KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Artūras Korsakas

# Veiklos taisyklių manipuliavimo mechanizmo ir duomenų bazės tarpusavio sąveikos tyrimas

Magistro darbas

Darbo vadovas
prof. dr. Rimantas Butleris
Konsultantė
dokt. Lina Tutkutė

Kaunas
2009

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Artūras Korsakas

# Veiklos taisyklių manipuliavimo mechanizmo ir duomenų bazės tarpusavio sąveikos tyrimas

Magistro darbas

Recenzentas

Dr. Liudas Motiejūnas
2009-05-

Vadovas

prof. dr. Rimantas Butleris
2009-05-

Konsultantė

Dokt. Lina Tutkutė
2009-05-

Atliko

IFM-3/1 gr. stud.
Artūras Korsakas
2009-05-

Kaunas
2009

# SUMMARY

Business rules manipulation mechanism and database interaction research. Nowadays, nearly all of the commercial and government organizations are highly dependent on software system. Due to the inherent dynamic nature of their business environment, software evolution is inevitable. The growing of needs in management of global organizations similarly is growing the expert systems designing companies, which offer their own organizations management systems. This paper substance is how to transform the data structures to the business rules management systems, without losing the database structure elements, and ensuring fully interaction with the database. Representing the concept of business rules, analyzing architecture of business rules manipulation mechanisms. Exploring the *Blaze Advisor* as tool of business rule management system, which is implement of components such as: rule sets, decision trees, decision tables, and etc. These components are developing the processes of organization, which helps to efficient maintain and control the operations of internal logic. A study carried out between the *Blaze Advisor* tool and database, and established how to filtering the data from database, and then a data are transformed into information, and under information using the components to get the solution. To give spirit of this work was formulated a method of transforming data structures into development environment of business rules.

# TURINYS

# PAVEIKSLŲ SĄRAŠAS

# LENTELIŲ SĄRAŠAS

# 1. ĮVADAS

Augant, stambių pasaulinių organizacijų poreikiams, šalia auga ir ekspertines sistemas kuriančios kompanijos, kurios siūlo savo kuriamas organizacijos valdymo sistemas. Nes šiai dienai, tradicinės informacinės sistemos nebetenkina šiuolaikinių organizacijų poreikių. Tokios sistemos, kuriose veiklos logika yra integruojama į programinį kodą, yra keičiamos į sistemas, kurios palaiko veiklos taisykles atskyrai nuo programinio kodo. Taip užtikrinamas greitesnis organizacijos veiklos valdymas. Taip pat, yra įvairių nuomonių ir siūlymų, kaip geriau įgyvendinti veiklos taisykles šiuolaikinėse informacinėse sistemose, kuriant saugyklas ar įgyvendinant veiklas duomenų bazės trigeriuose. Tačiau šiuo metu besivystantys manipuliavimo mechanizmai neturi sau konkurentų. Taigi, tokios kompanijos kaip Oracle, Microsoft, FICO ir kitos – siūlo savo komercinius produktus, kurie leidžia kurti, valdyti ir svarbiausia lengvai palaikyti organizacijos veiklą. Kiekviena kompanija siūlo savo išskirtinius principus kaip tai padaryti. Darbe yra pateikiami šių kompanijų mechanizmai – jų architektūros ir komponentai. Tiriama FICO kompanijos Blaze Advisor architektūra ir jos sprendimų komponentai. Analizuojama, kaip įrankio komponentai remią arba riboją sistemos objektų elgseną, t.y. kaip iš duomenų bazės duomenų yra formuojama informacija ir pagal ją pasinaudojus komponentais išgaunami sprendimai. Analizuojama problematinė sritis: kaip duomenų struktūras transformuoti į veiklos taisyklių kūrimo aplinką, neprarandant duomenų bazės struktūros elementų. Ir taip užtikrinant pilną sąveika su duomenų baze.

Darbo tikslas – išanalizuoti veiklos taisyklių manipuliavimo mechanizmo ir duomenų bazės tarpusavio sąveiką projektavimo etape ir sudaryti metodiką atspindinčią duomenų struktūrų transformavimą į veiklos taisyklių (VT) kūrimo aplinką. Šiam tikslui įgyvendinti yra iškelti uždaviniai:

- palyginti VT manipuliavimo mechanizmų architektūras;
- ištirti ir įvertinti *Blaze Advisor* įrankio architektūrą, kaip veiklos taisyklių valdymo sistemą;
- išanalizuoti kaip įrankis iš duomenų bazės esančių duomenų formuoja informaciją, ir pagal ją atlieka sprendimą;
- suformuoti metodiką: kaip duomenų struktūras transformuoti į VT kūrimo aplinką;
- parengti mokomąja medžiagą ir suskurti parodomuosius pavyzdžius;

Atsakymai į keturis pagrindinius klausimus: *kas?, kaip?, kam? ir kodėl?*. Sparčiai augant įmonių veiklai sunku valdyti statinį veiklos mechanizmą, jei keičiantis vidiniai įmonės logikai reikia iš naujo kviestis programuotojus tam, kad atliktų reikalingus veiklos pakeitimus programiniame kode. Tokie statiniai mechanizmai yra neefektyvūs, taigi, parankiau naudoti dinaminį mechanizmą – kai vidinę organizacijos logiką gali valdyti netechninis personalas.

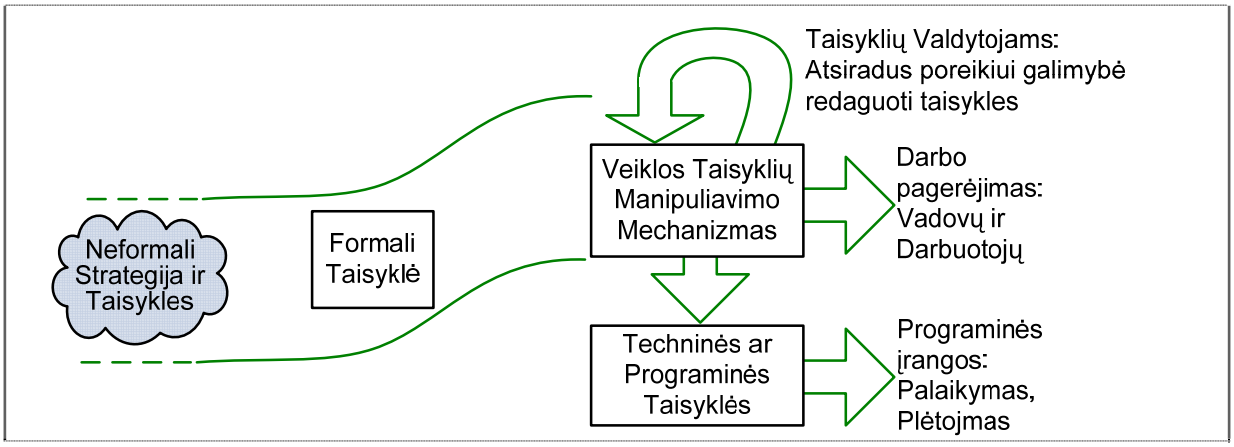
Tokios sistemos, šia diena, yra vadinamos veiklos taisyklių valdymo sistemomis (toliau VTVS). Taip pat, viena iš didžiausių problemų naudojant veiklos taisykles yra jų kiekis ir sunkiai nuspėjama tarpusavio veika. Taisyklėms veikiant kartu atsiranda įvairūs konfliktai, kai viena iš dviejų vykdomų taisyklių priima užduotį, o kita atmeta. Taigi VTVS leidžia lengviau rasti tokius konfliktus ir juos eliminuoti. Atsakymas į klausimą „kas?“ yra veiklos taisyklė. Veiklos taisyklėms valdyti yra sukurta įvairiausiu sistemų ir mechanizmų, taigi šiame darbe yra analizuojamos VTVS ir atsakymas į klausima „kaip?“ būtų Blaze Advisor ir kitos VT valdymo sistemos. Atsakymas į klausimą „kam?“ yra labai paprastas, tam kad automatizuoti visą organizacijos veiklą. Į klausimą „kodėl?“ atsakymų yra ganėtinai daug, bet pats pagrindinis būtų iš organizacijos pusės – keičiantis vidinei organizacijos logikai ar procesams – greitesnis ir lengvesnis veiklos taisyklės valdymas.

Šiame darbe yra pateikiami trys pagrindiniai skyriai. 2 skyriuje yra pateikta veiklos taisyklės sąvoka, analizuojamos veiklos taisyklių manipuliavimo mechanizmų architektūros. 3 skyriuje yra suformuluojama metodika kaip transformuojama duomenų struktūra į veiklos taisyklių aplinką, taip pat analizuojami *Blaze Advisor* įrankio komponentai ir sąveika su duomenų baze, pateikiami duomenų atvaizdavimo būdai. 4 skyriuje atliktas tyrimas sukurtai metodikai. 5 skyriuje įvertinama pasiūlyta metodika. Analitinės dalies skyriaus užbaigiamas analizės išvadomis, o darbo gale suformuojamos tyrimo išvados. Prieduose pateikiama sukurta mokomoji medžiaga darbui su *Blaze Advisor* įrankio komponentais.

# 2. VEIKLOS TAISYKLIŲ MANIPULIAVIMO MECHANIZMAI

## 2.1. Veiklos taisyklės (VT) sąvoka

Kaip profesorius Paul Harmon teigia: tai yra paprasta forma – teiginys užrašytas sakiniu, po kurio kažkas turi ar neturi būti atlikta. Šia prasme, taisyklės yra neginčijamos savo egzistavimu per tūkstantmečius. Dar ir ankstyvaisiais laikais, kai buvo pradėtas naudoti teisingumas (t.y teisminė diktatūra) – parodomas geras taisyklių panaudojimo pavyzdys. Taisyklėmis pagrįstas žinių atvaizdavimas ekspertinėse sistemose buvo panaudotas jau apie aštuntąjį dešimtmetį. Vokietijos matematikas ir filosofas Gottlobas Fregelis vienas iš pirmųjų savo darbe Begriffsschrift(1879) panaudojo matematinę logiką reprezentuojančia žinias pagrįstas taisyklėmis. Jis padėjo pamatą matematinės logikos sąvokoms: jei-tai, neigimas ir kintamųjų kvantifikavimas. Veiklos kryptis paprastai yra aprašyta bendra ir miglotais taisyklių sakiniais. Visos organizacijos priklauso nuo strategijos ir taisyklių aprašančių: tikslus ir uždavinius, kurie perduota pageidaujamą ar nepageidaujamą elgseną [11,14].




**2.1 pav.** Veiklos taisyklės formalizavimas.

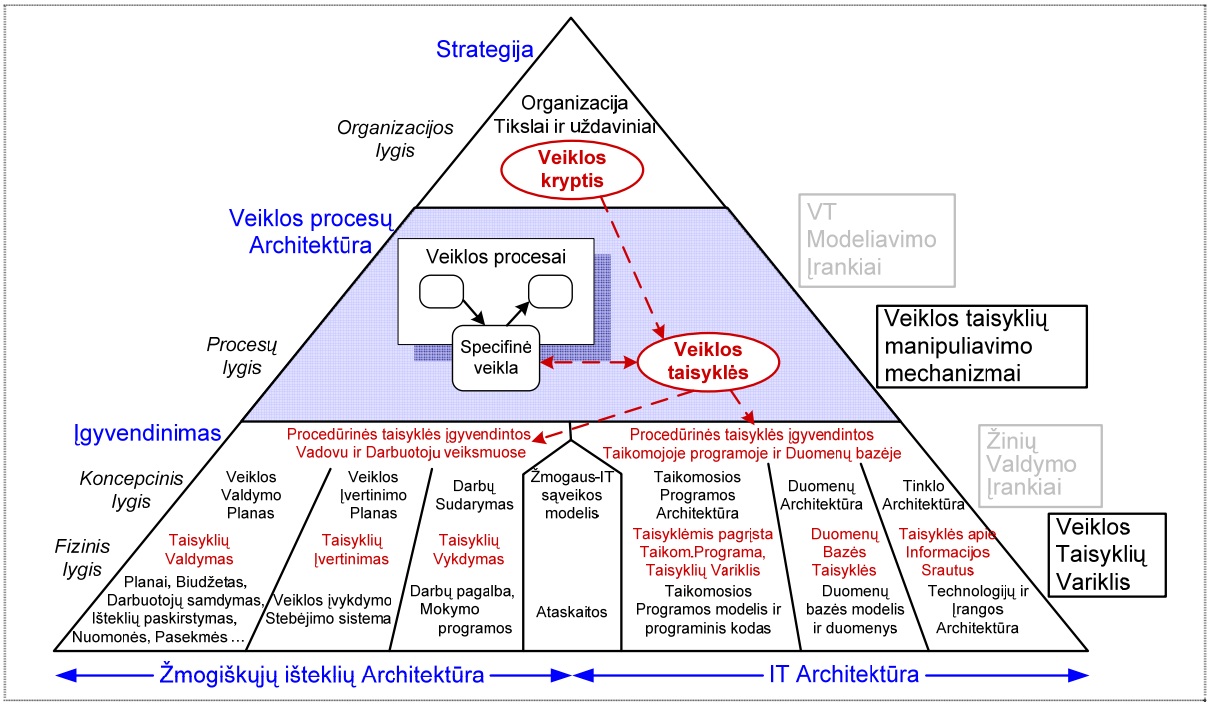
Sąlyginiai sakiniai jei-tai tapo populiarūs netik kompiuterių moksluose, tačiau ir loginiame programavime, ir taisyklėmis pagrįstose sistemose. Didžiąją dalį taisyklių, kurias kasdien vartojame savo kalbose yra laisvos-neformalios, tai yra naudojami terminai nėra gerai apibrėžiami. Kadangi nežinodami kaip tiksliai apibrėžiami terminai taisyklėse, tai tokiu atveju negalime šios taisyklės interpretuoti. Kad sistema galėtų interpretuoti taisykles, reikia užrašyti įprasta sistemai kalba (taisyklės formalizavimas sistemoje žr. 2.1 pav.). Tokiu atveju sistema galėtų reaguoti į sudėtingas situacijas visos sistemos veikimo eigoje. Taisyklės užrašomos susidedant iš prielaidų ir išvadų: „*jei situacija A, tai veiksmas B*“; „*jei prielaida X, tai išvada Y*“; „*jei sąlyga N, tai rezultatas M*“. Nors taisyklių sintaksė yra visada tokia pati, tačiau galima gauti skirtingas taisyklių interpretacijas. Skirtingos interpretacijos rodo įvairias galimybes kaip galima panaudoti taisykles. Jei sąlyginiai sakiniai (situacija, sąlyga) yra užrašyti sutartiniais sistemoje ženklais, tai tada tokios žinios gali būti apdirbtos kompiuteriu,

ir gautos naujos išvados (rezultatai ar veiksmai) gali būti panaudotos naujiems skaičiavimams. Kai kurių veiklos taisyklių pavyzdžiai apima [11,14]:

- Bendras taisykles, kurios naudojamos palaikyti visos organizacijos veiklos sritį.
- Specifines taisykles, kurios naudojamos prie tam tikrų specifinių situacijų.
- Kitas taisykles, kurios:
- Riboja transakcijas tarp objektų ar procesų.
- Apibrėžia ar riboja: sąryšius tarp objektų ar procesų; įvykius.
- Išgauna tam tikrus sprendimus ar faktus.
- Reikalingos patvirtinti/uždrausti tam tikrus veiksmus specifinėse situacijose.

## 2.2. Veiklos procesų architektūra

Šiame darbo skyrelyje norima parodyti, kurioje vietoje yra veiklos taisyklės – visoje organizacijos veikloje (žr. 2.2 pav.). Vienas iš svarbiausių faktų yra tas, kad veiklos taisyklės turėtų būti apibrėžtos, kai – apibrėžiamas procesas. Taisyklės procese apibūdina specifinius veiksmus sprendimų viduje, ir kaip vienas veiksmas susietas su kitu [11].




**2.2 pav.** Veiklos taisyklių naudojimas veiklos procesų architektūroje.

Veiklos taisyklės turėtų vienytis su organizacijos veiklos kryptimi, kuri yra apibrėžiama strategijos lygmenyje. Panašiai yra ir su procedūrinėmis ir techninėmis taisyklėmis, kai organizacija siekia įgyvendinti veiklos procesus, kurios turėtų būti priskirtos kartu su veiklos taisyklėmis – 3 ir 4 lygyje. Jei duotas procesas turi būti įgyvendintas darbuotojų ar vadovų, tai veiklos taisyklės turi būti įdiegtos mokymosi programose ar vadovėliuose, kuriais yra vadovaujamas įvykdymas. Jeigu yra automatizuotas procesas, tai veiklos taisyklės turi būti įdiegtos taikomojoje programoje arba duomenų bazėje [11].

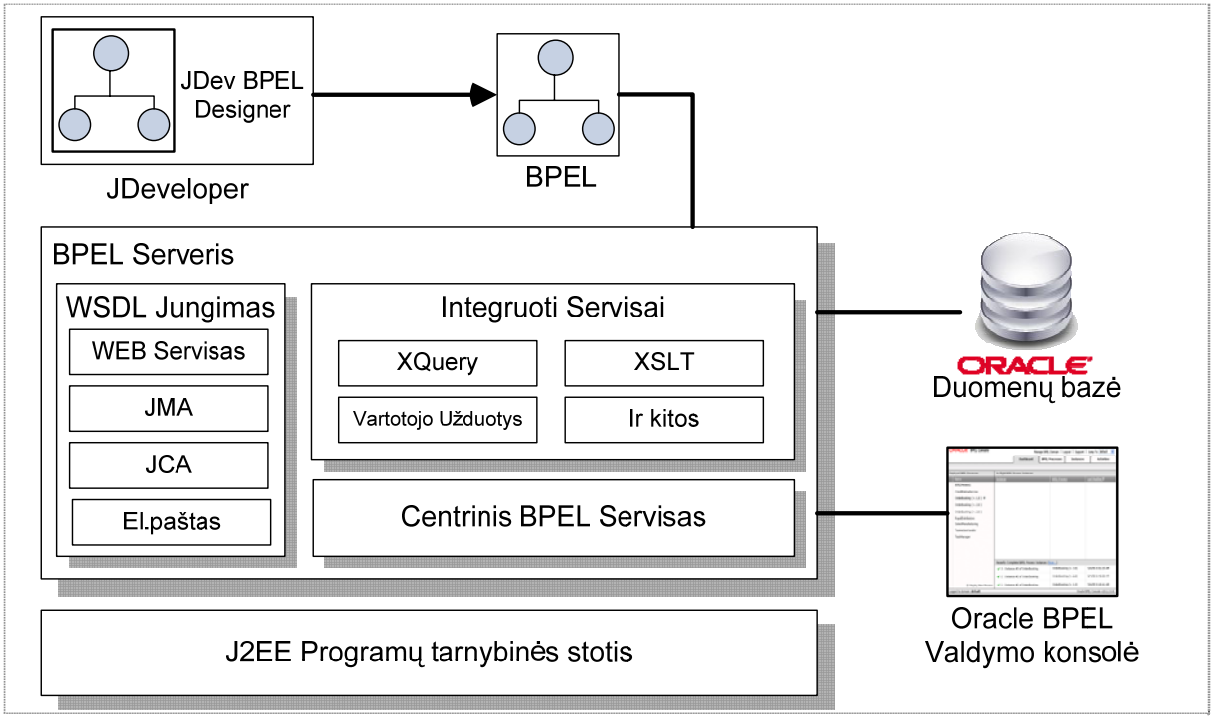
## 2.3. Darbų srautų valdymo varikliai

### 2.3.1. Oracle BPEL procesų valdymas

BPEL sutrumpinimo pažodinis vertimas iš anglų kalbos į lietuvių – verslo procesų vykdymo kalba (angl. *Business Process Execution Language* – BPEL). BPEL yra kaip standartas sudarytas iš daugialypių sinchroninių ir asinchroninių servisų – į bendrus ir transakcinius procesų srautus. Šio standarto moksliniai tyrinėjimai ir formavimasis vyko maždaug penkiolika metų tyrinėjant tokias pirmines kalbas kaip XLANG (yra apjungta su XML ir WSDL, angl. *Web Service Definition Language*) ir WSFL (angl. *Web Services Flow Language*). BPEL apibrėžiama kaip [13]:

- Web Services/WSDL kaip komponentų modelis;
- XML kaip duomenų modelis;
- Sinchroninių ir asinchroninių pranešimų keitimosi struktūra;
- Determinuotų ir nedeterminuotų srautų koordinavimas;
- Hierarchinis valdymas;

Oracle verslo procesų valdymo architektūra yra pagrįsta BPEL varikliu (architektūra žr.2.3 pav.), kuris aprūpina sistemos procesų projektavimą, išdėstymą, kontroliavimą ir valdymą pagrįstą BPEL standartais. Visa tai yra išvystoma Java principu, taigi kuriama sistema nėra pririšama prie konkrečios operacinės sistemos. Variklis susideda iš komponentų: *JDeveloper BPEL* projektavimo, *Eclipse BPEL* projektavimo įskiepių ir įdiegimo komponento, kuris yra ir serveris, ir valdymo komponentas – *Oracle BPEL Console* [7].

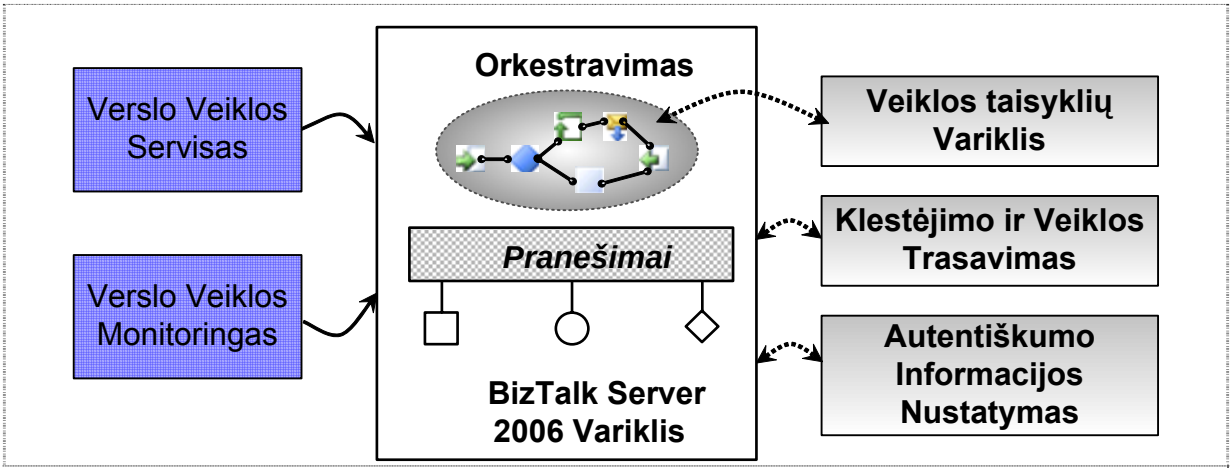



**2.3 pav.** Oracle BPEL procesų valdymo architektūra.

Projektavimo *JDeveloper BPEL* komponentas suteikią vartotojui patogią grafinę sąsają kurti BPEL procesus. Kas yra unikalu šiam komponentui, tai kad BPEL naudoją kaip savo gimtąjį formatą. Taip pat šis komponentas duodą galimybę peržiūrėti ir pakeisti BPEL šaltinį nemažinant įrankio naudingumo. *Oracle BPEL Console* aprūpima gerai apgalvotą internetinio puslapiu pagrįstą interfeisą, kurio pagalba *Oracle BPEL* serveryje yra valdomi, tvarkomi ir suderinami procesai. Darbo srautų procesai išvystomi keliais būdais: naudojant tiesiog BPEL scenarijus arba grafinį projektavimo interfeisą (pridėtas prie *JDeveloper* ar *Eclips*e programinės įrangos). Darbo srautai įgyvendinami panaudojant šią „drag-and-drop“ grafinę vartotojo sąsają, kuri yra perversta į BPEL skriptus. Taip pat leidžiama Java kodo fragmentus įterpti į darbo srautų vidų (naudojant <bpel:exec> veiklą). Audito tyrimai, procesų istorija ar ataskaitų informacija yra palaikoma automatiškai ir prieinama tiek per *Oracle BPEL Console*, tiek per Java taikomąją programą [7,13].

### 2.3.2. BizTalk serveris

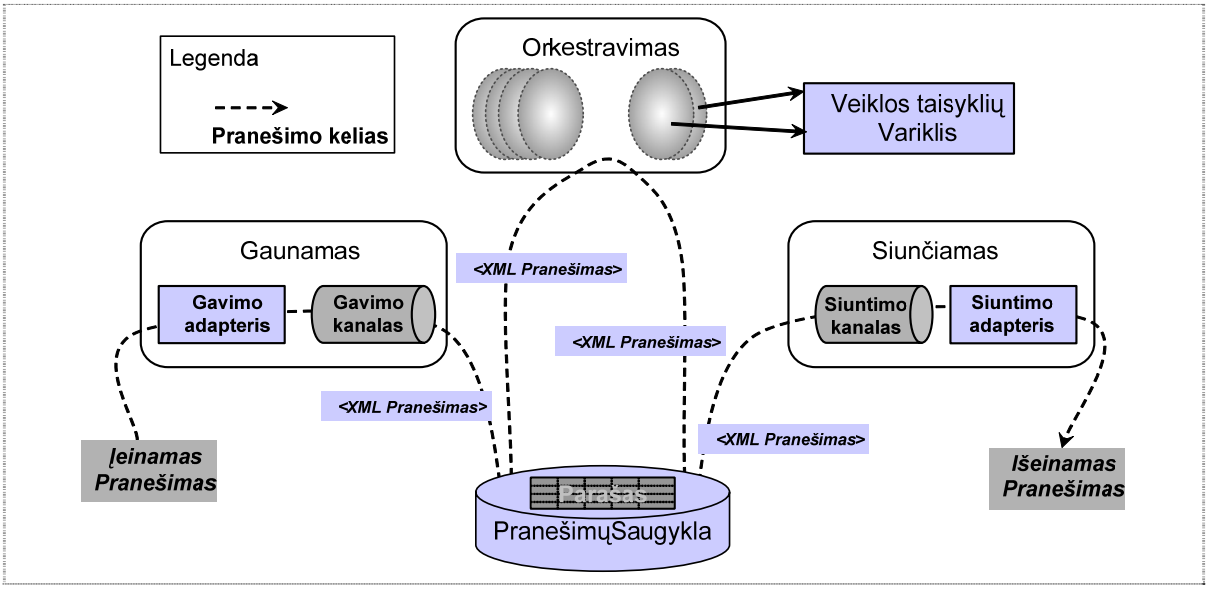
Microsoft BizTalk Server 2006 yra produktas skirtas komerciniai rinkai, ši versija yra jau ketvirtoji. Naudoja Microsoft .NET technologiją. Integruota su valdymo įrankiais, palaikančia SOA (angl. *Service Oriented Architecture*) paradigma, kurios naudoja Web tinklo serverį, WS-* specifikaciją ir veiklos kontroliavimo portalą. Servisų orkestravimas (angl. *Orchestration*, dažnai literatūros šaltiniuose dar vadinama kaip – servisų kompozicija) suteikia BizTalk serveriui sutelkti visą dėmesį ties sistema-su-sistema komunikacijai, kurios pagalba yra palaikomi verslo procesai, kai priklauso nuo skirtingos programinės įrangos sujungimo [7]. Variklio komponentai pavaizduoti 2.4 paveiksle [5].




