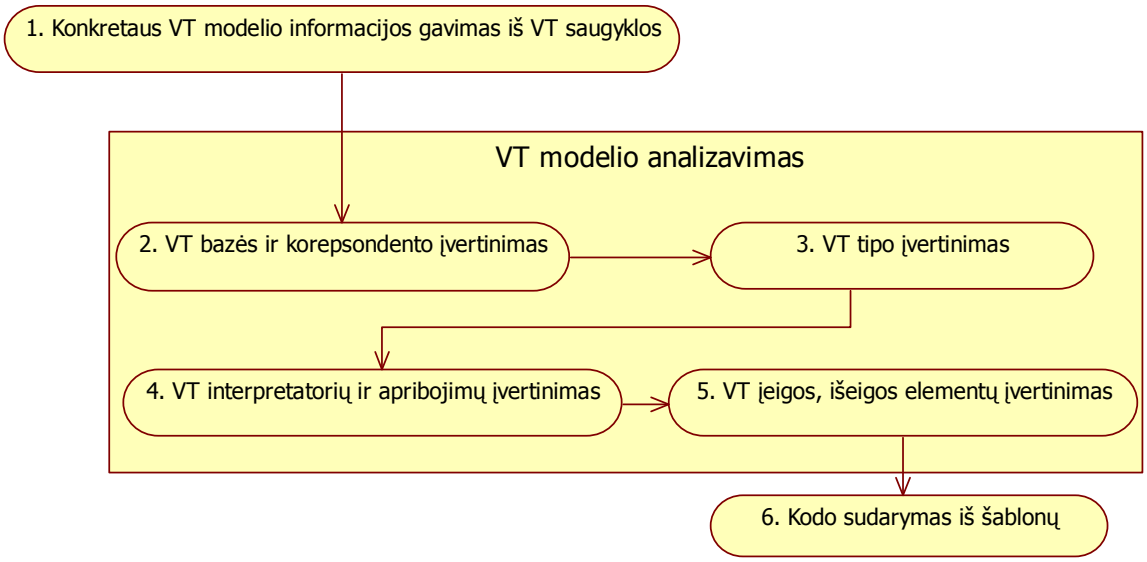
Atlikus Roso metodo analizę nuspręsta, jog šio tyrimo rėmuose tikslingiausia generuoti SQL kalbos kodą (akcentuojamas apribojimų generavimas), nes:

1. Roso VT diagramos iš esmės yra deklaratyvaus, o ne procedūrinio pobūdžio, jose dažniausiai bus fiksuojami apribojimai duomenų modelio objektams (lentelėms, atributams, ryšiams).

2. Siekiama užtikrinti greitą susijusių sistemos charakteristikų keitimąsi pasikeitus atitinkamai veiklos taisyklei. SQL apribojimai leidžia greitai įdiegti tam tikras darbo su duomenų bazėje saugoma informacija taisykles.

Roso metodas teigia, kad VT šeimos viena nuo kitos skiriasi tuo, kad sugeba atlikti tik sau unikalų patikrinimą arba testą. Remiantis šiuo išskirtinumu, kiekvienai VT kuriamas atskiras kodo generavimo šablonas, kuris realizuoja tai taisyklei unikalų testą generuojamoje kalboje.

Įvertinant šį ir kitus, aukščiau aptartus sprendimus, sukurta kodo generavimo iš VT modelių metodika, kurios apibendrinta schema pateikiama žemiau pateiktame 8 paveikslėlyje.



8 pav. Kodo generavimo koncepcinė schema

Metodiką sudaro šeši žingsniai, apibrėžiantys bendrą veiksmų seką įvertinant VT modelį:

1. Konkretaus VT modelio informacijos gavimas iš VT saugyklos. Šiuo konkrečiu atveju priimama, jog saugykla sukurta pagal minėtąjį Roso metamodelį. Tokiu būdu ne tik sukurta galimybė nesudėtingai realizuoti metodiką, praplečiant anksčiau sukurtąjį prototipą, bet ir sustiprintos prielaidos metodikos tyrimams bei potencialiam platesniam taikymui ateityje

2. VT bazės ir korespondento įvertinimas, nustatant bazės ir korespondento(-ų) tipus (tai gali būti lentelės, atributai, ryšiai). Pavyzdžiui,, jeigu bazė arba korespondentas yra atributas, surandama, kokiai lentelei priklauso atributas ir patikrinama, kokie yra lentelių tarpusavio ryšiai, nes juos reikia atspindėti generuojamame kode. Tarkim tarp lentelių esant sąryšiui 1:N, SQL kodą gali tekti papildyti WHERE sąlyga su konkrečia nurodoma reikšme ir t.t.

3. VT tipo įvertinimas. VT tipas apsprendžia, koks patikrinimas bus taikomas korespondentui bazės atžvilgiu, pvz., VT tipas SUM (sumuojama) nurodo, kad korespondento reikšmės privalo būti sumuojamos.

4. VT interpretatorių ir apribojimų įvertinimas. Specialieji elementai gali žymiai pakeisti VT prasmę, priklausomai nuo jų tipo. Pavyzdžiui, panaudojant interpretatorių, taisyklės galiojimą galima išplėsti nuo konkretaus egzemplioriaus iki visos aibės egzempliorių tikrinimo, - tokiu atveju SQL kodas turi būti suformuojamas visai egzempliorių aibei.

5. VT įeigos, išeigos elementų įvertinimas. Vertinamos modelyje nurodytos įeigos, išeigos taisyklės ir išeigos reikšmės. Teisingas šių galimų scenarijų įvertinimas svarbus, nes konkrečios VT patikrinimo rezultatas gali būti išsaugomas jos išeigos reikšmėje vėlesniam panaudojimui ir pan.

6. Kodo sudarymas pagal šablonus. Panaudojant apdorotą informaciją, iš pagal šablonus sugeneruotų teksto fragmentų gaunamas SQL kodas. Pavyzdžiui, generuojant SELECT sakinį perduodamas lentelių pavadinimų sąrašas, kuris panaudojamas užklausos FROM daliai formuoti.

3.1.2.1 Konkretaus VT modelio informacijos gavimas iš VT saugyklos

VT saugykloje saugoma aibė modelių, todėl norint sugeneruoti programinį kodą, pirmiausia reikia išrinkti konkretų VT modelį. Tai atliekama įvykdant reikalingas SQL užklausas ir šią informaciją perduodant toliau generatoriui analizuoti.

3.1.2.2 VT bazės ir korespondento įvertinimas

Bazė ir korespondentas yra elementai VT modeliuose visada žymintys tam tikrus duomenų tipus, bet tiek bazė, tiek korespondentas taip pat gali būti kita veiklos taisyklė.
Roso metodikoje duomenų tipai skirstomi taip:

- Duomenų objektai
    - Branduolio tipas
    - Charakteristikos tipas

    - Asociacijos tipas
- Duomenų elementai
    - Atributo tipas
- Junginiai
    - Ryšio tipas
    - Subtipo ryšys

Duomenų objektai reliacinėse duomenų bazėse išreiškiami kaip lentelės. Duomenų elementai (atributo tipas) išreiškiami kaip duomenų bazės lentelės atributai. Junginių ryšio tipas palaikomas per antrinius raktus, o subtipo ryšys gali būti reiškiamas kaip ryšys 1:1, kuris yra būtinas subtipui.

Remiantis modifikuotu Roso metodu ir jo metamodeliu VT bazės tipą nusako metamodelio lentelės *VTaisykle* atributas kaip parodyta 2 lentelėje. VT korespondento tipą nusako metamodelio lentelės *Koresp* atributas kaip parodyta 3 lentelėje. Greta pateikiamas galimas šių tipų atvaizdavimas SQL kode.

2 lentelė. Bazės tipai ir jų atvaizdavimas SQL

Atributas	***Aprašas***	***Galimos reikšmės***	***Atvaizdavimas SQL***
b_tipas	Veiklos taisyklės bazės pobūdis.	**E** – bazė yra esybė, **A** – bazė yra atributas, **R** – bazė yra ryšys, **S** – bazė yra ISA ryšys, **C** – bazė yra konstanta, **T** – bazė yra kita VT, **I** – bazė yra kitos VT išeigos reikšmė.	Esybė, Esybė.Atributas, WHERE sąlygos panaudojimas

3 lentelė. Korespondento tipai ir jų atvaizdavimas SQL

Atributas	***Aprašas***	***Galimos reikšmės***	***Atvaizdavimas SQL***
kor_tipas	Korespondento pobūdis.	**E** – korespondentas yra esybė. **A** – korespondentas yra atributas. **R** – korespondentas yra ryšys. **S** – korespondentas yra ISA ryšys. **C** – korespondentas yra konstanta. **T** – korespondentas yra kita VT negu ta, kuriai priskiriamas šis korespondentas. **I** – korespondentas yra kitos VT negu ta, kuriai priskiriamas šis korespondentas, išeigos reikšmė.	Esybė, Esybė.Atributas, WHERE sąlygos panaudojimas

3.1.2.3 VT tipo įvertinimas

VT tipas yra esminis veiklos taisyklės interpretavimo elementas. Kiekvienam tipui pagal klasifikaciją pateikiamos SQL kodo generavimo įžvalgos.

Egzempliorių patvirtinimo taisyklių tipų grupė Roso notifikacijoje (angl. instance verifiers) sudaryta iš šių taisyklių tipų:

1. **Privaloma** (angl. mandatory) patikrina ar yra bent vienas korespondento egzempliorius. Tokios taisyklės pavyzdys: užsakymas visada privalo turėti užsakymo datą. Tokiam taisyklių tipui galima generuoti *CHECK* tipo SQL kodą nurodant *IS NOT NULL* sąlygą.
2. **Apribota** (angl. limited) patikrina ar yra tam tikras ribotas kiekis korespondento egzempliorių. Tokios taisyklės pavyzdys: pirkėjas turi pateikti mažiausiai vieną užsakymą. Šio tipo taisyklėse visada nurodomas kiekis ir išskaičiavimo apribojimo tipas: apatinis, aukščiausias ar fiksuotas. *CHECK* apribojimas generuotinas panaudojant SQL *COUNT()* funkciją ir apribojimo tipus išreiškiant palyginimo operatoriais (>, <, = ir t.t.).

Tipo patvirtinimo (angl. type verifiers) grupės taisyklės veikia kaip loginis *IR (AND)* arba loginis *ARBA (OR)* ir jų grupę sudaro:

1. **Bendra** (angl. mutual) atlieka patikrinimą atsižvelgiant ar bazei egzistuoja keletas korespondentų vienu metu. Generuotinos loginės sąlygos kiekvienam iš bazės korespondentų ir apjungiamos loginiu *IR (AND)*. Taisyklės pavyzdys: užsakyme privalo būti nurodyta užsakymo data ir pirkėjas pateikęs užsakymą.
2. **Bendrai–atskirianti** (angl. mutually–exclusive) atlieka patikrinimą ar bazei egzistuoja ne daugiau kaip vienas korespondentas tuo pačiu metu. Taisyklės pavyzdys: pirkėjas gali būti arba vyriausybės, korporacinis arba individualus, bet niekada daugiau negu vienas iš šių tipų. Generuotinos loginės sąlygos patikrinant ar bazei vienas iš korespondentų įgauna loginę tiesos reikšmę *true* ir kiti korespondentai įgauna loginę netiesos reikšmę *false* panaudojant neigimą *NOT*.
3. **Bendrai–susiejanti** (angl. mutually–inclusive) atlieka patikrinimą ar bazei egzistuoja bent vienas korespondentas tuo pačiu metu. Taisyklė tenkinama, kai egzistuoja vienas korespondentas tenkinantis taisyklės reikalavimus, tačiau gali egzistuoti ir keli korespondentai tenkinantys taisyklės reikalavimus. Taisyklės pavyzdys: komanda visada privalo turėti bent vadybininką arba biudžetą. Generuotinos loginės sąlygos patikrinančios ar kiekvienas korespondentas įgyja loginę tiesos reikšmę *true*, šias

sąlygas tarpusavyje atskiriant loginiu *ARBA (OR)*, kadangi yra pakankama vienam korespondentui tenkinti sąlygas.

4. **Bendrai–uždraudžianti** (angl. mutually–prohibited) atlieka patikrinimą ar bazei neegzistuoja (nėra teisingas) bent vienas iš dviejų ar daugiau korespondentų tuo pačiu metu. Taisyklė tenkinama, kai bent vienas iš korespondentų įgyja loginę netiesos reikšmę *false*. Generuotinos loginės sąlygos patikrinančios ar kiekvienas korespondentas įgyja loginę netiesos reikšmė *false*, šias sąlygas tarpusavyje atskiriant loginiu *ARBA (OR)*, kadangi yra pakankama vienam korespondentui netenkinti sąlygas. Taisyklės pavyzdys: studentas vienu metu negali būti: sporto komandos narys, prasižengęs.

Pozicijos patvirtinimo (angl. postition verifiers) grupės taisyklės tikrina egzempliorių sekas pagal reikšmes (pozicinė, žemiausia, aukščiausia) ir chronologiją (chronologinė, seniausia, naujausia). SQL generavimui galima naudoti SQL funkcijas *MIN(), MAX(), FIRST(), LAST()*, bet tada korespondentai turėtų būti išrikiuoti pagal tam tikrą kriterijų, t.y. naudojamas atributas pagal kurį būtų taikomas rikiavimas *ORDER BY*, arba iš anksto priimtas apribojimas, kad korespondentai jau išdėstyti reikiama seka. Šią grupę sudaro:

1. **Pozicinė** (angl. positioned) nustato tam tikrą eiliškumą, pvz. antras, dvidešimt penktas, trečias nuo galo ir pan. Rikiavimas SQL pagal reikšmes.
2. **Žemiausia** (angl. lowest) atrenka žemiausias pozicijas. Panaudotina SQL funkcija *MIN()*.
3. **Aukščiausia** (angl. highest) atrenka aukščiausias pozicijas. Panaudotina SQL funkcija *MAX()*.
4. **Chronologinė** (angl. chronological) naudojamas nustatyti tam tikram chronologiniam eiliškumui. Rikiavimas SQL pagal datą, kadangi nagrinėjama ne korespondentų reikšmė, o amžius.
5. **Seniausia** (angl. oldest) atrenka seniausią. Galima naudoti SQL funkcijas *MIN(), MAX(), FIRST(), LAST()*.
6. **Naujausia** (angl. newest) atrenka naujausią. Galima naudoti SQL funkcijas *MIN(), MAX(), FIRST(), LAST()*.

Funkcinio patvirtinimo (angl. functional verifiers) grupės taisyklės tikrina korespondento reikšmes, kaip bazės egzempliorių funkcijas. Taikant tokius testus visada atsižvelgiama į taisyklės bazės egzempliorių tvarką ir taisyklės tipą. Kadangi VT modelio diagramose nurodomas tik taisyklės tipas, tačiau nenurodoma jokia informacija apie egzempliorių išsidėstymą, todėl siekiant generuoti SQL kodą reikia priimti išankstinį

apribojimą, kad duomenų egzemplioriai turi atributus, pagal kuriuos galima taikyti veiklos taisyklės patikrinimą. Funkcinio patvirtinimo taisyklių grupę sudaro:

1. **Funkcinė** (angl. functional), kuri yra bendras atvejis funkcinio patvirtinimo taisyklių grupėje. Taikant šią taisyklę konkretus testas turi būti nurodomas analitiko. SQL kodo generavimas šiai taisyklei būtų sudėtingas, kadangi taisyklės interpretacija atskleidžiama modelyje pateikiamu aprašu.
2. **Unikali** (angl. unique) pabrėžia reikšmių unikalumą, t.y. jokie du bazės egzemplioriai negali turėti tokių pačių korespondento reikšmių. Taikytinas SQL apribojimas *UNIQUE*.
3. **Svyruojanti** (angl. fluctuating) nurodo korespondento reikšmių varijavimą kiekvienam sekančiam bazės egzemplioriui. Šiuo atveju tai reikštų, kad du gretimi įrašai negali sutapti. Tam reikėtų nagrinėti duomenis poromis, jiems tarpusavyje pritaikant *DISTINCT*.
4. **Auganti** (angl. ascending) nurodo, kad kiekvienam sekančiam bazės egzemplioriui korespondento reikšmės privalo tik didėti. Reikalaujama *UNIQUE* ir kad sekoje visi korespondentai rikiuotųsi didėjimo tvarka.
5. **Krentanti** (angl. descending) nurodo, kad kiekvienam sekančiam bazės egzemplioriui korespondento reikšmės privalo tik mažėti. Reikalaujama *UNIQUE* ir kad sekoje visi korespondentai rikiuotųsi mažėjimo tvarka.
6. **Neatnaujinama** (angl. non–renewable) nurodo, kad korespondento reikšmė gali kartotis nepertraukiamai bazės egzempliorių sekoje, tačiau pertraukus šią seką, korespondentui draudžiama įgyti šią reikšmę.

Palyginamojo įvertinimo (angl. comparative evaluators) grupės taisyklės apibrėžia palyginimą tarp dviejų ar daugiau vienodo duomenų tipo egzempliorių. Šie duomenų tipai dažniausiai būna atributo tipo, kas reiškia, kad palyginamos atributų vertės. Palyginamojo įvertinimo grupę sudaro:

1. **Lygu** (angl. equal–to)
2. **Nelygu** (angl. not–equal–to)
3. **Daugiau–už** (angl. greater–than)
4. **Daugiau–arba–lygu** (angl. greater–than–or–equal–to)
5. **Mažiau–už** (angl. less–than)
6. **Mažiau–arba–lygu** (angl. less–than–or–equal–to)

Kadangi taisyklių prasmė aiški iš jų pavadinimų, kiekviena taisyklė nebus aptariama atskirai. SQL kodo generavime kiekvieną taisyklę galima atvaizduoti palyginimo operatoriais (>, <, =

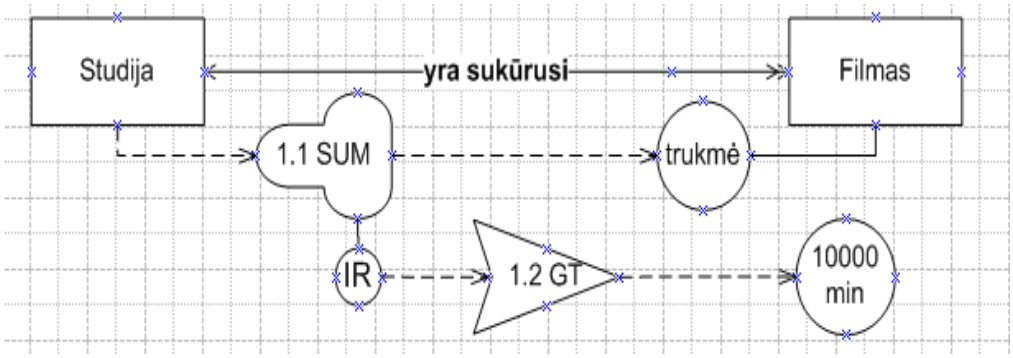
ir t.t.) atitinkamai jos pavadinimui. Taisyklės pavyzdys: projekto išlaidos privalo būti mažesnės arba lygios projekto biudžetui.

Matematinio įvertinimo (angl. mathematical evaluators) grupės taisyklės tikrina rezultatus standartinių skaičiavimų atliekamų su korespondentų reikšmėmis. Dažniausiai tai būna atributų tipo reikšmės. Jokios kitos taisyklės Roso metode negali atlikti matematinių skaičiavimų. Į matematinio įvertinimo taisyklių grupę patenka:

1. **Apskaičiavimo** (angl. calculated) taisyklė atspindi bendrą atvejį šios taisyklių šeimos. Naudojant šią taisyklę taikoma analitiko nurodyta skaičiavimo formulė. SQL kodo generavimas nepateikiamas tokioms taisyklėms, nes priklauso nuo apibrėžiamos formulės.
2. **Sumavimo** (angl. summed)
3. **Atimties** (angl. subtracted)
4. **Daugybos** (angl. multiplied)
5. **Dalybos** (angl. divided)
6. **Maksimumo** (angl. maximum)
7. **Minimumo** (angl. minimum)

Matematinio įvertinimo taisyklių grupė yra atvira, todėl ją sudaryti gali ne tik išvardintos, bet ir kitos, pvz. vidurkio, modos, procentilio ir kitos taisyklės. SQL kodo generavimas šios grupės taisyklėms grindžiamas SQL funkcijomis *max(), min(), sum(), avg()* ir kitomis matematinėmis funkcijomis. Taisyklės pavyzdžiai:

- viso departamento atlyginimus apskaičiuojame kaip sumą visų departamento darbuotojų atlyginimų
- studijos sukurtų filmų trukmė privalo būti didesnė nei 10000 minučių. Pastaroji taisyklė pavaizduota žemiau pateiktame 9 paveikslėlyje.



9 pav. VT modelio pavyzdys

Modelį sudaro dvi taisyklės: „susumuota“ tipo sąlyga (1.1) bei „daugiau-už“ tipo apribojimas (1.2). Pastarojo korespondentas - tai 1.1 taisyklės išeigos reikšmė, kurioje šiuo atveju atsispindi bendra konkrečios studijos sukurtų filmų trukmė. Taisyklėje apibrėžtas apribojimas

turi būti tenkinamas nuolat, todėl jį galima realizuoti SQL apribojimu, neleisiančiu atlikti duomenų modifikacijų, po kurių minėta sąlyga būtų pažeista. Taigi, pagal aptartąją metodiką bus sugeneruotas toks SQL apribojimas (priimama, jog lentelėje Filmas ryšį su lentele Studija realizuoja išorinis raktas studija – ši informacija be kita ko saugoma VT saugykloje):

```
CREATE ASSERTION BendraTrukme CHECK (10000 < ALL
(SELECT SUM(trukme)
FROM Filmas
GROUP BY studija));
```

Projekcijos valdikliai (angl. projection controllers) yra išskirtinė VT grupė, gebanti automatiškai įgalinti, kopijuoti arba iškviesti vykdymą. SQL generavimas šiai grupei gali įvairuoti. Dažniausiai tai būtų trigeriai, kurie priklausomai nuo sąlygos gali keisti duomenis taip, kad šie tenkintų taisyklę. Šią grupę sudaro:

1. **Įgalinta** (angl. enabled)
2. **Nukopijuota** (angl. copied)
3. **Įvykdyta** (angl. executed)

3.1.2.4 VT interpretatorių ir apribojimų įvertinimas

Interpretatoriai (angl. interpreters) ir apribojimai (angl. qualifiers) yra specialieji elementai galintys žymiai pakeisti VT prasmę. Naudojami elementai galintys būti tik interpretatoriais, apribojimais arba abiejų rūšių specialiaisiais elementais ir jie skirstomi į:

1. **Pagrindo**. Šie specialieji elementai dažniausiai žymi egzempliorių egzistavimą arba loginę tiesą. SQL kode šie elementai atvaizduojami *NULL, EXISTS, NOT* ir jų kombinacijomis Pagrindo specialiesiems elementams priklauso:
    - Tiesa (angl. true)
    - Netiesa (angl. false)
    - Be reikšmės (angl. unvalued)
    - Tiesa arba netiesa (angl. true ir false)
    - Netiesa arba be reikšmės (angl. false or unvalued)
    - Tiesa arba be reikšmės (angl. true or unvalued)
    - Tiesa, netiesa arba be reikšmės (angl. true, false or unvalued)
2. **Aktyvatorius**. Šie specialieji elementai dažniausiai žymi įvykius (angl. events), tokius kaip reikšmių sukūrimas, modifikavimas arba ištrynimas, todėl galimas panaudojimas SQL trigeriuose, tačiau dažnai laikoma, kad taisyklėse įvykio metu konkrečiam egzemplioriui nurodomas jo sukūrimo, modifikavimo laikai, kurie išsaugomi kaip egzemplioriaus išeigos reikšmės. Toks panaudojimas leidžia tikrinti taisyklę ne tik

įvykio metu, bet ir tam tikro įvykio metu, kuris buvo praeityje, todėl aktyvatoriai nebūtinai gali būti trigerius generavimą simbolizuojantys elementai. Aktyvatorių specialiesiems elementams priklauso:

- Sukūrimas (angl. creation)
- Modifikavimas (angl. modification)
- Trynimas (angl. deletion)
- Sukūrimas arba modifikavimas (angl. creation or modification)
- Modifikavimas arba trynimas (angl. modification or deletion)
- Sukūrimas arba trynimas (angl. creation or deletion)
- Sukūrimas, modifikavimas arba ištrynimas (angl. creation, modification or deletion)

3. **Iššaukimo būsenas**. Šie specialieji elementai žymi taisyklių, veiksmų sužadinimo būsenas. Iššaukimo būsenoms priklauso:
    - Neiškviesta (angl. not invoked)
    - Šiuo metu iškviesta (angl. currently invoked)
    - Iškviesta sėkmingai (angl. invoked successfully)
    - Iškviesta nesėkmingai (angl. invoked unsuccessfully)
    - Neiškviesta arba iškviesta sėkmingai, arba iškviesta nesėkmingai (angl. not invoked or invoked successfully, or invoked unsuccessfully)
    - Iškviesta sėkmingai arba iškviesta nesėkmingai (angl. invoked successfully, or invoked unsuccessfully)
    - Šiuo metu iškviesta arba iškviesta nesėkmingai (angl. currently invoked or invoked unsuccessfully)

    Šiems elementams kodo generavimo galimybės nėra tiriamos, kadangi Roso metode pateikiamas tik jų aprašymas, be jokio praktinio pritaikymo pavyzdžių.
4. **Specialiuosius interpretatorius**, kuriuos sudaro:
    - Neigimas (angl. negator) keičia prasmę į priešingą, todėl SQL kode verčiamas į *NOT*
    - Lygio pakėlimas (angl. elevator) taikomas, kai taisyklės taikymą norima perkelti nuo vieno konkretaus korespondento egzemplioriaus prie visos korespondentų egzempliorių aibės, tokiu atveju generuojamos SQL užklausos, kuriose atrenkami visi galimi įrašai vietoje konkretaus vieno, pvz., jeigu taisyklei sugeneruojamas *SELECT* sakinys su *WHERE* dalimi, kur nurodomas konkretus egzemplioriaus identifikacinis numeris, tai taisyklę papildžius Lygio

pakėlimu, tokios *WHERE* dalies atsisakoma, nes taikymas perkeliamas nuo konkretaus egzemplioriaus prie visos egzempliorių aibės.

- Atrinkimas taikomas bazei (angl. applied to anchor)
- Atrinkimas taikomas korespondentams (angl. applied to correspondents)
- Atrinkimas taikomas abiems (angl. applied to both)

  Visi trys atrinkimo tipai žymi aibės apribojimą, todėl taikomas SQL *WHERE* sakinys visais trimis atvejais. Atrinkimo taikomo bazei atveju *WHERE* dalyje atsižvelgiama į bazę, atrinkimo taikomo korespondento atveju atsižvelgiama į taisyklės korespondentą, o atrinkimo taikomo abiems atveju taikoma tokia SQL sakinio struktūra: *WHERE sąlyga_bazei AND sąlyga_korespondentui*.
- Inicijavimas tuojau pat (angl. immediate)
- Inicijavimas tarpinis (angl. intermediate)

  Inicijavimo interpretatoriams nėra galimybės generuoti SQL kodą.
- Įgalinimas konverguojantis (angl. converger)
- Įgalinimas perspėjantis (angl. suggester)
- Įgalinimas miegantis (angl. sleeper)

  Inicijavimo interpretatoriams nėra galimybės generuoti SQL kodą.

5. **Specialiuosius apribojimus**, kuriuos sudaro:
   - Bet kuris sąryšis (angl. any) yra specialusis elementas, kuris tiesiogiai neatvaizduojamas SQL kodu, tačiau yra svarbus taisyklės interpretavimui, kadangi nutraukia ryšius tarp taisyklės bazės ir korespondento. Paprastas pavyzdys galėtų būti: darbuotojas dirbantis departamente, bendru atveju būtų svarbu įvertinti darbuotoją kaip konkretaus departamento narį, o įvedus *bet kuris sąryšis* elementą, ši prasmė pasikeistų į darbuotojas dirbantis bet kuriame departamente, nesvarbu kuriame.
   - Visi korespondentai (angl. all) yra specialusis elementas taisyklės patikrinimą reikalaujantis tenkinti visiems korespondentams. Tarkime, gydytojas turi pacientus, kurie gali būti stabilioje, geroje, arba kritinėje būsenoje ir turime taisyklę, kuri patikrina ar yra bent vienas pacientas kritinėje būsenoje (korespondentas – kritinė paciento būsena). Šiam atvejui panaudojus specialųjį korespondentą taisyklė bus tenkinama ne tada kai bent vienas pacientas bus kritinėje būsenoje, tačiau kai visi gydytojo pacientai bus kritinėje būsenoje. Toks patikrinimas SQL kode gali būti realizuojamas panaudojant *COUNT()* funkciją ir patikrinant kiek yra korespondento egzempliorių konkrečiai bazei tenkinančių taisyklės nurodomą patikrinimą ir kiek iš viso yra šiai bazei

korespondentų egzempliorių, bei juos palyginant. Tam, kad būtų tenkinama taisyklė šis skaičius turėtų būti lygus.

- Rekursinis (angl. recursive) yra specialusis elementas, kuris nurodo į korespondentus susijusius per vieną ar daugiau rekursinio ryšio ar struktūros lygmenų. SQL kodas leidžia rekursiją išreikšti naudojant užklausas kitų užklausų viduje.
- Numeratorius fiksuotas (angl. fixed)
- Numeratorius ne maziau (angl. lower)
- Numeratorius ne daugiau (angl. upper)

  Numeratoriai atvaizduojami SQL kode, kaip konstantos ir dažniausiai galima naudoti palyginimo operacijose siekiant išreikšti numeratorių nurodomas ribas per SQL *COUNT()* funkciją.
- Loginis identifikatorius (angl. logical identificator), tai specialusis elementas išreiškiantis korespondentų egzempliorių seką, SQL kode jį galima išreikšti pasitelkiant rikiavimo funkcijas *ORDER BY*, tačiau šiuo atveju būtina įvesti apribojimą, kad korespondento egzempliorių duomenys būtų papildomi sekos atributais, be šios prielaidos SQL kodo generavimas negalimas.

3.1.2.5 VT įeigos, išeigos elementų įvertinimas

VT įeigos ir išeigos elementus galima skirstyti į dvi grupes:

1. Įeigos, išeigos taisyklės. Jos naudojamos kaip būdas jungti VT tarpusavyje ir dažniausiai išreiškia kokį nors konkretų duomenų aibės apribojimą. Dažniausiai naudojamos su atrinkimo (atrinkimas taikomas bazei, atrinkimas taikomas korespondentams, atrinkimas taikomas abiems) specialiaisiais interpretatoriais, todėl SQL atvaizduotinos kaip *WHERE* tipo sąlygos.
2. Išeigos reikšmės. Kartais VT tikrinimo rezultatas gali būti išsaugomas išeigos reikšmėje, todėl SQL atvaizduojamos kaip rezultatas konkrečių skaičiavimų.

3.1.2.6 Kodo sudarymas pagal šablonus

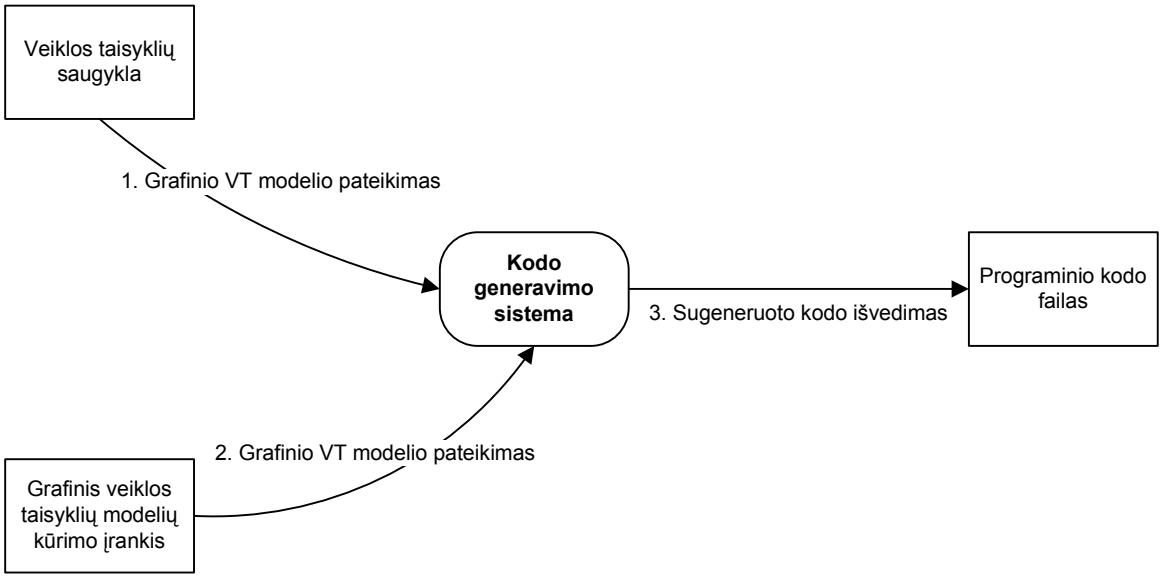
Paskutinis SQL kodo generavimo žingsnis sekantis po VT modelio analizės. Šio žingsnio metu šablono struktūra pagal ankstesniuose žingsniuose pritaikytus kriterijus susiejama su iš VT gauta informacija. Šioje metodikoje tai pasiekiama panaudojant ERB biblioteką.

3.2 Kodo generavimo iš VT modelių įrankio prototipo kūrimas

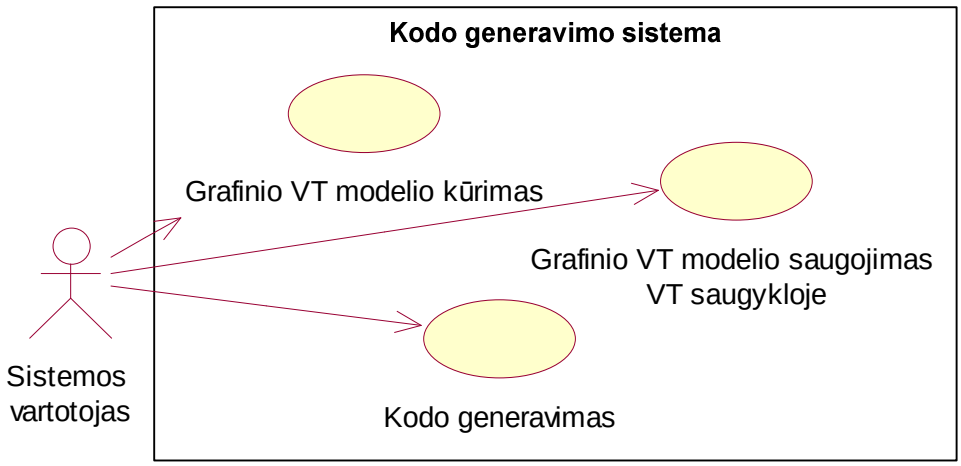
Siekiant įvertinti sukurtąją metodiką, buvo kuriamas ja grindžiamo įrankio prototipas. Kodo generavimo funkcionalumas turėjo papildyti jau anksčiau sukurtą MS Visio bazėje veikiantį Roso modeliavimo įrankio prototipą, kuris turėjo būti papildomai koreguojamas. Pagrindiniai prototipo metu atlikto reikalavimų analizės ir projektavimo darbų rezultatai pateikiami šiame skyriuje.

3.2.1 Veiklos kontekstas ir sistemos panaudos atvejai

Iš pradžių apibrėžiamas veiklos kontekstas (žr. 10 pav.) ir išskiriami kodo generavimo sistemos panaudos atvejai (žr. 11 pav.), kaip minimalaus funkcionalumo, kurį turi tenkinti generatorius reikalavimai.



10 pav. Veiklos kontekstas

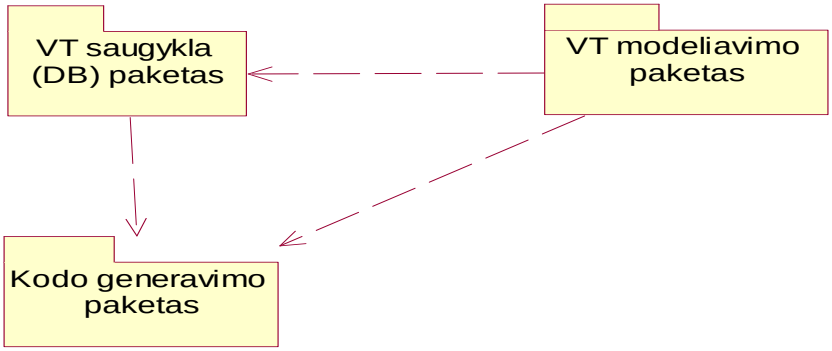


11 pav. Sistemos panaudos atvejai

3.2.2 Sistemos statinis vaizdas

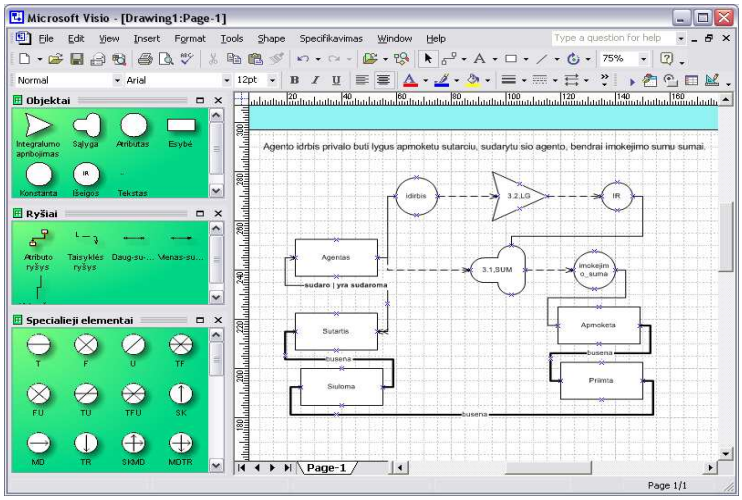
Šiame skyriuje pateikiamas sistemos skirstymas į paketus ir paketų detalizavimas. Paveikslėlyje pateikiamas sistemos skaidymas į paketus. Išskiriami 3 pagrindiniai paketai (žr. 12 pav.):

1. VT saugyklos (DB) paketas.
2. Grafinių VT modelių braižymo paketas.
3. Kodo generavimo paketas.



12 pav. Pagrindiniai sistemos paketai

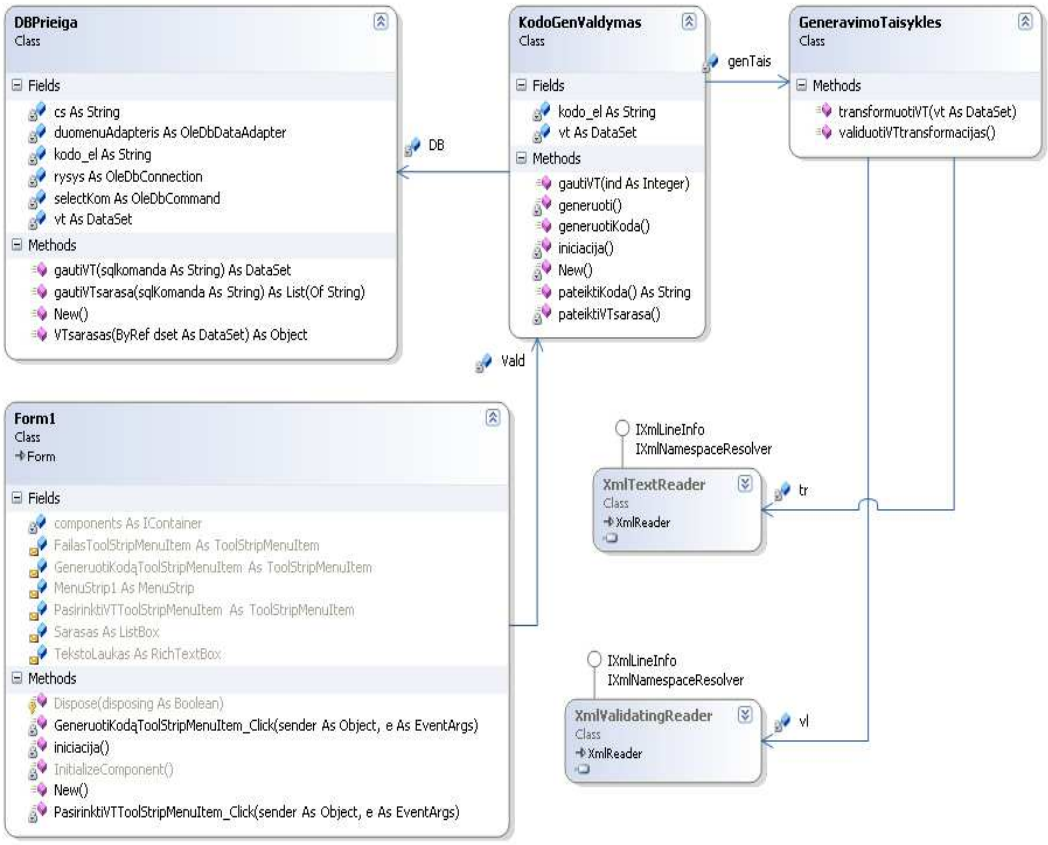
VT modeliavimo paketas. Naudojamas VT modeliavimo paketas yra iš anksto paruoštas įskiepis Microsoft Visio aplinkoje, kurio bendras vaizdas pateikiamas 13 paveikslėlyje. VT modeliavimo paketas papildomos trūkstamomis funkcijomis arba pataisomos esamos klaidos. Programuojama VBA kalba.