**2.4 pav.** BizTalk Server 2006 variklio komponentai.

BizTalk Server viduje yra naudojama XLANG kalbą, kuri atvaizduoja darbo sekų konstravimą naudojant grafinę vartotojo sąsają. Palaiko XPATH (angl. XML Path Language) reiškinius, ir taip leidžiama .NET komponentams būti panaudotiems darbų sekų viduje. Programavimas vyksta .NET pagrindu, tačiau reikalingas ir XLANG supratimas [7].

Kad būtų leista vartotojams sukurti verslo procesą, kuris apima daugialypes taikomąsias programas, BizTalk Server 2006 variklis turi aprūpinti du pirminius dalykus: pirmas – apibrėžti ir realizuoti verslo proceso logiką; antra – mechanizmu palaikančiu ryšį tarp programos ir verslo proceso naudojimo. Paveikslas 2.5 iliustruoja svarbiausius variklio komponentus, kurie sprendžią šias dvi problemas [5].



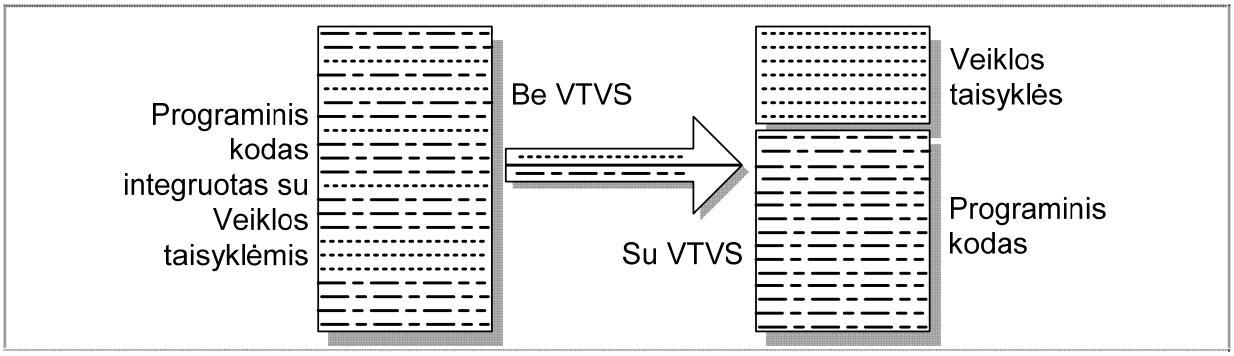

**2.5 pav.** BizTalk Server 2006 variklio architektūra.

Kaip pavaizduota diagramoje (žr. 2.5 pav.), žinutė yra gaunama per gavimo adapterį. Skirtingi adapteriai tiekia skirtingus komunikacijos mechanizmus, tokiu būdu žinutė galėtų būti gaunama tiek per interneto paslaugas, tiek nuskaitant iš duomenų failų ar kažkokiu kitu būdu. Tada žinutė yra apdirbama per gavimo kanalą. Šis kanalas gali turėti savyje įvairius komponentus, kurie žinutes perverčia iš paprasto formato į XML dokumentą, patvirtinant žinutę skaitmeniniu parašu ir kitokia informacija. Tada žinutė siunčiama į duomenų bazę, pavadintą *Pranešimų Saugykla*, kuri yra įgyvendinta panaudojant *MS SQL Server*. Ši logika taikoma verslo procesų įgyvendinimui su vienu ar daugiau orkestravimų, kur kiekvienas susideda iš įvykdymo kodų. Orkestravimas gali pasirinktinai panaudoti *Veiklos Taisyklių Variklį*, kuris verslo procesų vykdyme aprūpina paprastesnį ir lengvesnį būdą išreikšti sudėtingus taisyklių rinkinius. Kiekvienas orkestravimas kuria žinučių prenumeratą pagal tam tikrą rūšį, kurių jis nori gauti. Kai tinkama žinutė atvyksta į *Pranešimų Saugyklą*, tai ši žinutė yra išsisiunčiama laukiančiam orkestravimui, kuris atlieka veiksmus reikalingam verslo procesui įvykdyti. Apdirtas rezultatas tipiškai yra kita žinutė sukurta orkestravimo ir išsaugoma *Pranešimų Saugykloje.* Toliau žinutė siunčiama į siuntimo kanalą, kuris gali perversti iš XML formato (panaudoto BizTalk Server) į formatą reikalingą siunčiamu maršrutu, pridedant skaitmeninį parašą ar kitą informaciją. Tada žinutė yra išsiunčiama per siuntimo adapterį, kuris naudoja tinkamą mechanizmą susisiekimui su taikomąja programa, kuriai ši žinutė yra numatyta. [5].

## 2.4. Veiklos taisyklių valdymo sistemos (VTVS)

Augant organizacijos valdymo lygiui, taip pat didėjant veiklos taisyklių skaičiui, atsirado poreikis kurti tam specifikuotas sistemas. Šių sistemų kūrėjų požiūriai yra panašūs: į vieningą ir lengvesnį organizacijos veiklos valdymą. Tokia sistema yra vadinama – veiklos taisyklių valdymo sistema, toliau VTVS (angl. *Business Rule Management System* – BRMS). VTVS yra programinė įranga, skirta apibrėžti veiklos taisykles, padedanti išvystyti, valdyti ir palaikyti vidinę organizacijos logiką. Kitaip sakant sistemos panaudotos apibrėžti, išdėstyti, vykdyti ir valdyti veiklos taisykles, kurios yra naudojamos organizacijos ar verslo įmonės sistemose. VTVS yra kaip tiltas jungiantis verslą ir informacines technologijas, leidžianti analitikams kontroliuoti logiką ir net kodą [6,9,10,17].

Dažniausiai organizacijos valdymui sukurtose sistemose (kuriamos ne VTVS požiūriu) veiklos taisyklės yra turimos egzistuojančiame taikomajame kode ar procesų žinynuose, ar vadovuose. O žiūrint iš VTVS pusės: šios sistemos aprūpina reikalinga infrastruktūra tokia, kad vienoje vietoje yra išvystomos ir palaikomos veiklos taisyklės. Šios taisyklės yra kaip atskyra taikomosios programos dalis, kurią gali valdyti nebūtinai programuotojų grupė, o apmokytas organizacijos personalas. Pagrindinis sistemos kūrimo principas panaudojus VTVS požiūrį pavaizduotas 2.1 paveiksle.



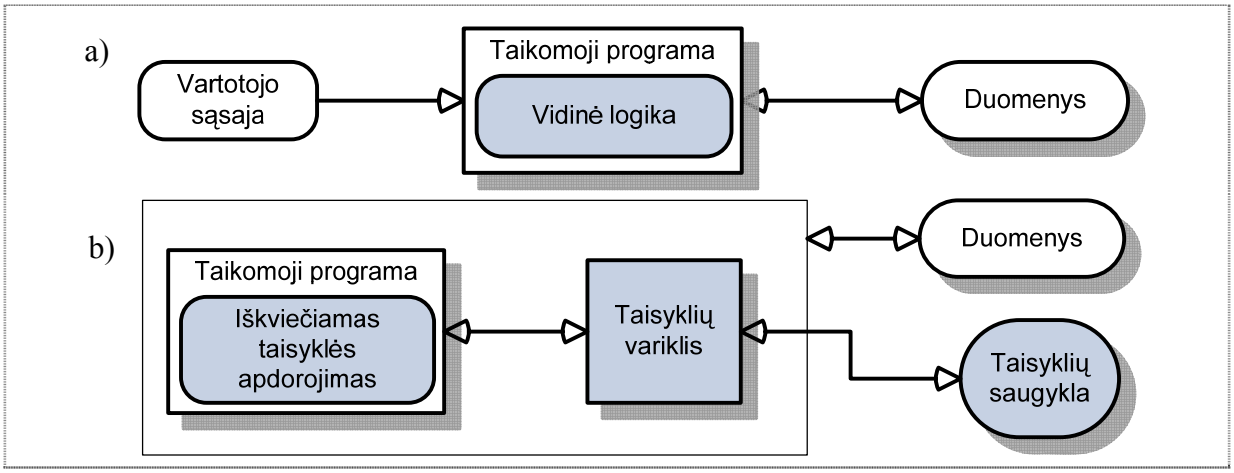

**2.6 pav.** Skirtumas tarp sistemų, kurios yra kuriamos VTVS pagrindu.

Taigi naudojant VTVS požiūrį išplečiamas kuriamų sistemų funkcionalumas. Dažniausiai VTVS sistemos turi savyje tokias pagrindines dalis [6,10]:

1. Taisyklių variklis (angl. *Rule Engine*), kuris aprūpina reikalingą infrastruktūrą iškviečiant veiklos taisykles.
2. Taisyklių saugykla (angl. *Rule Repository*), kurioje yra saugomos taisyklės t.y. taisyklių rinkiniai. Taisyklių saugykla naudojasi taisyklių variklis.
3. Taikomųjų programų kūrimo įrankis pagrįstas veiklos taisyklėmis, kuris palaiko grafinę vartotojo sąsają. Dauguma įrankių turi savyje įdiegtą programą ar tinklo serverį tam, kad būtų galima patikrinti išvystytas taisykles.

Įprastai veiklos taisyklės VTVS išvystomos ir užrašomos sąlyginių sakinių pagalba. VTVS yra mokslinių tyrinėjimų ir sistemų plėtojimosi rezultatas, kitaip sakant sprendimo

išvadų variklis, kuris yra nagrinėjamas kompiuterių mokslo dirbtinio intelekto sferoje. VTVS tūri aprūpinti taisyklių: redagavimu, saugojimu ir vykdymu. Sekančiame 2.7 paveiksle pavaizduotos dvi architektūros: kur *a* dalyje yra pavaizduota tradicinė, o *b* – šiuolaikinės sistemos panaudojus VTVS [9].



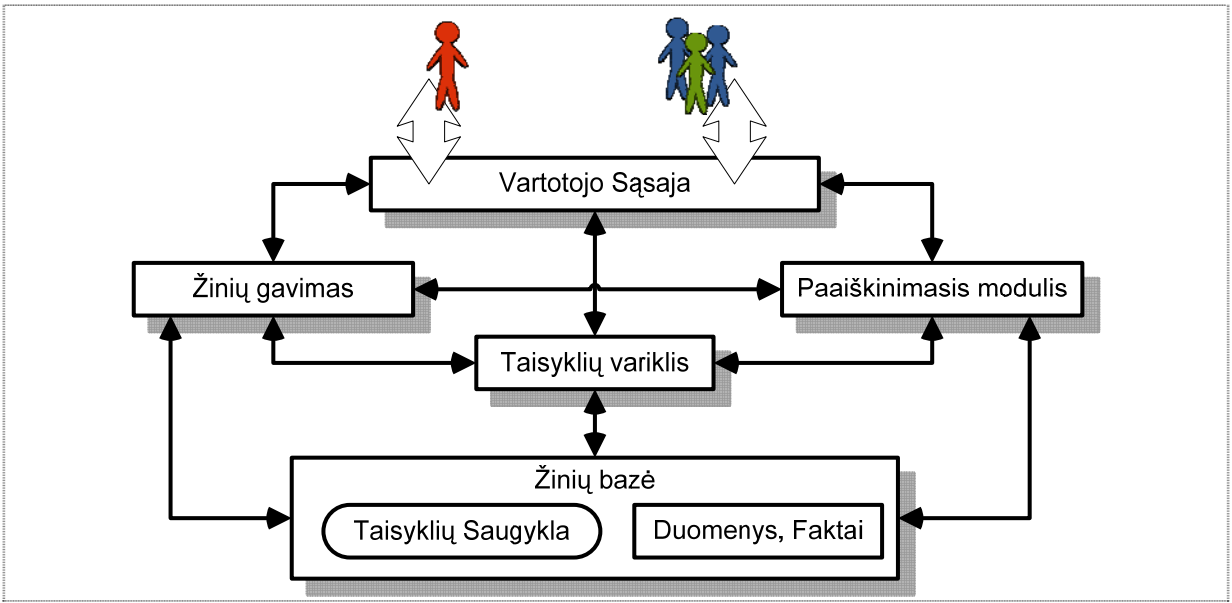

**2.7 pav.** Architektūra: a) tradicinė, b) panaudojus veiklos taisykles.

Kaip jau buvo minėtą anksčiau, pagrindinis VTVS tikslas yra kurti, valdyti ir palaikyti organizacijos struktūrą, taigi VTVS turi turėti tokias ypatybes, pareigas ir galimybes [9,10]:

- Veiklos taisyklių saugyklos kaupimas ir priežiūra, kurios pagalba yra valdoma organizacijos politiką, logiką ir procedūras. Veiklos taisyklių centralizavimas geriną nuoseklumą, neprieštaringumą ir sumažiną galimas dviprasmybes.
- Veiklos taisyklių saugyklos sujungimas su taikomosiomis programomis tam, kad taisyklės būtų panaudotos priimant visus sprendimus.
- Taisyklių formavimas į nepriklausomus taisyklių rinkinius ir išvadų gavimas jais pasinaudojant. Taisyklių užrašymo kalba yra artima gimtajai kalbai (šiuo atveju anglų kalbai) leidžianti analitikams keisti veiklos taisyklėms ir taip mažinant klaidų skaičių.
- Ne tik analitikams, tačiau ir organizacijos netechniniam personalui leidžiantys kurti suprantamas ir svarbiausias veiklos taisykles, su minimaliomis sistemos kūrimo žiniomis.
- Automatizuoti ir lengvinti vidinius ar išorinius organizacijos procesus.
- Lengvas taikomųjų programų kūrimas, taip pat vartotojui aiškūs, suprantami ir logiški taikomosios programos dialogai ir langai.

VTVS sistemos aprūpina vartotoją keletą įrankių, kurių pagalba yra valdomos taisyklės ir taisyklių rinkiniai: tekstinis ir grafinis redaktoriai yra kaip vartotojo grafinės sąsajos dalis. Taisyklės, ypač jų argumentai ar išvados, veikia duomenis. Paprastai duomenys kaupiami ir valdomi duomenų bazėje, kurios palaiko ODBC tvarkyklę, tokios kaip Oracle, MS SQL Server, MySQL ar Access. Dauguma sistemų naudoja taikomąjį serverį, kuris atskiria

vartotojų interfeisą, veiklos logika ir duomenų valdymą. VTVS sistema dažniausiai palaikoma keliais taikomaisiais serveriais. Kiti įrankiai aprūpina taisyklių testavimą ir tikrinimą ar versijos valdymą. Detalesnė VTVS architektūra pavaizduota 2.8 paveiksle [14].




**2.8 pav.** Detalesnė VTVS architektūra.

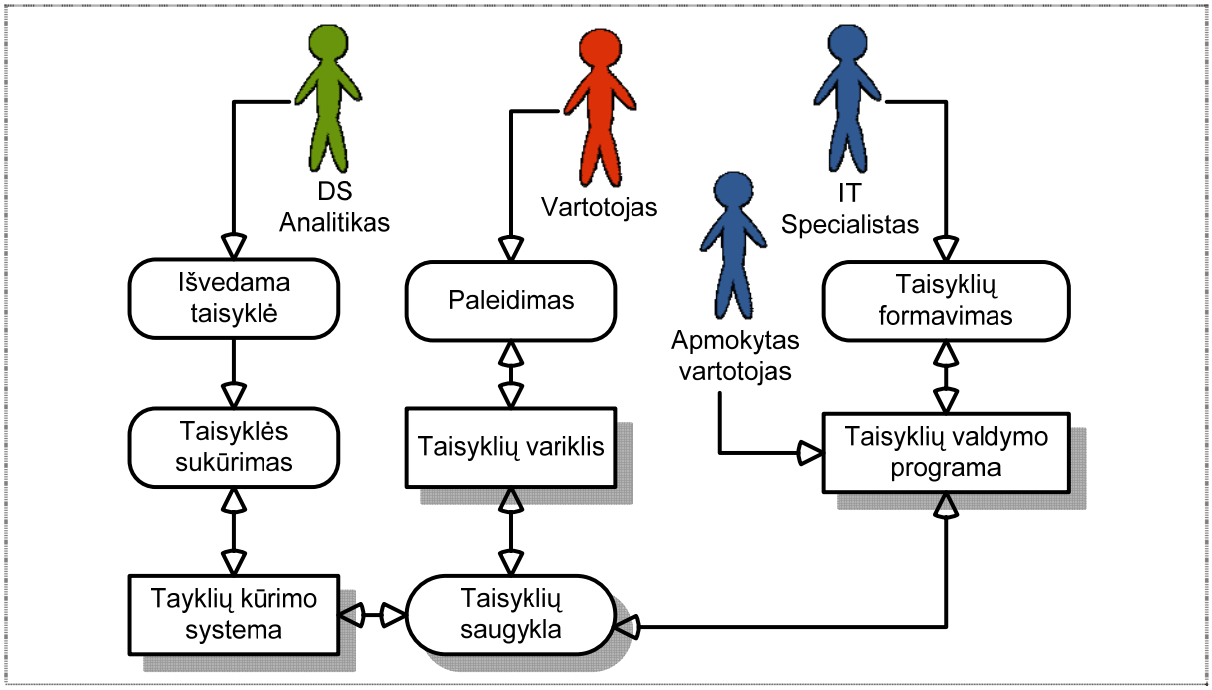
VTVS sistemos seka patyrusių ekspertinių sistemų keliu. Du skirtingi vartotojai bendrauja su taisyklėmis pagrįsta sistema. Taisyklių(žinių) inžinierius arba analitikas palaiko naujausias žinias sistemoje. Prideda naujas taisykles ir tikrinti taisyklių rinkinius. Tikrinimas yra labai svarbus: nes taisyklių rinkiniai neturi turėti savyje taisyklių, kurios priveda prie prieštaraujančių rezultatų. VTVS sistemos siūlo tam tikrus taisyklių tikrinimo įrankius, tam kad surastų prieštaravimus taisyklių rinkiniuose. Žinių ekspertas gali būti veiklos ekspertu, kuris tiesiogiai bendrauja su sistema. Kiti vartotojai naudojasi sistema – bendraudamas su klientais. Sistema tai pat gali būti panaudota kitos sistemos pavyzdžiui: transakcijų automatizavimas atliekant pirkimo užsakymus [14].

**2.1 lentelė.** Vaidmenys ir atsakingumas išvystant VTVS.

| **Aktoriai** | **Vaidmuo** |
|---|---|
| IT Specialistas | Kurti taisyklių formas.<br>Vystyti taisyklių valdymo sistemą (TVS). |
| DS Analitikas | Taisyklių išvedimas.<br>Taisyklių formavimas.<br>Taisyklių pakeitimas naudojant TVS. |
| Apmokytas vartotojas | Koordinuoti su įvairiais taisyklių savininkais ir nustatyti jų prioritetus ir įgyvendinti taisyklių pakeitimus.<br>Valdyti taisyklių saugyklą. |
| Vartotojas | Galutinis vartotojas kuris naudojasi taikomąja programa. |

Kadangi kuriamos VTVS sistemos yra ganėtinai nemažos ir apimančios didelius veiklos taisyklių srautus, tai svarbu nurodyti ir sistemoje veikiančius aktorius ir jų vaidmenis jose.

Taigi 2.1 lentelėje yra parodyti pagrindiniai aktoriai ir jų vaidmenys kuriant ir palaikant tokias VTVS sistemas, taip pat 2.9 paveiksle – tipinė veiklos forma [6].




**2.9 pav.** Tipinė VTVS veiklos forma.

Šioje darbo dalyje yra nagrinėjamas bendros VTVS būdingos ypatybės padedančios kurti, valdyti ir palaikyti organizacijos taikomąsias programas. Pateikiami bendri pagrindiniai ir svarbiausi komponentai būdingi VTVS. Taip pat nagrinėjami *Blaze Advisor* komponentai ir tiriama kaip VTVS sistema. Sekančiuose skyriuose peržiūrėsime būtinus šiai dienai VTVS komponentus.

### 2.4.1. Komponentai ir techninės VTVS ypatybės

Žinios apie tam tikrus priežastinius santykius (angl. *causal relationships*) yra paprastai kaupiamos tokiose taisyklių formose: *„jeigu [sąlyga(os)] tai [veiksmas(ai)]“* (angl. *if [condition(s)] then [actions(s)]*). Tokios taisyklės (sąlyga lygu veiksmas) dirba su žiniomis apie tam tikrus įvykius ar objektus. Dar vienas svarbus būdas nusakantis žinias yra procedūros. Yra ir kitokių būdų nusakyti žinias, tačiau taisyklės, procedūros ir objektai yra svarbiausi elementai panaudoti VTVS. Kuriant taisyklėmis pagrįstas taikomąsias programas 2.2 lentelėje pavaizduoti būdingi VTVS komponentai[9,10].

**2.2 lentelė.** VTVS komponentai

| **Komponentai** | **Apibūdinimas** | **Naudotojas** |
|---|---|---|
| Taisyklių užrašymo kalba | Naudojama išreikšti ir konstruoti veiklos taisykles. | Taisyklių variklis |
| Veiklos taisyklės | Veiklos logika išreikšta „jei (sąlyga) tai (veiksmai)“ forma sugrupuota į rinkinius (grupes). | DS Analitikas |
| Taisyklių valdymo programa | Naudoja dalykinės srities analitikai įvedant veiklos taisykles. | Taisyklių variklis |

| Komponentai | Apibūdinimas | Naudotojas |
|---|---|---|
| Taisyklių saugykla | Taisyklių atmintis, priėjimo valdymas. | Taisyklių variklis |
| Taisyklių variklis | Programinė įranga įvertina ir reaguoja veiklos taisyklėmis į informaciją pateiktą iš klientinės programos. | Taisyklių variklis |
| Klientinė programa | Atsakingas už sąveika su taisyklių varikliu, duomenų pateikimas varikliui, ir gautų sprendimų duomenų priėmimas iš variklio. | IT Specialistas |
| Programavimo objektas | Veiklos srities programavimo objektai (pvz.: Java objektai, pagristi Java kalbos programomis) | IT Specialistas |

Visuose VTVS produktuose, taisyklės užrašomos kaip sakiniai, paprastai turėdamos savyje žodžius *if* ir *then*. Vienas iš mokslininkų tyrinėjantis veiklos taisykles Tony Morgan 2002 metais rekomendavo patogesnį taisyklių užrašymo stilių: nutaikytą į dviprasmybės pašalinimą; į aiškų santykių sukūrimą, vengiant miglotos terminologijos; pašalinimą daug žodžiavimo; ir kiti. Mokslininko T.Morgan stilius yra nepaprastai artimas gimtajai anglų kalbai. Jis siūlo pereiti iš 1 į 2 taisyklių formos (žr. 2.3 lentelė) [10].

**2.3 lentelė.** Taisyklės formatas.

| Nr. | Taisyklė | Vertimas |
|---|---|---|
| 1 | A loan may be approved if<br>the status of the customer is high<br>and the loan is less than 2000<br>unless the customer has a low rating | Paskolai galima pritarti, jei kliento padėtis yra aukšta, ir paskola yra mažesnė negu 2000, tik tada kai klientas neturi žemo reitingo |
| 2 | if the customer status is high<br>and the loan is less than 2000<br>and the customer does not have a low rating<br>then approve the loan<br>if the customer status is high<br>and the loan is less than 2000<br>and the customer has a low rating<br>then don't approve the loan | jei kliento padėtis yra aukšta, ir paskola yra mažesnė negu 2000 ir klientas neturi žemo reitingo, tai pritarti paskolai,<br>jei kliento padėtis yra aukšta, ir paskola yra mažesnė negu 2000 ir klientas turi žemą reitingą, tai nepritarti paskolai |

Taisyklių šablonai (angl. *rule templates*) yra struktūrinė konstrukcija vienodoms taisyklėms kurti. Pagrindinis šablonų tikslas – apjungti taisykles su tokia pat struktūra tik skiriasi tam tikri duomenys. Taisyklių šablonų kūrimo metu yra atsižvelgiama į tam tikrus aspektus ir paliekami tušti laukai, kurie vėliau yra užpildomi reikalinga informacija. Taisyklių šablonas iš dalies atstovauja apibrėžtai veiklos taisyklei, kuri turi savyje tam tikrus laukus, kuriems būtina užpildyti trūkstama informacija. Šablonai gali būti panaudoti kuriant daugialypes taisykles su panašia struktūra, kur keičiasi tik užpildomų laukų reikšmes [10].

Vienas iš svarbiausių dalykų, tai kad VTVS tikrintų realiu laiku taisyklių sintaksę, tuo pačiu metu kada jos būna įvestos. Struktūrinė taisyklių kalba yra naudinga, jei sintaksės tikrinimo kontroleris pabrėžia reikšminius žodžius, kintamąjį ir vertes – panaudojant skirtingas spalvas. Turi būti aiški sąsaja tarp modelio objekto ir taisyklių. Tam tikros žinios

yra aiškiai procedūrinės, pavyzdžiui: negalime apskaičiuoti mokesčio įsiskolinimo, jei iš pradžių nežinome pajamų ir išlaidų. Taisyklės pavaizdavimas gali būti labai sunkus ir nepatogus, jei sukauptos žinios yra procedūrinės, pavyzdžiui: matematinį ar finansinį skaičiavimą. VTVS turi turėti taisyklių rinkinius, kuriuos iškviečia procedūros tam, kad atliktų skaičiavimus ir grąžintu reikšmes [10].

Taisyklių rinkiniai (angl. *ruleset*) yra taisyklių grupės, kurios turi tą pačią struktūrą ir yra įvertintos kaip rinkinys tame pačiame proceso sraute. Taisyklių srautai (angl. *ruleflow*) suteikia žingsnių įvykdymo seką, apibrėždami funkcijų ar taisyklių rinkinių įvykdymą kiekviename proceso sraute [10].

Sprendimo medžiai (angl. *decision trees*) ir sprendimo lentelės (angl. *decision tables*) padeda įsivaizduoti taisyklių „*if then*“ žinias. Sprendimo medžiai naudojami taisyklėms iliustruoti kaip medžio struktūra. Tai naudinga pagalba klaidų ieškojime ar komunikacijai tarp vartotojų, kūrėjų ar analitikų. Sprendimų medžiai padeda peržiūrėti taisyklių srautus, taip pat išvengiamas nereikalingų taisyklių vykdymas. Eidami reikalingo medžio šaka jie renkasi taisyklių kelią, ir taip sugreitėja sprendimų priėmimas. Sprendimo lentelės naudojamos toms pačioms žinioms ir taisyklėms pavaizduoti, tik rezultatai būna pateikti lentelių formate. Tačiau yra vienas sprendimo lentelės trūkumas, jei yra apibrėžiamas didelis kiekis sąlygų, tai tokiu atveju sprendimų lentelės tampa labai didelės. Toks užrašymas duoda didelį skaičių taisyklių vienoje vietoje, ir taip taisyklės bus sunkiai skaitomos ir suprantamos. Tačiau yra ir sprendimo lentelių pranašumas, kai organizacija jau laiko žinias tokiose formose kaip: kainų, tarifų ir kitas panašias struktūrines lenteles [10].

Sekančiuose skyreliuose yra analizuojama VTVS architektūra ir jos komponentai, šiai analizei atlikti yra nagrinėjamas *Blaze Advisor* įrankis. Pateiktuose [10] ir [3] literatūros šaltiniuose yra nagrinėjamos tokios VTVS sistemos ir jos galimybės.