13 pav. Veiklos taisyklių modeliavimo įrankio langas

VT saugykla. VT saugyklos vaizdas ir detalesnė informacija pateikiama 2.2.4 „Grafinis veiklos taisyklių modelis. Modifikuotas Roso metodas“ skyriuje.

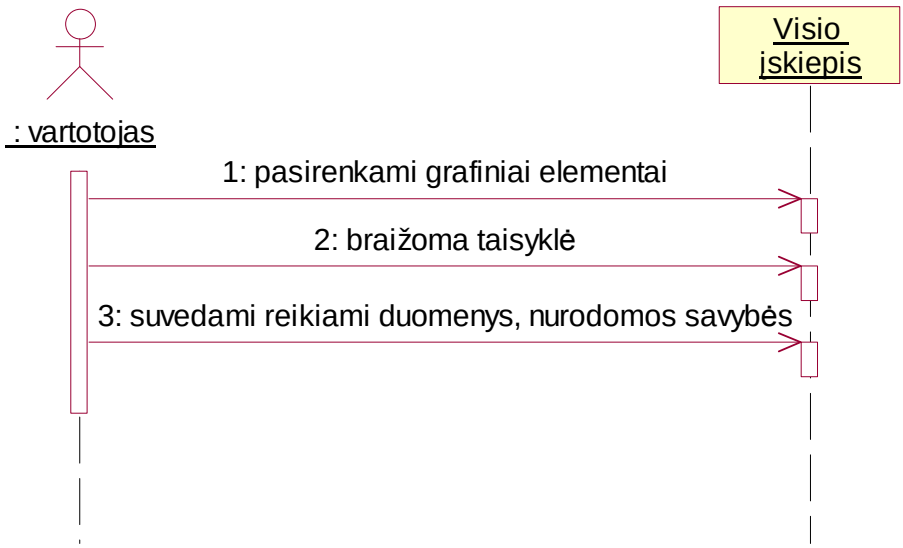
Kodo generavimo paketas. 14 pav. pateikiama kodo generavimo paketo klasių diagrama. Kodo generavimo paketui naudojama Ruby ir erb (eRuby) šablonų mechanizmas.



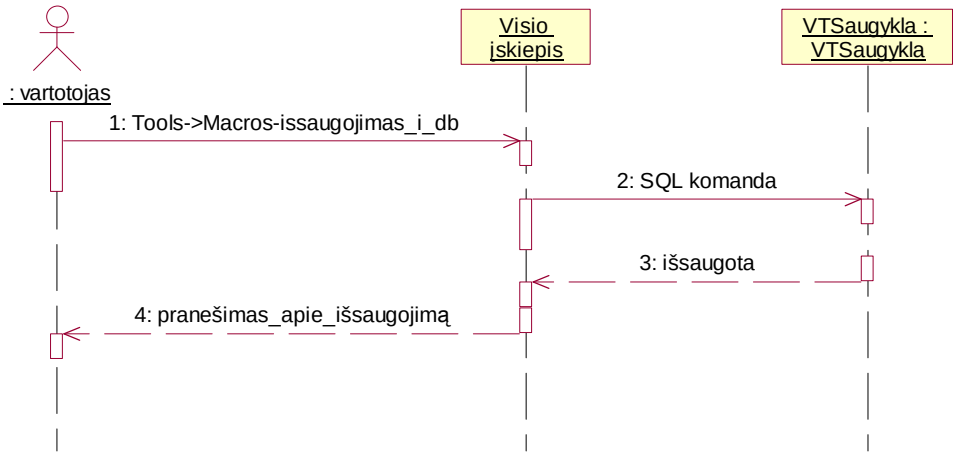
14 pav. Kodo generavimo paketas (klasių diagrama)

3.2.3 Sistemos dinaminis vaizdas

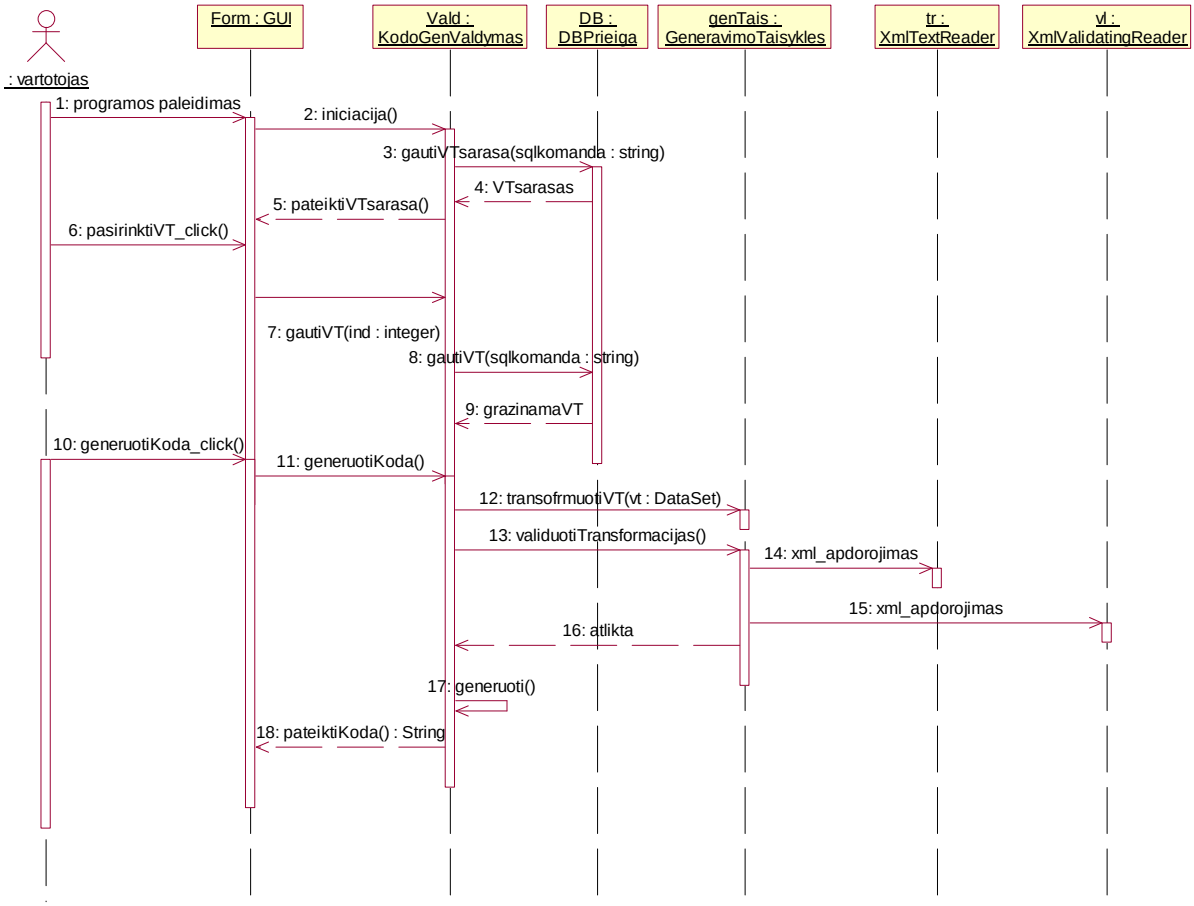
Sekų diagramos. 15 pav. pateikiama VT modelio kūrimo sekų diagrama. 16 pav. pateikiama VT modelio išsaugojimo sekų diagrama. 17 pav. pateikiama kodo generavimo sekų diagrama.



15 pav. VT modelio kūrimo sekų diagrama

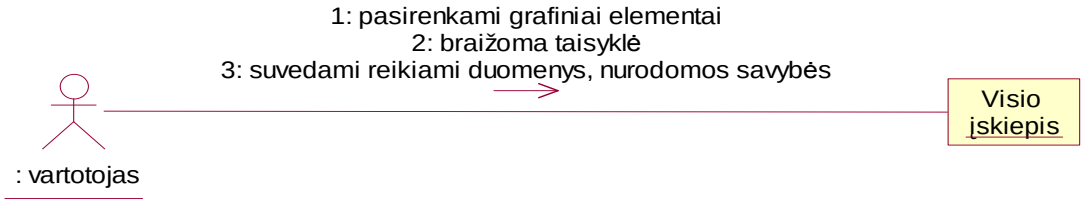


16 pav. VT modelio išsaugojimo sekų diagrama

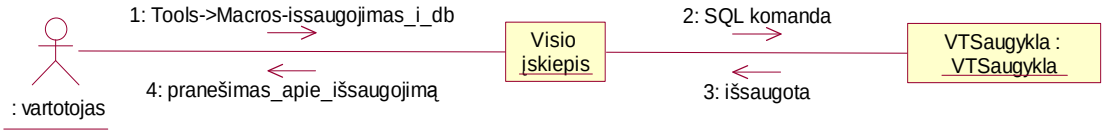


17 pav. Kodo generavimo sekų diagrama

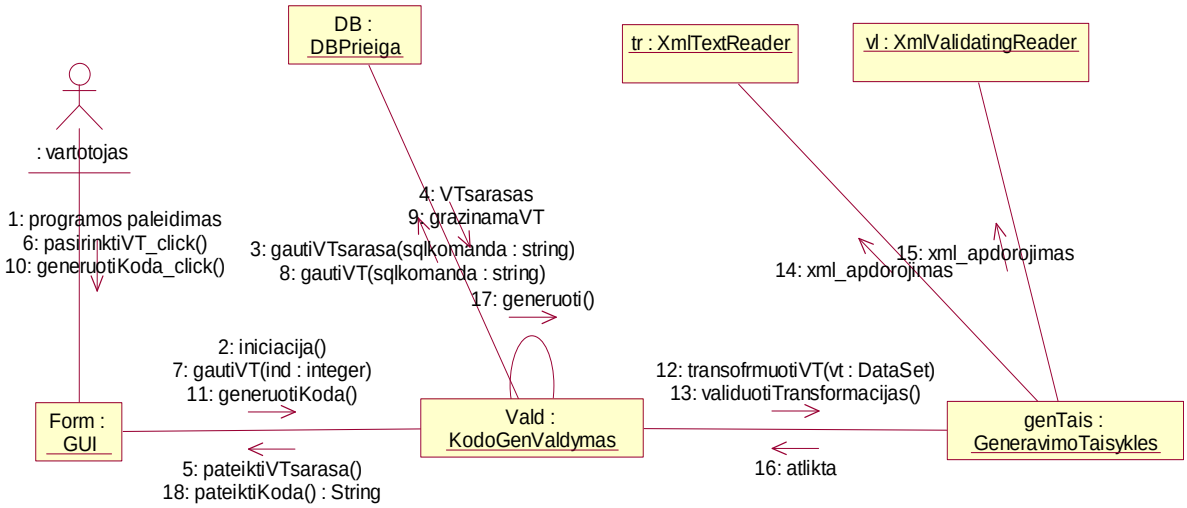
Bendradarbiavimo diagramos. Pateikiamos VT modelio kūrimo bendradarbiavimo (žr. 18 pav.), VT modelio išsaugojimo bendradarbiavimo (žr. 19 pav.) ir kodo generavimo bendradarbiavimo (žr. 20 pav.) diagramos.



18 pav. VT modelio kūrimo bendradarbiavimo diagrama

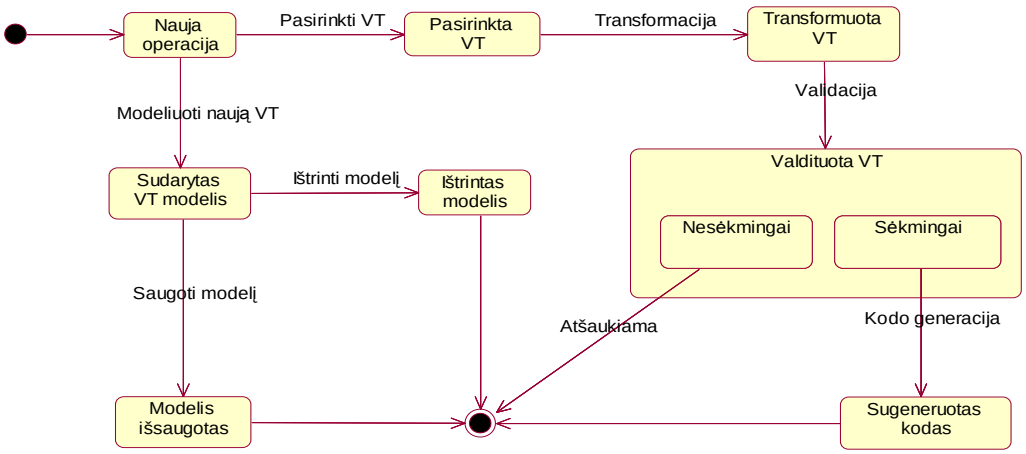


19 pav. VT modelio išsaugojimo bendradarbiavimo diagrama



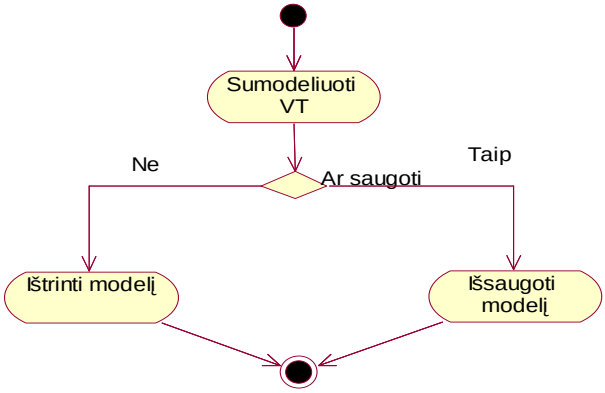
20 pav. Kodo generavimo bendradarbiavimo diagrama

Būsenos diagrama. Žemiau, 21 paveikslėlyje pateikiama apibendrinta kuriamo projekto būsenų diagrama.

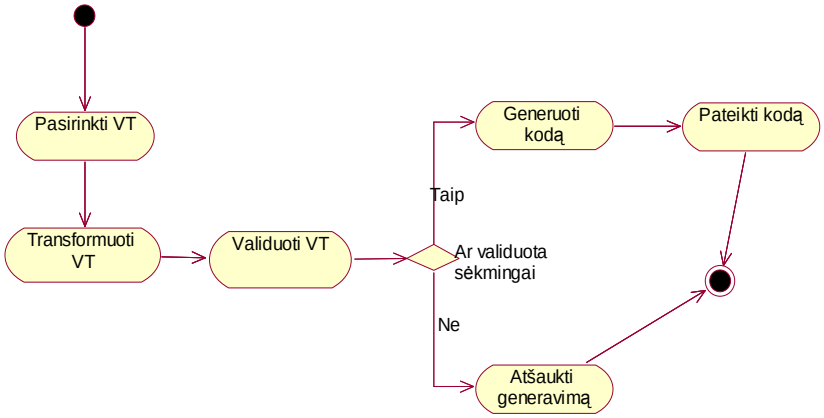


21 pav. Apibendrinta kuriamos programinės įrangos projekto būsenų diagrama

Veiklos diagramos. 22 ir 23 paveikslėliuose pateikiamos projekto veiklos diagramos.



22 pav. VT modeliavimo ir išsaugojimo veiklos diagrama

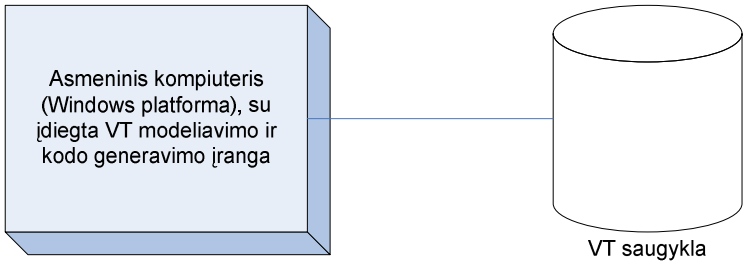


23 pav. Kodo generavimo veiklos diagrama

3.2.4 Sistemos išdėstymo vaizdas

Sistemos išdėstymo vaizdas pateikiamas žemiau (žr. 24 pav.). Iš jo matyti, kad sistemos išdėstymas yra paprastas:

- asmeninis kompiuteris, kuriame įdiegta Windows šeimos operacinė sistema, VT modeliavimo ir kodo generavimo programinė įranga.
- VT saugykla (DB)



24 pav. Išdėstymo vaizdas

3.3 Kodo generavimo iš VT modelių metodikos išvados

1. Pasiūlyta kodo generatoriaus architektūra kodo generavimui iš VT modelių.
2. Aptartas darbo su ERB šablonais principas.
3. Remiantis Roso metodo analizės rezultatais nustatyta, jog tikslinga generuoti SQL kodą.
4. Nustatyta, kad kodo generavimui tikslinga naudoti šablonus.
5. Peržvelgti visi Roso metode naudojami grafiniai elementai, trumpai aptarta jų prasmė ir pasiūlytos metodikos gairės SQL kodo generavimui iš VT modelių.
6. Pateiktas kodo generavimo iš VT modelių įrankio prototipo projektas.

4. SQL kodo generavimo iš grafinių veiklos taisyklių modelių eksperimentas

Šiame skyriuje pateikiama informacija kaip buvo vykdomas SQL kodo generavimo iš grafinių veiklos taisyklių modelių eksperimentas.