### 2.4.2. Blaze Advisor

Viena iš lyderiaujančių pasaulinėje rinkoje kompanijų *Fair Isaac*(šiandien FICO) sukūrė ir toliau tobulina veiklos taisyklių valdymo sistemą pavadinta *Blaze Advisor.* Šį įrankį sudaro pagrindiniai veiklos taisyklėms įgyvendinti principai: sprendimų variklis, taisyklių saugykla ir jos valdymas. Įrankis skirtas kurti taikomąsias programas pagrįstas veiklos taisyklėmis, su integruota programų kūrimo aplinka (angl. *integrated develoment enviroment* – IDE). Šių kuriamų taikomųjų programų pagrindinis tikslas yra apimti tokias sritis kaip: draudimo analizės, paskolų svarstymas, kreditai paskolos, produktų rekomendavimas, klientų įvertinimas, apgavysčių nustatymas ir kitos probleminės sritys. Kurios gali būti išsprendžiamos pasinaudojant sąlyga-veikslas sakiniais (pvz.: jeigu metinės pajamos didesnės negu 100000, tada paskolos rizikos laipsnis mažas), taip pat sprendimų medžiais ir darbų

eigos logika. Šis įrankis yra išties efektyvus ir veiksmingas kuriant ir diegiant taikomąsias programas, tačiau painus ir sudėtingas pradedantiesiems. Kompanija *Fair Isaac* atliko didelį darbą paslepiant sudėtingumą su IDE aplinka, kuri gerai suprojektuota (angl. *well desgine*), kitaip tariant atitinką posaki: *tai ką matai atitinka tai ką gausi* (angl. *What You See Is What You Get* – WYSIWYG) [1].

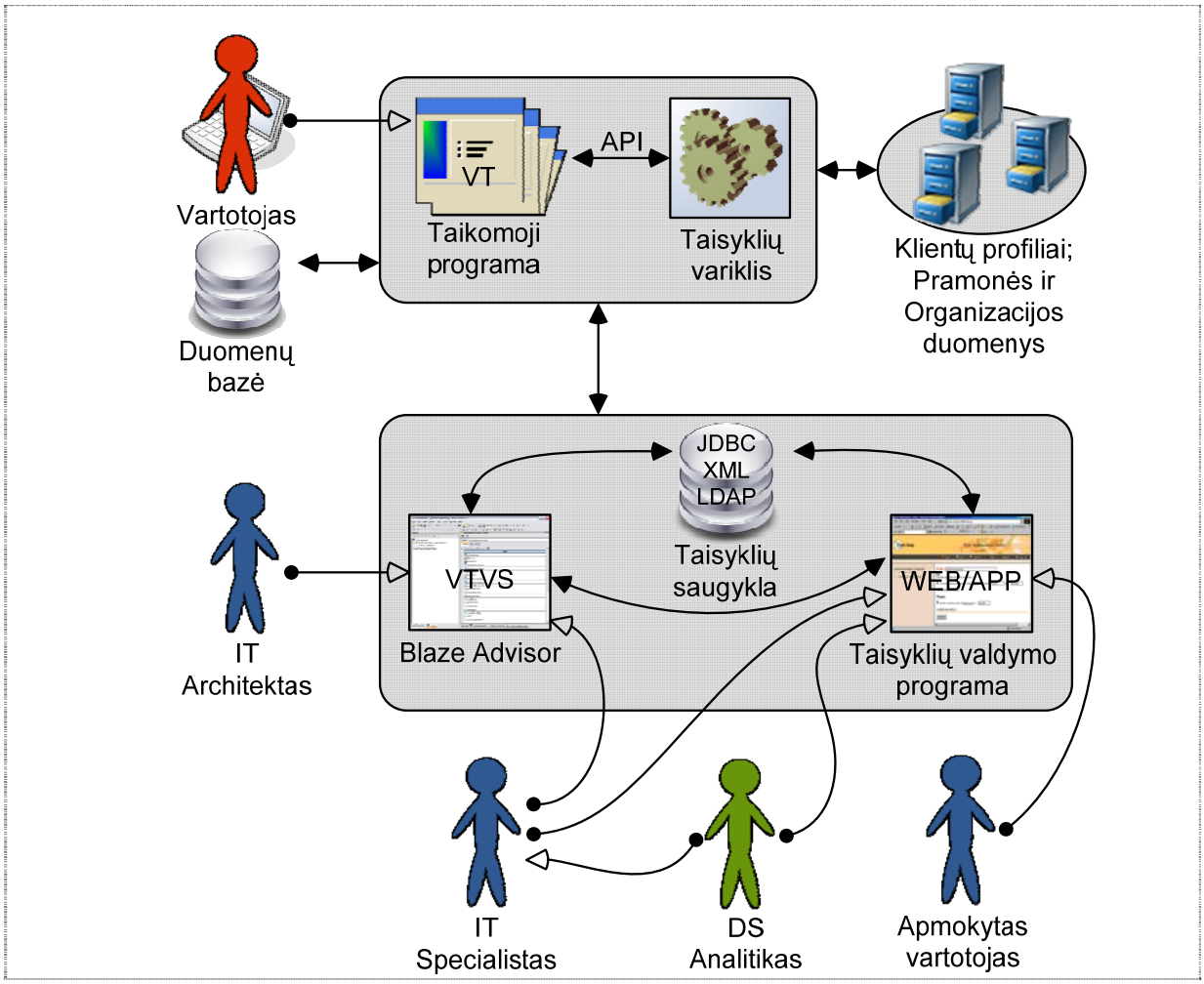
Šis įrankis tūri tris skirtingas panaudojimo galimybes: projektavimo aplinka, interfeiso kūrimo aplinka ir vartotojo orientuota taisyklių valdymo aplinka. Išdėstymo vedlys (angl. *wizard*) supaprastina šį procesą. Projektavimo aplinka *Blaze Builder* veikia *Microsoft Windows* ir *Linux Red Hat* sistemose. Kuriamos sistemos gali būti paleidžiamos beveik visose operacinėse sistemose, kuriose veikia JAVA virtuali mašina.

Programinis kodas atrodo lyg anglų kalbos sakiniai, tačiau tai yra tikra programavimo kalba. Apima sąlygas, veiksmus, medžių šakas, ciklus ir prieigą prie duomenų bazių. IDE turi pažengusią klaidų suradimo ir pašalinimo aplinką, kuri analizuoja ir aptinka taisyklių ciklus, taisykles be veiksmų, taisykles su ta pačia sąlyga ir skirtingais veiksmais, ir kitas potencialias anomalijas. Derinimo aplinka taip pat rodo taisyklės egzekucijos srautą, ir leidžia nustatyti sustojimo taškus ir vykdyti trasavimą, projekto verifikavimą taip pat galima atlikti taisyklių testavimą ir teisingumo tikrinimą pasinaudojus moduliu *brUnit*. Šie IDE įrankiai yra pažįstami daugumai programuotojų, kurie suranda patrauklią ir tinkamą vartoti regimąją derinimo aplinką.

Kaip jau buvo minėta įrankio programinis kodas yra panašus į anglų kalbą, nes naudojamos tokios intuityvios frazės kaip: „*satisfy*“, „*does not start with*“, „*of every*“ ir „*is any*“. Taisyklės turi ir į anglų kalbą panašią sintaksę: *„if at least two children satisfy (age < 10), then set discount to 5"* arba *„SeniorCustomer is any customer such that (age > 65)"*. Tačiau tokio kodo aspektai nėra tai, kas daro kalbą tinkamą verslo žmonėms. Nes jeigu verslo žmogus nebus susipažinęs su kodiniu testo gyvybės raidos ciklu, tai neleis į pilną išsivystymo lygį. Kad būtų pilnas taisyklių saugyklos valdymas įrankyje yra sukurta taisyklių valdymo programa toliau TVP (angl. *Rules Maintenance Applicatio* – RMA). Taisyklės ir jų dalys (tokios kaip jungiamieji žodžiai ir kintamieji) verslo žmonėms yra kaip parametrai ir sąlygos, ir tai šablonuose iš anksto yra apibrėžiama. Tokiu būdu yra suvaržomos netechninio vartotojo tiesioginio valdymo prieigos. Sprendimo lentelės ir sprendimo medžiai padeda vartotojams įsivaizduoti taisykles ir darbo eigą [1].

Taigi sprendimo variklių pagrindinis tikslas yra pakeisti ir iš naujo išdėstyti veiklos taisykles, kurios yra organizacijos operacijų širdis. Tačiau yra posakiai: mūsų jėgos yra dažnai mūsų silpnybė. Kitaip sakant, lengvas prototipų kūrimas gali iššaukti priežastį: sunkus programos palaikymas. Sudėtingiems taisyklių srautams reikalingas atsargus išbandymas,

kitaip tariant – testavimas. Verslo vartotojai turi prieigą prie meistriškai padarytos šablonų priemonės, kurios pagalba galima greičiau ir patogiau valdyti organizacijos vidinę logiką.



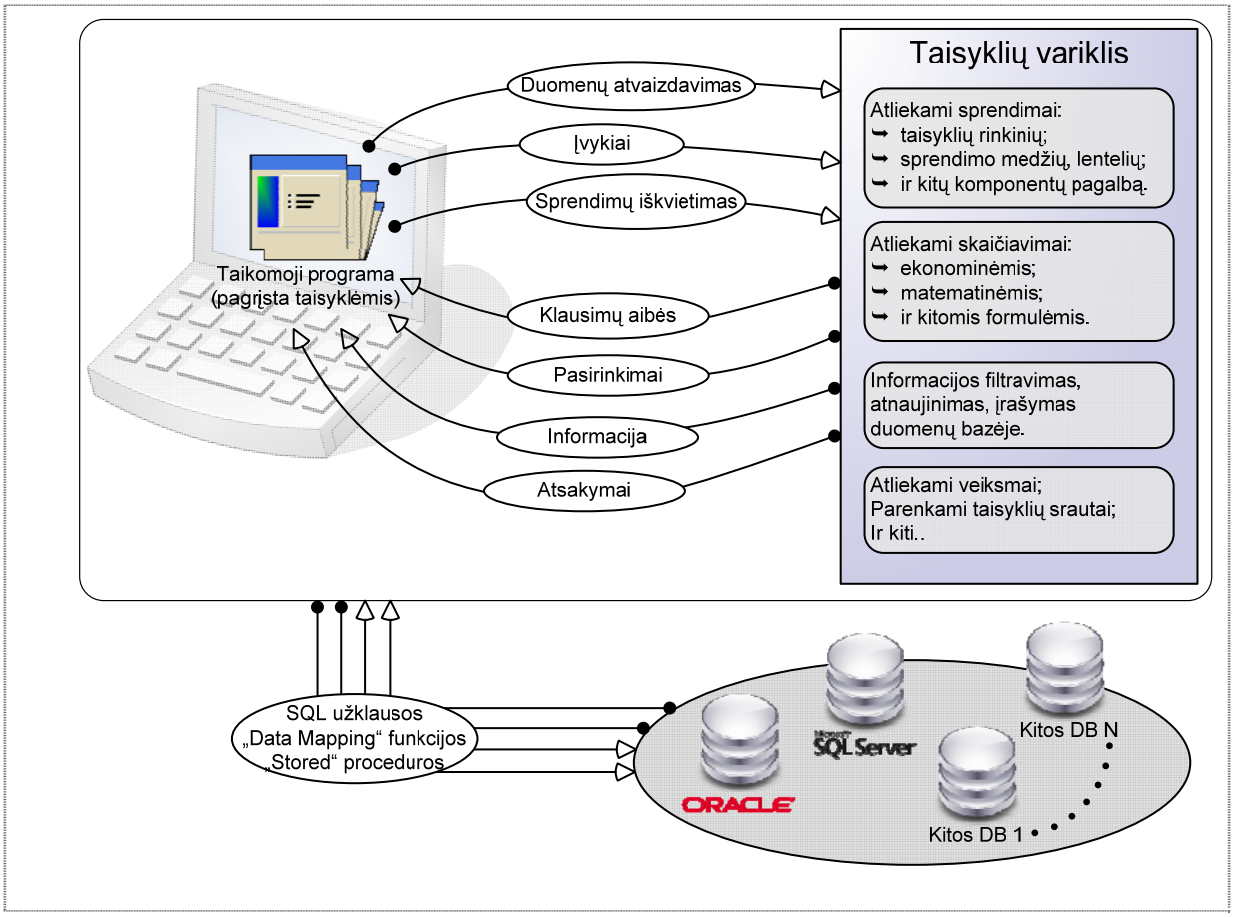

**2.10 pav.** Architektūra ir ją valdantys pagrindiniai aktoriai.

Toliau apžvelgsime *Blaze Advisor* įrankio architektūra, kuri pavaizduota 2.4 paveiksle. Pateikiamas trumpas įrankio aprašymas [8]:

- Taikomoji programa paprašo taisyklių variklio, kad įvykdytų tam tikrą paslaugą (pvz.: verslo vertinimas, situacijos klasifikavimas, problemos diagnozavimas, rekomenduotų veiksmą, ar patvirtintų duomenis).
- Kai būtina įvykdyti paslaugą, tada komponentas prisijungia prie bet kurio duomenų šaltinio, ir gauna reikalingą informaciją.
- Taisyklės kurios yra valdomos taisyklių variklio yra saugomos taisyklių saugykloje, kuri aprūpiną einamą serverį. Taisyklių saugykloje yra naujausios taisyklės, kurios yra atnaujinamos taisyklių valdymo programa.
- Įrankio IDE aplinka yra panaudota techniniam paslaugos sprendimo apibrėžimui. IDE yra skirta taisyklių kūrimui, taisyklių testavimui, projektuoti taisyklėmis pagrįstas paslaugas, generuoti taisyklių valdymo programas, naudotis taisyklių šablonais.

- Taisyklių palaikymo programa leidžia kontroliuoti taisyklių redagavimą ir sukūrimą netechniniam personalui. Tai vyksta per standartines tinklo technologijas (angl. *standard web technologies*). Sukurtos ir suredaguotos taisyklės yra kaupiamos toje pačioje taisyklių saugykloje.

Paraiškos iš taikomosios programos inicializuoja taisyklių serverio priėmimą. Paraiškos gali būti netik kaip duomenų bazės atnaujinimo įvykiai, bet ir paraiškos iš kitų taisyklių serverių. Kai iškviečiamas taisyklių variklis, gali reikėti įvykdyti kelis reikalingus įvykius, tokius kaip: duomenų bazės filtravimas ar duomenų skaičiavimas ir gražinamos vertės į objekto atmintį; duomenų gavimas iš kelių objektų (pagrįstas viena lentele ar duomenų bazės vaizdu (angl. *view*)), kurie yra naudojami taisyklėms ir įvykdomi taisyklių variklyje; dinaminių (laikinų) objektų sukūrimas, kuris gali būti panaudotas taisyklių serveriui apdoroti rezultatus ir kiti. Prie įrankio galima prisijungti ne vieną, o kelias duomenų bazes. Minimali duomenų gavimo schema parodyta 2.11 paveiksle.




**2.11 pav.** Informacijos gavimo srautai.

**Struktūrinė taisyklių kalba.** *Blaze Advisor* VTVS aprūpina galingą taisyklių atvaizdavimo sintakse pavadinta struktūrinė taisyklių kalba (*Structured Rule Language – SRL*). SRL sintaksė yra formali ir arti natūralios kalbos (arti anglų kalbos reikšminių žodžių), kuri leidžia kompaktiškiems ir išplėstiems taisyklių atvaizdavimo stiliams [8,18].

Paimkime paprasta veiklos taisyklę, išreikšta trimis skirtingais SRL būdais (žr. 2.3 lentelę). Visi trys formatai yra galiojantys, jie yra funkciškai ekvivalentiški, ir nėra jokio skirtumo tarp jų atliekant taisyklių variklyje. Taisyklių kūrimo autoriai gali apjungti šiuos tris sintaksės stilius taip kaip jiems yra patogiau [8].

**2.4 lentelė.** Trys skirtingi taisyklės užrašymo būdai įrankyje.

| Būdas | Taisyklė |
|---|---|
| 1 | **If** *Užsakovas.skola* **>** *Užsakovas.lėšos*<br>**then** *Užsakovas.prašymas.statusas* = Nepriimti. |
| 2 | **If the** *skola* **of the** *Užsakovas* **is greater than the** *lėšos* **of the** *Užsakovas*<br>**then the** *statusas* **of the** *prašymas* **of the** *Užsakovas* **is** Nepriimti. |
| 3 | **If** *Užsakovas***'s** *skola* **exceeds** *Užsakovas***'s** *lėšos*<br>**then set the** *statusas* **of** *Užsakovas***'s** *prašymas* **to** Nepriimti. |

Tokie terminai kaip „*is greater than*“ ir „*exceeds*“ yra apibrėžti iš anksto SRL kalbos viduje. Yra daugiau kaip 100 reikšminių žodžių ir kalbos frazių apibendrinančių taisyklės apibrėžimą (suprantamai kaip natūralią ir intuityvią kalbą). Specialios kalbos konstrukcijos skirtos, tam kad naudojant *string*, *currencies*, *dates* tipo kintamuosius, trūkstamus duomenis ir operacijas būtų apibrėžiama struktūrą ar parametru. Jeigu reikia atlikti sudėtingesnius procedūrinius skaičiavimus taisyklės viduje, tai leidžia SRL kalba t.y taisyklės viduje atlikti reikalingus skaičiavimus. Tai leidžia taisyklėms likti nepriklausomoms nuo jų vykdymo aplinkos, tokiu atveju vartotojas negalėtų atlikti pakeitimų įvedant savo sprendimų logiką [8].

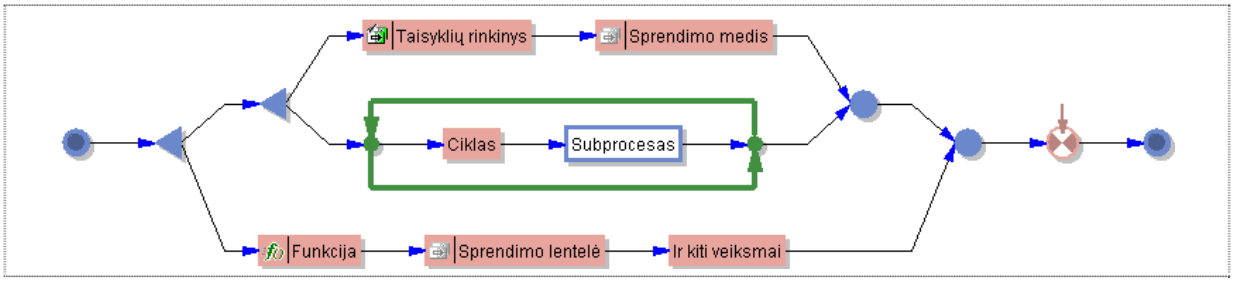
SRL kalba yra susieta su XML failų formatu, tai yra taisyklės, kurios yra užrašomos įrankyje yra saugomos XML dokumentu. Ši struktūrinė taisyklių užrašymo kalba leidžia apibrėžti tokias esybes [18,16]:

1. taisykles jungti į taisyklių rinkiniuos;
2. šablonas „*pattern*“ apibrėžia apribojimų rinkinius priklausomai nuo egzistuojančių duomenų;
3. įvykius – sąlyga-veiksmas taisykles, veikiantis taisyklių variklio kontekste arba einamajame taisyklių rinkinyje;
4. funkcijas: procedūriniai algoritmai ar iš bibliotekų;
5. klases, objektus, sąrašus, kintamuosius ir kiti.

**Taisyklių grupavimas.** Organizacijos gali turėti daug taisyklių, atspindinčių įvairių skirtingų sprendimų tipų. Ganėtinai dažnai galima identifikuoti taisyklių poaibius, kurie prisideda prie tam tikro sprendimo ar apimančią visą proceso dalį, bet tai tiesiogiai nepaveikia kitų programos sprendimų priėmimo dalių. Įrankis leidžia sugrupuoti tokias taisykles į vadinamuosius taisyklių rinkinius (angl. *ruleset*). Kompanijos viduje funkcinės grupės gali dirbti ties joms „priklausančiomis“ taisyklėmis, gaudamos prieigą tik prie tam tikro taisyklių rinkinio. Tokiu būdu galima greitai surasti ir redaguoti taisykles, apimančias tam tikrą

funkciją, nustatant taisyklių rinkinio vietą. Taisykles ir kitus sprendimo proceso komponentus galima sugrupuoti ir sukaupti loginiuose aplankuose ar fizinėse rinkmenose [8].

**Taisyklių srautų sudarymas.** Sistema turi leisti nustatyti aiškius taisyklių įvykdymo prioritetus einant tam tikru keliu. Tokia galimybė šiame įrankyje yra pavadinta taisyklių srautais (angl. *ruleflow*), kuri apibrėžia žingsnis po žingsnio operacijų seka. Taisyklių srautai aiškiai apibrėžia apdorojimo kelią. Jie leidžia taisyklių rinkiniams būti atliktiems kaip subprocesai, su kiekvienu įvykdymu atliekamos tam tikros užduotys, ir taip apimamas net visas sprendimas. Kaip taisyklių rinkiniai turi būti atlikti priklauso nuo tam tikro proceso žingsnio, tai gali būti statiškai apibrėžtas, ar net nustatyti sąlygomis iš ankstesnių užduoties žingsnių, ar net prie kažkokio atskiro atvejo. Todėl taisyklių srautai turi palaikyti šakojimąsi ar grįžtamą ryšį, vidaus ir išorinį atvejo kontroliavimą, ir užduotis, kurios apima ne tiktai taisyklių rinkinius, bet taip pat ir funkciniai iškvietimai į išorines sistemas ir analitinius modelius [8]. Galimas, tačiau ne konkretus taisyklių srautas įrankyje parodytas 2.12 paveiksle.




**2.12 pav.** Taisyklių srauto pavyzdys.

**Sprendimų lentelės.** Daugelis organizacijų įgyvendina pagrindinius sprendimus vartojant lenteles. Tai ypač teisinga, jeigu veiklos taisyklės yra pagrįstos mažu skaičiumi sąlygų: su kiekviena taisykle atitinkančia tam tikrą duomenų reikšmių kombinacija.

| RizikosLaipsnis | didelis | nedidelis | mazas | didelis | nedidelis |
|---|---|---|---|---|---|
| Pajamos | 25.000 <= ..< 35.000 | 25.000 <= ..< 35.000 | 25.000 <= ..< 35.000 | 35.000 <= ..< 45.000 | 35.000 <= ..< 45.000 |
| Limitas | 2.500 | 3.500 | 4.000 | 3.000 | 3.500 |

Eilutėmis

| Pajamos | Narys | Limitas |
|---|---|---|
| 25.000 <= ..< 35.000 | Brondzinis Studentas | 2.500 |
| 25.000 <= ..< 35.000 | Sidabrinis Studentas | 3.000 |
| 25.000 <= ..< 35.000 | Auksinis Studentas | 4.000 |
| 35.000 <= ..< 45.000 | Brondzinis Studentas | 3.000 |
| 35.000 <= ..< 45.000 | Sidabrinis Studentas | 3.500 |
| 35.000 <= ..< 45.000 | Auksinis Studentas | 4.500 |
| >= 45.000 | Brondzinis Studentas | 4.200 |
| >= 45.000 | Sidabrinis Studentas | 4.700 |
| >= 45.000 | Auksinis Studentas | 5.200 |

Stulpeliais

Eilutėmis ir stulpeliais

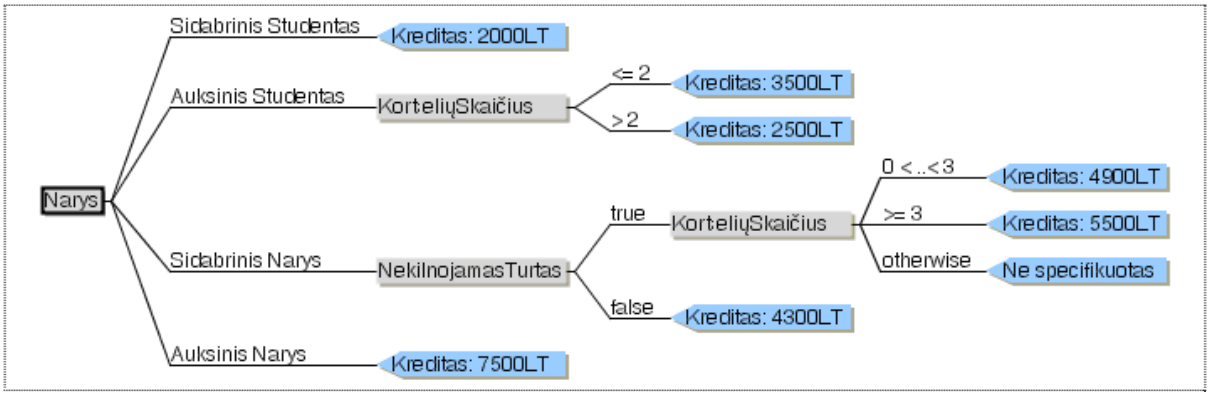
| Narys | Brindzinis Stud... | Sidrabrinis Stud... | Auksinis Stud... |
|---|---|---|---|
| RizikosLaipsnis | didelis | nedidelis | mažas |
| NekilnojamasTurtas | false | true | true |
| Pajamos | Limitas | Limitas | Limitas |
| 25.000 <= ..< 35.000 | 2.500 | 3.000 | 4.000 |
| 35.000 <= ..< 45.000 | 3.000 | 3.500 | 4.500 |
| >= 45.000 | 4.200 | 4.700 | 5.200 |

**2.13 pav.** Sprendimų lentelių pavyzdžiai.

Įrankis palaiko sprendimo lenteles kaip būdą apibrėžti ir palaikyti taisykles kompaktiškame formate. Projektuotojai gali nuspręsti kaip rodyti savo lentelių formatų įvairovę. Apimdami vienguba ar dvigubą ašį, daugialypes sąlygų eiles, prie vienos reikšmės ar

prie kelių reikšmių diapazono. Galimas lentelių spalvų ir šriftų keitimas. Ląstelių turinys gali būti grąžinamas paprastomis reikšmėmis arba sudėtingesnių vidinių SRL sintaksės leistinų veiksmų rezultatas [8]. Sprendimų lentelių pavyzdžiai pavaizduoti 2.13 paveiksle.

**Sprendimų medžiai.** Sprendimų medžiai leidžia surasti grandinę sąlygų sueinančių į vieną tinkamą veiksmą ar rezultatą, elementarus pavyzdys pateikiamas 2.8 paveiksle. Peržiūrėjus šakas nuo sprendimo medžio pradžios (viršūnės), galima patvirtinti ar visos taikomos galimybės buvo sudarytos. Įrankis palaiko lanksčius sprendimų medžius, kurie apibrėžią, bet kokio dydžio šakų lygmenų skaičių ir sąlygų. Kaip ir sprendimų lentelėse, taip ir sprendimų medžiuose mazgai gali būti apibrėžti kaip grąžinamomis reikšmėmis ar sudėtingesnių taisyklių kalbos konstrukcijų rezultatas. Medžių redaktorius leidžia ant bet kokios medžio šakos ar mazgo dinaminį sąlygų ir veiksmų atnaujinimą [8].




**2.14 pav.** Sprendimų medžio pavyzdys.

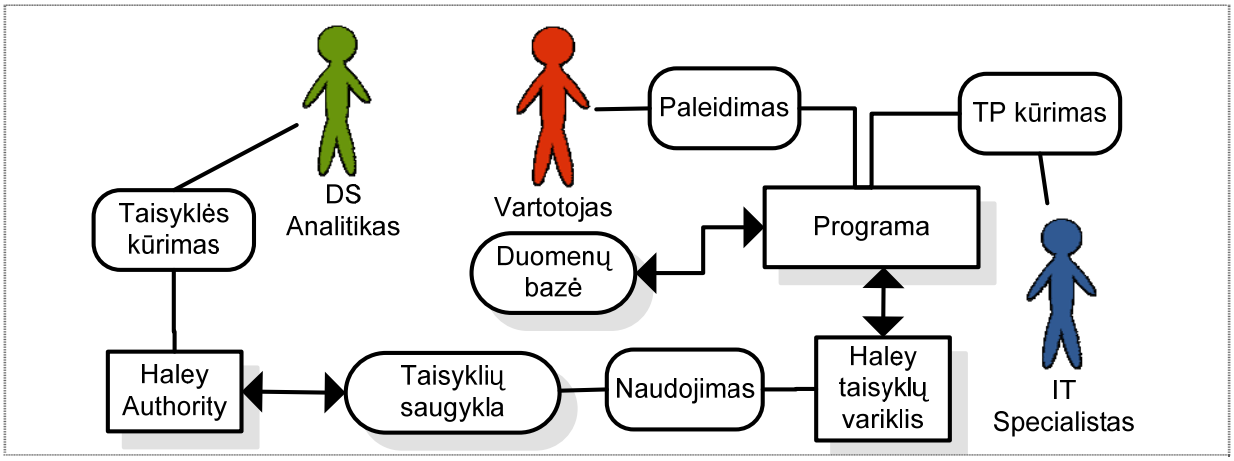
**Taisyklių šablonai.** Šablonų pagalba netechniniams vartotojams yra parodomos taisyklės, kuriuose yra galimas reikšmių pakeitimas. Šių šablonu pagalba vartotojai galėtų panaikinti aktyvaus naudojimo taisykles, taip pat sukurti naujas taisykles. Bet jie neturėtų teisės imtis tam tikrų pašalinių kenkėjiškų veiksmų: į sistemą įvedant taisykles kurios veikia neteisingą informaciją. Yra sukurta tokia apsaugojimo struktūra pavadinta taisyklių šablonais (angl. *rule templates*). Yra sukuriamos taisyklės su tam tikrais tuščiais laukeliais, kuriuos yra būtina užpildyti. Taisyklių kūrėjai gali sukurti daug skirtingų šablonų taisyklių įvairioms rūšims. Kai tik šablonas yra sukurtas, sekančiame žingsnyje yra įvedamas šablono savininkas (angl. *value holder*). Sistemoje leidžiama taisyklių kūrėjui sukurti kontrolę ant kiekvienos lauko reikšmės. Kai kurie laukai gali būti apibrėžti kaip statiniai sąrašai, kurie yra įkeliami iš duomenų bazes ar iš duomenų šaltinio [8].

**Kitos įrankio ypatybės.** Įverčiu modelis (angl. *scorecard*) duoda lengvą būdą sukombinuoti didelį skaičių faktorių į vieną apimantį matavimą. Dažnai taisyklės turi savo gyvenimo trukmę, tai įrankyje galima įvesti taisyklių galiojimo laiką. Taisyklių rinkinių viduje, retkarčiais pageidaujama garantuoti, kad kai kurios taisyklės įvyktų anksčiau negu kitos. Tokiu būdu įrankyje galima nustatyti taisyklės prioritetą (angl. *priorities*). Taisyklių

prioritetai gali būti nustatyti kiekvienai taisyklei, leidžiančiai pirmumo teisią viename taisyklių rinkinyje. Įvykiai (angl. *events*) – specifinė taisyklių operacija, kuri gali būti iškviečiama, kai keičiasi informacija ar naujiems duomenims išgauti. Klausimų aibė (angl. *question sets*) – skirta gauti duomenims iš vartotojo, kuris užpildo laukelius atsakydamas į klausimų aibę priklausiančia vienai ar kitai klausimų grupei. Įrankis leidžia susigeneruoti taisyklių valdymo programą toliau TVP (angl. *Rule Maintenance Application* – RMA). TVP taisyklių valdymas yra labai paprastas, skirtas labiau ne techniniam personalui. Pirmiausia sukuriami taisyklių šablonai kurie yra naudojami ateityje [8].

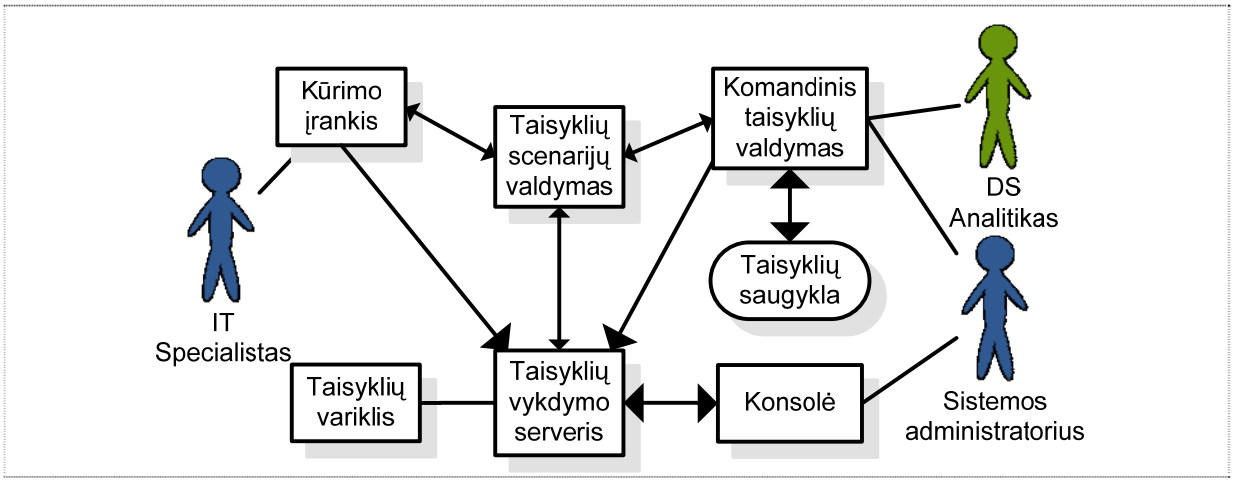
### 2.4.3. Kitos VTVS

HaleyAuthority yra taisyklių kūrimo ir žinių valdymo sistema, kuri generuoja kodą HaleyRules taisyklių varikliui. Taisyklių variklis naudoja taisyklių kalbą, pavadintą Eclipse, kuri kilusi iš CLIPS ir OPS-5 kalbų. Tačiau nėra jokio ryšio su IBM kompanijos Eclipse IDE. Įrankis veikia Windows aplinkoje, tačiau turi internetinės sąsajos komponentus. Taisyklių variklis yra palaikomas tiek Javos, tiek C pagrindu, turintis „out-of-the-box“ sąsaja su galimybę integruoti Java, .NET, C++ ir Visual Basic taikomąsias programas. HaleyAuthority architektūra parodyta 2.15 paveiksle [10].




**2.15 pav.** HaleyAuthority architektūra.

ILOG JRules duoda galimybę organizacijoje pagreitinti ekonominį vystymąsi. Suteikia strategijos valdymą įgyvendinta veiklos taisyklėmis. Atsiradus poreikiui keisti strategijos planą – veiklos taisykles, ir nekviečiant IS kūrėjų. Be to, šis įrankis yra funkcionalus kuriant taikomąją programą, kuris garantuoja produktyvumą techninėms grupėms, taip pat aprūpiną administravimo, valdymo įrankiais šią sistemą. Grafinė vartotojo sąsaja palengvina analitikams veiklos taisyklių valdymą ir redagavimą [3].




**2.16 pav.** ILOG JRules architektūra.

ILOG JRules architektūra pavaizduota 2.16 paveiksle [12]. Komandinis taisyklių valdymas yra lengvai suprantamas ir paprastai naudojamas. Palengvinantis analitikų darbą – greitai ir saugiai valdyti veiklos taisykles, ir taip sutaupomi pinigai ir laikas. Galimas taisyklių šablonų kūrimas. Kūrimo įrankis susideda iš taisyklių rinkinių vystymo ir testavimo, kurias vykdo taisyklių vykdymo serveris. Visos veiklos taisyklės saugomos centrinėje taisyklių saugykloje. Kūrimo įrankis, taisyklių saugykla ir taisyklių vykdymo serveris yra bendrai sinchronizuojami, tam kad viskas vyktų lygiagrečiai. Taisyklių serveris yra galingas naudojantis J2SE ir J2EE vykdymo aplinkas, kurio pagalba vystomos veiklos taisykles palaikančios servisais orientuotą architektūrą (SOA) [3].

## 2.5. Išvados

1) Išanalizavus veiklos taisyklių manipuliavimo mechanizmus galime teigti, kad toks veiklos taisyklės specifikavimas ir klasifikavimas yra efektyvus, nes: visos taisyklės yra kaupiamos vienoje saugykloje, taip atsiranda galimybė patogiau jas kaupti ir redaguoti; visos taisyklės yra užrašytos formalia taisyklių kalba (vystomi sprendimo komponentai), kurios yra interpretuojamos taisyklių variklio; taisyklės yra vystomos veiklos procesuose, kurie geriausiai atspindi organizacijos veiklos sritį.

2) Analizuojant VTVS sistemas buvo prieita prie tokios išvados, kad nėra vieningos metodikos, kuri apibrėžtų kuriamos informacinė sistemos (IS) raidos etapus. Atspindinčius pagrindinius aspektus į kuriuos būtina atsižvelgti kuriant taikomąją programa pagrįstą VTVS sąveikai su duomenų baze.

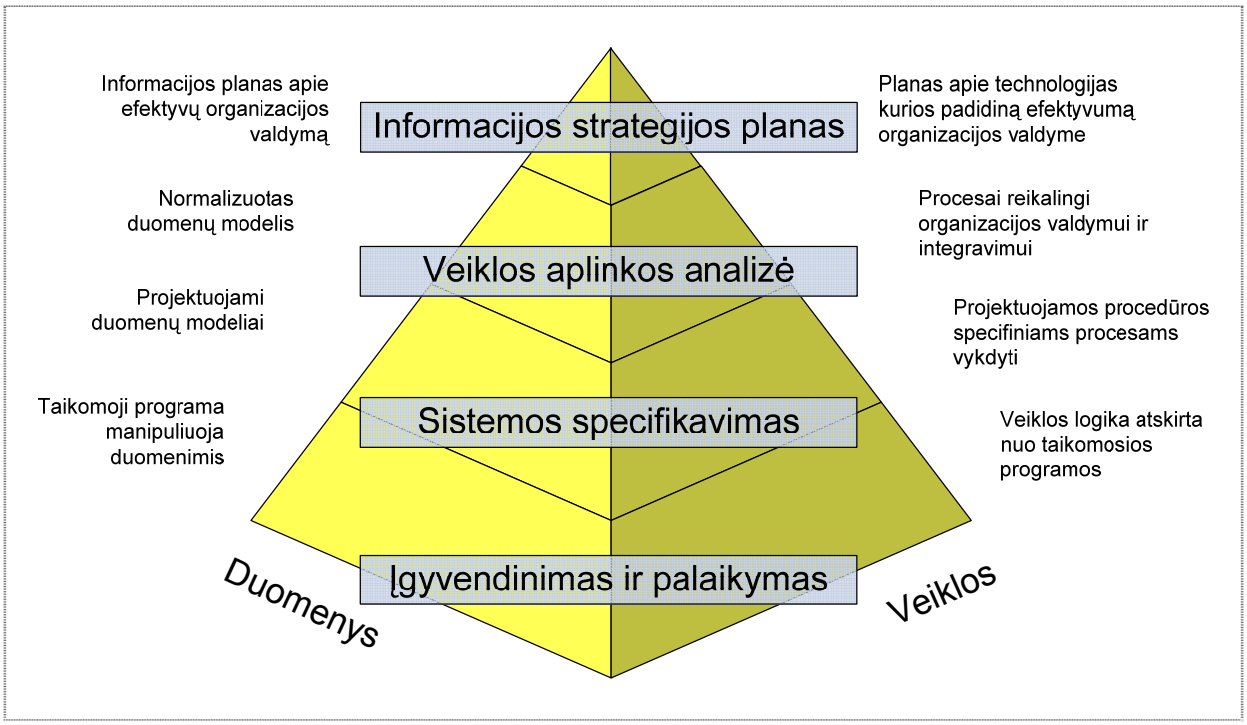
3) Kadangi taisyklės dažnai savyje turi atkartojimo aspektus tokius kaip pasikartojantys terminai, sąlygos ir faktai, kurie yra susieti su organizacijos veiklos procesais. Ir jeigu tokie terminai dažnai keičiasi, tai jie veikia eilę veiklos procesų ir vadinasi veikia eilę veiklos taisyklių. Taigi įrankis turi aprūpinti galimybę tinkamai kaupti ir atkartoti tokius elementus.

# 3. DUOMENŲ STRUKTŪRŲ (DS) TRANSFORMAVIMO Į VT KŪRIMO APLINKĄ METODIKA

Šiame skyriuje yra pateikiama kuriamos informacines sistemos projektavimo metodika, pagrįsta veiklos taisyklių valdymo sistemų principu. Plačiau nagrinėjamas projektavimo etapas: kaip paruošiama duomenų bazė, iš kurios duomenų VTVS formuoja informacija „maitinanti“ veiklos taisykles. Metodikos esmė yra sutelkta į tai kaip paruošti duomenų bazę t.y tam kad, konstruojant VTVS sistemą būtų neprarandami DB struktūros elementai. Jeigu prarandami struktūriniai elementai, tai netinkamai sukuriamos arba neįmanoma sukurti veiklos taisyklių. Netinkamai sujungus VT valdymo sistemą ir duomenų bazę prarandamas tarpusavio ryšys, taigi prarandama sąveika tarp DB esančių duomenų ir jais užpildančių veiklos taisyklių.

## 3.1. DS ir veiklos taisyklių sąveika IS kūrimo etapuose

Metodikai konstruoti buvo pasirinkta informacijos inžinerijos koncepcija (žr. 3.1 pav.). Ši piramidė [15] šaltinyje nagrinėjama ir su kitomis informacijos inžinerijos koncepcijomis. Piramidė nusako keturis lygius: strategijos, veiklos, analizės, sistemos projektavimo ir įgyvendinimo – kurie apima tris principus: duomenis, veiklą ir technologiją. Taigi darbe yra nagrinėjamos šios dalys: VTVS kaip technologija, veikla pagrįsta veiklos taisyklėmis, o duomenys saugomi duomenų bazėje. Pereinant visus keturis piramidės etapus ir tris nagrinėjamas dalis galima suformuoti VTVS paruošimo metodiką.




**3.1 pav.** IS piramidė pritaikyta VTVS.

Kiekviena šiuolaikinė IS turi turėti savo kūrimo kelią, kuriama pereinami pagrindiniai sistemos specifikavimo, duomenų bazės projektavimo etapai ir kiti. Šiame skyriuje

analizuojamas 3.1 lentelėje ir 3.2 paveiksle pavaizduotas IS kūrimo kelias, tačiau visas pagrindinis dėmesys yra sutelktas į VTVS ir DB sąveika. Lentelėje 3.1 ir 3.2 paveiksle parodyti etapuose veikiančių aktorių veikla IS kūrimo procese, kai pereinama nuo organizacijos plano, tikslų ir uždavinių – iki konkretaus IS produkto.

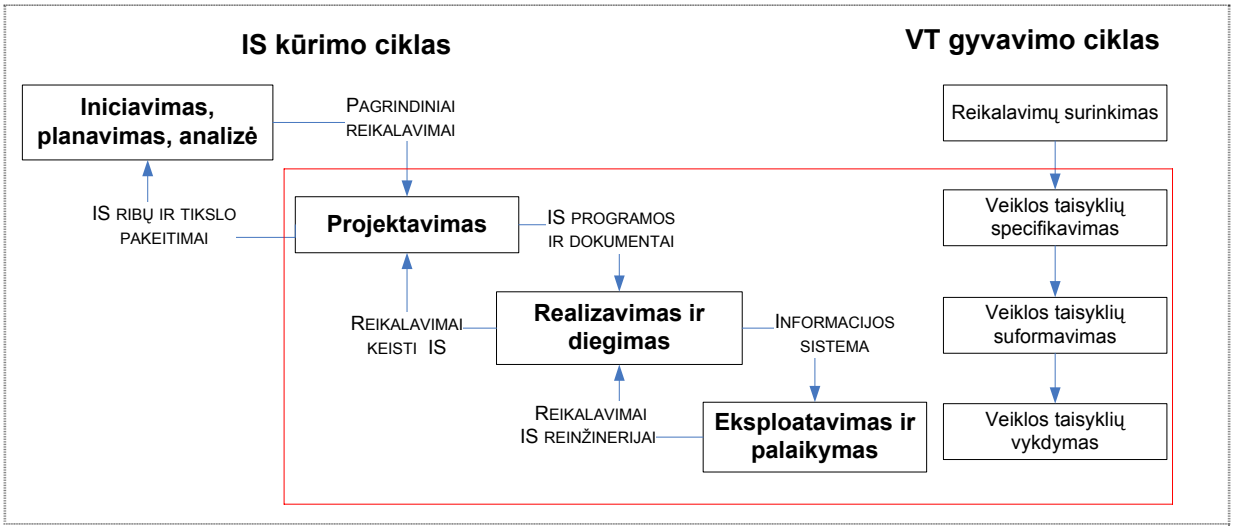
**3.1 lentelė.** IS kūrimo etapuose veikiantys aktoriai.

| **Etapai** | **Reikšmė** | **Aktoriai** |
|---|---|---|
| *Iniciavimas, planavimas, analizė* | Informacijos strategijos planas. | Savininkas |
| | VTVS pasirinkimas ir planavimas. | IT Architektas<br>DS Analitikas |
| | IS sistemos kūrimo analizė. Reikalavimų analizė. | IT Architektas |
| | Dalykinės srities analizė. Taisyklių gavimas iš veiklos krypties, jų formalizavimas ir užrašymas. Taisyklių grupavimas ir priskyrimas veiklos objektams. | DS Analitikas |
| *Projektavimas* | Sistemos specifikavimas, modelio validavimas. Reikalavimų, funkcijų specifikavimas. Vartotojo sąsajos specifikavimas. | IT Architektas |
| | Duomenų bazės projektavimas. | DB Analitikai |
| *Realizavimas ir diegimas* | Programavimas VTVS sistema. DB programavimas. | IT Specialistas |
| | Sistemos testavimas, validavimas, trasavimas, taisyklių testavimas. | IT Specialistas |
| | Vartotojo sąsajos testavimas. | Vartotojas |
| *Eksploatavimas ir palaikymas* | Atsiradus poreikiui atnaujinti VTVS sistemą – ne veiklos logiką. | IT Specialistas |
| | Palaiko taisykles taisyklių saugykloje. | Ap.Vartotojas |
| | Įvedinėja naujas, pašalina senas, redaguoja esamas taisykles – taisyklių saugykloje. | DS Analitikas |

Pateiktoje 3.1 lentelėje pavaizduoti aktoriai yra kaip atskyri specialistai, tačiau nebūtinai šiuos darbus atlieka vienas konkretus specialistas, tai gali atlikti ir specialistų grupės. Pavyzdžiui, DS Analitikai analizuoja dalykinę sritį, išskiria organizacijos veiklos kryptį. Taip pat šiam aktoriui kitus priskirtus veiksmus gali atlikti kiti grupės nariai – taisyklių ekspertai, kurie išvedinėja taisykles, formuoja ir užrašinėja jas naudojamos sistemos kalba (pvz.: *Blaze Advisor* naudoja SRL kalbą). Taip pat ir su kitomis grupėmis, kur IT Architektus sudaro IS Analitikai, Dizaineriai, DB Projektuotojai ir panašiai.

Šioje metodikoje visas dėmesys yra sutelktas į duomenų bazės ir VTVS sąveika – projektavime, realizavime ir palaikyme (kaip parodyta raudona linija 3.2 paveiksle). Tikslas yra toks, kad pereinant šiuos etapus sukurtas IS produktas – veiktų sklandžiai valdant visas organizacijos veiklas, priimant svarbiausius ir specifinius sprendimus. IS produktas gali būti netik vieno padalinio veiklai valdyti, tačiau gali būti ir geografinėje padėtyje pasiskirsčiusių padalinių bendros veiklos valdymui ir palaikymui. Projektavime iš pagrindinių reikalavimų yra specifikuojama ir projektuojama IS, taip pat projektuojamas duomenų modelis. Realizavime yra konstruojama ir įgyvendinama IS, pagal sudaryta specifikaciją. Taip pat šiame etape yra labai svarbus, netik projekto testavimas, tačiau ir veiklos taisyklių testavimas.

Eksploatavime ir palaikyme yra laibai svarbus veiklos taisyklių valdymas. IS kūrimo ir VT gyvavimo ciklai pavaizduoti 3.2 paveiksle.




**3.2 pav.** IS kūrimo ir VT gyvavimo ciklai.

Pirminiame etape planavimas yra vienas iš svarbesnių, nes sudaromas IS planas. Organizacijos savininkas inicijuoją savo viziją apie kuriamą IS – pateikdamas organizacijos tikslus ir uždavinius, ir norimos veiklos planą. Pagal pateiktą viziją architektai ir analitikai sudaro, jau konkretų sistemos prototipą (planą). Įvertinami kuriamos IS tikslai, apibrėžiamos problemos ir sudaromos alternatyvos joms spręsti tiek techniškai, tiek veiklos prasme. Parengiamos patvirtinimų ir svarstymų ataskaitos. Toliau atliekama analizė – nagrinėjamas sudarytasis IS planas, vystoma reikalavimų analizė, įvertinama dabartinė sąveikos būklė, t.y. organizacijos procedūrų ir duomenų būklė. Detalizuojamas naujos sistemos funkcionalumas, apibrėžiami ryšiai tarp procesų ir esybių, ir taip pat atliekamos kitos išsamios analizės. Visa analizė dokumentuojama ir suderinama su organizacijos savininku. Sudaromas objektinis modelis: komunikacinės ir duomenų srautų, statinės ir pragmatinės integracijos priklausomybės.

Sekančiuose skyreliuose, yra aptariami IS pagrindiniai projektavimo, realizavimo ir palaikymo etapai – pateikiamos diagramos nusakančios pagrindines veiklas IS kūrimui pasinaudojant VTVS.

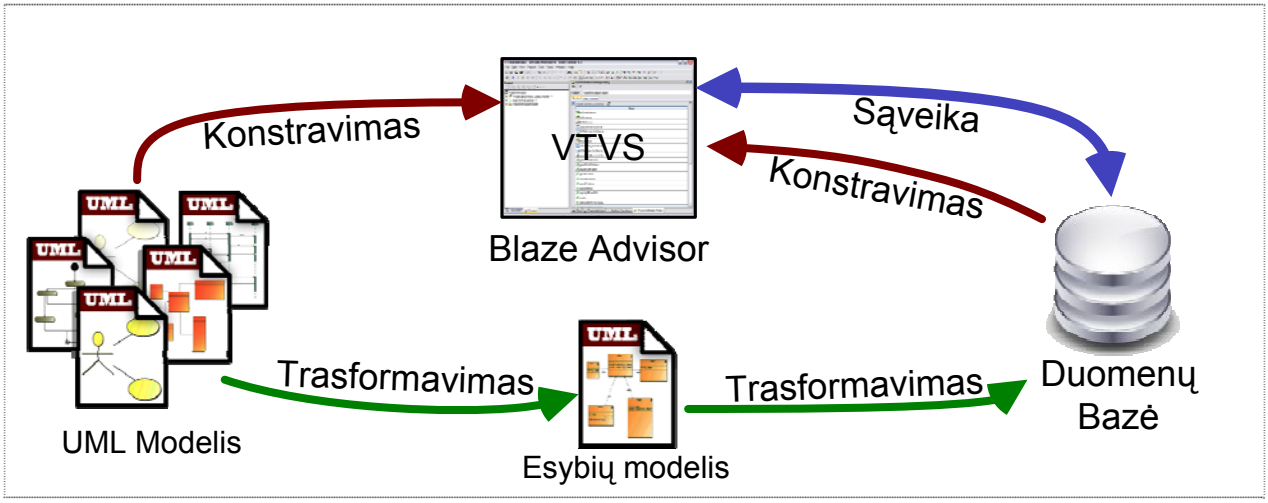
### 3.1.1. Projektavimas

Šis etapas yra vienas sudėtingiausių iš viso IS vystymosi, nes šiame etape yra aprašoma visa sistemos elgsena. Pasirenkama modeliavimo kalba ir įgyvendinami jos principai. Elgsenos specifikavimui braižomos klasių, veiklos, būsenų, užduočių, sekų ir kitos diagramos. Pilnai parengiama reikalavimų specifikacija ir projekto dokumentacija. Specifikuojami funkciniai ir nefunkciniai, vartotojo, sistemos, duomenų modelio ir programinės įrangos reikalavimai. Jeigu yra numatomas prototipas, tai specifikuojami ir

modelio reikalavimai. Etapo pabaigoje, apibendrinami reikalavimai ir suformuluojami galutiniai sistemos tikslai. Toliau nagrinėjamas duomenų bazės projektavimas kaip iš klasių diagramos transformuojama į reliacinę DB.

**Duomenų bazės projektavimas.** Duomenų bazių projektavimas yra vienas iš didžiausiu IS kūrimo uždavinių. Nes IS ne tik kaupia, saugo ir klasifikuoja duomenys, o taip pat panaudojus VTVS komponentus jie transformuojami į tam tikrą informaciją, ir taip atliekami sprendimai. Todėl projektuojant duomenų bazę, būtina atsižvelgti į tai kaip VTVS sistema interpretuoja sąveika su duomenų baze. Svarbu parinkti tinkamiausią duomenų bazės transformacijos strategiją.

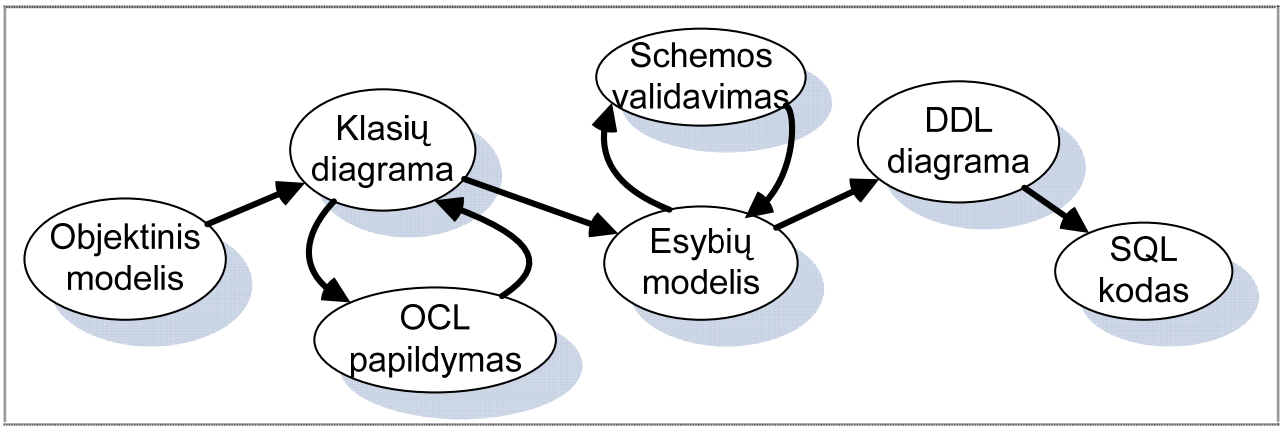
Šiam darbui atlikti yra pasiremta UML kalbos modeliavimu ir specifikavimu (žr. 3.3 paveikslą). Kai iš UML modelio sudarytoji klasių diagrama papildyta stereotipais – transformuojama į esybių modelį. Toliau pakoreguotą esybių modelį transformuojama į reliacinę duomenų bazę. Galimos ir kitos duomenų modelio transformacijos.