4.1 Eksperimento prielaidos ir apribojimai

Eksperimento metu siekiama pritaikyti praeitame skyriuje pateiktos SQL kodo generavimo iš grafinių VT modelių metodikos gaires. Šiam tikslui įgyvendinti priimame keletą išankstinių prielaidų ir apribojimų:

1. Veiklos taisyklių modelių kūrimui turi būti naudojamas KTU IS katedroje sukurtas Microsoft Visio įskiepio prototipas, kuris remiasi VT saugyklos metamodeliu.
2. Esamas VT modeliavimo pagal modifikuotą Roso metodą įrankio prototipas skirtas tik grafinio taisyklių modeliavimo efektyvumui tirti ir yra neparengtas kodo generavimo galimybių tyrimams, todėl jo funkcionalumas plečiamas papildant jį kodo generavimo galimybėmis veiklos taisyklių modeliams.
3. Dėl Roso metode pateikiamos didelės gausos veiklos taisyklių eksperimento metu neįmanoma patikrinti kodo generavimo visoms taisyklėms, todėl apsiribojama tik keletu veiklos taisyklių tipų iš visos klasifikacijos.
4. Sukuriamas kodo generatorius, kurio realizacijai pasitelkiama Ruby programavimo kalba, o šablonai kodo generavimui realizuojami panaudojant ERB šablonų mechanizmą [14].
5. Eksperimento metu įvertinama kodo generavimo metodika, VT modeliavimo įrankio prototipo ir VT saugyklos (metamodelio) tinkamumas kodo generavimui.

4.2 Eksperimento eiga

4.2.1 Prototipo parengimas eksperimentui

Kadangi vienas iš eksperimento tikslų įvertinti VT modeliavimo įrankio prototipo ir VT saugyklos (metamodelio) tinkamumą kodo generavimui, buvo pradėta nuo būtinų funkcijų kodo generavimui realizavimo. Kadangi prototipas (žr. 25 pav.) jau turėjo modeliavimo ir saugojimo funkcijas, tikėtasi, jog užteks tik papildyti prototipą kodo generavimo funkcija, tačiau šioje vietoje jau buvo susidurta su tokiomis problemomis:

1. VT modeliai prototipe išsaugojami nekorektiškai
2. VT modelių užkrovimo metu, neužkraunama dalis informacijos

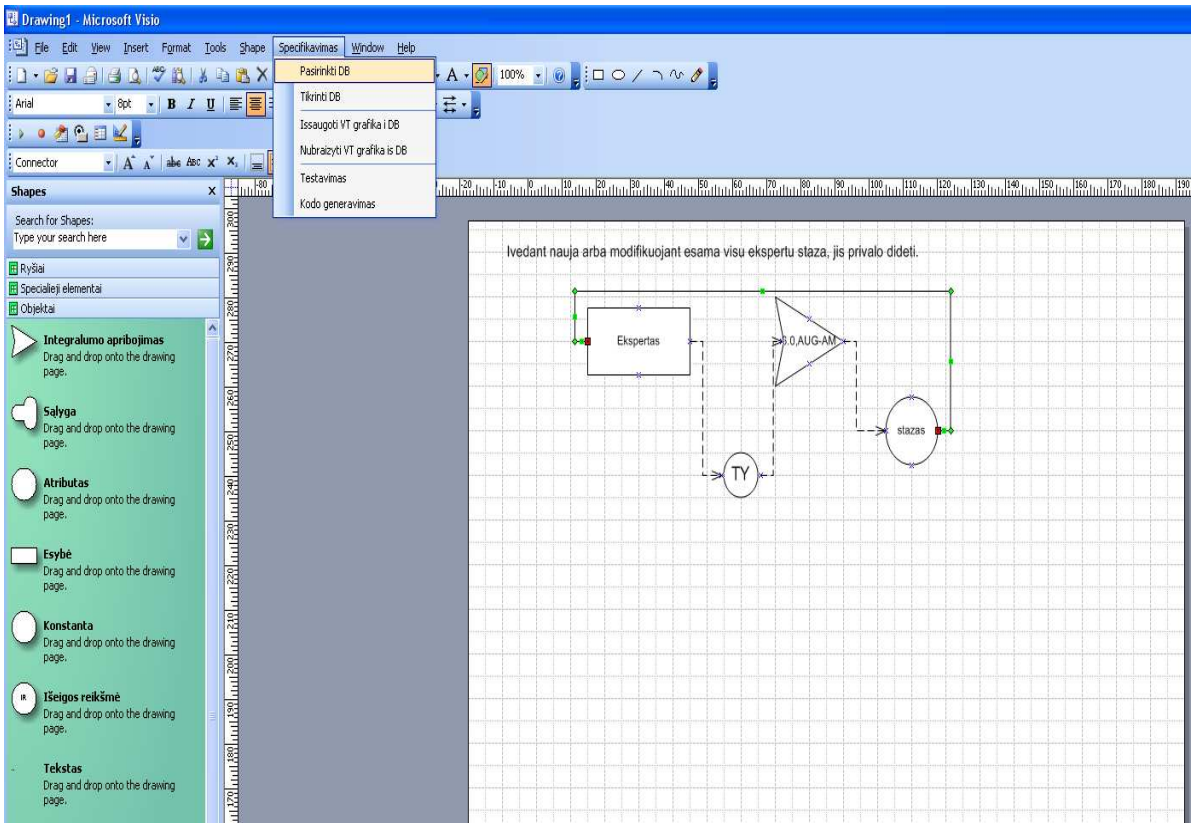
3. VT modeliavimui reikiami duomenys į metamodelio duomenų modelio posistemę privalo būti suvesti iš anksto. Deja, nepateikiamas joks įrankis šiam darbui atlikti, apart duomenų suvedimo rankiniu būdu į VT saugyklos lenteles.

Problemų sprendimas iššaukė tam tikrus prototipo patobulinimus, kurie pateikti 4 lentelėje.

4 lentelė. Prototipo patobulinimai

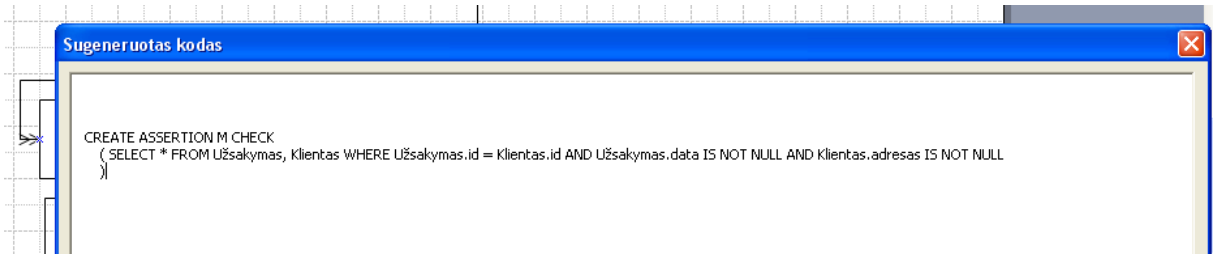
Nr.	***Patobulinimas***
1.	VT modelio išsaugojimo funkcijos klaidų ištaisymas
2.	VT modelio užkrovimo funkcijos ištaisymas
3.	Papildymas kodo generavimo funkcija

Pirmąsias dvi problemas teko spręsti, kadangi be jų negalimas kodo generavimas. Trečios problemos atvejus priimtas sprendimas suvesti duomenų modelio posistemės duomenis rankiniu būdu. Sėkmingai išsprendus problemas, prototipo įrankis papildytas kodo generavimo funkcija (žr. 26 pav.) ir realizuotas kodo generatoriaus architektūra kaip aptarta metodikoje. Kodo generavimo langas matyti 27 pav.



25 pav. Eksperimentinės sistemos prototipo vaizdas

26 pav. Įrankio meniu su įvesta kodo generavimo funkcija



27 pav. Sugeneruoto kodo langas

4.2.2 Dalykinės srities parinkimas

Tolimesnis žingsnis eksperimente buvo veiklos taisyklių parinkimas ir kodo generavimo joms analizė. Taisyklės parenkamos iš įvairių probleminių sričių, tai parodo metodikos nepriklausymą nuo konkrečios srities. Pasirinktos šios veiklos taisyklės:

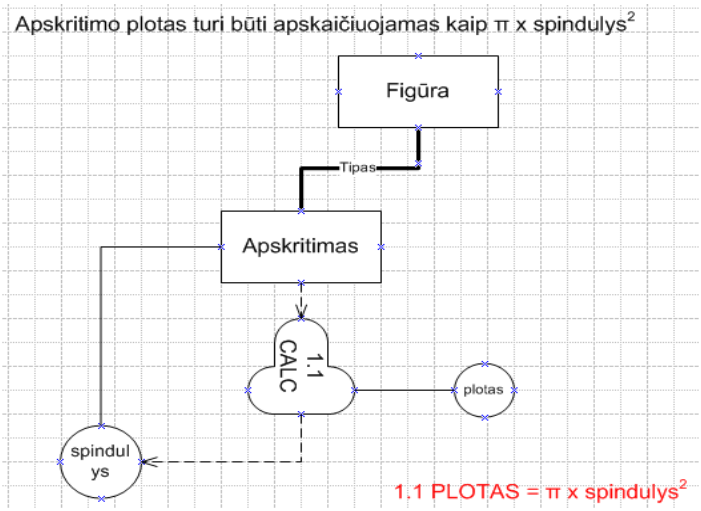
1. Apskaičiavimo
2. Sumavimo
3. Privaloma
4. Mažiau-arba-lygu
5. Lygu
6. Lygu (su apribojimu)
7. Apribota
8. Bendra
9. Bendrai–susiejanti
10. Unikali
11. Unikali (su apribojimu)
12. Privaloma (su apribojimu)

4.2.3 Taisyklių modeliavimas ir kodo generavimas

Žemiau pateikiami VT modelių grafiniai vaizdai ir SQL kodo generavimo rezultatai jiems.

Apskaičiavimo:

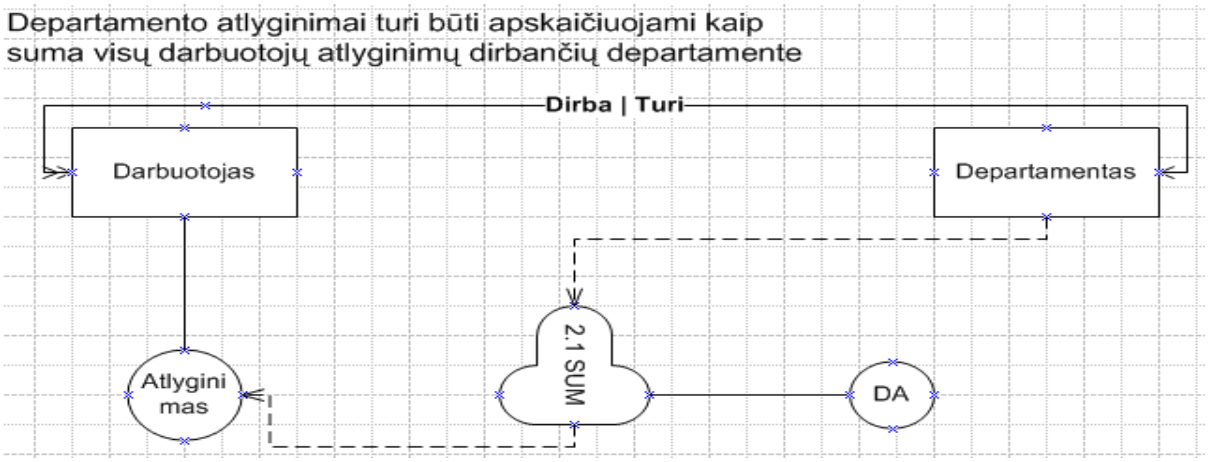
Taisyklėje pateiktas aprašas, kaip apskaičiuoti apskritimo plotą. Šios taisyklės modeliui sugeneruoti SQL kodo nepavyko. 28 paveikslėlyje matomas raudonai pažymėtas tekstas, kuris būtinas taisyklės interpretacijai, tačiau naudojant esamą prototipą nėra išsaugojamas, kas automatiškai reiškia, jog nepatenka į VT saugyklą, tokiu būdu užkertant kelią SQL kodo generavimui. Iš čia seka svarbus pastebėjimas, kad vien grafinio modelio a*pskaičiavimo* taisyklei interpretuoti nepakanka, reikalingas papildomas tekstinis aprašas, aprašantis a*pskaičiavimo* taisyklės interpretavimą.



28 pav. Apskaičiavimo taisyklės modelis

Sumavimo:

Nagrinėjamas sumavimo taisyklės pavyzdys leidžia sugeneruoti *CHECK* tipo apribojimą. Pavyzdyje (žr. 29 pav.) pateiktos dvi esybės (*Darbuotojas ir Departamentas*) susijusios ryšiu *Dirba | Turi*. Viso departamento darbuotojų atlyginimai apskaičiuojami sumuojant kiekvieno iš departamente (tai apibrėžia ryšys tarp esybių) dirbančio darbuotojo atlyginimą. Taisyklė SQL kode realizuojama pasitelkiant SQL *SUM()* funkciją. *WHERE* sakinio dalis priima, kad Departamentas turi savo identifikacinį numerį, kuriam paliekama įvedimo vieta, kad taisyklę pritaikytų konkrečiam departamentui. Bendru atveju toks apribojimas turėtų būti taikomas kiekvienam departamentui.



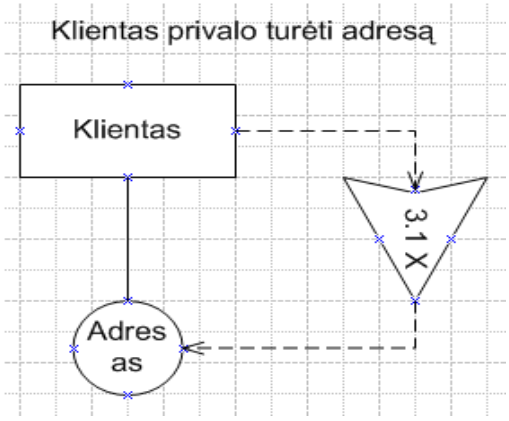
29 pav. Sumavimo taisyklės modelis

Sugeneruotas kodas:

```
CREATE ASSERTION SUM CHECK
      (DA= (SELECT SUM(Darbuotojas.Atlyginimas)
FROM Darbuotojas, Departamentas WHERE Departamentas = '')
      )
```

Privaloma:

Nagrinėjamas *privaloma* taisyklės pavyzdys taip pat leidžia sugeneruoti *CHECK* tipo apribojimą. Pavyzdyje (žr. 30 pav.) pateikta esybė *Klientas* ir jos atributas *Adresas*. Esybė nurodoma kaip taisyklės *privaloma* bazė, o esybės atributas kaip taisyklės *privaloma* korespondentas. Taisyklė *privaloma* pareikalauja, kad atributo reikšmė būtų užpildyta. Iš sugeneruoto SQL kodo, matyti, kad esybė neatsispindi sugeneruotame kode, tačiau sugeneruotą kodą būtų galima taikyti paskelbiant esybės *Klientas* lentelės sukūrimą ir nustatant atributui *Adresas* sugeneruotą apribojimą.



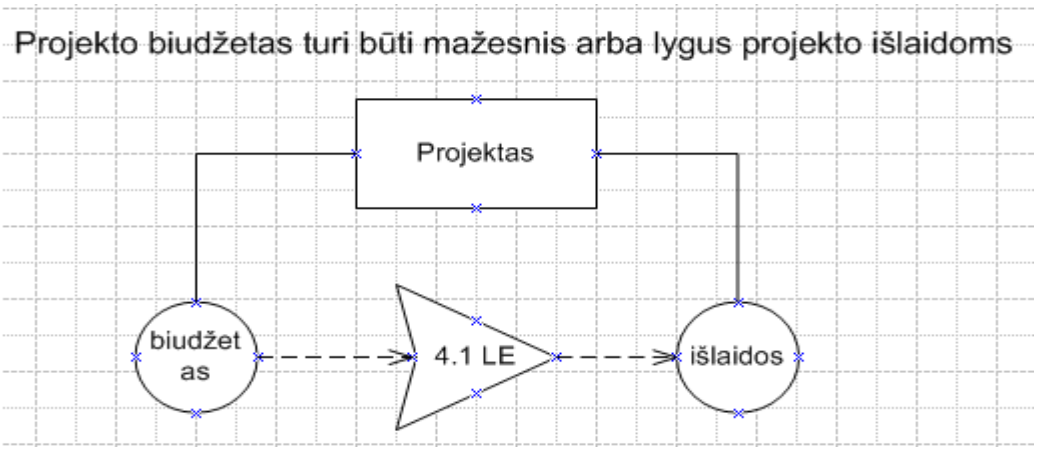
30 pav. Privaloma taisyklės modelis

Sugeneruotas kodas:

```
CHECK (Adresas IS NOT NULL)
```

Mažiau-arba-lygu:

Nagrinėjamas mažiau-arba-lygu taisyklės pavyzdys pateiktas 31 pav. sako: *projekto biudžetas turi būti mažesnis arba lygus projekto išlaidoms*. Pateiktoje taisyklės grafinėje išraiškoje matyti esybė *Projektas*, bei du jos atributai: *biudžetas* (taisyklės bazė) ir *išlaidos* (taisyklės korespondentas). Taisyklės tipas *mažiau-arba-lygu* žymimas LE (angl. less-than-or-equal) išreiškia reikalavimą keliamą esybės atributams. Šios taisyklės rezultatas *CHECK* tipo apribojimas.



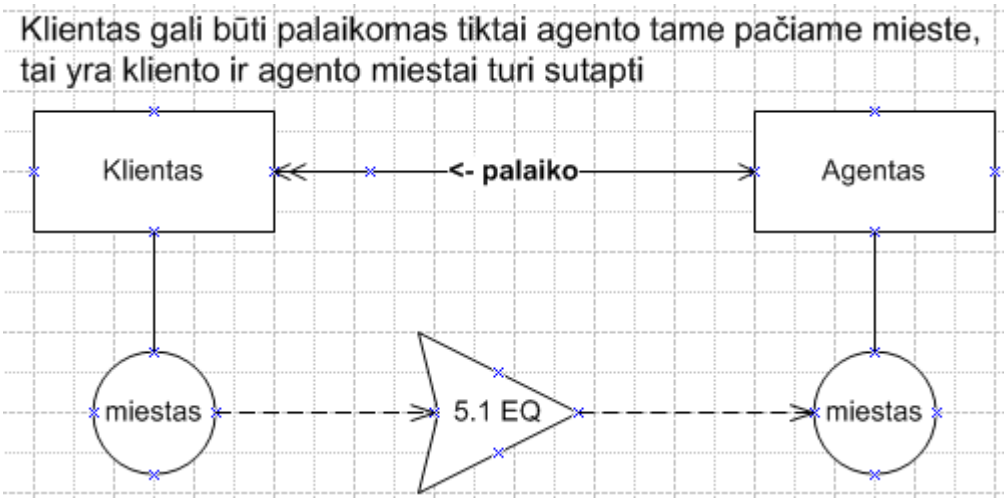
31 pav. Mažiau-arba-lygu taisyklės modelis

Sugeneruotas kodas:

```
CHECK (biudžetas <= išlaidos)
```

Lygu (angl. equal–to) taisyklė:

Nagrinėjamas *lygu* taisyklės pavyzdys pateiktas 32 pav. teigia: *klientas gali būti palaikomas tiktai agento tame pačiame mieste*. Pateiktoje taisyklės grafinėje išraiškoje matyti dvi esybės: *Klientas* ir *Agentas*, bei jų atributai *miestas*. Esybės *Klientas* atributas *miestas* yra taisyklės bazė, o esybės *Agentas* atributas *miestas* yra taisyklės korespondentas. Taisyklės tipas *lygu* (EQ) nurodo reikalavimą lygybės tarp atributų. Ryšys vienas-su-daug tarp esybių nurodo reikalavimą, kad atributai *miestas* sutaptų tiktai agentams susijusiems su klientais šiuo ryšiu. Šios taisyklės SQL kodo generavimo rezultatas yra *ASSERTION* tipo apribojimas, generavimo rezultatas pateikiamas žemiau.