**3.3 pav.** Projektavimo esmė DB atžvilgiu.

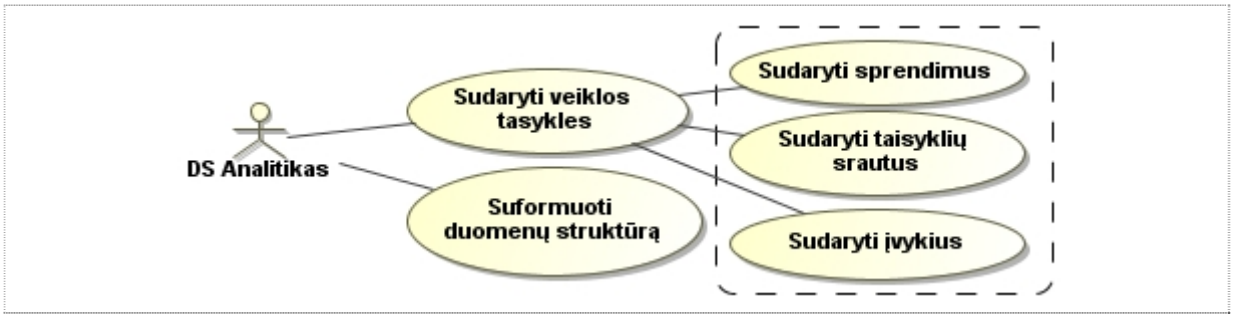
Pavyzdžiui, šiame darbe nagrinėjamas *Blaze Advisor* duomenų bazės lenteles atvaizduoja objektinėmis Java klasėmis. Taigi svarbu suprojektuoti DB taip, kad konstruojant iš lentelių klases, nebūtų prarasti DB struktūros elementai. Pravartu iš objektinio modelio sudaryti klasių diagramą – iš kurios transformuojamas SQL kodas. Taip išlieka glaudus ryšys tarp DB lentelių ir atvaizduojamųjų klasių. Paveiksle 3.3 žaliomis rodyklėmis pavaizduotas transformavimas iš klasių diagramos į reliacinę duomenų bazę; ruda linija pavaizduotas konstravimas: iš DB lentelių konstruojamos objektinės klasės, iš UML modelio konstruojama IS elgsena; tuo tarpu mėlyna linija vaizduoja sąveika tarp VTVS ir DB, t.y. atsižvelgiama į tai, kad projektuojama IS yra VTVS principo.

Detalizuotas duomenų modelio transformavimas, grįstas klasių diagramomis (žr. 3.3 paveiksle išskirta žaliomis rodyklėmis), pavaizduotas 3.4 paveiksle.




**3.4 pav.** Duomenų modelio (duomenų bazės) transformavimas.

**Taisyklių suformavimas.** Kadangi kuriama informacinė sistema yra grindžiama VTVS, taigi reikia atsižvelgti ir į tai kad: organizacijos veikla yra atskiriama nuo programinio kodo. Taigi analizės arba projektavimo etape reikia išskirti ir veiklos taisyklių gavimą – iš organizacijos veiklos srities. Kokios užduotys atliekamos formuojant taisykles pavaizduotos 3.5 paveiksle.




**3.5 pav.** Užduočių Diagrama (UD): Taisyklių formavimas.

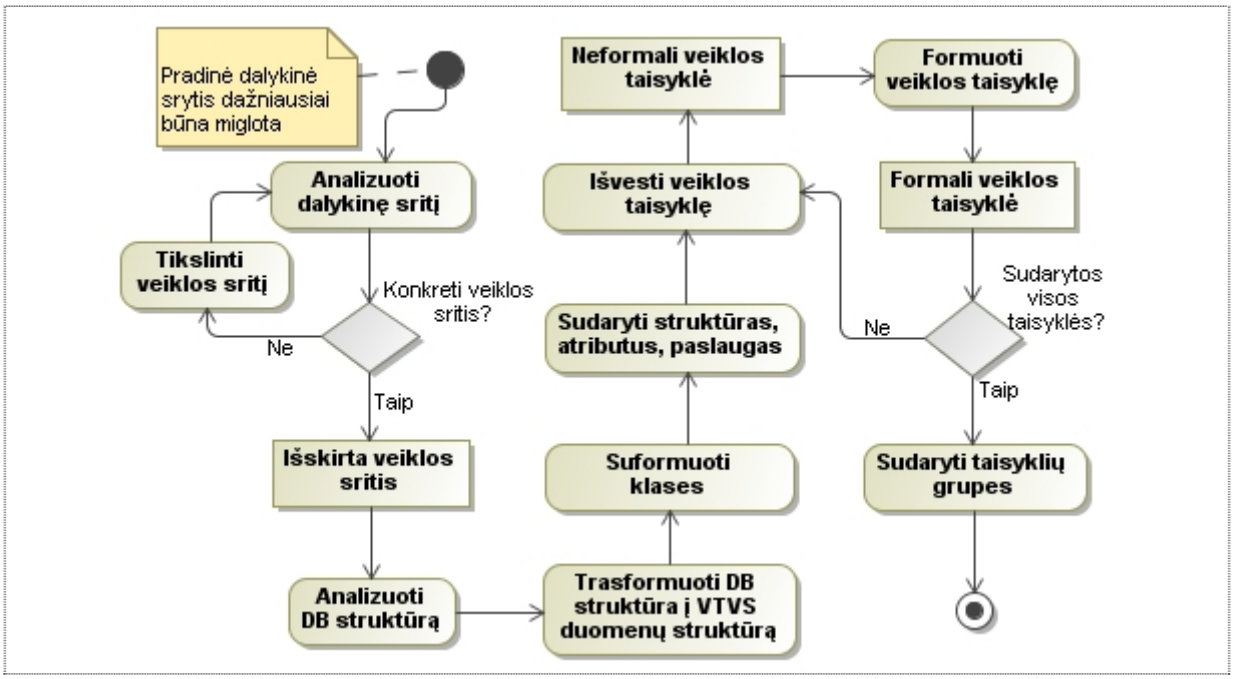
Suformuotos ir užrašytos formalios veiklos taisyklės, toliau yra grupuojamos į sprendimus pasinaudojant: taisyklių rinkiniais, sprendimo medžiai ar lentelėmis. Taip pat svarbu sudaryti įvykius ir taisyklių srautus. Šiame darbe nėra nagrinėjami principai, kaip teisingai suformuoti veiklos taisykles Taisyklių nepriklausomumo principai nagrinėjami [4] literatūros šaltinyje, kur pateikiami 10 straipsnių nagrinėjančių veiklos taisykles, pirmi trys pateikiami 3.2 lentelėje.

**3.2 lentelė.** Veiklos taisyklių manifestas.

| **Straipsnis** | **Nagrinėjama** |
|---|---|
| 1. Pirmaeiliai, ne antraeiliai reikalavimai | Taisyklės yra pirmaeilės svarbos reikalavimai.<br>Taisyklės yra esminė, atskira veiklos ir technologijos modelių dalis. |
| 2. Atskirtos nuo procesų, ne jų dalis | Taisyklės aiškiai riboja ir/arba remia elgseną.<br>Taisyklės nėra nei procesas, nei procedūra, todėl neturėtų būti nė vieno iš jų sudėtinė dalis.<br>Taisyklės veikia procesus ir procedūras. Visos taisyklės, kurių nuosekliai laikomasi visose tiesiogiai susijusiose veiklos srityse, turėtų būti suderintos. |

| Straipsnis | Nagrinėjama |
|---|---|
| 3. Sistemingos, ne atsitiktinės, žinios | Taisyklės kuriamos faktų, o faktai − terminais išreikštų sąvokų pagrindu.<br>Terminai išreiškia veiklos sąvokas; faktai išreiškia teiginius apie sąvokas; taisyklės riboja ir remia faktus.<br>Taisyklės turi būti aiškiai išreikštos. Jokia taisyklė apie jokias sąvokas ar faktus negali būti numanoma.<br>Taisyklės yra bazinių veiklos žinių pagrindas.<br>Taisykles reikia puoselėti, saugoti ir valdyti. |

Analizuojant organizacijos dalykinę sritį (dažniausiai pradinė DS būna miglota) analitikai iš DS ir organizacijos veiklos išvedinėja veiklos taisykles (žr. 3.6 pav.). Dažniausiai pirminės taisyklės yra neformalios (t.y. taisyklės užrašytos natūralia kalba). Tokios taisyklės yra netinkamos sistemos vykdymui. Taigi taisyklės yra formuojamos pagal pasirinktos sistemos formatą: dažniausiai užrašinėjamos taisyklės sąlyginiais *jei-tai* sakiniais. Analizuojami ir sudaromi organizacijoje veikiantys objektai ir sudaroma duomenų struktūrą. Iš atskyrų taisyklių sudaromos taisyklių grupes ir priskiriama atitinkančią taisyklių grupė kiekvienam objektui veikiančiam sistemoje.




**3.6 pav.** Veiklos diagrama (VD): DS analizavimas, taisyklių suformavimas.

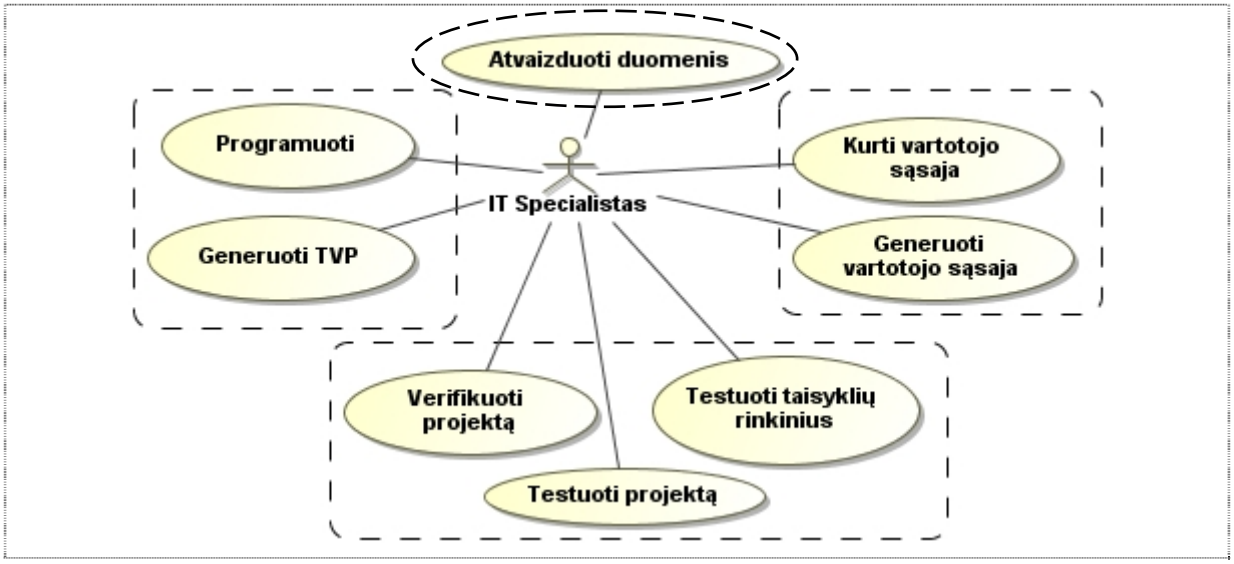
### 3.1.2. Realizavimas ir diegimas

Kai yra atliktas sistemos planavimas, analizavimas ir specifikavimas – tai po jų seką IS realizavimas arba kitaip tariant – įgyvendinimas. Sistemos kūrimas priklauso nuo pasirinktos technologijos, šiame darbe analizuojamas VT valdymo sistema *Blaze Advisor* ir jo kuriamos IS. Pradiniame realizavimo etape pirmiausia seka taisyklių saugyklos saugojimo būdo pasirinkimas – priklausomai nuo konkrečios VTVS sistemos galimybių.

Pavaizduotoje užduočių diagramoje 3.7 paveiksle – labai svarbus akcentas: duomenų atvaizdavimas. Šioje dalyje yra suprojektuotos duomenų bazės prisijungimas ir lentelių atvaizdavimas VT valdymo sistemoje. Pagal specifikaciją ir duomenų bazės modelį konstruojamas lentelių atvaizdavimas klasėmis. Toliau vystomas pagal poreiki programavimas, vystomas sistemos funkcionalumas, tiek veiklos taisyklių įgyvendinimas VT valdymo sistemos komponentais. Kitaip tariant užrašomos suformuluotos veiklos taisyklės: taisyklių rinkiniais, sprendimo medžiais ar lentelėmis, ir kitais komponentais. Pagal sistemos poreiki sukuriami įvykiai ir sudaromi taisyklių srautai, ir panaudojami kiti komponentai išvystantys informacinę sistemą VTVS pagalba. Pagal sudaryta vartotojo sąsajos specifikavimą sudaromi vartotojo sąsajos „langai“ – kuo patogesni ir efektyvesni visam IS valdymui, kuris veikia visą organizacijos veiklą.

Svarbus IS realizavimo etape yra testavimas. Tai nėra tik vartotojo sąsajos testavimas, kai vartotojas testuoja informacinę sistemą, ir teikia pastabas apie ją. Tačiau labai svarbu, kad VT valdymo sistema užtikrintu, ne tik projekto kūrimo metu verifikavimą ir testavimą, tačiau ir veiklos logikos patikrinimą, t.y. taisyklių rinkinių testavimą. Taip užtikrinimas visos logikos teisingumas, aptinkamos anomalijos ir prieštaravimai. Tokiu būdu palengvinamas visos IS kūrimas – pašalint netikėtas klaidas realizavimo etape.

IT Specialistų grupei, t.y programuotojų pagrindinės užduotys realizuojant informacinę sistemą pavaizduota užduočių diagramoje 3.7 paveiksle.

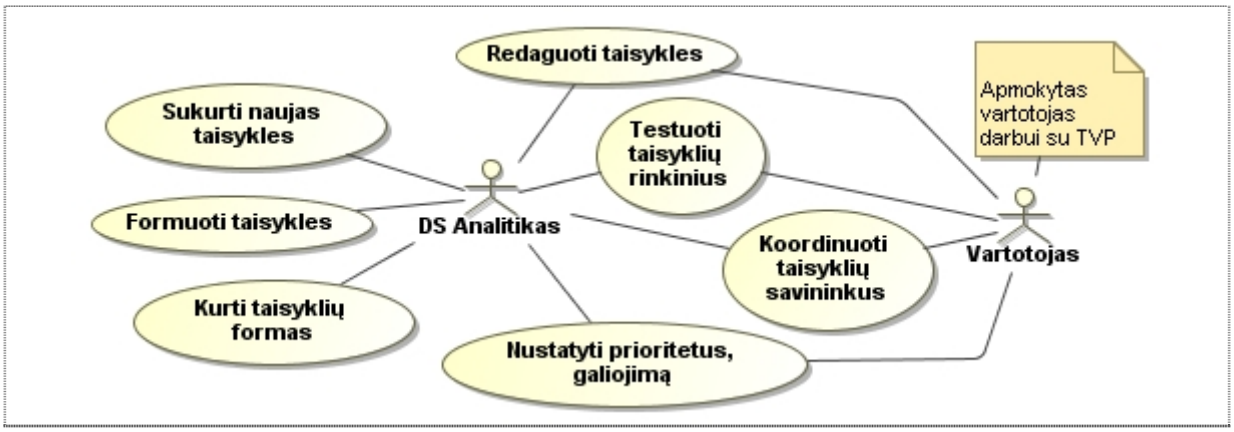



**3.7 pav.** UD: Realizavimas.

Kai IS pasiekia galutini kūrimo tašką ir išvystytos visos suprojektuotos galimybės – vyksta IS diegimas organizacijoje. Supažindamas ir apmokomas personalas darbui su naująja IS. Apmokomas personalas darbui su taisyklių valdymo programa, kuri palaiko taisyklių saugyklą – įvedinėjamos naujos, redaguojamos esamos ir pašalinamos nereikalingos VT.

### 3.1.3. Eksploatavimas, tobulinimas ir palaikymas

Eksploatavime galutinis vartotojas naudojasi sukurta informacine sistema, kurios vidinę logiką valdo papildomi apmokyti vartotojai per TVP. Taigi pirmiausiai reikia apibrėžti palaikymo etape kas yra Vartotojas. Vartotojas – tai netik galutinis vartotojas kuris naudojasi IS, tačiau gali būti ir apmokytas vartotojas darbui su TVP. Tokio vartotojo kompetencijoje yra visoje taisyklių saugykloje veikiančių taisyklių būklė. Tobulinimas ir palaikymas IS vystymosi kelyje yra išties artimi vienas kitam, tačiau pagrindinis skirtumas – IS tobulina IT Specialistas, o palaiko DS Analitikas arba apmokytas Vartotojas. Tobulinimas priklauso nuo to ar naudojantis IS atsiranda poreikis kurti naujas IS funkcijas, įdiegiant naujas vartoto sąsajos formas. Tačiau šis etapas neliečia vidinių organizacijos veiklos taisyklių keitimosi. Palaikymo etape DS Analitikai išvedinėja ir formuoja naujas taisykles (žr. 3.8 pav.). Priklausomai nuo to kaip organizacija planavo IS palaikymą, tai nuo to priklauso ir darbų pasiskirstymas IS palaikymo etape. Tarkime sistema nėra smarkiai besikeičianti t.y. keičiasi tik taisyklių reikšmės ir įverčiai. Tokiu atveju pakanka apmokyti vartotoja ir leisti jam naudotis TVP. Kurio kompetencijoje yra taisyklių redagavimas, galiojimo nustatymas, prioritetų nustatymas ir kiti uždaviniai. Tačiau jei sistema yra dažnai besikeičianti – atsiranda naujos veiklos taisyklės, tokiu atveju reikalingas taisyklių ekspertas (DS Analitikas). Šis ekspertas netik išvedinėja ir formuoja ar redaguoja taisykles, tačiau ir kurią naujas taisyklių formas.




**3.8 pav.** UD: Palaikymas.

Tokiu IS sistemų privalumas yra greitai besikeičiančių veiklos taisyklių redagavimas organizacijoje. Išties IS palaikymas vyksta greitai ir efektyviai, o tobulinimas reikalingas tik tam tikrais atvejais, kai iškviečiami programuotojai naujoms funkcijoms įgyvendinti.

## 3.2. DS transformavimo į VT kūrimo aplinką algoritmas

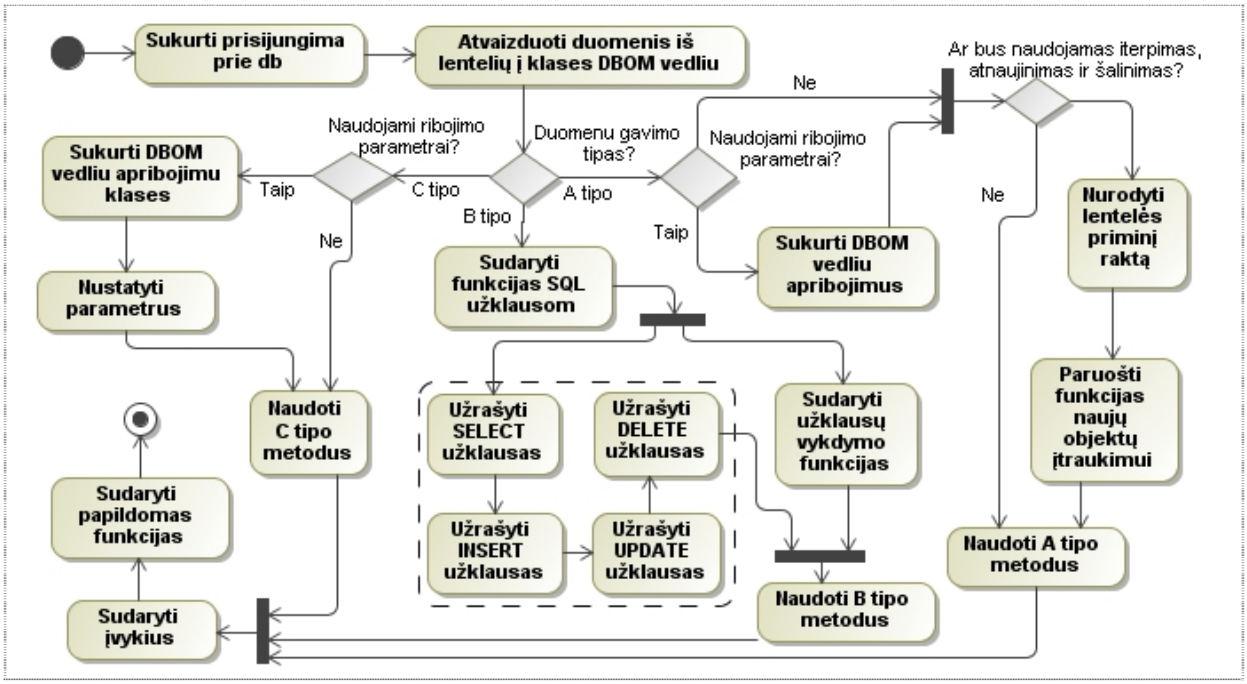
Šiame skyrelyje yra sudarytas algoritmas kaip VTVS sistema *Blaze Advisor* paruošiama darbui su duomenų baze, kurioje apibrėžiami pagrindiniai principai: į ką reikia atsižvelgti kūrimo veikloje.

### 3.2.1. Taisyklių saugyklos paruošimas

Prieš pradedant kurti IS sistemą reikia nustatyti kokiu būdu geriau kaupti veiklos taisykles. *Blaze Advisor* palaiko tris skirtingus taisyklių saugyklos saugojimo būdus: XML, LDAP ir DB. Pagal nutylėjimą taisyklės yra saugomos XML failais. Norint naudoti kitus saugojimo būdus, reikalingas paruošimas: LDAP reikalingas serveris palaikantis LDAP protokolą; norint naudotis DB reikalingas ne tik serveris, tačiau ir jo paruošimas. Yra sukurti papildomi įrankiai, kurių pagalba yra paruošiama DB, t.y. paleidus įrankį – nurodoma duomenų bazė į kurią sugeneruojamos reikalingos lentelės ir jos stulpeliai.

### 3.2.2. Duomenų bazės lentelių atvaizdavimas klasėmis

Vienas iš svarbiausių akcentų realizavimo etape yra duomenų atvaizdavimas. Duomenų saugojimo būdų yra įvairių, tačiau vienas iš efektyviausių ir naudingiausių yra duomenų bazė. Atvaizdavimo būdų yra įvairių priklausomai nuo klasių architektūros ir duomenų bazės. *Blaze Advisor* įrankyje yra keletą akcentų, kai atvaizduojamų klasių esybių atnaujinimas yra negalimas naudojantis standartiniais metodais. Kai yra ribojimas lentelės pirminio rakto atnaujinimas, tokiu atveju programuotojas turi pats užrašinėja atnaujinimo SQL užklausas. Arba panaikinti atnaujinimą veikiančius trigerius (pirminį raktą), tačiau tada programuotojas turi užtikrinti DB vientisumą taikomojoje programoje, kad pakoregavus įrašą neatsirastu anomalijų. Pavaizduotoje 3.9 paveiksle veiklos diagramoje yra pavaizduoti 3.3 skyriuje nagrinėjamų duomenų gavimo ir saugojimo būdų veiklos principai.




**3.9 pav.** VD: Sąveika su duomenų baze.

Priklausomai nuo duomenų bazės pirmiausiai įrankyje yra sukuriama prieiga prie duomenų bazės. Jungimuisi nurodoma kokia yra naudojama valdyklė, nuoroda į DB ir prisijungimo duomenys. Sėkmingai prisijungus prie duomenų bazės, toliau seka duomenų

bazės lentelių atvaizdavimas objektinėmis klasėmis. Klasių konstravimas vykstą DBOM vedliu – pereinant pagrindinius akcentus: išrenkami reikalingi lentelių stulpeliai, pažymimi pirminiai raktai, sudaromi ribojimai ir kiti. Jeigu nėra nurodomi projektavime kokiu būdu išgauti duomenis iš duomenų bazės, taigi programuotojas pats pasirinktą duomenų gavimo būdą – ir pagal jį seką atskyras paruošimas (kokie filtravimo būdai naudojami BA skaityti skyriuje: 3.3.2.„Duomenų filtravimas“). Taigi programuotojas pats renkasi pagal poreikį jam palankiausią atvaizdavimą. Jeigu pasirenkami BA standartiniai filtravimo metodai:

- einat vedliu išrenkamos lentelės ir jų stulpeliai(pagal poreiki galima pavadinimus pervadinti), jeigu pasirenkama daugiau lentelių tokiu atveju yra galimas tik filtravimas;
- jeigu reikalingas įterpimas, tai pasirenkama tik viena lentelė ir jos stulpeliai, nurodomas pirminis raktas. Sukuriamos funkcijos kuriose paruošiamas naujų objektų įtraukimas į duomenų bazę;
- abiem atvejais sukuriami ribojimai (jeigu jie yra reikalingi) ir jie panaudojami SQL užklausai konstruoti *WHERE, ORDER BY* ir kituose skiltyse pasinaudojus DBOM vedlio pagalba.

Jeigu programuotojas pasirenka SQL užklausų užrašinėjimo kelią, tai priklausomai nuo programuotojo lygio – priklauso ir realizuojamo darbo lygis. Pagrindinis principas yra užrašyti visas SQL užklausas, sudaryti funkcijas kurių pagalba yra užklausos vykdomos, ir programuotojas pats prisijungia/atsijungia prie DB. Jeigu DB yra sukuriamos priskirtosios procedūros, taigi pravartu programuotojui jas naudoti. Tokiu būdu, jeigu paskirtosios procedūros yra naudojamos be parametru, tai vykdymo metodus galima naudoti tiesiogiai. Tačiau jeigu yra parametrituotos procedūros, tai DBOM vedliu yra sukuriami papildomi ribojimai, kurie yra realizuoti kaip atskyros klasės.

Duomenų atvaizdavimas išties yra labai svarbus, nes iš duomenų bazės filtruojami duomenys, ir jie yra užpildomi klasėmis. Pagal tokią informacija toliau yra atliekami sprendimai, taigi jeigu atvaizdavimo metu prarandami duomenys, tai galimi ir tam tikri sprendimų praradimai, ir taip neveikia pilnai VT. Labai svarbu dirbti sistemingai ir pagal projektą, nes atvaizduojant lenteles BA įrankyje klasėmis, kai yra naudojami ribojimai ir nurodomi pirminiai raktai, tai įrankis sukuria vienai lentelei ir jos valdymui net 3 klases.

Kitiem papildomoms reikmėms sudarinėjami taisyklių įvykiai – papildomai informacijai išgauti ar pasikeitus sistemoje tam tikroms reikšmėms iškviečiamas papildomas veiksmas. Sudarinėjamos pagal poreiki funkcijos, kuriose gali būti įgyvendinami algoritmai ar minimalūs skaičiavimai.

### 3.2.3. VT sprendimo komponentais

Prieš taisyklių užrašymą, įrankio komponentais svarbu išskirti veiklos taisykles į dvi grupes: įrankyje ir duomenų bazėje veikiančias taisykles (kokios taisyklės laikomos DB skaityti sekančiame skyrelyje). Šis taisyklių atskyrimas gali vykti ir analizės etape, tai tokiu atveju programuotojui tektų tik užrašinėti taisykles priklausančiai grupei. Panaudojimo diagramoje parodytoje 3.10 paveiksle – išskiriami taisyklių užrašymo būdai.

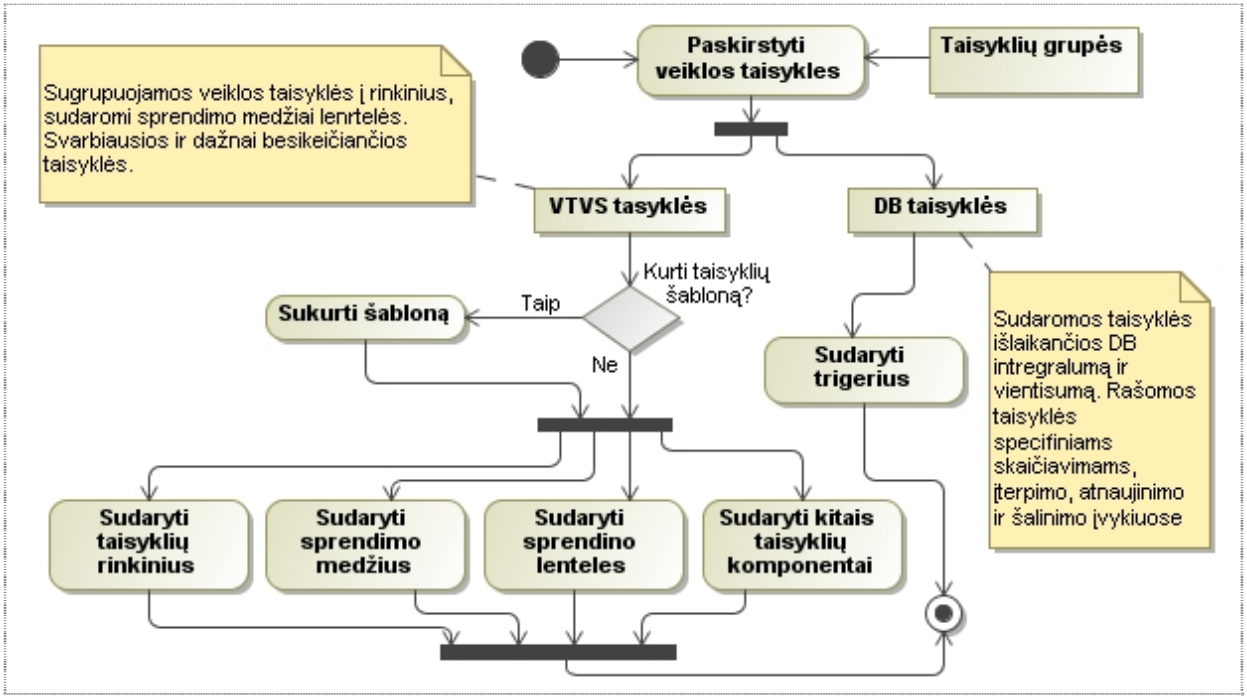
Jeigu yra priskiriama taisyklių grupė įgyvendinant įrankio komponentams, taigi pirmiausiai apjungiamos panašia struktūrą taisyklės. Tai atliekama pasinaudojus šablonais, kai yra užrašoma taisyklė ir nurodomi tušti laukai, kurie vėliau yra užpildomi. Programuotojas renkasi kokiu būdu užrašinėti taisykles. Jei iš taisyklių grupės išeiną sudarytį medi, tai įrankyje pravartu sudaryti sprendimų medį. Tokių medžių privalumas – lengvas taisyklių eiliškumo įsivaizdavimas įrankyje. Programuotojas gali lengvai atskirti sprendimo kelią einant nuo viršūnės iki lapo. Jeigu organizaciją laiko tam tikrą informaciją lentelių pavidalu, tai pravartu naudoti taisyklių lenteles. Tokiu lentelių pranašumas, kaip ir medžių yra grafinis apipavidalinimas. Tačiau yra ir trūkumas: didelės lentelės yra išties sunkiai skaitomos. Priklausomai nuo VTVS sistemos, yra ir kitokiu taisyklių užrašymo būdų (pvz.: BA įrankyje įverčiu modelis). Vienas iš standartinių ir dažniausiai naudojamų komponentų – taisyklių rinkiniai. Tokios taisyklės neturi grafinės sąsajos, užrašoma naudojama sistemoje kalba. Nors atrodo, kad tai yra skirtingi užrašymo būdai, tačiau taisyklių variklyje visi komponentai yra lygūs. Tiek patys sprendimo medžiai ir lentelės yra išsaugomi, kaip ir taisyklių rinkiniai t.y. *jei-tai* sakiniais.

### 3.2.4. VT duomenų bazės trigeriais

Duomenų bazės trigeriuose galima naudoti veiklos taisykles, kada įvyksta DB įvykiai tokie kaip INSERT, UPDATE ar DELETE. Yra įvairių nuomonių ir siūlymų, nagrinėjamų [13] literatūros šaltinyje, kaip veiklos taisykles integruoti DB trigeriuose. Tačiau ar tai verta daryti, kai yra kuriama taikomoji programa pagrįsta VTVS. Tokiu atveju trigeriuose saugomas taisykles reikėtų palaikytu su papildomais valdymo įrankiais. Ir taip neišnaudojamos visos VTVS architektūros galimybės, kuri VTVS pilnai palaiko veiklos taisyklės sąvoką – duomenų ir įvykių kombinacija. Tačiau neverta duomenų bazę palikti visai be veiklos, ir įgyvendinti: DB integralumui ir apsaugai palaikyti; stulpelių reikšmių pakeitimams, kai keičiasi tos pačios ar kitos lentelės reikšmės; ir stulpelių reikšmių patikrinimui ar perskaičiavimui.

Tačiau išlieka problema, kai duomenų bazės įrašai yra koreguojami iš kitų įrankių, tokiu atveju atnaujinamų duomenų neveikia jokia veiklos taisyklė, kuri standartiškai veikia IS.

Taigi tokiu atveju vertėtu nors minimaliai naudoti veiklos taisykles trigeriuose, t.y. tokias taisykles kaip „validation rules“ ir kitos.




**3.10 pav.** VD: Sprendimų sudarymas, komponentais.

Taisyklės įgyvendintos DB trigeriuose yra sudėtingos valdyti iš VTVS įrankio (jeigu taisyklės dažnai keičiasi), taigi trigeriuose vertėtų naudoti tik tokias taisykles, kurios vykdant sprendimą papildo tam tikra informacija įterpimo, atnaujinimo ar pašalinimo metu. Trigeriuose gali būti atliekami prieš ar po atnaujinimą – perskaičiavimo sprendimai, ir jeigu yra atliekamas neteisingas atnaujinimas, tai sustabdoma ir siunčiamas pranešimas informacinei sistemai. Ir kitokie sprendimai, kurie ilgainiui nesikeičia arba išvis nesikeičia. Veiklos diagramoje parodytoje 3.10 paveiksle yra pavaizduotas taisyklių priskyrimas sprendimo komponentams ir DB trigeriais.

### 3.2.5. Taisyklių srautų sudarymas

Taip pat svarbus akcentas yra taisyklių srautų sudarymas. Tai yra grafinė vartotojo sąsaja, kurioje programuotojas sudėtingiems sprendimams sudarinėja procesus. Kuriant tokius procesus palengvėja taisyklių srautų valdymas. Tokiais atvejais patogu sudarinėti ir svarbiausia pakoreguoti procesą. Kaip įgyvendinti tokie VTVS procesai pavaizduota – 2 skyriaus 2.12 paveiksle: taisyklių srautai.

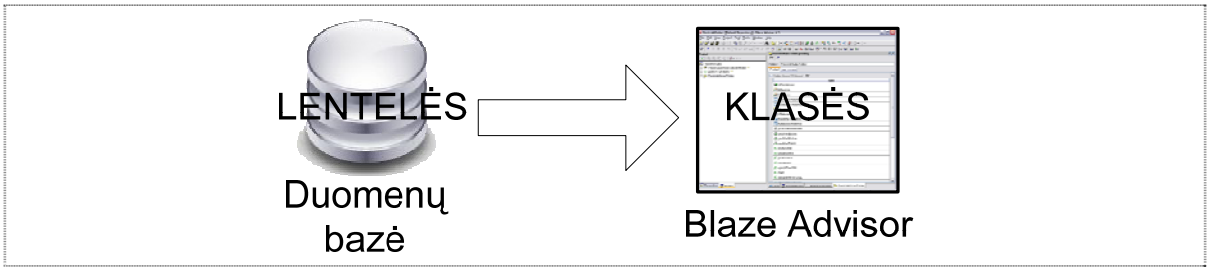
### 3.2.6. Projekto ir taisyklių rinkinių testavimas

Testavimo etapas susideda iš smulkesnių dalių: taisyklių testavimo, sistemos trasavimo, vartotojo sąsajos testavimo. Šio etapo metodikoje yra nagrinėjamos BA įrankio galimybės. Taisyklių testavimui galima pasinaudoti *brUnit* moduliu, kuris leidžia įvykdyti testus veiksmams, kurių pagalba galima palyginti laukiamą rezultatą su tikru rezultatu. Pasinaudojus

šiuo moduliu galima: rašyti testus pasinaudojant „assertion“ metodais; atskirti testavimo kodą nuo veiklos taisyklių; organizuoti testus bandomajam atvejui; bet kuriuo metu vykdyti testus; peržiūrėti ar testai atitinka ar neatitinka reikalavimų. Šį modulį galima naudoti tiek kūrimo aplinkoje, tiek taisyklių valdymo programoje. Taip pat vystant sistemą, bet kuriuo metu galima pasinaudoti tikrinimo *Verify* įrankiu. Šio įrankio pagalba aptinkamos projekto anomalijos sprendimo esybėse. Anomalija yra loginė problema, kai yra naudojami vienas ar daugiau algoritmų – gali susidaryti konfliktai tarp sprendimų. Šis įrankis yra naudojamas įvairiems konfliktams aptikti ar semantinėms klaidoms patikrinti. Galimas veiklos taisyklių vykdymo statistikos generavimas. Sugeneruojamos ataskaitos, kuriose pateikiama informacija apie projekto vykdymą ar taisyklių serviso. Generuojamos ataskaitos gali apimti: taisyklių vykdymą; objektų panaudojimą taisyklių vykdymo metu; statinė ir dinaminė informaciją apie projekto trasavimą; ir kitos. Testavimas susideda taip pat iš pačios sistemos ir vartotojo sąsajos testavimo. Sistemos metu kuriant sprendimus grindžiamus taisyklėmis, pradžioje patartina testuoti su nedideliu rinkiniu taisyklių, o palaipsniui pildant taisyklių rinkinius iš gauti pilnus sprendimus. Vartotojos sąsają testuoja ne tik kūrėju grupė, tačiau ir vartotojai. Atsižvelgiama į vartotojo pastabas, ir pagal pageidavimus pakoreguojama vartotojo sąsaja.

## 3.3. DS atvaizdavimas

Šiuolaikinės verslo sistemos dažniausiai yra kuriamos objektinio programavimo paradigma, naudojamos tokios kalbos kaip JAVA arba C#. Vienas iš populiariausių duomenų saugojimo būdų yra reliacinės duomenų bazės. Taigi šiame skyrelyje yra nagrinėjama tokia problematinė sritis kai atvaizduojami duomenis iš DB lentelių į klases (žr. 3.11 pav.). Duomenų atvaizdavimas nagrinėjamas [2] literatūros šaltinyje.




**3.11 pav.** Duomenų atvaizdavimas.

Nagrinėjami ryšių kardinalumai vienas-su-vienu, vienas-su-daug ir daug-su-daug. Aptariami pagrindiniai Blaze Advisor įrankio duomenų atvaizdavimo principai, ir pateikiami elementarūs suvokimui pavyzdžiai.

### 3.3.1. Duomenų tipai

Kadangi įrankis iš duomenų bazės lentelių konstruoja klases, tai toliau apžvelgsime pagrindinius duomenų tipus. Elementarus pavyzdys, kaip įrankis interpretuoja DB duomenų

tipus, parodytas 3.3 lentelėje. Kaip matome iš šios lentelės tipas *integer* nesikeičia einant nuo lentelės iki klasės, tuo tarpu *varchar* tipas keičiasi į *string*.

**3.3 lentelė.** Duomenų tipo perėjimo pavyzdys.

| DB lentelė | Perėjimas | Klasė |
|---|---|---|
| INTEGER[10]StudentasID <PK><br>INTEGER[10]FakID <FK><br>VARCHAR[20]StudVardas<br>VARCHAR[20]StudPavarde<br>NUMERIC [20]Stipendija<br>INTEGER[10]PazymNr | StudentasID → id<br>FakID → fakultetasId<br>StudVardas → vardas<br>StudPavarde → pavarde<br>Stipendija → stipendija<br>PazymNr → pazymNumeris | integer id<br>integer fakultetasId<br>string vardas<br>string pavarde<br>BigDecimal stipendija<br>integer pazymNumeris |

Pagrindiniai duomenų tipai pereinant nuo duomenų bazės į BA įrankį pavaizduoti 3.4 lentelėje. Kitus neparodytus lentelėje tipus įrankis interpretuoja, kaip objekto tipą. Įrankis naudoja metodą *ddPropertyValueFilter()*, kuris leidžią *null* reikšmes pervesti į specifines įrankio reikšmes *unknown* arba *unavailable*.

**3.4 lentelė.** Duomenų tipai.

| DB duomenų tipas | BA duomenų tipas |
|---|---|
| BIT | boolean |
| TINYINT, SMALLINT, INTEGER, BIGINT | integer |
| FLOAT, REAL, DOUBLE | real |
| NUMERIC, DECIMAL | BigDecimal (java.math.BigDecimal) |
| NUMBER (Oracle) | real, jeigu tikslumas p yra daugiau už 0<br>integer, jeigu tikslumas p yra lygus 0 |
| CHAR, VARCHAR | string |
| DATE | SQLDate (java.sql.Date) |
| TIME | Time (java.sql.Time) |
| TIMESTAMP | Timestamp (java.sql.Timestamp) |
| NULL | null |
| Kiti... | Object |

### 3.3.2. Duomenų filtravimas

Šiame skyriuje yra aptarta kaip įrankis jungiasi prie duomenų bazės ir išgauna reikalingą informaciją. Viskas vyksta įrankyje atvaizduojant klases su įrašais esančiais duomenų bazėje – einant pasirinkimo vedliu, o duomenis išgaunami pasinaudojus SRL kalba. BA įrankis leidžia kreiptis į duomenų bazę keliais būdais pasinaudojant *Database Business Objetc Model* (toliau DBOM) vedliu (jo pagalba galima sukurti: prisijungimą prie duomenų bazės; pasirinkti duomenis iš duomenų bazės lentelių ar vaizdų; sukurti SQL užklausas duomenų filtravimui; apibrėžti klases naudojančias tinkamas jai lenteles). Taigi vienas iš būdų yra einant pasirinkimų DBOM vedliu: užrašomos SQL užklausos susiejant klasę su duomenų bazes lentelėmis, ir pasinaudojus standartinėmis funkcijomis vykdomas duomenų keitimasis. Kitas būdas einant tuo pačiu DBOM vedliu, neužrašinė jokiu SQL komandų, o vedlį panaudoti kaip nurodymą į kokią duomenų bazę bus kreipiamasi. Tokiu atveju programuotojas, pats užrašinėja „ranka“ SQL komandas ir rūpinasi

prisijungimu(atsijungimu) prie(nuo) duomenų bazės, ir pačių užklausų įvykdymų. Taip pat galima pasinaudoti priskirtosiomis procedūromis (angl. *stored procedures*).

Sakykime yra kaupiama informacija apie verslo objektus reliacinėje duomenų bazėje ir taip pat yra užrašytos taisyklės veikiančios šiuos objektus. Pavyzdžiui duomenų bazėje turime lentele pavadinta *Studentas* su tokiais laukais: *StudentasID, Vardas, Pavardė* ir *PažymėjimoNumeris*. Taigi su tokia lentele galime atlikti tokius veiksmus:

- Susieti lentelę (*Studentas*) atitinkančia jai klase su parametrais *StudentasID, Vardas, Pavardė* ir *PažymėjimoNumeris*. Jeigu yra poreikis įrankis leidžia pervadinti parametru pavadinimus.
- Užpildyti reikalingais duomenimis klasę atitinkančią duomenų bazės lentelės eilutėmis.

Nebūtinai viena klasė yra susiejama su viena lentele, galima apjungti kelias lenteles į vieną klasę. Taip pat dvi skirtingos klasės gali naudoti vieną bendrą lentelę. Kaip jau buvo minėta yra keletą būdų kaip įrankis išgauna duomenys iš DB, taigi 3.5 lentelėje yra parodyti trys duomenų išgavimo būdai: A – susietos klasės su duomenų bazės lentelėmis; B – SQL užklausų vykdymo metodai; C – iškviečiant priskirtąsias procedūras.

**3.5 lentelė.** Duomenų gavimo metodai.

| **Būdas** | **Metodai** | **Reikšmė** |
|---|---|---|
| A | `className.getAllInstances();`<br>`className.fetchAllInstances();`<br>`className.xxxInstances(`<br>`mappedClassParams params);` | Gražina visas className reikšmes, su fiksuoto ilgio masyvu. Gali būti panaudoti ir papildomi apribojimų parametrai. |
| | `className.getNextInstance();`<br>`className.fetchNextInstance();` | Klasės „one by one“ seka |
| | `className.getNextInstances(n);`<br>`className.fetchNextInstances(n);` | Gražina n skaičių className reikšmių, su fiksuoto ilgio masyvu, kur n yra integer tipo argumentas, kuris nurodo kiek reikšmių gražinama |
| | `className.getNthFetchedInstance(n);` | Reikšmė n pozicijų esamoje rezultatų aibėje, kur n yra rezultato pozicija |
| | `className.getInstance(id);` | Reikšmė susieta su pirminiu lentelės raktu, kur id yra *classNameId* reikšmė |
| B | `className.connect();`<br>`call connect on className.` | Prisijungimas prie norimos duomenų bazės |
| | `className.executeUpdate(stringSQL);`<br>`call executeUpdate on clName using stringSQL.` | Atnaujinama duomenų bazės informacija, siunčiama SQL komanda, kai nereikalingas grąžinimas rezultatas |
| | `className.executeQuery(stringSQL);`<br>`call executeQuery on className using stringSQL.` | Siunčiama užklausa į duomenų bazę. Šis metodas gali grąžinti rezultatų aibę. |

| Būdas | Metodai | Reikšmė |
|---|---|---|
| | `className.disconnect();`<br>`call disconnect on className.` | Atsijungia nuo duomenų bazės |
| C | `DbResultSet executeCall(stringSQL);`<br>`DbResultSet executeCall(stringSQL, mappedClassParams params);` | Iškviečia priskirtąsias procedūras, kai reikalingas rezultatas. Pavyzdžiui kai vykdoma SELECT užklausa. |
| | `integer executeUpdateCall(strSQL);`<br>`integer executeUpdateCall(strSQL, mappedClassParams params);` | Iškviečia priskirtąsias procedūras, kai nereikalingas rezultatas. Pavyzdžiui kai vykdoma INSERT ar UPDATE užklausas. |

Jeigu pasinaudojus DBOM vedliu kaip statinėms SQL užklausoms užrašyti, tai standartinės funkcijos iškviečiamos be papildomų parametrų. Tačiau tokios SQL užklausos retai pasitaiko (kai iš DB lentelės filtruojami visi esantys įrašai), taigi apibendrinant ir papildant SQL užklausas papildomais ribojimais, tokia galimybė įrankyje yra pasinaudojant apribojimų parametrus. Taigi naudojantis tokiais apribojimų parametrais užrašant SQL užklausas, leidžiama nustatyti parametrus reikalingomis reikšmėmis iš taisyklių bazės. Taip duomenų bazės užklausos susiaurinamos iki reikalingų duomenų veikiančių taisyklių apdorojimuose.

Tarkime jau minėtame pavyzdyje su lentele *Studentas*, reikia išfiltruoti vieną studentą pagal pažymėjimo numerį. Tokiu atveju einant DBOM vedliu įtraukiamas vienas apribojimo parametras. Po užpildymo yra sukuriamos dvi klasės *Studentas* (studento informacijai kaupti) ir *StudentasParams* (kaupiama parametrų informacija). Parametrų klasėje yra užpildomi įtrauktieji apribojimų parametrai, šiuo atveju yra pildomas pažymėjimo numerio ribojimas. Taigi iškviečiant standartines funkcijas su papildomu ribojimo parametrų, duomenų bazė grąžiną reikalingą informaciją.

A tipo funkcijos pasižymi privalumu tuo, kad užrašytas programinis kodas yra išties trumpas palyginus su B ir C būdais. Tokiu atveju kai kuriama sistema ne vieno programuotojo, o grupės – toks būdas yra patogus skaitymui kito programuotojo užrašytam kodui. Nes toks A būdas yra labiau pririštas prie pačio BA įrankio (t.y. lentelių suliejimas į klases, kai užrašomos SQL užklausos su DBOM vedliu), ir iš dalies šis būdas palengvinama neįgudusio programuotojo darbą dirbant su DB. Naudojant B tipo funkcijas, kai programuotojas pats užsirašinėja užklausas, programuotojas įgauną didesnę laisvę ir fantazijai išgaunant informaciją iš DB. Tačiau A ir B būdai vienas kitam niekuo nenusileidžia, nes abiem būdais galime gauti vienodus rezultatus. B tipo funkcijas patartina naudoti tokiu atveju, kai yra gaunama ne objekto informacija, o ištraukiant papildomą informaciją. Pradedančiajam programuotojui kai užrašinėjamos SQL užklausos su ribojimais (naudojant A būdą), pats pradinis darbas įrankyje yra ganėtinai painus, tačiau įgavus patirties ir suvokimą – darbas įgauną pagreitį. Naudojamų C būdo funkcijų privalumai yra lygus priskirtųjų procedūrų

privalumams. Šiuo būdu programuotojui nereikia užrašinėti SQL užklausų, o tiesiog pakanka žinoti visas DB priskirtąsias procedūras ir jų įeinančius/išeinančius parametrus. Taigi programuotojas pats renkasi kelia kaip iš DB gauti informaciją, atsižvelgiant į tam tikrus aspektus.

### 3.3.3. Duomenų įtraukimas, atnaujinimas ir šalinimas

Šiame skyriuje daugiau dėmesio yra skirta kaip įrankio priemonėmis interpretuojamas duomenų įtraukimas, atnaujinimas ir šalinimas. Visą informacijos keitimąsi galima atlikti ir su skirtosiomis DB galimybėmis, tai yra SQL užklausų pagalbą ar priskirtosiomis procedūromis (3.2 skyriaus 3.3 lentelėje B, C būdai). Taigi toliau nagrinėsime tik pačio įrankio siūlymą, kaip interpretuoti darbą su DB.

Norint redaguoti ar šalinti įrašus iš DB pirmiausia turime ją užpildyti. Pildymas prasideda nuo pirminio rakto priskyrimo, kuris yra atsakingas už įrašus esančius DB. Taigi prieš įtraukiant naujus įrašus turi būti atlikti tokie žingsniai:

- Sukuriamas naujas klasės objektas (pvz.: *NaujasStudentas* objektas priklauso klasei *Studentas*; objektas turi būti globalus, o ne statinis):

```
NaujasStudentas is a Studentas,
set NaujasStudentas to the outcome of newInstance on
Studentas.
```

- Sukuriamas naujas objektas (*NaujasStudentasId*) apibrėžiantis objekto (*Studentas*) pirminį raktą:

```
NaujasStudentasId is a StudentasId.
```

  Šis objektas leidžia nustatyti lentelės stulpelį atitinkantį pirminį raktą apibrėžtos klase.

- Nustatyti naujo objekto pirminį raktą:

```
set NaujasStudentasId's StudentasID to IdReikšmė.
```

- Nustatyti naujo objekto informaciją.
- Iškviesti naujo objekto įkėlimo funkcijas:

```
call insert on NaujasStudentas using NaujasStudentoId.
```

- Papildomos funkcijos DB atnaujinimui:

```
call update on Studentas,
call commit on Studentas.
```

Atlikus šiuos žingsnius į DB yra įtraukiami nauji įrašai, šiuo atveju buvo parodyta, kaip įtraukiamas naujas studentas. Funkcija *insert* yra iškviečiama su papildomu parametru, kuris apibrėžią naujo objekto pirminį raktą. Sekančios parodytos funkcijos *update* ir *commit* yra iškviečiamos tik tuo atveju, jeigu yra nenurodytas automatinis DB atnaujinimas. Norint sumažinti srautą į DB, patartina nenaudoti automatinio atnaujinamo, o programuotojas metodais nurodo kokiu metu yra atnaujinami DB įrašai. Kai vyksta DB pasikeitimas (įrašų

įtraukimas, atnaujinimas ar šalinimas) įrankis naujas reikšmes laiko atmintyje, tol kol neįvykdomas metodas *commit()*. Metodų reikšmės parodytos lentelėje 3.6.

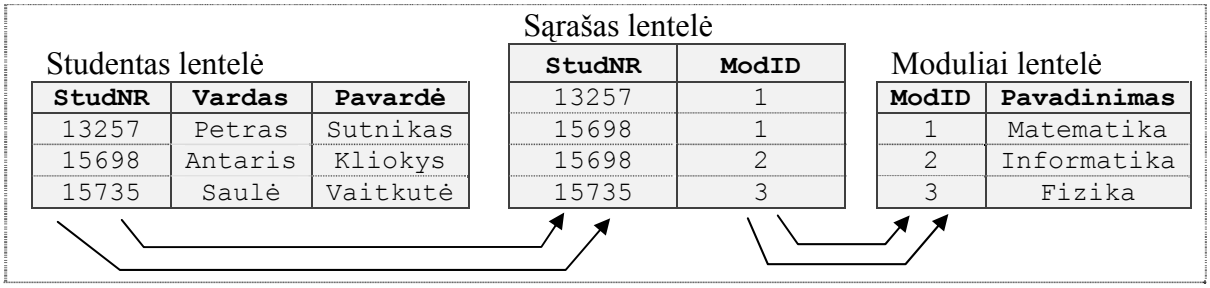
**3.6 lentelė.** Naudojant kartu su A tipo filtravimo metodais.

| **Metodai** | **Reikšmė** |
|---|---|
| `newObjectName.insert(newObjectId)` | Naujo objekto įtraukimas į DB, kur parametras yra pirminio rakto objektas. |
| `objectName.destroy()` | Šalina objektą iš DB. |
| `mappedClassName.update()` | Atnaujina įrankyje informaciją apie objektą. |
| `mappedClassName.commit()` | Jei nenaudojama automatinis atnaujinimas, naudojamas po kiekvienos insert(), destroy() ir update() funkcijos. |
| `mappedClassName.reset()` | Šis metodas tiesiogiai nesikreipia į DB, naudojamas pašalinti objektą iš projekto nenaudojant destroy() metodo. |

Kaip jau buvo minėta, galima naudoti ir tiesioginius DB įrašų įtraukimus, atnaujinimus ar šalinimus, pasinaudojant SQL užklausų rašymų ar priskirtųjų procedūrų naudojimų. Dar vienas iš svarbesnių aspektų, kuris lemia *update()* metodo veikimą yra DB projektuotojo rašyti trigeriai. Jeigu ribojamas atnaujinimas įrašų, pirmiausiai veikiantis trigeris pirminio rakto atnaujinimą, tai *update()* metodas bus neįvykdomas. Parodomas pranešimas, kad įrašas priklauso nuo kitos su ja susijusios lentelės ryšiu ir atnaujinimas nutraukiamas. Kai vykdomas įrankyje atnaujinimas, bet kokio klasės lauko (atvaizduojamos lentele), ir tai gali būti nebūtinai pirminio rakto reikšmė. Tai jeigu yra veikiantis trigeris pirminį raktą, tai nutraukiamas atnaujinamas ir parodoma klaida. BA įrankis siunčia pilną užklausą į DB, norėdamas atnaujinti pilną įrašo informacija – atnaujindamas pilnai visus laukus įskaitant ir pirminį raktą. Sekančiuose skyriuose bus parodoma, kaip įrankis priima tam tikrus sprendimus pasinaudojant komponentais, ir kaip manipuliuoja duomenimis iš DB.