32 pav. Lygu taisyklės modelis

Sugeneruotas kodas:

```
CREATE ASSERTION EQ CHECK
    ( SELECT * FROM Agentas, Klientas WHERE Agentas.id = Klientas.id
AND Agentas.miestas = Klientas.miestas
    )
```

Sugeneruotame kode naudojama *ASSERTION* konstrukcija, kadangi taisyklėje esybės susietos ryšiu *palaiko, SELECT* sakinyje atsiranda *WHERE* dalis, kurioje pagal ryšio reikšmę sulyginami esybių *Agentas* ir *Klientas* raktiniai id atributai (pagal nutylėjimą) ir taisyklės *lygu* rezultatas esybių *Agentas* ir *Klientas* atributų *miestas* sulyginimas (*Agentas.miestas = Klientas.miestas*), šios dvi sąlygos apjungiamos *AND* operatoriumi. Visgi sugeneruotam SQL kodui trūksta funkcionalumo. Norint, kad SQL kodas išreikštų taisyklės skelbiamą apribojimą sugeneruotas kodas turėtų atrodyti šitaip:

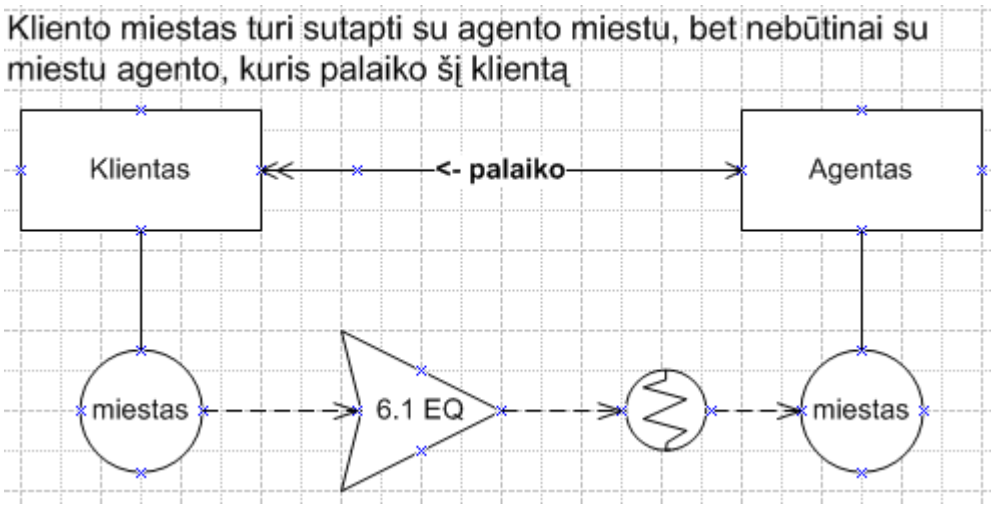
```
CREATE ASSERTION EQ CHECK
```

```
    ( NOT EXISTS (SELECT * FROM Agentas, Klientas WHERE Agentas.id =
Klientas.id AND Agentas.miestas <> Klientas.miestas)
    )
```

Taigi, turėtų atsirasti *NOT EXISTS* ir = lygybės ženklas pasikeisti į <> (nelygu) iš to seka išvada, kad pilnaverčiam taisyklės išreiškimui SQL kode kartais reikalingi tam tikri papildomi SQL elementai, kurių nenurodome VT modelyje. Šiuo atveju tai *NOT EXISTS* ir <> (nelygu). Priešingu atveju kodo generavimas yra tik dalinis, t.y. teisingai generuojami tik tam tikri SQL kodo fragmentai išreikšti taisyklėje.

Lygu (angl. equal–to) taisyklė su apribojimu:

Nagrinėjama aukščiau pateiktos taisyklės *lygu* modifikacija. Modifikuoto modelio vaizdas pateiktas 33 pav., kuriame matyti, kad VT taisyklės modelyje korespondento ryšyje panaudotas apribojimas (angl. qualifier) *bet kuris sąryšis* (angl. any).



33 pav. Taisyklės lygu su apribojimu modelis

Šio apribojimo prasmė yra nutraukimas bet kokio ryšio tarp bazės ir korespondento (šiuo atveju nutraukiamas ryšys vienas-su-daug *palaiko* tarp esybių *Klientas* ir *Agentas*, kurioms priklauso bazė ir korespondentas).

Sugeneruotas kodas:

```
CREATE ASSERTION EQ CHECK
    ( SELECT * FROM Agentas, Klientas WHERE Agentas.miestas =
Klientas.miestas
    )
```

Gautas rezultatas nuo ankstesnio skiriasi tuo, kad trumpesnė *WHERE* dalis, tai apribojimo *bet kuris sąryšis* įtakotas rezultatas. Šiuo atveju vėlgi gaunamas tik dalinis SQL kodas, kadangi pilnai funkcionalus kodas vėlgi neturi *NOT EXISTS* ir <> (nelygu). Pilna VT modelio išraiška SQL kode turėtų būti:

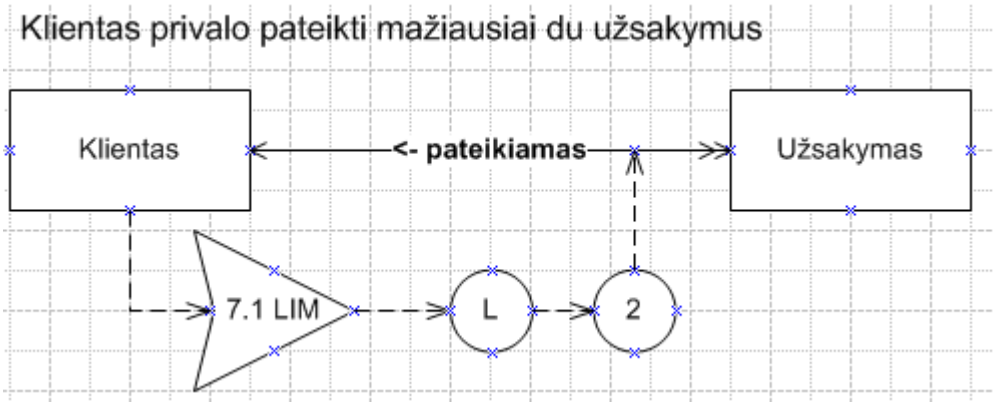
```
CREATE ASSERTION EQ CHECK
    ( NOT EXISTS (SELECT * FROM Agentas, Klientas WHERE
Agentas.miestas <> Klientas.miestas)
    )
```

Apribota (angl. limited):

34 pav. pateiktas taisyklės *apribota* modelis. Modelis išreiškia taisyklę: *klientas privalo pateikti mažiausiai du užsakymus*. Taisyklės bazė yra esybė *Klientas*, o korespondentas yra ryšys (<- *pateikiamas*). Taisyklės bazė šiuo ryšiu susijusi su kita esybe *Užsakymas*, VT patikrinimas išreiškiamas taisyklės *apribota LIM* simboliu ir jai priklausančiais numeratoriais *L* ir *2*.

Sugeneruotas kodas:

```
CREATE ASSERTION LIM CHECK
    ( SELECT * FROM Klientas, Užsakymas WHERE Klientas.id =
Užsakymas.id GROUP BY Klientas.id HAVING COUNT (*) >= 2
    )
```



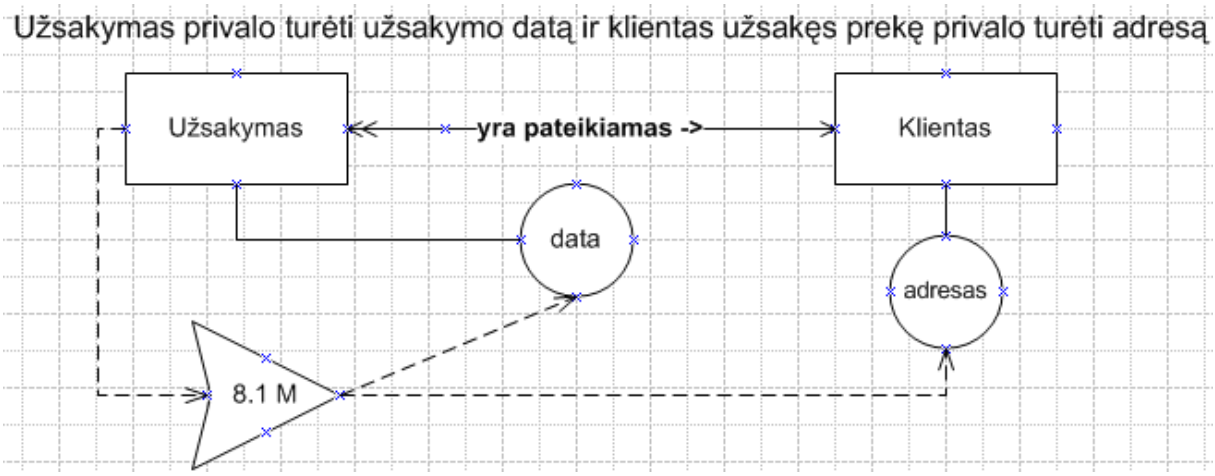
34 pav. Taisyklės apribota modelis

Gautas apribojimas yra teisingas, tačiau jis tik atrenka užklausą atitinkančius duomenis, bet neužtraukia duomenų pakeitimų, kurie sąlygotų pateikto apribojimo pažeidimą. Norint, kad SQL kodas išreikštų taisyklės skelbiamą apribojimą sugeneruotas kodas turėtų atrodyti šitaip:

```
CREATE ASSERTION LIM CHECK
    ( NOT EXISTS (SELECT * FROM Klientas, Užsakymas WHERE
Klientas.id = Užsakymas.id GROUP BY Klientas.id HAVING COUNT (*) <
2)
    )
```

Reikalinga papildyti kodą *NOT EXISTS* ir sąlygą >= (daugiau arba lygu) pakeisti sąlyga < (mažiau).

Bendra (angl. mutual):



35 pav. Taisyklės bendra modelis

35 pav. pateikta taisyklės *bendra* modelis. Modelis išreiškia taisyklę: *užsakymas privalo turėti užsakymo datą ir klientas užsakęs prekę privalo turėti adresą*. Taisyklės bazė yra esybė *Užsakymas*, o korespondentai yra du: esybės *Užsakymas* atributas *data* ir esybės *Klientas* atributas *adresas*. Taisyklėje esybės *Užsakymas* ir *Klientas* tarpusavyje yra susiję ryšiu vienas-su-daug (*yra pateikiamas->*). Šiuo atveju sugeneruotinas SQL *ASSERTION* apribojimo kodas:

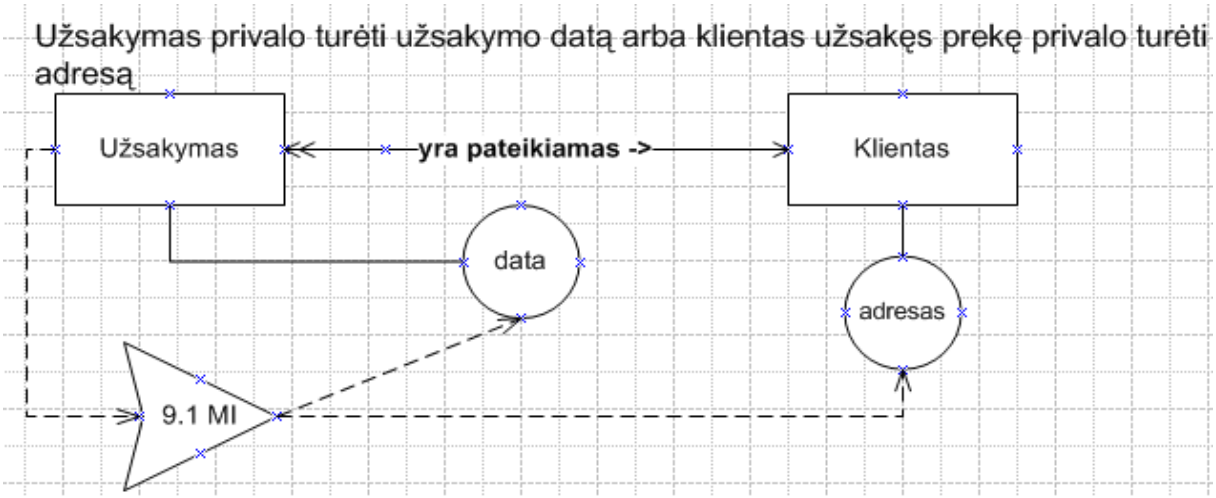
```
CREATE ASSERTION M CHECK
    ( SELECT * FROM Užsakymas, Klientas WHERE Užsakymas.id =
Klientas.id AND Užsakymas.data IS NOT NULL AND Klientas.adresas IS
NOT NULL
    )
```

Esminė taisyklės *bendra* generavimo metodika pasitvirtina *IS NOT NULL* ir sąlygų apjungime panaudojant *AND*. Visgi, sugeneruotas kodas yra dalinis, kadangi toks *ASSERTION* panaudojimas netenkintų taisyklėje išreikšto apribojimo. Tam reikėtų kodą papildyti *NOT EXISTS*, bei panaikinti neiginius *NOT*. Galutinis variantas turėtų atrodyti šitaip:

```
CREATE ASSERTION M CHECK
    ( NOT EXISTS (SELECT * FROM Užsakymas, Klientas WHERE
Užsakymas.id = Klientas.id AND Užsakymas.data IS NULL AND
Klientas.adresas IS NULL)
    )
```

Bendrai-susiejanti (angl. mutually-inclusive):

36 pav. pavaizduota taisyklės *bendra* modifikacija pritaikant taisyklę *bendrai-susiejanti*. Pasikeičia tik tekstinis aprašas ir taisyklės simbolis. Šiuo atveju taisyklė reiškia: *užsakymas privalo turėti užsakymo datą arba klientas užsakęs prekę privalo turėti adresą*.



36 pav. Taisyklės bendrai-susiejanti modelis

Šiuo atveju sugeneruotas SQL *ASSERTION* apribojimo kodas pagal apibrėžtą metodiką yra toks:

```
CREATE ASSERTION MI CHECK
    ( SELECT * FROM Užsakymas, Klientas WHERE Užsakymas.id =
Klientas.id AND Užsakymas.data IS NOT NULL OR Klientas.adresas IS
NOT NULL
    )
```

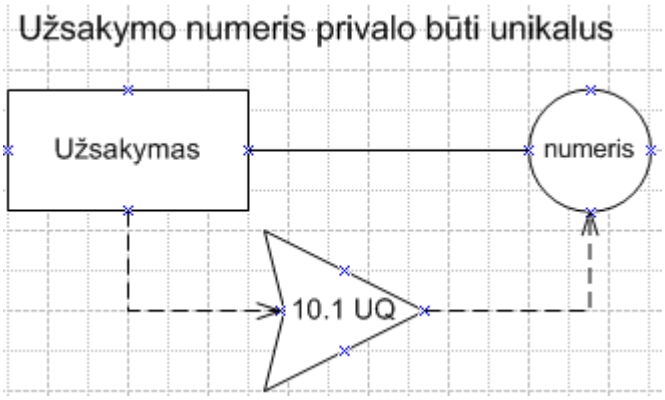
Kaip ir ankstesniame pavyzdyje gaunamas SQL kodas yra dalinai išreiškiantis taisyklę. Toks *ASSERTION* panaudojimas netenkintų taisyklėje išreikšto apribojimo. Tam reikėtų kodą papildyti *NOT EXISTS*, bei panaikinti neiginius *NOT*, bei *OR* pakeisti *AND*. Galutinis variantas turėtų atrodyti šitaip:

```
CREATE ASSERTION MI CHECK
    ( NOT EXISTS (SELECT * FROM Užsakymas, Klientas WHERE
Užsakymas.id = Klientas.id AND Užsakymas.data IS NULL AND
Klientas.adresas IS NULL)
    )
```

Unikali:

Nagrinėjamas taisyklės *unikali* pavyzdys pateiktas 37 pav. teigia: *užsakymo numeris privalo būti unikalus*. Taisyklės bazė yra esybė *Užsakymas*, o jos atributas *numeris* – taisyklės korespondentas. Apribojimas išreikštas taisyklės simboliu *UQ*. Sugeneruojamas SQL apribojimas:

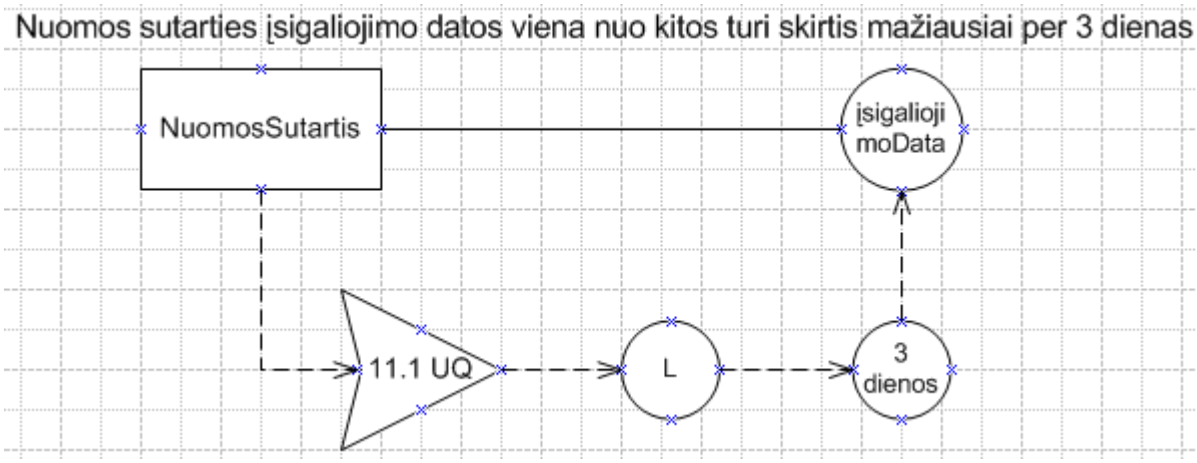
```
CONSTRAINT UQ UNIQUE (numeris)
```



37 pav. Taisyklės unikali modelis

Unikali (su apribojimu):

Taisyklės *unikali (su apribojimu)* modelio vaizdas pateiktas 38 pav. Taisyklė teigia: *nuomos sutarties įsigaliojimo datos viena nuo kitos turi skirtis mažiausiai per 3 dienas*. Taisyklės bazė yra esybė *NuomosSutartis*, o jos atributas *įsigaliojimoData* – taisyklės korespondentas. Apribojimas išreikštas taisyklės simboliu *UQ* ir apribojimo elementais *L* ir *3 dienos*.



38 pav. Taisyklės unikali su apribojimu modelis

Sugeneruojamas SQL kodas:

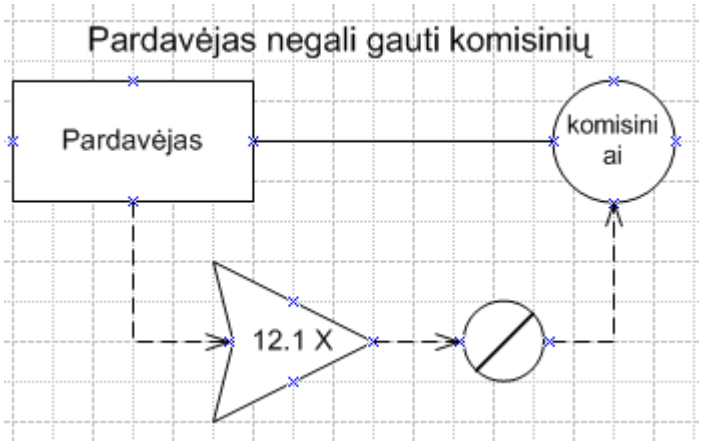
```
CONSTRAINT UQ UNIQUE (įsigaliojimoData)
```

SQL kode atsižvelgiama į tai, kad atributo *įsigaliojimoData* reikšmės turi būti unikalios, tačiau likusi dalis taisyklės neinterpretuojama.

Privaloma (su apribojimu):

Taisyklės *privaloma (su apribojimu)* modelio vaizdas pateiktas 39 pav. Taisyklė teigia: *pardavėjas negali gauti komisinių*. Taisyklės bazė yra esybė *Pardavėjas*, o jos atributas *komisiniai* – taisyklės korespondentas. Taisyklės tipas nurodo, kad atributas yra būtinas, o apribojimas *be reikšmės* (angl. unvalued) šią taisyklės reikšmę verčia priešinga ir reikalauja,

kad atributas neturėtų reikšmės. Bendru atveju generuotinas *IS NOT NULL* apribojimas yra dar kartą paneigiamas ir gauna priešingą reikšmę: *IS NULL* Generuojamas SQL *CHECK* apribojimas.



39 pav. Taisyklės privaloma su apribojimu modelis

Gautas SQL kodas:

```
CHECK (komisiniai IS NULL)
```

4.3 Rezultatai

SQL kodo generavimo iš grafinių VT modelių eksperimento rezultatai pateikiami 5 ir 6 lentelėse. Iš 5 lentelės matyti, kad buvo tiriama 12 veiklos taisyklių modelių ir didžiajai daliai taisyklių pavyko sugeneruoti SQL kodą.

5 lentelė. Eksperimento apibendrinimas

Viso tirta taisyklių	***Kokiam kiekiui taisyklių sugeneruotas kodas***	***Taisyklių, kurioms sugeneruotas kodas, dalis procentais, %***
12	11	91,6

Žemiau pateikiamoje 6 lentelėje atskleidžiama detalesnė informacija apie kiekvieno VT modelio generavimo rezultatus ir rezultatų procentinė išraiška visoje eksperimento imtyje.
Iš pastarosios 6 lentelės matyti, kad daugiau kaip 58% taisyklių sugeneruojamas tik dalinai funkcionalus SQL kodas. Iki pilnai funkcionalaus SQL kodo sugeneruotą kodą tenka papildyti *NOT EXISTS*, bei tam tikrus operatorius pakeisti jiems priešingais (AND į OR, = į <> ir t.t.). 33,33% procentams modelių sugeneruojamo SQL kodo papildyti nereikia, nes jis yra pilnai funkcionalus.

6 lentelė. Kodo generavimo rezultatai kiekvienam VT modeliui

Taisyklės modelio numeris	***Kodo generavimo rezultatas***	***Kokią dalį procentais sudaro eksperimento imtyje gautas kodo generavimo rezultatas, %***
1	0	

2	1	Kodo generavimo rezultatas 0 gautas 8,33% taisyklių. Kodo generavimo rezultatas 1 gautas 58,33% taisyklių. Kodo generavimo rezultatas 2 gautas 33,33% taisyklių.
3	2	
4	2	
5	1	
6	1	
7	1	
8	1	
9	1	
10	2	
11	1	
12	2	

Paaiškinimas. „Kodo generavimo rezultatas“ reikšmės:
0 – nepavyko sugeneruoti SQL kodo
1 – sugeneruotas dalinai funkcionalus SQL kodas
2 – sugeneruotas pilnai funkcionalus SQL kodas

4.4 SQL kodo generavimo eksperimento išvados

1. Eksperimento metu nustatyta, kad VT modeliavimo įrankio modelių išsaugojimo ir užkrovimo funkcijos veikia nekorektiškai. VT modelių išsaugojimo ir užkrovimo problemas išsprendus prototipas papildytas kodo generavimo funkcijomis.
2. Nustatyta, kad VT saugyklos metamodelis neleidžia pilnavertiškai išsaugoti a*pskaičiavimo* veiklos taisyklės tipo modelių, kas užkerta kelią SQL kodo generavimui. Iš to seka išvada, kad tam tikrais atvejais būtų galima pagerinti kodo generavimo galimybes, jeigu *apskaičiavimo* VT modeliai būtų išsaugomi pilnavertiškai.
3. Ne visi Roso notacijos elementai gali būti atvaizduoti SQL kode, tačiau įtakoja VT prasmę.
4. Dažniausiai VT modeliams sugeneruojamas dalinai funkcionalus SQL kodas. Tokį kodą galima nesunkiai papildyti iki pilnai funkcionalaus.
5. Nustatyta, jog Roso VT diagramos gali būti naudojamos SQL apribojimams generuoti. Programinio kodo generavimo iš grafinių veiklos taisyklių modelių galimybės priklauso nuo veiklos taisyklės tipo.

5. Išvados

1. Dalykinės srities analizės metu apžvelgta veiklos taisyklių koncepcija, šios srities standartai (SBVR, PRR). Iš klasifikavimo bei struktūrizavimo metodų išskirtas modifikuotas Roso metodas kaip vienintelis metodas pasiūlantis grafinę notaciją veiklos taisyklių modeliavimui.
2. Išnagrinėti kodo generatorių pavyzdžiai įgalina pritaikyti metodikas, kuriant kodo generatorių darbui su grafiniais veiklos taisyklių modeliais, tačiau pagrindine problema išlieka tokių metodikų nebuvimas Roso metodui.
3. Kodo generavimo problemai spręsti, išanalizavus modifikuotą Roso metodą, buvo pasiūlyta SQL kodo generavimo metodika iš grafinių veiklos taisyklių modelių.
4. Ištyrus galimus kodo generatoriaus architektūros variantus parodyta, kad SQL apribojimų kodo generavimui iš grafinių veiklos taisyklių yra tikslinga taikyti kodo generavimo šablonus.
5. Realizavus kodo generatorių, atliktas SQL kodo generavimo grafiniams VT modeliams eksperimentas praktiškai pritaikant dalį pasiūlytos kodo generavimo metodikos. Eksperimentas parodė, kad daugiau nei 90% atvejų gautas pilnai arba dalinai (su nedideliais trūkumais) taisyklę išreiškiantis kodas. Iš to galima daryti išvadą, kad pasiūlytą kodo generavimo metodiką galima taikyti SQL kodo generavimui iš grafinių veiklos taisyklių modelių.

6. Literatūra

[1] **Wikipedia.** Business rules. Iš interneto puslapio [interaktyvus] [žiūrėta 2008 11 07], prieiga per internetą: http://en.wikipedia.org/wiki/Business_rules

[2] **Business rules.** Iš interneto puslapio [interaktyvus] [žiūrėta 2008 11 07], prieiga per internetą: http://www.agilemodeling.com/artifacts/businessRule.htm

[3] **Wikipedia.** Verslo taisyklė. Iš interneto puslapio [interaktyvus] [žiūrėta 2008 11 07], prieiga per internetą: http://lt.wikipedia.org/wiki/Verslo_taisykl%C4%97

[4] **Ross, R. G**. The Business Rule Book: Classifying, Defining an Modeling Rules. *Business Rule Solutions*. 1997.

[5] **Baškevičius, S., Kapočius, K.** Veiklos taisyklių struktūrizavimo informacijos sistemų projektavimo metu įrankio realizacija. *Informacinės technologijos – 2006. Tarpuniversitetinė magistrantų ir doktorantų konferencija. Pranešimų medžiaga. Vilnius, VU leidykla*, 2 dalis, p. 91-96.

[6] **Butleris, R., Kapočius, K.** Struktūrizuotų veiklos taisyklių saugyklos architektūra. *Informacijos mokslai, Nr. 17, Vilnius, Vilniaus universiteto leidykla*. 2001, pp. 46-57.

[7] **Object Management Group**. Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 OMG Available Specification. Iš *Object Management Group internetinio puslapio* [interaktyvus], [žiūrėta 2008-01-18], prieiga per internetą: http://www.omg.org/docs/formal/08-01-02.pdf.

[8] **Object Management Group**. Production Rule Representation (PRR), Beta 1. Iš *Object Management Group internetinio puslapio* [interaktyvus], [žiūrėta 2008-06-15], prieiga per internetą: http://www.omg.org/docs/dtc/07-11-04.pdf.

[9] **Butleris, R., Kapočius, K.** The Business Rules Repository for Information Systems Design. *The 6th East-European Conference ADBIS'2002. Konferencijos pranešimų medžiaga. Bratislava, Slovakia, Vydavatel'stvo STU*, Vol. 2. 2002, pp.64-77.

[10] **Wikipedia.** Code generation (compiler). Interneto puslapis [interaktyvus] [žiūrėta 2007 10 10], prieiga per internetą: http://en.wikipedia.org/wiki/Code_generation_%28compiler%29

[11] **Object Management Group**. MDA Specifications. Iš *Object Management Group internetinio puslapio* [interaktyvus], [žiūrėta 2008-06-15], prieiga per internetą: http://www.omg.org/mda/specs.htm.

[12] **Herrington, J.** Code generation in action. *Manning Publications Co*. 2003, pp. 37-49

[13] **Y. Matz.** Ruby programavimo kalbos interneto puslapis, [interaktyvus]. [žiūrėta 2009-03-20], prieiga per internetą: http://www.ruby-lang.org/en/about/.

[14] **Britt J.** ERB - Ruby Templating, [interaktyvus]. [žiūrėta 2009-03-20], prieiga per internetą: http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html.

[15] **Hay D. et al.** (2001) Defining Business Rules - What Are They Really? GUIDE projektas, revizija 1.3, [interaktyvus], [žiūrėta 2008.06.26]. Prieiga per internetą: http://www.businessrulesgroup.org/first_paper/br01c0.htm

[16] **V. Štuikys, V., G. Ziberkas, R. Damaševičius (2000)**. Open PROMOL – the Program Modification Language (Reference Manual). Iš *Kauno technologijos universiteto internetinio puslapio* [interaktyvus], [žiūrėta 2009-03-09], prieiga per internetą: http://soften.ktu.lt/~stuik/group/promol/docs/PROMOL-manual.pdf.

[17] **Kapočius K.** Veiklos taisyklių struktūrizavimo modeliai ir jų taikymas kuriant informacijos sistemas: daktaro disertacija. KTU informatikos fakultetas. [Kaunas], 2006, p. 198.

7. Terminų ir santrumpų žodynėlis

VT – veiklos taisyklė.

IS – informacinė sistema.

CASE – (angl. Computer-Aided Software Engineering) – programinės įrangos projektavimas kompiuterio pagalba.

SBVR (angl. Semantics of Business Vocabulary and Business Rules) – OMG (angl. Object Management Group) priimtas standartas, kurį sukūrė Business Rule Team konsorciumas.

OMG (angl. Object Management Group) – konsorciumas, kurio pagrindinis tikslas standartų rengimas ojektiškai orientuotų paskirstytųjų sistemų.

XMI (angl. XML Metadata Interchange) – XML metaduomenų tarpusavio apsikeitimas

API (angl. application programming interface) – programų kūrimo sąsaja.

MDA (angl. Model-Driven Architecture) – modeliu paremta architektūra.

PRR (angl. Production Rule Representation) – standartas pasiūlytas OMG skirtas patenkinti poreikį turėti bendrą produkcinių taisyklių atvaizdavimą, naudojamą įvairių gamintojų taisyklių VT varikliuose.

C++, Java, PHP, C#, Ruby, Perl, Python – progamavimo kalbos.

UML (angl. Unified Modelling Language) – vieninga modeliavimo kalba.

PIM (angl. Platform Independent Model) – nuo platformos nepriklausantis modelis.

PSM (angl. Platform Specific Model) – nuo platformos priklausantis modelis.

J2EE, .NET – karkasai programinės įrangos kūrimui.

MOF (angl. Meta-Object Facility) – standartas parengtas OMG konsorciumo.

OCL (angl. Object Constraint Language) – objektų apribojimo kalba.

CWM (angl. Common Warehouse Metamodel) – modeliavimo specifikacija reliaciniams, nereliaciniams, daugiamačiams objektams.

SQL (angl. Structured Query Language) – struktūrizuota užklausų kalba.

DB – duomenų bazė.

XML (angl. Extensible Markup Lanuage) – išplėstinė žymų kalba.

Ruby – Interpretuojama programavimo kalba.

ERB – eRuby šablonų mechanizmo realizacija.

Open PROMOL (angl. Program Modification Language) – metakalba bendriniams komponentams kurti.

Priedai

1 Priedas. Nedalomų taisyklių tipų savybės pagal Roso metodą

Parengta pagal [Ross, 1997].

TAISYKLIŲ TIPŲ GRUPĖ / TIPAS-SANTRUMPA	REIKŠMĖ	REIKŠMĖ, JEIGU VYKDOMA	REIKŠMĖ, JEIGU TIESIOG TESTUOJAMA
I. Egzempliorių patvirtinimo *Išskirtinė savybė*: tik šio tipo taisyklės gali skaičiuoti egzempliorius. *Išeigos vertės tipas*: egzempliorių kiekis/*EK*.			
Privaloma – P	turi tam tikros savybes egzempliorių	privalo turėti	ar turi?
Apribota – A	turi nurodytą tam tikros savybės egzempliorių kiekį	egzempliorių kiekis privalo neviršyti nurodyto limito	ar egzempliorių kiekis neviršija nurodyto limito?
II. Tipo patvirtinimo *Išskirtinė savybė*: tik šio tipo taisyklės gali veikti kaip loginiai operatoriai IR arba ARBA. *Išeigos vertės tipas*: tipų kiekis/*TK*.			
Bendra – B	turi nurodytą kiekį tipų	tipų kiekis privalo neviršyti nurodyto limito	ar tipų kiekis neviršija nurodyto limito?
Bendrai-atskirianti – BA	neturi dviejų ar daugiau tipų vienu metu	negali turėti daugiau negu vieną iš kelių skirtingų tipų vienu metu	ar neturi daugiau negu vieną skirtingą tipą vienu metu?
Bendrai-susiejanti – BS	vienu metu turi bent vieną iš dviejų ar daugiau tipų	privalo turėti bent vieną iš dviejų ar daugiau tipų vienu metu	ar vienu metu turi bent vieną iš dviejų ar daugiau tipų?
Bendrai-uždraudžianti –BU	vienu metu neturi visų iš dviejų ar daugiau tipų	negali turėti visų iš dviejų ar daugiau tipų vienu metu	ar neturi visų iš dviejų ar daugiau tipų vienu metu?
III. Pozicijos patvirtinimo *Išskirtinė savybė*: tik šio tipo taisyklės gali formuoti išvestinius faktus. *Išeigos vertės tipas*: taisyklėms POZ, ZEM, AUK – išvestas faktas/*IF*; CHRO, SEN, NAU – data-laikas/*DL*.			
Pozicinė – POZ	konkreti vertė (arba vertės), apibrėžianti poziciją uždaroje sekoje	privalo būti nurodytoje pozicijoje	ar yra nurodytoje pozicijoje?

TAISYKLIŲ TIPŲ GRUPĖ / TIPAS-SANTRUMPA	REIKŠMĖ	REIKŠMĖ, JEIGU VYKDOMA	REIKŠMĖ, JEIGU TIESIOG TESTUOJAMA
Žemiausia – ZEM	pirmoji arba mažiausia uždaroje reikšmių sekoje	turi būti žemiausia	ar žemiausia?
Aukščiausia – AUK	paskutinė arba didžiausia uždaroje reikšmių sekoje	privalo būti aukščiausia	ar aukščiausia?
Chronologinė – CHRO	konkreti amžiaus vertė uždaroje sekoje	privalo būti nurodyto amžiaus pozicijoje	ar yra nurodyto amžiaus pozicijoje?
Seniausia – SEN	anksčiausia (seniausia) uždaroje sekoje	privalo būti anksčiausia	ar anksčiausia?
Naujausia – NAU	pati naujausia uždaroje sekoje	privalo būti naujausia	ar pati naujausia?
IV. Funkcinio patvirtinimo *Išskirtinė savybė*: tik šio tipo taisyklės gali tikrinti duotojo duomenų tipo reikšmes kito duomenų tipo reikšmių sekoje (t.y., kaip funkcijos). *Išeigos vertės tipas*: egzemplioriaus numeris/*EN*			
Funkcinė – FNKC	visi tipo egzemplioriai tenkina vartotojo pateiktą funkciją	privalo tenkinti vartotojo pateiktą funkciją	ar tenkina vartotojo pateiktą funkciją?
Unikali – UNIK	bet kurie du tipo egzemplioriai yra skirtingi	privalo būti unikalūs	ar unikalūs?
Svyruojanti – SV	visi tipo egzemplioriai skiriasi nuo gretimų egzempliorių	privalo skirtis	ar skiriasi?
Auganti – AUG	visi tipo egzemplioriai išdėstyti didėjimo tvarka	turi didėti	ar didėja?
Krentanti – KRE	visi tipo egzemplioriai išdėstyti mažėjimo tvarka	turi mažėti	ar mažėja?
Neatnaujinama – NAT	visi panašūs tipo egzemplioriai seka vienas po kito	turi būti nuosekli	ar nuosekli?
V. Palyginamojo įvertinimo *Išskirtinė savybė*: jokio kito tipo taisyklės negali pritaikyti tiesioginio reikšmių sulyginimo testų. *Išeigos vertės tipas*: palyginimo vertė/*PV*			
Lygu – LG	reikšmės lygios	privalo būti tokia pati, kaip	ar tokia pati?
Nelygu – NLG	reikšmės nelygios	privalo būti ne tokia pati, kaip	ar ne tokia pati?

TAISYKLIŲ TIPŲ GRUPĖ / TIPAS-SANTRUMPA	REIKŠMĖ	REIKŠMĖ, JEIGU VYKDOMA	REIKŠMĖ, JEIGU TIESIOG TESTUOJAMA
Daugiau-už – DU	didesnė pagal reikšmę	privalo būti didesnė už	ar didesnė?
Daugiau-arba-lygu – DLG	didesnė arba lygi pagal reikšmę	privalo būti didesnė arba lygi	ar didesnė už arba lygi?
Mažiau-už – MU	mažesnė pagal reikšmę	privalo būti mažesnė pagal reikšmę	ar mažesnė?
Mažiau-arba-lygu – MLG	mažesnė arba lygi pagal reikšmę	privalo būti mažesnė arba lygi	ar mažesnė už arba lygi?
VI. Matematinio įvertinimo *Išskirtinė savybė*: Jokio kito tipo taisyklės negali atlikti matematinių skaičiavimų. *Išeigos vertės tipas*: rezultatas/*REZ*			
Apskaičiuota – APSK	reikšmė tokia pati, kaip ir vartotojo nurodyto skaičiavimo rezultatas	privalo būti lygi skaičiavimo rezultatui	ar lygi rezultatui?
SUM, ATM, DGB, PDL ir t.t.	reikšmė tokia pati, kaip ir nurodyto skaičiavimo rezultatas	privalo būti lygi rezultatui	ar lygi rezultatui?
VII. Projekcijos valdikliai *Išskirtinė savybė*: jokio kito tipo taisyklės negali automatiškai “projektuoti” vienos vertės (pvz., loginio vieneto) į kitą vertę, pvz: automatiškai įgalinti, kopijuoti arba iškviesti vykdymą. *Išeigos vertės tipas*: taisyklei IGL – įgalintų egzempliorių kiekis/*IEK*, KOP – nukopijuotų egzempliorių kiekis/*KEK*, IVK – įvykdytų egzempliorių kiekis/*VEK*.			
Įgalinta – IGL	reikalauja papildomo būvio	privalo paskleisti būvį	ar būvis paskleistas?
Nukopijuota – KOP	reikalauja vertės kopijavimo	privalo nukopijuoti vertę	ar vertė nukopijuota?
Įvykdyta – IVK	reikalauja įvykdymo	privalo būti įvykdyta	ar įvykdyta?

2 Priedas. Išvestinių VT tipai pagal Roso metodą

Vertimas į lietuvių kalbą parengtas pagal [Ross, 1997].

A. Egzempliorių testavimo išvestinės taisyklės

EGZ egzistuojanti-anksčiau : Patikrina, ar bent vienas korespondento egzempliorius egzistavo prieš sukuriant bazės egzempliorių.
P pirmesnioji : Patikrina, ar korespondento egzempliorius egzistavo tam tikru metu prieš sukuriant bazės egzempliorių.
TES besitęsianti : Patikrina korespondento egzempliorius siekiant nustatyti ar bent vienam iš jų galiojo visos žemiau pateiktos sąlygos:
- egzempliorius egzistavo prieš sukuriant bazės egzempliorių,
- egzempliorius egzistavo visą bazės egzistavimo laikotarpį,
- egzempliorius nėra ištrinamas (jeigu apskritai yra trinamas) kartu su bazės egzemplioriumi.