### 3.3.4. DB ryšių kardinalumų atvaizdavimas įrankyje

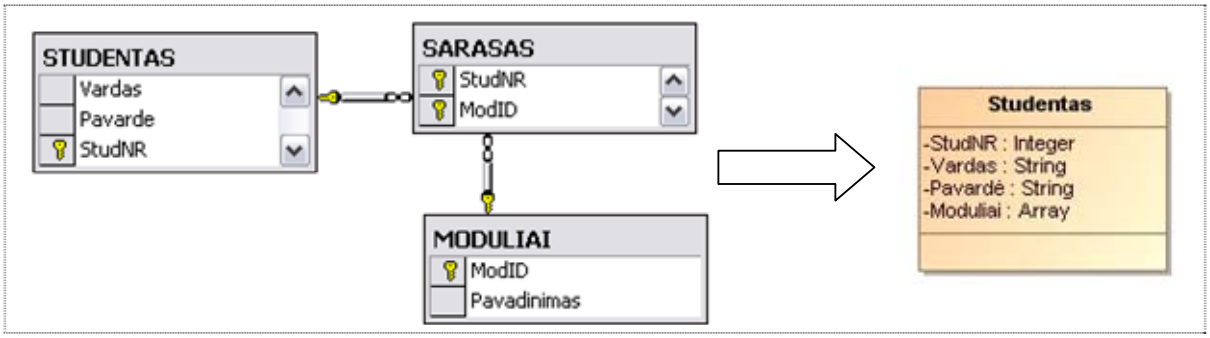
Dažniausiai reliacinėje duomenų bazėje yra naudojamas vienas-su-daug ryšys, rečiau naudojami vienas-su-vienu ar daug-su-daug. Vienas iš problematiškiausiu ryšių yra daug-su-daug, kai jo įgyvendinimui reliacinėje DB yra naudojama tarpinė lentelė kaip parodyta 3.12 paveiksle.




**3.12 pav.** Lentelių daug-su-daug ryšys.

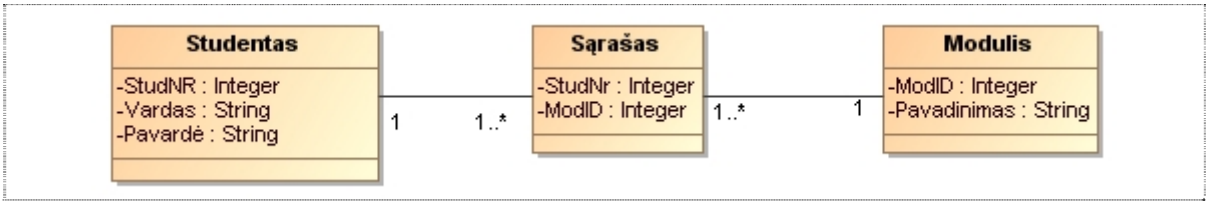
Šiuolaikiniame IT amžiuje yra tyrinėjama ir siūloma didele įvairovė kaip atvaizduoti duomenų bazių lentelių duomenis objektinėmis klasėmis. Vienas iš būdų kaip galima tai padaryti yra vartojamas angliškas terminas „data mapping“, verčiamas į lietuvių kalbą kaip duomenų atvaizdavimas. Tokiu būdu tyrinėjama, kaip duomenų bazės lentelės atvaizduojamos objektinėmis klasėmis. Šiame skyrelyje yra koncentruojamasi į BA įrankio komponentus, kurių pagalba konstruojamas duomenų atvaizdavimas. Kadangi nagrinėjamas BA įrankis yra pagrįstas JAVA koncepcija, tai yra galimas JAVA klasių įtraukimas į projektą.

Panagrinėkime elementarų daug-su-daug ryšį parodytą 3.12 paveiksle. Sakykime studentai gali turėti daug studijuojamų modulių, o kiekvienas modulis turi daug studentų. Tokiam ryšiui įgyvendinti reikia tarpinės lentelės, kurioje yra renkama informacija apie studentus ir modulius (kaip 3.13 pav. parodyta SĄRAŠAS; kaip atrodo įrašai DB parodyta 3.12 paveiksle). BA įrankis leidžia DBOM vedlio pagalba sukurti lentelių atvaizdavimą klasėmis. Kaip atvaizduojami duomenis priklauso nuo klasių kūrėjo. Viena iš galimų realizacijų kaip atvaizduoti tokią duomenų struktūra, pavaizduota 3.13 paveikslo dešinėje pusėje (klasė Studentas). Tokiam atvejui kiekvienam studentui sudaromas dvimatis masyvas su modulio pavadinimu ir identifikatoriumi (pirminiu raktu atitinkančiu modulį DB). Tačiau tokia klasė yra patogi tik objekto informacijos pateikimui, nes atsiradus poreikiui papildyti, atnaujinti ar pašalinti informaciją: susiduriama su sunkumais. Atvaizduojamos klasės informacijos atnaujinimo metodų naudojamas tampa problematiškas. Nes papildžius modulių masyvą, reikia programuotojui pačiam rūpintis apie informacijos atnaujinimą duomenų bazėje. Nes standartiniams metodams naujų duomenų įtraukimui reikia nurodyti pirminį raktą, tai galimas tik naujo studento įtraukimas, o jau modulių ir sąrašo lentelių standartiniais metodais papildyti neįmanoma. Taigi toks atvaizdavimas iš dalie yra neefektyvus.




**3.13 pav.** Daug-su-daug ryšio atvaizdavimas viena klase.

Galimas ir tiesioginis DB lentelių atvaizdavimas viena klase, kai kuriamos klasės objektas – kaupia visą informacija apie visas tris lenteles. Tokiu atveju kiekviena lentelės eilutė atitinka atvaizduojamos klasės objektą. Ir pasinaudojus *P* šablonų struktūra, galimas taisyklių rašymas tokiems sukurtiems objektams, ir tokiu atveju taip apjungiami vieno studento objektai. Tačiau ir toks būdas nėra efektyvus susidūrus su informacijos atnaujinimu.




**3.14 pav.** Ryšio atvaizdavimas klasėmis priklausiančiomis konkrečiai lentelei.

Ruošiant atvaizduojamąsias klases iš DB lentelių: ne tik informacijos pateikimui, tačiau ir duomenų pildymui, atnaujinimui, šalinimui. Naudinga kiekvieną lentelę atvaizduoti konkrečia klase. Tokiu atveju galimas duomenų filtravimas iš DB ir taip pat šias klases panaudoti su standartiniais metodais – lentelių įrašų atnaujinimui. Tokios 3.13 lentelių struktūros atvaizdavimas konkrečia klase pavaizduotas 3.14 paveiksle.

## 3.4. VT šablonų naudojimas grindžiamas transformuotomis DS

Norint užrašyti taisykles visiems egzistuojantiems tam tikro tipo objektams, įrankyje yra naudojama šablonų struktūra (angl. *Pattern* – toliau *P* šablonas). Šie šablonai palengvina sujungimą sudėtinių klasių ar interfeisų į vieną aibę, pagal tam tikrą tipą. Pirmiausia deklaruojama klasė ar interfeisas pasinaudojus šiais *P* šablonais, tada užrašomos taisyklės veikiančios kiekvieną objektą taisyklių variklyje. Paleidus taisyklių variklį vyksta taisyklių vykdymas, ir jeigu yra panaudojami *P* šablonai, tai tada ši struktūra yra nagrinėjama kaip sutrumpinimas notacijos apie apibrėžtus objektus.

Šie *P* šablonai palengvina programuotojo darbą, kai įrankyje yra kuriamos susietosios klasės su DB lentelėmis. Tokiais atvejais kai yra naudojami metodai objektų informacijai išgauti iš duomenų bazės. *P* šablonų kūrimas galimas dvejais būdais: pasinaudojus įrankio juosta arba SLR kalba. Abiem būdais deklaruojant šabloną reikia:

- Privaloma nurodyti unikalų vardą projekto viduje, ar taisyklių rinkinio viduje (jeigu naudojamas ne globalus; jei užrašoma SRL kalba naudoji žodeliai: *is any*)
- Privaloma apibrėžti tipą naudojamos klasės ar interfeiso. (pvz.: *S is any Studentas*)
- Jei reikia sudaromas sąrašas sąlygų ar apribojimų, ar nurodoma objektų kolekcija. (pvz.: *S is any Studentas such that Stipentija > 0*)

Sakykime duomenų bazėje turime studentų sąrašą priklausančių fakultetams. Iš duomenų bazės traukiama visa informacija apie studentus pasinaudojus metodu *Studentas.fetchAllInstances()*. Šis metodas sukuria įrankyje dinaminius studento objektus, o kiekis priklauso nuo esančių DB įrašų skaičiaus. Šablonas gali būti globus arba statinis, taip pat galima priskirti – taisyklių rinkiniams. Tarkime deklaruojame *P* šablonų struktūrą *S* taisyklių rinkiniui *Tikrinti* ir užrašoma taisyklė *TikrintiFakultetą*:

```
if the Fakultetas of S is equal to "Informatika"
then print("Studentas:" S's Vardas " " S's Pavardė " " S's Fakultetas).
```

Įvykdžius taisyklių rinkinį gaunamas norimas rezultatas, tai yra išspausdinami studentai su vardais pavardėmis ir norimo fakulteto pavadinimu. Toks būdas palengviną taisyklių rašymą. Priešingu atveju reikėtų kiekvienam objektui sukurti po atskirą taisyklę. Kaip jau buvo minėtą šablonai gali būti tiek globalus tiek priskirti taisyklių rinkiniams. Globalaus privalumas yra tas, kad galima naudoti tiek bet kokioje taisyklėje, tiek bet kokiame taisyklių rinkinyje. Ir taip objektai priskirti šiai struktūrai bus ribojami visame projekte – visais taisyklių rinkiniais, sprendimų medžiais ar lentelėmis. Statinis riboja objektus tik vienoje sukurtoje srityje, priklausomai kur yra kuriamas.

## 3.5. VT sprendimų priėmimo mechanizmas grindžiamas transformuotomis DS

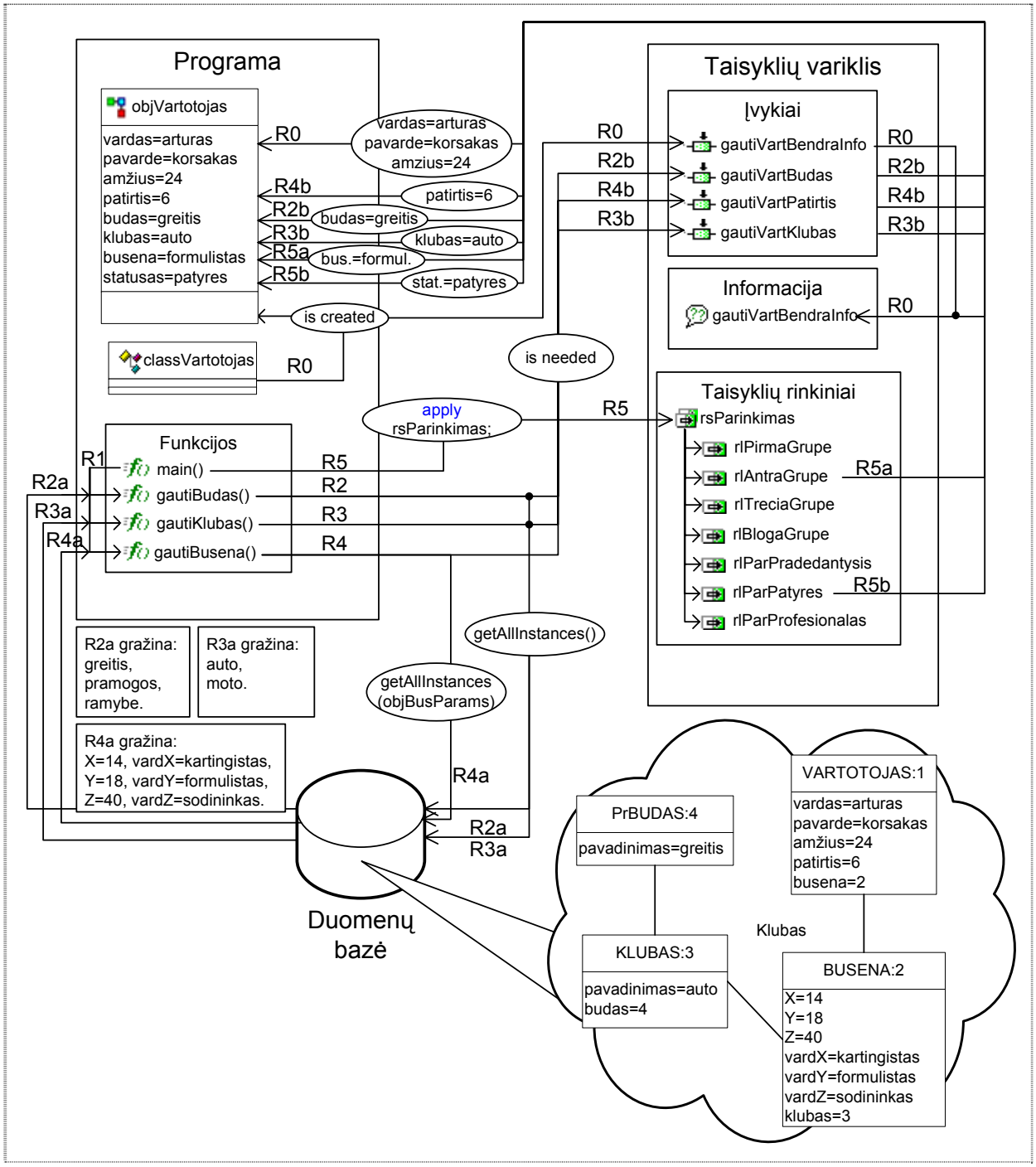
Kad įsivaizduotume, pati įrankio veikimo principą, pirmiausia pabandykime panagrinėti elementarų pavyzdį. Sakykime DB laikoma informacija apie skirtingų žmonių grupės sujungtas pagal tam tikrus pomėgius. Įrankis pagal vartotojo mėgstama praleidimo būdą parinks jam atitinkanti klubą. Šiame skyrelyje bus aptarta pagrindiniai sprendimų komponentai. Trumpa minimalios sistemos specifikacija ir „žingsnis po žingsnio“ kūrimo etapai pateikti 1 priede (pirmi 6 punktai: trumpa specifikacija, nuo 7: kūrimo etapai).

Kaip ir visuose programinės įrangos projektų kūrimo įrankiuose, pirmas žingsnis, yra projekto sukūrimas, kuris pavaizduotas 7 punkte. Kadangi šiame darbe pagrindinis informacijos šaltinis yra DB, tai iš to seka, kad tolimesniam darbui įgyvendinti prie įrankio reikia prijungti pasirinktą DB. Įrankio prijungimas prie *Microsoft SQL Server 2000* ir klasių suliejimas su DB esančiomis lentelėmis yra pateiktas 8 ir 9 punktuose. Duomenų filtravimui iš DB reikalingos papildomos funkcijos, kurios užpildo susietųjų klasių objektų informacija (funkcijos pateiktos 10 punkte). Šiuo atveju šių funkcijų paskirtis yra siusti duomenų bazei užklausas ir gautą informaciją užpildyti esamuose objektuose. Sekančiuose 11 ir 12 punktuose yra parodyta kaip sukuriami taisyklių rinkiniai ir kaip užsirašomos taisyklės ir įvykiai. Šių taisyklių paskirtis yra iš užpildytų objektų informacijos, gauti tam tikrą sprendimą (panaudojant taisykles) ar tiesiog iš vartotojo gauti papildomą informaciją (panaudojant įvykius). Įvykdžius šiuos pirmus 12 žingsnių gaunama, minimali sistemėlė, kurios rezultatas pateiktas 13 punkte. Toliau smulkiau panagrinėsime, kaip keičiasi duomenys einant nuo DB iki susietų klasių objektų ir veikiant šiuos objektus taisyklėmis.

### 3.5.1. Taisyklių rinkinių naudojimas

Kai paleidžiamas programos veikimas, pirmiausiai sukuriamas klasės *classVartotojas* objektas *objVartotojas* (toliau OV). Tada iškviečiami įvykiai su parametru „*is created*“(žr. 3.1 lentelę), šiuo atveju parodytas ryšys R0, kai įvyksta įvykis *gautiVartBendraInfo*. Toliau šis įvykis iškviečia klausimų seką, kurioje vartotojas įveda duomenis, ir išsaugo

*objVartotojas* savybių reikšmes (kaip parodyta R0 ryšyje). Minimalios sistemos srautai įrankyje parodyti 3.15 paveiksle.




**3.15 pav.** Bendra sistemos komunikacija, panaudojus taisyklių rinkinius

Tolesnis programos veikimas, valdomas pagrindine funkcija *main()* (pavadinimas privalomas kaip parodytas ir negali būti pakeistas). Pagrindinė funkcija iškviečia kitas funkcijas, kurios manipuliuoja duomenis. Pirmiausia iškviečiama funkcija *gautiBudas()*(F1), kuri kreipiasi į standartinį įrankio metodą *getAllInstances()*, kuris siunčia duomenų bazei užklausa ir gauną atsakymą (kaip parodytą R2a ryšyje). Toliau pagal gauta informacija vartotojui parodoma pasirinkimų galimybė, iškviečiamas įvykis *gautiVartBudas* su parametru „*is needed*“ (tai reiškia, kad šis įvykis yra kviečiamas tik

tada, kai programai yra būtina užpildyti vieną ar kitą parametrą). Kai vartotojas pasirinka vieną būdą iš kelėtos galimų, tai pasirinkta reikšmė įrašoma į objekto OV esybę (parodyta R2b ryšyje). Funkcijos *gautiKlubas()*(F2) veikimo principas yra panašus į funkcijos F1 veikimą, kai siunčiama užklausa parodyta R3a, o vartotojo informacijai gauti R3b ryšiuose. Kai funkcija F2 baigia darbą toliau iškviečiama funkcija *gautiBusena()*(F3), kurios principas yra panašus į prieš tai aptartas funkcijas (R4 ryšys), skiriasi tik kreipimasis į duomenų baze. Funkcija *getAllInstances()* iškviečiama su papildomu apribojimo parametru, kuris sukuriamas rašant užklausa į duomenų bazę (žr. 1 priedo 8 punkto 10 žingsnyje).

Kai yra suvesta pilna informacija apie vartotoją, tai data iškviečiamas taisyklių variklyje esantis taisyklių rinkinys komandos *apply* pagalba. Šioje minimalioje sistemoje iš pagrindinės funkcijos iškviečiamas taisyklių rinkinys su komanda „a*pply rsPasirinkimas"* (parodyta R5). Toliau vykdomos veiklos taisyklės ir kaip parodyta R5a ir R5b ryšiuose užpildomos objekto OV savybės. Įvykdžius šią sistemą gaunamas norimas rezultatas, kuris yra 1 priede 13 punkte. Sistema parinko vartotojui, pagal jo paliktą informacija, būseną ir statusą grupėje.

**3.7 lentelė.** Įvykių iškvietimo reikšmės.

| **Įvykio iškvietimo parametras** | **Reikšmė** |
|---|---|
| *is changed* | naudojamas, kai keičiasi informacija, ir reikia pakeisti su tuo objektu, parametru, esybe susijusią kitą informaciją |
| *is created* | naudojamas, kai sukuriamas ar susiejamas (*mapping*) objektas; kviečiamas anksčiau už *is initialized* |
| *is deleted* | naudojamas, prieš panaikinant (ištrinant) ar atsiejant (*unmapping*) objektą |
| *is initialized* | iškviečiamas po *is created*, naudojamas inicializuoti objektą |
| *occurs* | naudojamas iškviesti išorinius įvykius |
| *is needed* | naudojamas, kai reikalinga informacija, kitu atveju reikšmė yra *unknown* |

Nagrinėjamame pavyzdyje yra taikomos veiklos taisyklės – taisyklių rinkiniai. Tokiam pat rezultatui galima gauti taisyklių rinkinius pakeitus sprendimo lentelėmis ar sprendimų medžiais (toliau atitinkamai lentelės ar medžiai).

### 3.5.2. Sprendimo medžių ir lentelių naudojimas

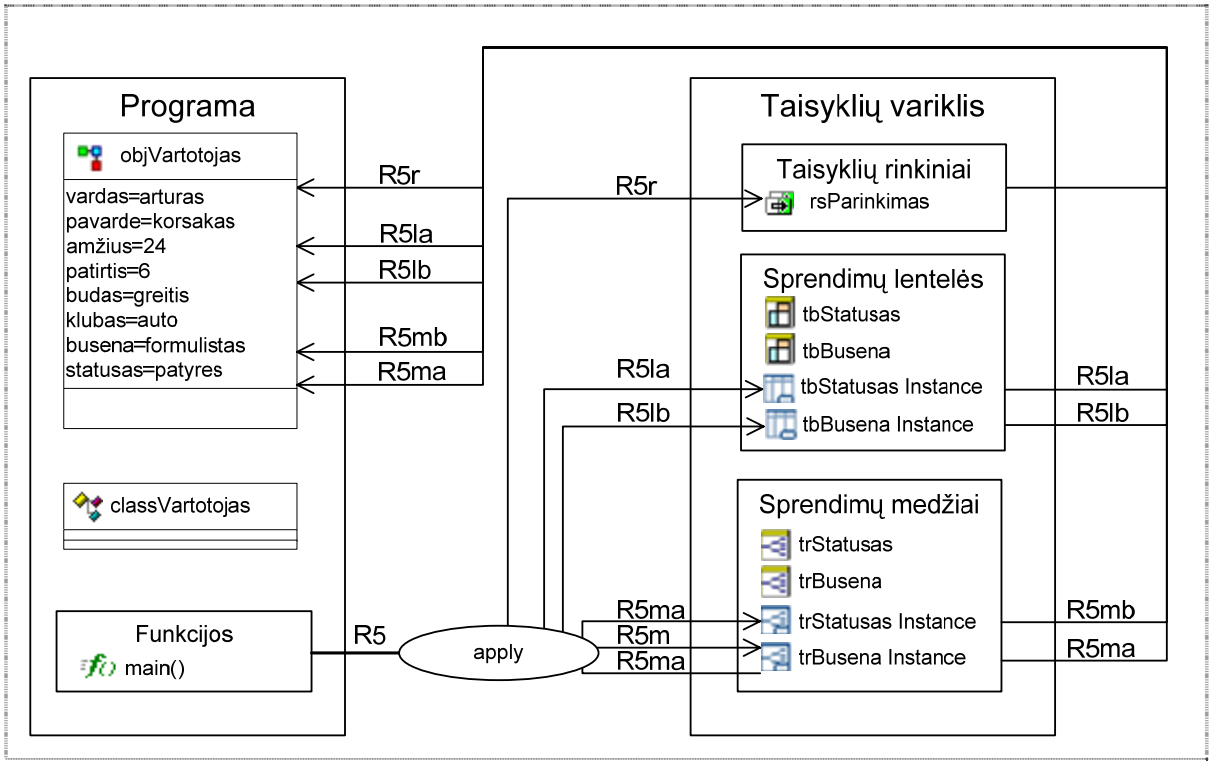
Medžių ir lentelių komponentai yra kaip taisyklių rinkinių praplėtimas, kitaip tariant, tai ir yra taisyklių rinkiniai su savomis ypatybėmis (praplėstas grafiniu apipavidalinimu). Parodytame 3.15 paveiksle komunikacijos R5 ryšį apibendrinsime ne tik su taisyklių rinkiniais, tačiau ir su lentelėmis, ir medžiais (atitinkamai kūrimo žingsniai parodyti 1 priedo 14 ir 15 punktuose).

Įvedus į minimalią sistemą medžius ir lenteles (žr. 3.16 paveikslą) yra matoma, kad panaudojus tris skirtingus komponentus – gaunamas rezultatas yra identiškas, su vienodais

duomenimis. Iš funkcijų ar tam tikrų įvykių galime iškviesti sprendimo komponentus, ar net iš pačių savęs – kitus sprendimo komponentus.

Šių trijų komponentų iškvietimas vyksta komanda *apply* ir pridedant sprendimo vardą, ir reikalingus parametrus, pavyzdžiui:

- taisyklių rinkiniai – *apply rsParinkimas*;
- sprendimų lentelės – *apply tbStatusas_Instance(objVartotojas)*;
- sprendimų medžiai – *apply trBusena_Instance(objVartotojas, objBusena)*.

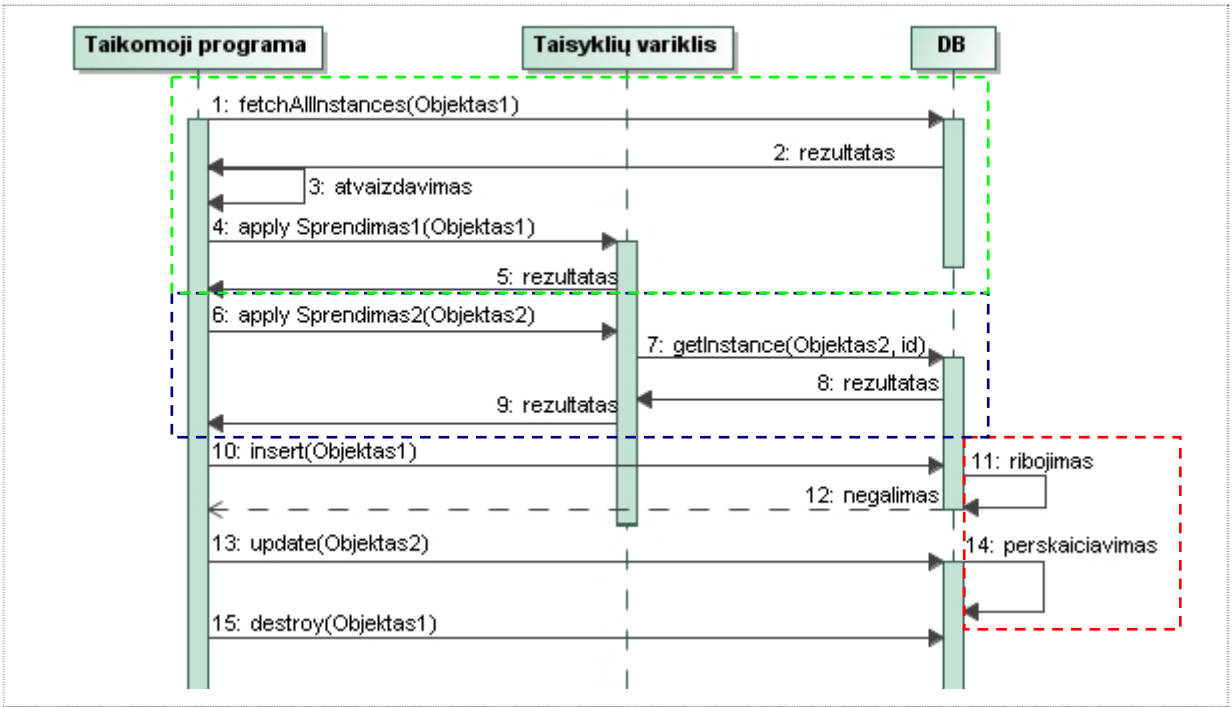



**3.16 pav.** Sistemos komunikacija panaudojus rinkinius, lenteles ar medžius

Šių trijų komponentų tikslas yra vienas ir pagrindinis – pagal tam tikrus dėsnius atlikti vieną ar kitą sprendimą. Visi trys komponentai gali dirbti tiek vieni atskyrai, tiek bendrai iškviečiant kitokio pobūdžio sprendimus. Sprendimų lentelės ir medžiai yra kaip praplėtimas taisyklių rinkinių. Kadangi kuriant didelį projektą, kur yra ne viena veiklos taisyklė, o dešimtys, šimtai ar tūkstančių, tai yra problematiška sekti visas veiklos taisykles. Todėl yra kuriami tokie įrankiai kaip *Blaze Advisor*, *JRules, HaleyAuthority, JBoss* ir kiti. Jeigu susidaro daug veiklos taisyklių, tai patogu grupuoti jas į taisyklių rinkinius, tačiau sunku yra peržvelgti taisyklių rinkinius ar jų tam tikrus sprendimus. Tuo tarpu sprendimų lentelės ir medžiai yra patogus sekti tam tikras sprendimų reikšmes. Sprendimų lentelės tinka mažesnėms taisyklių grupėms sekti, nes kuo daugiau taisyklių tuo lentelių matmenys didėja ir sunkėja peržiūra. O žiūrint į sprendimų medžius, yra aiškiai matomi kaip vyksta tam tikri sprendimai, kokie duomenys jais „vaikšto“ ir panašiai. Sprendimo medžiai yra skirti didesniam taisyklių rinkiniui užrašyti.