PR proporcinė : Tikrina, ar numatytoji procentinė bazės egzempliorių dalis turi korespondento egzempliorių.
BP bendrai-priklausoma : Nurodo, kad visi iš dviejų ar daugiau tipų turi būti teisingi (*true*) arba visi neteisingi (*not true*). Kitaip tariant, ši taisyklė nurodo, kad kiekvienas bazės egzempliorius arba turi kiekvieno korespondento tipus, arba nė vieno iš jų.
FP funkciškai-priklausoma : Tikrina, ar du duomenų objekto egzemplioriai turi tą pačią vieno atributo tipo reikšmę ir ar tą pačią reikšmę turi ir kiti(-as) atributų tipai(-as). Jeigu taip, tuomet pastarieji tipai vadinami funkciškai-priklausomais nuo pirmųjų tipų.
SS susieta : Tikrina, ar bazės egzempliorius yra susijęs (arba, jeigu neigiamoj formoj, – nesusijęs) su korespondento egzemplioriumi (kuris yra duomenų objektas) dviem ar daugiau sąryšių tipais (ir gal būt vienu ar daugiau potipių sąryšių).

B. Pozicijos patikrinimo išvestinės taisyklės

Šios išvestinė taisyklės yra panašios į *Pozicijos patvirtinimo* tipų grupės taisykles, tačiau išskiriami tokie skirtumai:
- Šios išvestinės taisyklės nesuformuoja išvestinių faktų.
- Jos nesuformuoja egzempliorių, kuriuos gali panaudoti kitos taisyklės, porų ar kombinacijų.
- Kiekvienas jų bazės egzempliorius turės tik vieną teisingą reikšmę, vieną išeigos reikšmę ir vieną iškvietimo reikšmę.

POZ-SD pozicinė-susiejanti-daug : atitinka taisyklę Pozicinė.
ZEM-SD žemiausia-susiejanti-daug : atitinka taisyklę Žemiausia.
AUK-SD aukščiausia-susiejanti-daug : atitinka taisyklę Aukščiausia.
CHRO-SD chronologinė-susiejanti-daug : atitinka taisyklę Chronologinė.
SEN-SD seniausia-susiejanti-daug : atitinka taisyklę Seniausia.
NAU-SD naujausia-susiejanti-daug : atitinka taisyklę Naujausia.

C. Atributų modifikavimo išvestinės taisyklės

DID didėjanti : Tikrina, ar nauja atributo tipo reikšmė yra didesnė už prieš tai buvusiąją.
MAZ mažėjanti : Tikrina, ar nauja atributo reikšmė yra mažesnė už prieš tai buvusiąją.
Likusios šio tipo išvestinės taisyklės yra panašios į *Funkcinio patvirtinimo* tipų grupės taisykles, tačiau išskiriami tokie skirtumai:
- Šių tipų taisyklės paprastai yra taikomos tik vienam bazės egzemplioriui, o ne visiems.
- Šių taisyklių taikymo sekoje atsižvelgiama ne tik į esamas, bet ir į buvusias reikšmes. Kuomet taisyklė yra sustabdoma, o vėliau – vėl paleidžiama, ji darbą tęsia nuo ten, kur jį baigė.

UNIK-AM unikali-atributų-modifikavimui : Atitinka taisyklę Unikali (atsižvelgiant į skirtumus).
SV-AM svyruojanti-atributų-modifikavimui : Atitinka taisyklę Svyruojanti (atsižvelgiant į skirtumus).
AUG-AM auganti-atributų-modifikavimui : Atitinka taisyklę Auganti (atsižvelgiant į skirtumus).
KRE-AM krentanti-atributų-modifikavimui : Atitinka taisyklę Krentanti (atsižvelgiant į skirtumus).
NAT-AM neatnaujinama-atributų-modifikavimui : Atitinka taisyklę Neatnaujinama (atsižvelgiant į skirtumus).

D. Sekos valdymo išvestinės taisyklės

IN inicijuojanti : Nurodo, kad kuomet atvira loginė seka neturi esamos pozicijos, tokia pozicija gali būti nustatoma tik žemiausiam (arba pirmam) korespondentui.
JP judėjimo-pirmyn : Nurodo, kad bet koks judėjimas atviroje loginėje sekoje turi būti judėjimas į priekį.
PR progresyvi : Nurodo, kad kiekvienas pereiga pirmyn atviroje loginėje sekoje turi būti pereiga į kitą, nuosekliai sekantį po esamo, sekos elementą.
RR regresyvi : Nurodo, kad kiekvienas pereiga atgal atviroje loginėje sekoje turi būti pereiga į kitą, nuosekliai sekantį prieš esamą, sekos elementą.
PIN pakartotinio-inicijavimo : Nurodo, kad kiekvienas esamos padėties svyravimas (išskyrus gal tik pirmos pozicijos svyravimą) atviroje loginėje sekoje turi prasidėti žemiausiu korespondentu.
CKN ciklinė : Nurodo, kad kiekvienas esamos pozicijos svyravimas turi būti pilnas.

E. Nuoseklumų specifikavimo išvestinės taisyklės

SABL šabloninė : Nurodo, kad sekos egzemplioriai turi savybes iš tam tikros apibrėžtos sekos.

F. Kompozicijos struktūrų testavimo išvestinės taisyklės

APR apribota : Naudojama sąryšių tipams rekursyvios struktūros viduje patikrinti. Nurodo, kad bazės egzempliorius (visada sąryšio tipas) privalo (arba neprivalo) turėti nepertrauktą korespondento(-ų) (taip pat visada sąryšio tipas), kuris prasideda ir baigiasi tais pačiais iš anksto susieto objekto egzemplioriais, egzempliorių seką.
ATS atspindinti : Tikrina, ar į kompozicijos struktūrą įeinančio duomenų objekto egzempliorius yra susietas su savimi pačiu “per” vieną rekursijos lygį (bent viena kryptimi).
CKS cikliška : Tikrina, ar į kompozicijos struktūrą įeinančio duomenų objekto egzempliorius yra susietas su savimi pačiu “per” vieną ar daugiau rekursijos lygį (bent viena kryptimi).
TR tranzitinė : Tikrina, ar į kompozicijos struktūrą įeinančiam duomenų objekto egzemplioriui yra tenkinama ši sąlyga: turint bet kokį kitą šio duomenų objekto egzempliorių, susietą su nagrinėjamu egzemplioriumi “per” daugiau negu vieną rekursijos lygį, šis *kitas* egzempliorius taip pat yra susietas ir per vieną rekursijos lygį.
MZG mazgas : Tikrina, ar kompozicijos struktūros duomenų objekto egzempliorius duotąja kryptimi yra mazgas. Turi būti tenkinamos dvi sąlygos:

- egzempliorius nurodytąja kryptimi yra įtrauktas bent į vieną rekursijos lygį,
- jo nėra nė vienoje pilnoje rekursijoje priešinga kryptimi.

G. Sąlyginio laiko įvertinimo išvestinės taisyklės

TR trukmės : Tikrina, kiek praėjo laiko nuo to vartotojo nurodyto laiko momento. Tai atliekama atimant atributo, saugančio šį laiką, reikšmę iš esamo laiko reikšmės.
LAUK laukianti : Tikrina, keik dar liko laiko iki vartotojo nurodyto laiko momento.
TRMP trumpiausia : Nustato dviejų ar daugiau korespondento egzempliorių egzistavimo trukmes, jas palygina ir randa mažiausią iš jų.
ILG ilgiausia : Nustato dviejų ar daugiau korespondento egzempliorių egzistavimo trukmes, jas palygina ir randa didžiausią iš jų.

H. Atnaujinimų įvertinimo išvestinės taisyklės

ATN-KIEK atnaujinimų-kiekio : Įvertina taisyklės korespondentų atnaujinimų, įvykusių esant vienam taisyklės bazės egzemplioriui, kiekį.
ATN-DZN atnaujinimų-dažnio : Įvertina taisyklės korespondentų atnaujinimų dažnį esant veinam taisyklės egzemplioriui.
ISL įšaldanti : Patikrina, ar korespondento egzemplioriai buvo atnaujinti, ar ne. Integralumo apribojimo formoje ši taisyklė draudžia atnaujinimus, o egzemplioriai, kuriems ji pritaikoma, vadinami *įšaldytais*.
ISLV įšaldanti-vartotojams : Analogiška taisyklei ISL, įšaldymo būsena galioja tik vartotojams, automatiški atnaujinimai leidžiami.

I. Veiklos koordinavimo išvestinės taisyklės

ATN-B bendro-atnaujinimo : Tikrina, ar bent vienas atnaujinimas įvyksta dviems ar daugiau korespondentų vienu metu.
ATN-P pakartotinio-atnaujinimo : Analogiškai taisyklei ATN-B, tačiau tikrinimas atliekamas kiekvieno apdorojimo žingsnio metu atskirai, o pati taisyklė gali turėti tik vieną korespondentą.
SNCH sinchronizavimo : Tikrina, ar pasikeitus bazės reikšmei, pasikeičia 9arba naudojant paneigtą – nepasikeičia) korespondento(-ų) reikšmė(-s).
PLSK paleidimo-sekos : Šio tipo taisyklės esant reikalui gali būti panaudotos nustatyti taisyklių (ar kitų komponentų, kuriuos reikia įgalinti) įgalinimo (paleidimo) seką, jeigu jos yra iškviečiamos to paties įvykio.

J. Egzempliorių įgalinimo išvestinės taisyklės

IGP įgalinimo-su-atkūrimu : Analogiška nedalomų taisyklių tipui IGL (Įgalinta). Skiriasi tuo, jog ištrynus bazės egzempliorių, taisyklė automatiškai ištrins (arba kitu atveju sukurs) ir korespondento egzempliorius.
IGPA įgalinimo-su-atkūrimu-apribota : Analogiška taisyklei IGP, tačiau taikoma norint išvengti klaidų, kylančių, kuomet bet kuriam tokios taisyklės korespondento egzemplioriui vienu metu egzistuoja keletas bazės egzempliorių ir vienas (bet ne visi) iš šių egzempliorių yra ištrintas ir tuo pačiu reikia automatiškai atkurti korespondento egzempliorių. Tokiu atveju taisyklė užtikrina, kad tokio egzemplioriaus atkūrimas įvyks tik tuomet, kai bus ištrintas paskutinysis iš bazės egzempliorių.
PER perjungimo : Analogiška taisyklių tipui IGP, tačiau korespondento(-ų) egzempliorius(-iai) turi būti pakeistas(-i) nepriklausomai nuo ankstesnės reikšmės, t.y. jeigu įgalinta, turi tapti neįvertinta, jeigu neįvertinta – įgalinta.
PERP perjungimo-su-atkūrimu : Taisyklė labai panaši į PER, tačiau veikia kaip IGP, t.y. ištrynus taisyklės bazės egzempliorių, korespondento egzemplioriai yra grąžinami į pradinę būseną.
LKP laiko-paskyrimo : Korespondento egzemplioriams nurodo laiko limitą.
IAP intervalų-tarp-atnaujinimų-paskyrimo : Įvertina laiko intervalus tarp vieno ar daugiau korespondento tipo(-ų) egzempliorių atnaujinimų. Naudojama kaip integralumo apribojimas, ji veikia kaip taimeris, ir gali automatiškai atlikti atnaujinimus.

K. Egzempliorių kopijavimo išvestinės taisyklės

PKOP nukopijuota-su-atkūrimu : Ši taisyklė panaši į nedalomą taisyklę *Nukopijuota*, todėl ji reikalauja, kad bazės egzemplioriaus reikšmė būtų nukopijuota (paskleista) į visus korespondento(-ų) egzempliorius. Be to, jeigu bazės egzempliorius ištrinamas, korespondento(-ų) reikšmės atkuriamos, t.y. nustatomos tokios reikšmės, kokios buvo prieš pradedant veikti taisyklei.
ZEM-PKOP žemiausia-nukopijuota-su-atkūrimu : Panaši į taisyklę PKOP, tačiau taikoma norint išvengti klaidų, kylančių tuomet, kai bet kuriam tokios taisyklės korespondento egzemplioriui vienu metu egzistuoja keletas bazės egzempliorių, ir šie egzemplioriai turi dvi ar daugiau reikšmių. Tokiu atveju, bus nukopijuojama žemiausia iš šių reikšmių.
AUK-PKOP aukščiausia-nukopijuota-su-atkūrimu : Analogiška taisyklei ZEM-PKOP ir taikoma tais pačiais tikslais, tačiau parenkama ir nukopijuojama aukščiausia reikšmė.
SEN-PKOP seniausia-nukopijuota-su-atkūrimu : Analogiška taisyklei ZEM-PKOP ir taikoma tais pačiais tikslais, tačiau parenkama ir nukopijuojama seniausia reikšmė.
NAU-PKOP naujausia-nukopijuota-su-atkūrimu : Analogiška taisyklei SEN-PKOP ir taikoma tais pačiais tikslais, tačiau parenkama ir nukopijuojama naujausia reikšmė.

L. Egzempliorių iškvietimo išvestinės taisyklės

IVP įvykdyta-su-atkūrimu : Atitinka nedalomą taisyklę *Įvykdyta*, t.y. reikalauja, kad kiekvienas korespondento egzemplioriaus būtų įgalintas (įvykdytas, jeigu veiksmas; paleistas, jeigu taisyklė) – turėtų įgalinimo reikšmę “šiuo metu įgalintas”, “įgalintas sėkmingai” arba “įgalintas nesėkmingai”. Tačiau ši taisyklė taip pat pakeičia korespondento egzemplioriaus(-ių) reikšmes (į “neįgalinta”), jeigu bazės egzempliorius yra ištrinamas.
PAL paleista : Gali būti panaudota siekiant, kad viena taisyklė įgalintų (paleistų) kitą taisyklę (kuri apibrėžiama kaip pirmosios taisyklės korespondentas).

IVN numatyta-vykdymui : Taisyklė, ekvivalentiška išvestinei taisyklei LKP (Laiko-paskyrimo), tačiau čia korespondentas(-ai) visada yra veiksmas(-ai). Taisyklė nurodo, per kiek laiko turi būti įvykdyti tokio veiksmo(-ų) egzemplioriai.

Išvestinių taisyklių tipai pagal Roso metodą (pagal [Ross, 1997])

A **Egzempliorių testavimas**	B **Pozicijos patikrinimas**	C **Atributų modifikavimas**	D **Sekos valdymas**	E **Nuoseklumų specifikavimas**	F **Kompozicijos struktūrų testavimas**	G **Sąlyginio laiko įvertinimas**	H **Atnaujinimų įvertinimas**	I **Veiklos koordinavimas**	J **Egzempliorių įgalinimas**	K **Egzempliorių kopijavimas**	L **Egzempliorių iškvietimas**
EGZ egzistuojanti-anksčiau	*POZ-SD* pozicinė-susiejanti-daug	*DID* didėjanti	*IN* inicijuojanti	*SABL* šabloninė	*APR* apribota	*TR* trukmės	*ATN-KIEK* atnaujinimų-kiekio	*ATN-B* bendro-atnaujinimo	*IGP* įgalinimo-su-atkūrimu	*PKOP* nukopijuota-su-atkūrimu	*IVP* įvykdyta-su-atkūrimu
P pirmesnioji	*ZEM-SD* žemiausia-susiejanti-daug	*MAZ* mažėjanti	*JP* judėjimo-pirmyn		*ATS* atspindinti	*LAUK* laukianti	*ATN-DZN* atnaujinimų-dažnio	*ATN-P* pakartotinio-atnaujinimo	*IGPA* įgalinimo-su-atkūrimu-	*ZEM-PKOP* žemiausia-nukopijuota-su-	*PAL* paleista
TES besitęsianti	*AUK-SD* aukščiausia-susiejanti-daug	*UNIK-AM* unikali-atributų-modifikavimui	*PR* progresyvi		*CKS* cikliška	*TRMP* trumpiausia	*ISL* įšaldanti	*SNCH* sinchronizavimo	*PER* perjungimo	*AUK-PKOP* aukščiausia-nukopijuota-su-	*IVN* numatyta-vykdymui
PR proporcinė	*CHRO-SD* chronologinė-susiejanti-daug	*SV-AM* svyruojanti-atributų-	*RR* regresyvi		*TR* tranzitinė	*ILG* ilgiausia	*ISLV* įšaldanti-vartotojams	*PLSK* paleidimo-sekos	*PERP* perjungimo-su-atkūrimu	*SEN-PKOP* seniausia-nukopijuota-su-	
BP bendrai-priklausoma	*SEN-SD* seniausia-susiejanti-daug	*AUG-AM* auganti-atributų-modifikavimui	*PIN* pakartotinio-inicijavimo		*MZG* mazgas				*LKP* laiko-paskyrimo	*NAU-PKOP* naujausia-nukopijuota-su-	
FP funkciškai-priklausoma	*NAU-SD* naujausia-susiejanti-daug	*KRE-AM* krentanti-atributų-	*CKN* ciklinė						*IAP* intervalų-tarp-atnaujinimų-		
SS susieta		*NAT-AM* neatnaujinama-atributų-									

3 Priedas. Straipsnio publikacija

(Pridedama atšviesta kopija)

VILNIAUS UNIVERSITETAS
VYTAUTO DIDŽIOJO UNIVERSITETAS
KAUNO TECHNOLOGIJOS UNIVERSITETAS

14- OSIOS TARPUNIVERSITETINĖS MAGISTRANTŲ IR DOKTORANTŲ MOKSLINĖS KONFERENCIJOS „INFORMACINĖS TECHNOLOGIJOS“ PRANEŠIMŲ MEDŽIAGA

2009

14-osios tarpuniversitetinės magistrantų ir doktorantų mokslinės konferencijos „Informacinės technologijos“ pranešimų medžiaga 2009.05.08

PROGRAMINIO KODO GENERAVIMAS IŠ GRAFINIŲ VEIKLOS TAISYKLIŲ MODELIŲ

Jonas Geležis[1], Kęstutis Kapočius[2]

[1]*Kauno technologijos universitetas, Informacijos sistemų katedra, jonas.gelezis@stud.ktu.lt*
[2]*Kauno technologijos universitetas, Informacijos sistemų katedra, kestutis.kapocius@ktu.lt*

Santrauka: Programinio kodo generavimo iš veiklos taisyklių modelių sritis iki šiol yra menkai ištirta ir tai neigiamai veikia veiklos taisyklių koncepcijos plėtrą. Straipsnyje nagrinėjama kodo generavimo iš grafinių IS (informacinių sistemų) reikalavimus atspindinčių modelių problema. Pristatomas modifikuotu Roso metodu grindžiamo veiklos taisyklių modeliavimo IS projektavimo stadijoje metodo tyrimas, siekiant sukurti adekvačią programinio kodo generavimo iš taisyklių diagramų metodiką.

Raktiniai žodžiai: Roso metodas, veiklos taisyklės, informacijos sistemų projektavimas, kodo generavimas.

1. Įvadas

Tradiciniai informacijos sistemų projektavimo metodai, didėjant operatyvaus informacijos apdorojimo poreikiui dinamiškai kintančioje aplinkoje, nėra pakankamai efektyvūs. Šių problemų sprendimui taikomas informacijos sistemų (IS) projektavimo būdas, pagrįstas veiklos taisyklių (VT) koncepcija. Kuriant IS pagal VT koncepciją, taisyklių identifikavimo, klasifikavimo, užrašymo fiksuota forma ir susiejimo su kitais IS projekto objektais klausimai yra ypač aktualūs. Minėtus VT tvarkymo veiksmus apimantis procesas vadinamas VT struktūrizavimu. Iš eilės žinomų teorinių VT struktūrizavimo modelių išsiskiria Roso (*Ronald Ross*) metodas, leidžiantis VT išreikšti grafinėmis diagramomis [1]. Šis metodas buvo konkretizuotas ir praplėstas, sukuriant pagal jį struktūrizuotų VT, duomenų modelio elementų ir kitos svarbios informacijos saugyklos loginį modelį (t.y. metodo metamodelį), kuris panaudotas sukuriant ir taisyklių modeliavimo įrankio prototipą [3, 4].

Vis tik šiuolaikiniai automatizuoto projektavimo (CASE) įrankiai turi programinio kodo generavimo iš tam tikrų modelių (ypač standartizuotų, pvz., UML) funkciją. Tuo tarpu kodo generavimo iš veiklos taisyklių modelių sritis yra menkai ištirta ir nėra konkrečios metodikos, kaip toks generavimas turėtų būti vykdomas. Šiame straipsnyje pristatomo tyrimo metu siekiama sukurti ir minėtame prototipiniame VT modeliavimo įrankyje įdiegti detalią metodiką, kurios pagalba grafiniai Roso veiklos taisyklių modeliai galėtų būti transformuojami į programinį kodą.

Straipsnio 2 skyriuje pateikiamas trumpas Roso metodo aprašymas. Tolimesniuose skyriuose aptariama pasirinkta kodo generavimo technologija bei pristatoma sukurtos metodikos koncepcija.

2. Grafinis veiklos taisyklių struktūrizavimo modelis, grindžiamas modifikuotu Roso metodu

Nuo pat veiklos taisyklių koncepcijos atsiradimo buvo siūlomi įvairūs jų klasifikavimo ir struktūrizavimo modeliai. Bendrosios VT identifikavimo, klasifikavimo ir fiksavimo idėjos, pirmąkart apibendrintos 1997 m. GUIDE projekte [6], „evoliucionavo“ iki oficialių standartų SBVR [9], PRR [8] ir kitų. Vis tik lygiagrečiai su GUIDE sukurtas Roso metodas iki šiol išlieka ryškiausiu veiksmo teiginių ir išvedimo tipų taisyklių modeliavimo grafinėmis diagramomis metodų. KTU Informacijos sistemų katedroje atlikto tyrimo metu Roso metodas buvo pritaikytas lietuvių kalbai, sukurtas metamodelis, apibrėžta taisyklių struktūrizavimo metodika [3, 4]. Vis tik automatizuoto VT modelių transformavimo į tam tikrą programinį kodą klausimas išliko atviras. Žemiau apžvelgiamos metodo charakteristikos, kurios buvo aktualios sprendžiant šią problemą.

Pagal Roso metodą taisyklės gali būti atominės ir išvestinės [1]. Išskiriami 32 atominių (nedalomų) taisyklių tipai, kurie grupuojami į 7 šeimas. Išvestinės taisyklės nėra atominės ir gali būti sudarytos iš keleto atominių taisyklių arba kitų išvestinių taisyklių. Išskiriami 58 išvestinių taisyklių tipai, kurie pagal naudojimo sritį grupuojami į 12 šeimų. Į lietuvių kalbą išversti VT šeimų pavadinimai pateikiami 1 lentelėje. Būtina pažymėti, kad ypač daug dėmesio R. Rosas skiria taisyklių tipų aprašymui, todėl šis modelis klasifikavimo atžvilgiu ypač tikslus.

1 lentelė. Veiklos taisyklių tipų šeimos pagal Roso metodą (pagal [1])

Atominių taisyklių šeimos	*Išvestinių taisyklių šeimos*	
I. Egzempliorių patvirtinimo	A. Egzempliorių testavimo	B. Pozicijos patikrinimo
II. Tipo patvirtinimo	C. Atributų modifikavimo	D. Sekos valdymo

III. Pozicijos patvirtinimo	E. Sekos specifikavimo	F. Kompozicijos struktūrų testavimo
IV. Funkciniai patvirtinimo	G. Sąlyginio laiko įvertinimo	H. Atnaujinimų įvertinimo
V. Lyginamojo įvertinimo	I. Veiklos koordinavimo	J. Egzempliorių įgalinimo
VI. Matematinio įvertinimo	K. Egzempliorių kopijavimo	L. Egzempliorių iškvietimo
VII. Projekcijos valdikliai		

Kuriant grafinę veiklos taisyklių diagramą, reikia atlikti tokius žingsnius:

1. Nustatyti, ar VT yra integralumo apribojimas, ar sąlyga. *Integralumo apribojimas* - tai taisyklė, kurios rezultatas visada privalo būti loginis vienetas (loginė tiesa) [1]. Taigi, apribojimai privalo būti tenkinami. *Sąlyga* - tai taisyklė, kurios rezultatas gali būti arba loginis vienetas, arba loginis nulis [1]. Rezultatas gali būti ir nežinomas. Sąlygos naudingos, kai reikia nustatyti, kokias taisykles vykdyti arba kokius testavimus atlikti (t. y. kokias kitas taisykles iškviesti), atsižvelgiant į tam tikrą situaciją. Sąlygas galima išdėstyti nuosekliai, nurodant, kas turi vykti, jeigu aukštesnio lygio taisyklė yra tenkinama.

2. Nustatyti VT bazę. Kiekviena taisyklė visuomet apibrėžia tam tikrą į duomenų modelį įeinančio tipo egzempliorių. Šis tipas vadinamas *baze*. Dažniausiai bazė - tai duomenų objektas, atributo tipas ir pan., tačiau tai gali būti ir kita VT ar jos išeigos reikšmė. Taisyklė gali būti traktuojama kaip jos bazės savybė [1]. Bazė yra svarbiausias elementas interpretuojant taisyklę, be to, bazės egzemplioriai gali būti įtraukti į taisyklės atliekamą patikrinimą. Šis patikrinimas daromas su kiekvienu bazės egzemplioriumi.

3. Nustatyti VT korespondentą. Visose taisyklėse, be bazės, nurodomi ir objektai, reikalingi taisyklės apibrėžto tipo testui atlikti. Tokie *korespondentais* vadinami objektai būtini kiekvienoje taisyklėje. Korespondentu gali būti ne tik duomenų tipas (pvz., duomenų objektas, atributas ir pan.), bet ir kita taisyklė, jos išeigos reikšmė arba veiksmas [1].

4. Parinkti taisyklės tipą. Tik esant nurodytam tipui galima teisingai interpretuoti taisyklę. Taisyklė gali priklausyti tiek atominiam tipui, tiek išvestiniam. Visi taisyklių tipai tinka ir integralumo apribojimams, ir sąlygoms. Taisyklės tipo santrumpa įrašoma apribojimo arba sąlygos simbolio viduje.

5. Susieti bazę ir korespondentą. Be pagrindinių taisyklės elementų dažnai svarbu užfiksuoti ir teisingą bazės ir korespondento sąryšį. Šis sąryšis privalo plaukti iš paties duomenų modelio. Svarbu nepamiršti, jog VT koncepcijos kontekste duomenų modelį galima traktuoti kaip terminų (lentelės, atributai) ir faktų (įvairūs sąryšiai tarp lentelių, lentelė-atributas sąryšiai) rinkinį.

Pažymėtina, jog VT niekuomet negali įvesti naujo termino ar fakto (sąryšio). Visi svarbūs faktai ir terminai turi būti išreikšti duomenų modelyje. Taigi, kiekvienas taisyklės bazės ir korespondento(-ų) sąryšį atspindintis objektas turi būti paimtas iš duomenų modelio. Svarbi ir kita prielaida: kiekvienas pavaizduotas taisyklės bazės ir korespondento(-ų) sąryšis detalizuojantis objektas (duomenų tipas) yra laikomas svarbiu taisyklei interpretuoti. Taigi, nesvarbūs objektai neturi būti įtraukiami į diagramą.

Analizuojant automatinio VT diagramų transformavimo galimybes, svarbu atsižvelgti ir į notacijoje numatytus papildomus grafinius simbolius, interpretatorius ir apribojimus, galinčius pakeisti standartinę VT interpretaciją tiek jos bazės, tiek ir korespondentų atžvilgiu. Be specialiųjų simbolių modeliuose gali būti naudojamos konstantos, taisyklių išeigos reikšmės ir kitos papildomos priemonės [1].

Kaip jau buvo užsiminta skyriaus pradžioje, kuriant programinio kodo generavimo iš Roso VT diagramų metodikos specifikaciją, turi būti atsižvelgiama ne tik į minėtas metodo savybes, bet ir į anksčiau sukurtojo metamodelio struktūrą. Šio straipsnio tikslas – pristatyti bendrąją sukurtos metodikos koncepciją, todėl minėtas metamodelis čia detaliai nenagrinėjamas.

3. Kodo generavimo technologija

Siekiant generuoti programinį kodą, reikalingas kodo generatorius. Apžvelgus prieinamas atvirojo kodo alternatyvas, nuspręsta generatorių realizuoti Ruby programavimo kalba [10]. Ši kalba puikiai pritaikoma tiek mažoms, tiek didelėms objektiškai orientuotoms programoms kurti.

Kodo generavimui iš modelių naudojami šablonai, o darbui su jais labai svarbios šablonų sistemos. Be jų generuojamas kodas išsidėsto viename lygmenyje su kitas funkcijas atliekančiu generatoriaus kodu, pvz. generuojant Java kodą Java programavimo kalba parašytu generatoriumi, tampa sudėtinga atskirti kur aprašomas generuojamas programos kodas, o kur generatoriaus vykdomasis kodas. Nagrinėjamu atveju buvo nuspręsta naudoti Ruby kalbai pritaikytą ERB tekstinių šablonų sistemą [5]. ERB pasižymi tuo, jog yra kompaktiškas, paprastas, patikimas ir portatyvus. Naudojant ERB, išsprendžiama vykdomojo ir generuojamo programinio kodo atskyrimo problema, nes skirtingos paskirties kodas patalpinamas atskiruose failuose [7].

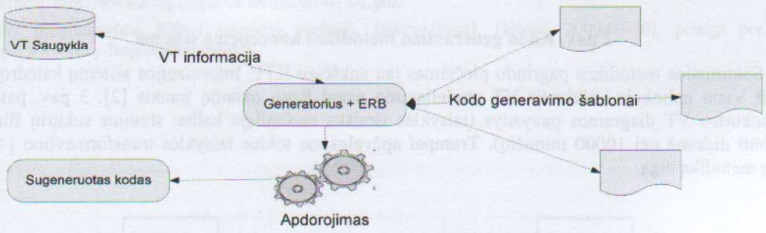
Aukščiau minėtų elementų tarpusavio sąveika apibendrinta 1 paveikslėlyje pateiktoje koncepcinėje siūlomo kodo generatoriaus architektūros schemoje. Nagrinėjamu atveju generatorius tiesiogiai komunikuoja su VT saugykla, kurioje saugoma informacija apie VT modelius, kurie turės būti transformuojami į programinį kodą.

4. Kodo generavimo iš VT modelių metodika

Atlikus Roso metodo analizę nuspręsta, jog šio tyrimo rėmuose tikslingiausia generuoti SQL kalbos kodą (akcentuojamas apribojimų ir trigerių generavimas), nes:

1. Roso VT diagramos iš esmės yra deklaratyvaus, o ne procedūrinio pobūdžio, jose dažniausiai bus fiksuojami apribojimai duomenų modelio objektams (lentelėms, atributams, ryšiams).

2. Siekiama užtikrinti greitą susijusių sistemos charakteristikų keitimąsi pasikeitus atitinkamai veiklos taisyklei. SQL trigeriai ir apribojimai leidžia greitai įdiegti tam tikras darbo su duomenų bazėje saugoma informacija taisykles.



1 pav. Kodo generatoriaus architektūra

Roso metodas teigia, kad VT šeimos viena nuo kitos skiriasi tuo, kad sugeba atlikti tik sau unikalų patikrinimą arba testą. Remiantis šiuo išskirtinumu, kiekvienai VT kuriamas atskiras kodo generavimo šablonas, kuris realizuoja tai taisyklei unikalų testą generuojamoje kalboje.

Įvertinant šį ir kitus, aukščiau aptartus sprendimus, sukurta kodo generavimo iš VT modelių metodika, kurios apibendrinta schema pateikiama 2 pav. Metodiką sudaro šeši žingsniai, apibrėžiantys bendrą veiksmų seką įvertinant VT modelį:

1. Konkretaus VT modelio informacijos gavimas iš VT saugyklos. Šiuo konkrečiu atveju priimama, jog saugykla sukurta pagal minėtąjį Roso metamodelį [3, 4].. Tokiu būdu ne tik sukurta galimybė nesudėtingai realizuoti metodiką, praplečiant anksčiau sukurtąjį prototipą, bet ir sustiprintos prielaidos metodikos tyrimams bei potencialiam platesniam taikymui ateityje

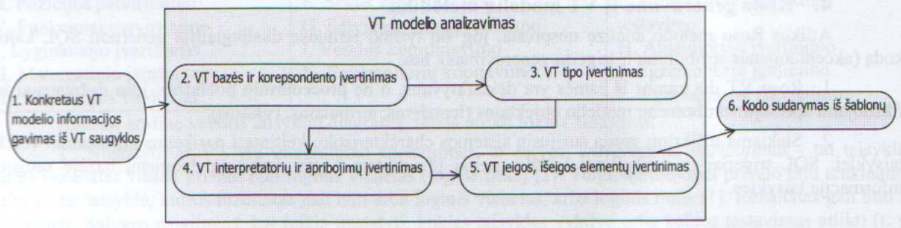
2. VT bazės ir korespondento įvertinimas, nustatant bazės ir korespondento(-ų) tipus (tai gali būti lentelės, atributai, ryšiai). Pavyzdžiui,, jeigu bazė arba korespondentas yra atributas, surandama, kokiai lentelei priklauso atributas ir patikrinama, kokie yra lentelių tarpusavio ryšiai, nes juos reikia atspindėti generuojamame kode. Tarkim tarp lentelių esant sąryšiui 1:N, SQL kodą gali tekti papildyti WHERE sąlyga su konkrečia nurodoma reikšme ir t.t.

3. VT tipo įvertinimas. VT tipas apsprendžia, koks patikrinimas bus taikomas korespondentui bazės atžvilgiu, pvz., VT tipas SUM (sumuojama) nurodo, kad korespondento reikšmės privalo būti sumuojamos.

4. VT interpretatorių ir apribojimų įvertinimas. Specialieji elementai gali žymiai pakeisti VT prasmę, priklausomai nuo jų tipo. Pavyzdžiui, panaudojant interpretatorių, taisyklės galiojimą galima išplėsti nuo konkretaus egzemplioriaus iki visos aibės egzempliorių tikrinimo, - tokiu atveju SQL kodas turi būti suformuojamas visai egzempliorių aibei.

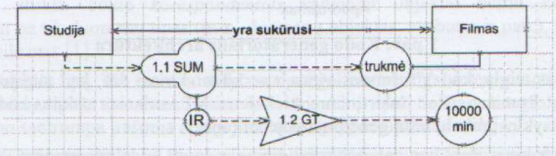
5. VT įeigos, išeigos elementų įvertinimas. Vertinamos modelyje nurodytos įeigos, išeigos taisyklės ir išeigos reikšmės. Teisingas šių galimų scenarijų įvertinimas svarbus, nes konkrečios VT patikrinimo rezultatas gali būti išsaugomas jos išeigos reikšmėje vėlesniam panaudojimui ir pan.

6. Kodo sudarymas pagal šablonus. Panaudojant apdorotą informaciją, iš pagal šablonus sugeneruotų teksto fragmentų gaunamas SQL kodas. Pavyzdžiui, generuojant SELECT sakinį perduodamas lentelių pavadinimų sąrašas, kuris panaudojamas užklausos FROM daliai formuoti.



2 pav. Kodo generavimo metodikos koncepcinė schema

Sukurtosios metodikos pagrindu plečiamas jau anksčiau KTU Informacijos sistemų katedroje sukurtas Microsoft Visio aplinkoje veikiantis VT modeliavimo pagal Roso metodą įrankis [2]. 3 pav. pateiktas šiuo įrankiu sukurtos VT diagramos pavyzdys (taisyklės išraiška natūraliaja kalba: studijos sukurtų filmų trukmė privalo būti didesnė nei 10000 minučių). Trumpai apžvelgsime tokios taisyklės transformavimo į SQL pagal sukurtąją metodiką eigą.



3 pav. VT modelio pavyzdys

Modelį sudaro dvi taisyklės: „susumuota“ tipo sąlyga (1.1) bei „daugiau-už“ tipo apribojimas (1.2). Pastarojo korespondentas - tai 1.1 taisyklės išeigos reikšmė, kurioje šiuo atveju atsispindi bendra konkrečios studijos sukurtų filmų trukmė. Taisyklėje apibrėžtas apribojimas turi būti tenkinamas nuolat, todėl jį galima realizuoti SQL apribojimu, neleisiančiu atlikti duomenų modifikacijų, po kurių minėta sąlyga būtų pažeista. Taigi, pagal aptartąją metodiką bus sugeneruotas toks SQL apribojimas (priimama, jog lentelėje `Filmas` ryšį su lentele `Studija` realizuoja išorinis raktas `studija` – ši informacija be kita ko saugoma VT saugykloje):

```
CREATE ASSERTION BendraTrukme CHECK (10000 < ALL
(SELECT SUM(trukme)
FROM Filmas
GROUP BY studija));
```

5 Išvados

Straipsnyje nagrinėjamas informacijos sistemų kūrimas pagal veiklos taisyklių koncepciją, akcentuojant modifikuotą Roso metodą. Šis sprendimas iš kitų metodų išsiskiria tuo, jog taiko grafinę notaciją struktūrizuotiems VT modeliams sudaryti IS projektavimo stadijoje.

Pristatyto tyrimo metu buvo sukurta SQL kodo generavimo iš Roso VT diagramų metodikos koncepcija. Metodika bus toliau detalizuojama ir realizuojama, praplečiant KTU Informacijos sistemų katedroje sukurtą VT modeliavimo įrankio prototipą. Aptartas sprendimas aktualus ne tik stiprinant jau anksčiau apibrėžto Roso metodo metamodelio taikymo ir plėtros galimybes, bet ir tiriant grafinių priemonių naudojimo veiklos taisyklėms modeliuoti kuriant informacijos sistemas perspektyvas.

Literatūra

[1] **Ross, R. G**. The Business Rule Book: Classifying, Defining an Modeling Rules. *Business Rule Solutions*. 1997.

[2] **Baškevičius, S., Kapočius, K.** Veiklos taisyklių struktūrizavimo informacijos sistemų projektavimo metu įrankio realizacija. *Informacinės technologijos – 2006. Tarpuniversitetinė magistrantų ir doktorantų konferencija. Pranešimų medžiaga. Vilnius, VU leidykla*, 2 dalis, p. 91-96.

[3] **Butleris, R., Kapočius, K.** Struktūrizuotų veiklos taisyklių saugyklos architektūra. *Informacijos mokslai, Nr. 17, Vilnius, Vilniaus universiteto leidykla*. 2001, pp. 46-57.

[4] **Butleris, R., Kapočius, K.** The Business Rules Repository for Information Systems Design. *The 6th East-European Conference ADBIS'2002. Konferencijos pranešimų medžiaga. Bratislava, Slovakia, Vydavatel'stvo STU*, Vol. 2. 2002, pp.64-77.

[5] ERB - Ruby Templating. [interaktyvus]. [žiūrėta 2009-03-20], prieiga per internetą: http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html.

[6] Hay D. et al. (2001) Defining Business Rules - What Are They Really? GUIDE projektas, revizija 1.3, [interaktyvus], [žiūrėta 2008.06.26]. Prieiga per internetą: http://www.businessrulesgroup.org/first_paper/br01c0.htm

[7] **Herrington, J.** Code generation in action. *Manning Publications Co*. 2003.

[8] **Object Management Group**. Production Rule Representation (PRR), Beta 1. Iš *Object Management Group internetinio puslapio* [interaktyvus], [žiūrėta 2008-06-15], prieiga per internetą: http://www.omg.org/docs/dtc/07-11-04.pdf.

[9] **Object Management Group**. Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 OMG Available Specification. Iš *Object Management Group internetinio puslapio* [interaktyvus], [žiūrėta 2008-01-18], prieiga per internetą: http://www.omg.org/docs/formal/08-01-02.pdf.

[10] Ruby programavimo kalbos interneto puslapis, [interaktyvus]. [žiūrėta 2009-03-20], prieiga per internetą: http://www.ruby-lang.org/en/about/.