## 3.6. VT variklio ir transformuotų DS sąveika

Sekų diagramoje pavaizduotoje 3.17 paveiksle yra parodoma nedetalizuota sąveika tarp trijų elementų: taikomosios programos, taisyklių variklio ir duomenų bazės. Kaip parodyta žalia linija (atliekami 5 žingsniai), bet kuriuo metu iš taikomosios programos galima siusti užklausą į duomenų bazę. Tuo tarpu DB gražina rezultatą ir jis atvaizduojamas dinaminiu objektu. Objekto laukai yra užpildyti, taigi galimas sprendimo iškvietimas. Priklausomai nuo sprendimo pobūdžio (tai gali būti taisyklių rinkiniai ar sprendimo medžiai ar lentelės) atliekami veiksmai ir gražinamas (arba nebūtinai) rezultatas. Mėlyna linija pažymėti 6-9 žingsniai parodo, kad ir iš taisyklių variklio galimas užklausų siuntimas (ne tik iš taikomosios programos).

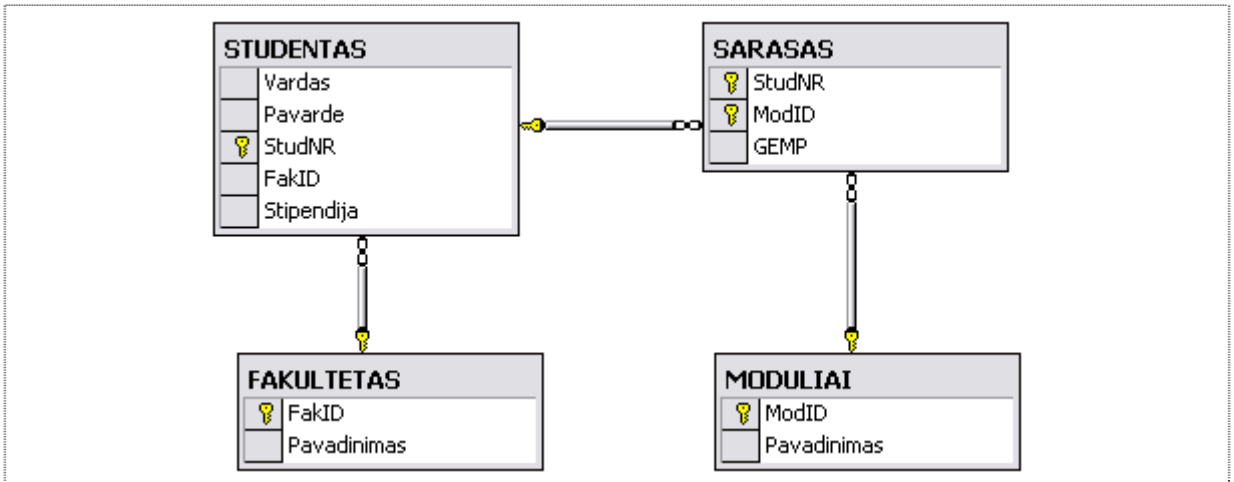



**3.17 pav.** Sekų diagrama: Sąveika.

Taigi taikomoji programa iškviečia tam tikrą sprendimą, o papildomai informacijai išgauti taisyklių variklis kreipiasi dar ir į DB (sprendimo metu gali būti iškviečiami ir įvykiai papildomai informacijai gauti iš vartotojo). Nebūtinai visos taisyklės ar sprendimai yra įgyvendinami įrankio komponentais – galimas ir trigeriu elgsenos ribojimas ar veiksmai (kaip parodyta raudona linija).

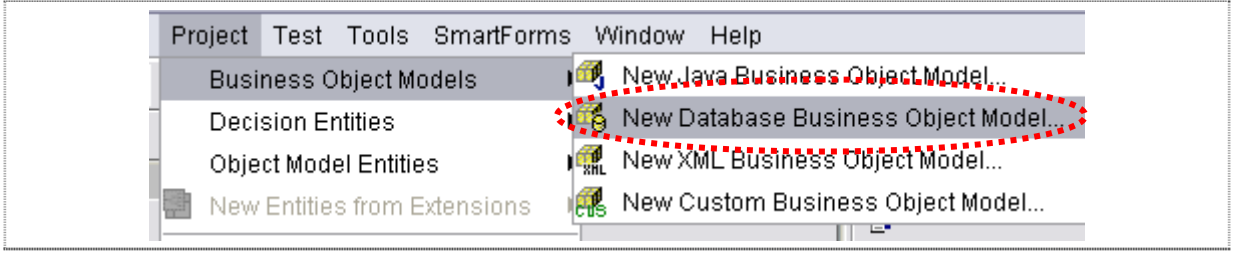
# 4. DS TRANSFORMAVIMO Į VT KŪRIMO APLINKĄ EKSPERIMENTAS

Panagrinėkime *Blaze Advisor* įrankyje lentelių daug-su-daug ryšio atvaizdavimą klasėmis. Imkime 3.17 paveikslėlyje esamą lentelių schemą, ir papildykime sąrašo lentelę įvesdami stulpeli su studento galutiniu modulio egzamino pažymių, ir taip pat studento lentelėje – stipendijos stulpelį. Su tokia pakeista schema galėsime atlikti sprendimą: pagal studento vidurkį parinkti stipendiją ir jos reikšmę įrašyti į DB. Kadangi studentų egzaminų pažymių saugojimas yra perteklinis, tai vidurkio reikšmė DB yra nesaugoma. Vidurki priskiriame atvaizduojamai studento klasei, kuri apskaičiuosime pasinaudojus taisyklių rinkiniu. Galimas ir kitoks vidurkio apskaičiavimas pasinaudojus pačiomis DB galimybėmis rašant transakcijas ar priskirtąsias procedūras. Taigi toliau yra nagrinėjama 4.1 paveiksle pateiktą duomenų bazės lentelių diagrama.



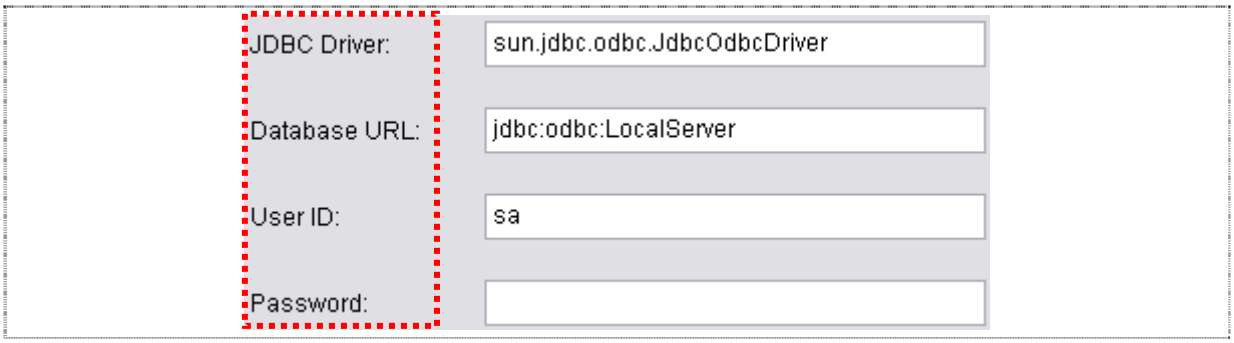

**4.1 pav.** DB lentelės.

Sukuriamas naujas duomenų atvaizdavimas (žr. 4.2 pav.): *Project* ➤ *Business Object Models* ➤ *New Database Business Object Model*. Šio DBOM vedlio paskirtis yra sukurti prisijungimą prie duomenų bazės ir atvaizduoti duomenų bazės lenteles klasėmis.



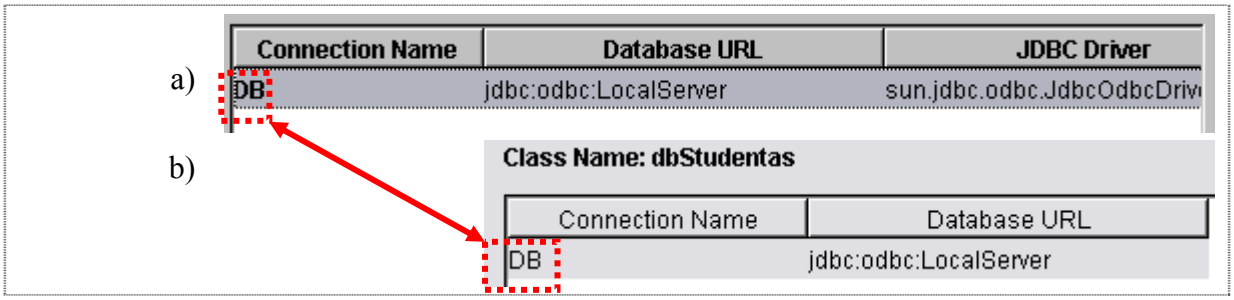

**4.2 pav.** Sukuriamas naujas DBOM.

Toliau vystomas prisijungimas prie duomenų bazės, kuris vyksta per Java duomenų bazės jungimosi valdykle, nurodoma duomenų nuoroda ir vartotojo prisijungimo duomenys (žr. 4.3 pav.). Kiekviena duomenų bazė turi savo prisijungimo principus. Šiame darbe yra prisijungiama per ODBC valdyklę.



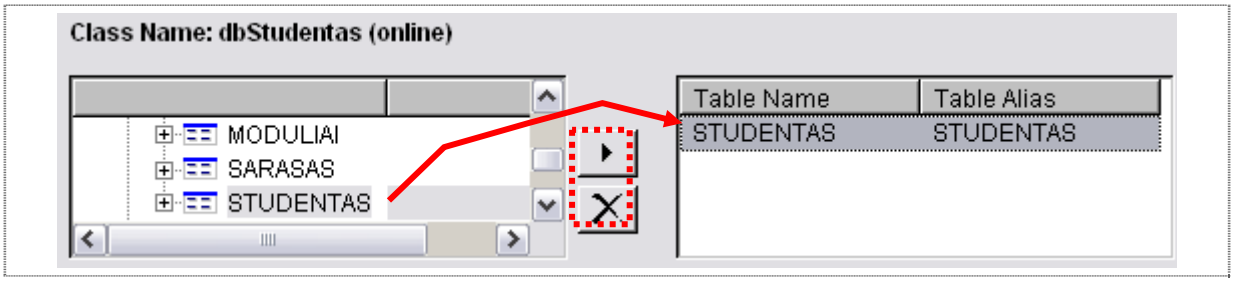

**4.3 pav.** Prisijungimas prie DB.

Kai sukuriamas prisijungimas prie DB (žr. 4.4 pav. a dalį), toliau seka vienas iš svarbiausių dalių, t.y klasių sudarymas. Įtraukiama nauja klasė su savo pavadinimu (patartina prieš kiekvieną klasės pavadinimą, kuri atvaizduoja DB lentelę, naudoti raidžių kombinacija *db*, tokiu būdu toliau įrankyje aiškiai matoma atvaizduojama klasė). Pradinėje DBOM dalyje yra nurodoma iš kokios duomenų bazės yra kuriama klasė (žr. 4.4 pav. b dalį).



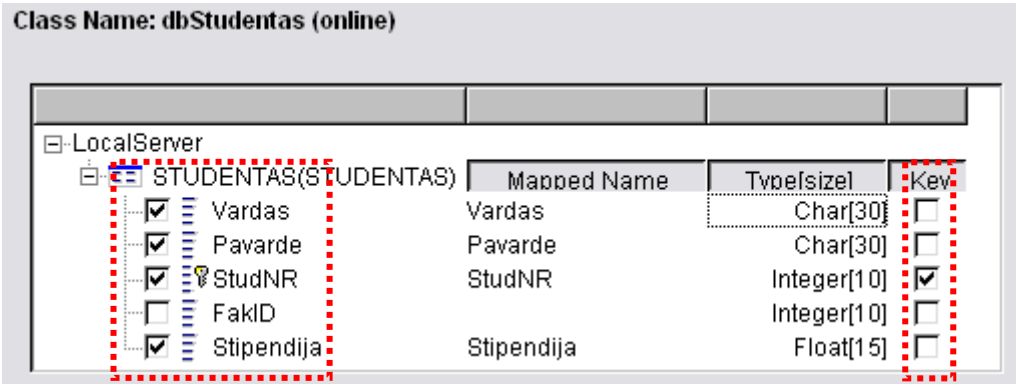

**4.4 pav.** Atvaizdavime pasirenkama DB.

Sekančioje dalyje pasirenkamos lentelės iš kurių yra konstruojama atvaizduojama klasė (žr. 4.5 pav.). Šiuo atveju užtenka vienos lentelės, nes pagal specifikaciją kiekviena lentelė turi savo priklausančią klasę. Taigi įkėlimą tik STUDENTO lentelė, tam kad būtų galimi panaudoti standartiniai BA įrankio filtravimo ir atnaujinimo metodai. Šioje dalyje taip pat galima atvaizduoti ne tik lenteles, tačiau ir duomenų bazės vaizdus. Šiuo atveju nėra naudojamos priskirtosios procedūros, tačiau jeigu reikėtu sukurti ribojimų klases, tai šia dalį reikėtų praleisti „nepaliestą“.



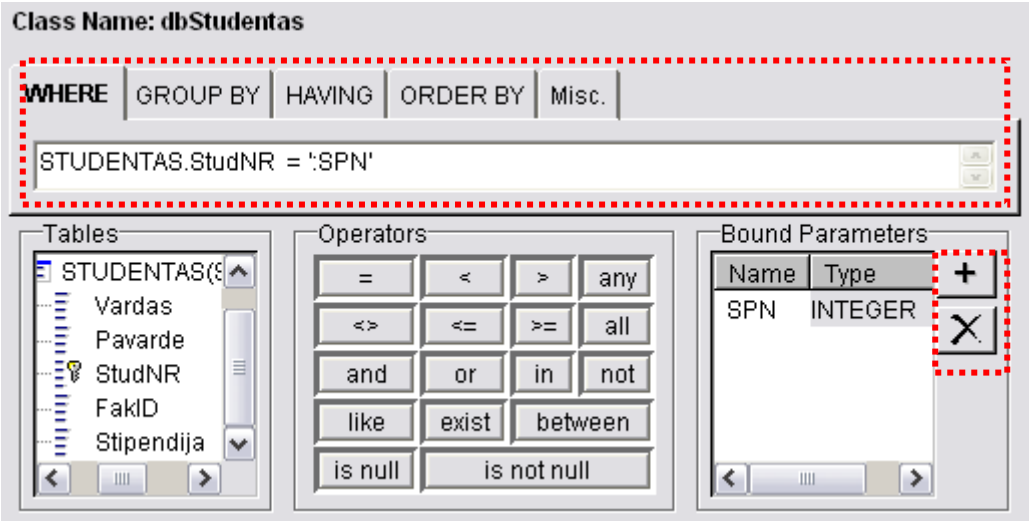

**4.5 pav.** DB lentelių pasirinkimas.

Toliau einant DBOM vedliu yra pasirenkami reikalingi lentelės stulpeliai ir pervadinami pagal poreiki pavadinimai (žr. 4.6 pav.). Taip pat labai svarbu nepamiršti pažymėti varnele pirminių raktų, pagal kuriuos vėliau sukuriamos papildomos klasės. Šių klasių pagalba BA įtraukinėja įrašus į duomenų baze nurodydamas pirminių raktų reikšmes.



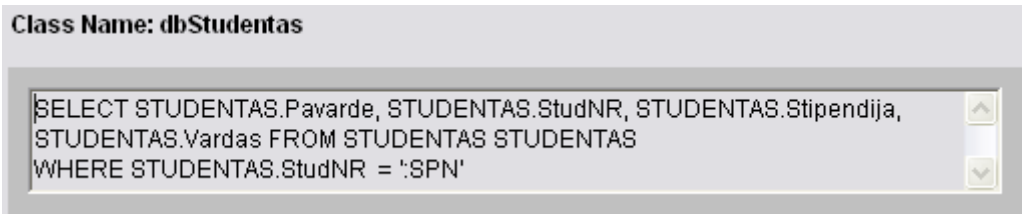

**4.6 pav.** DB lentelės laukų pasirinkimas.

Paskutinė ir bene svarbiausia DBOM vedliu yra užrašinėti likusią SQL užklausos dalį. Išsirenkami reikalingi lentelių stulpeliai ir operatoriai, ir iš jų sudaromos likusios užklausos dalys, tokios kaip: *WHERE, GROUP BY, ORDER BY* ir kiti (žr. 4.7 pav.). Šiam atvejui pakanka filtruoti duomenis pagal studento numerį, tačiau tam reikalinga sukurti ribojimų klasę. Tai atliekama įtraukiant *Bound Parameters* naują parametrą nurodant: parametro vardą ir duomenų tipą.



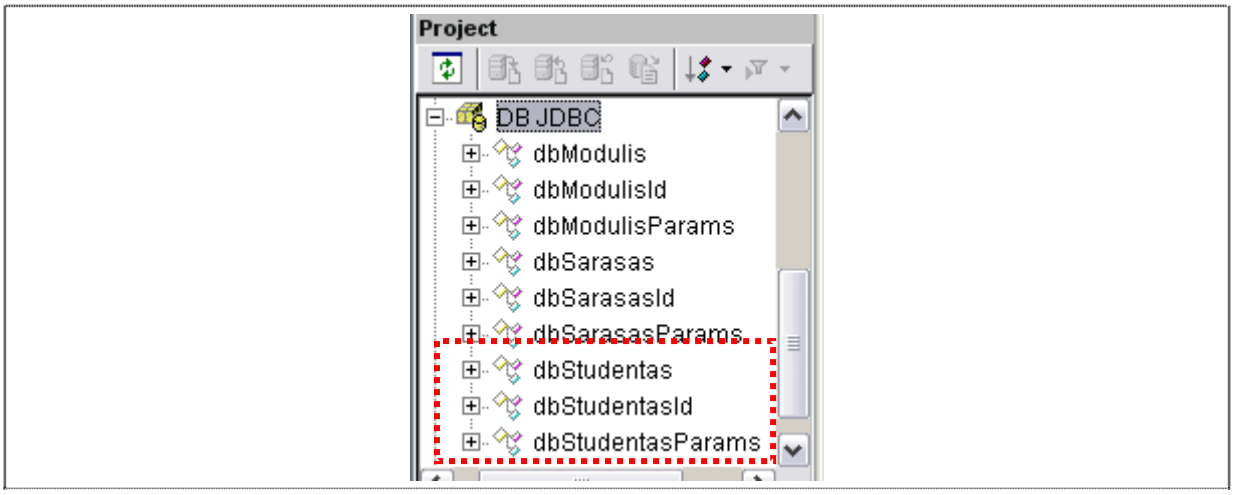

**4.7 pav.** Pildoma užklausa su papildomais ribojimo parametrais.

Kai užbaigiamas užklausos rašymas *Blaze Advisor* parodo sukonstruotą SQL užklausą (žr. 4.8 pav.). Tokiu būdu kai SQL užklausos yra ganėtinai didelės, tai peržiūrėjus visą SQL užklausos kodą ir randus klaidu grįžtama į DBOM vedlį jas ištaisyti. Šiame pavyzdyje užklausa yra elementari, kuri riboja tik studento numerį. Šią užklausą įvykdžius gaunami minimalus duomenys apie studentą.




**4.8 pav.** Gauta SQL užklausa DBOM vedliu.

Kai yra sukonstruojamos SQL užklausos, kuriose yra pažymėti pirminiai raktai, taip pat įtraukiami ribojimo parametrai, tai Blaze Advisor sukuria 3 klases (žr. 4.9 pav.). Sukuriama pagrindinė atvaizduojamoji klasė su pasirinktu pavadinimu (pvz. dbStudentas). Jeigu yra pažymėti pirminiai raktai, tai sukuriama klasė su pasirinktu pavadinimu ir gale pridedama *Id* (jeigu pasirenkamas pavadinimas dbStudentas, tai papildomos klasės pavadinimas yra dbStudentasId). Atitinkamai ir su ribojimo parametrai, kai gale prikabinamas žodelis *Params* (pvz. dbStudentasParams). Taigi atliekami tokie patys žingsniai su kitomis sąrašo ir modulio lentelėmis, kaip buvo atlikta su studento ir gaunamos 9 klasės pavaizduotos 4.9 paveiksle.

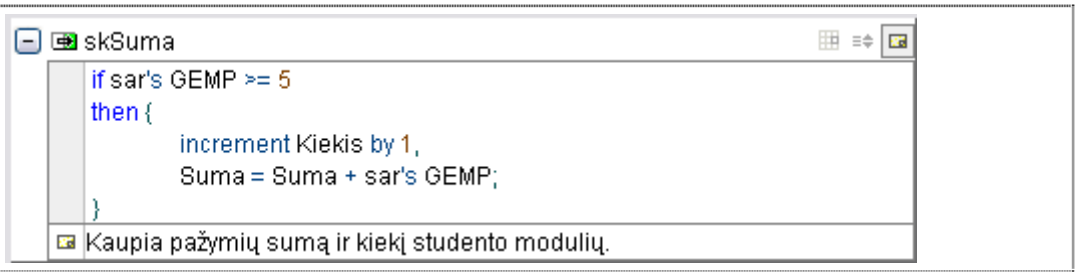



**4.9 pav.** Sukurtos atvaizduojamos klasės.

Kadangi į atvaizduojamas klases įtraukinėti papildomų laukų *Blaze Advisor* neleidžia, tai tolimesniam darbui reikia sukurti naują vaikinę klasę tėvinei atvaizduojamai studento klasei. Tokiu būdu yra galimas papildomų laukų įtraukimas, šiuo atveju reikia įvesti studento pažymių vidurki. Tiek su tėvine tiek su vaikine klase yra galimas standartinių metodų iškvietimas.

Kai paruošiamos klasės darbui su duomenimis, sekančiame etape yra vystomos veiklos taisyklės. Šiuo atveju yra panaudotos elementarios taisyklės:

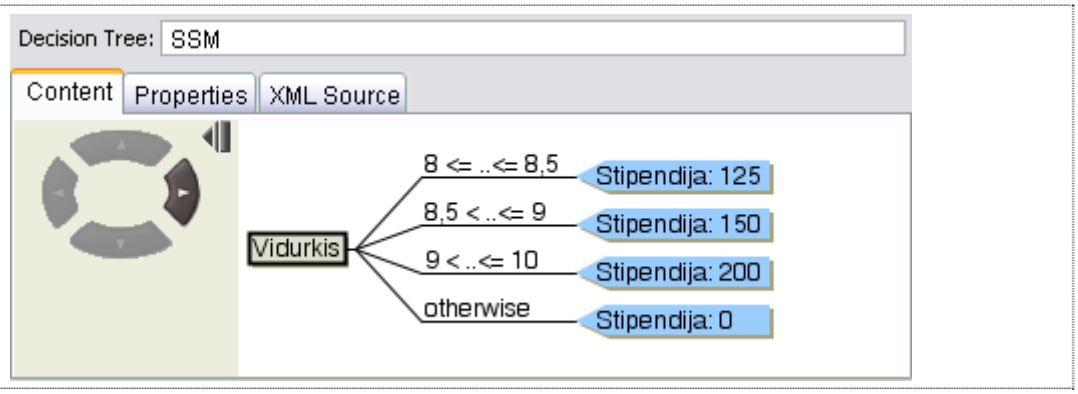
1. Kad galėtume apjungti vienodo tipo objektus, kurie kaupia studento modulių egzaminų pažymius – panaudojamas „pattern“ šablonas. Taip užtenka užrašyti vieną taisyklę visiems šio tipo objektams. Taigi 4.10 paveiksle pavaizduota taisyklė kaupia sumą ir kiekį, jeigu modulio pažymys didesni arba lygus 5, pagal formules:
   - $Suma = \sum_{i=1}^{n} GEMP_n;$ – kaupiama pažymių suma;
   - $Kiekis = Kiekis + 1;$ – skaičiuojamas kiekis pažymių.




**4.10 pav.** Taisyklė kaupią sumą ir skaičiuoja pažymių kiekį.

2. Studento stipendijos parinkimui yra sukuriamas elementarus vieno lygio medis užrašytas sprendimo medžiu(žr. 4.11 pav.), pagal formule:

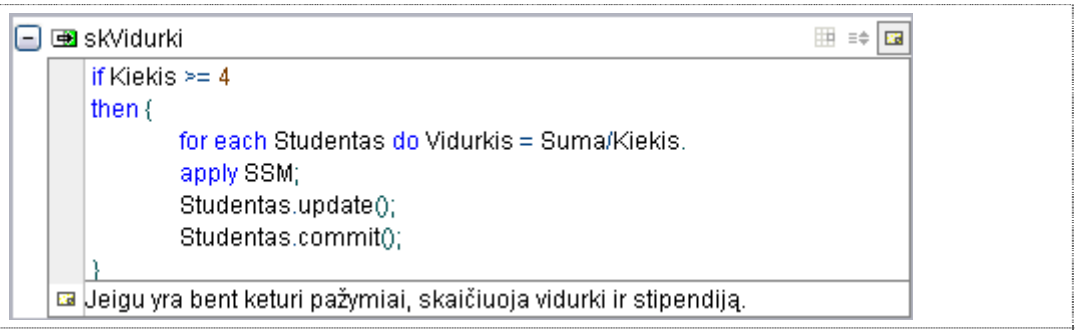
   - $Stipendija = \begin{cases} 200, & 9 < Vidurkis \le 10 \\ 150, & 8{,}5 < Vidurkis \le 9 \\ 125, & 8 \le Vidurkis \le 8{,}5 \\ 0, & kitaip \end{cases}.$




**4.11 pav.** Minimalus sprendimų medis apskaičiuojantis stipendiją.

3. Pagrindinė taisyklė kurioje skaičiuojamas ne tik sukauptas vidurkis pagal sumą ir kiekį, tačiau ir iškviečiamas sprendimų medis ir atnaujinama reikšmė duomenų bazėje. Vidurkis skaičiuojamas, jeigu pažymių kiekis didesnis arba lygus negu 4, pagal formulę:

   - $Vidurkis = \frac{Suma}{Kiekis}$.




**4.12 pav.** Taisyklė skaičiuojanti vidurkį.

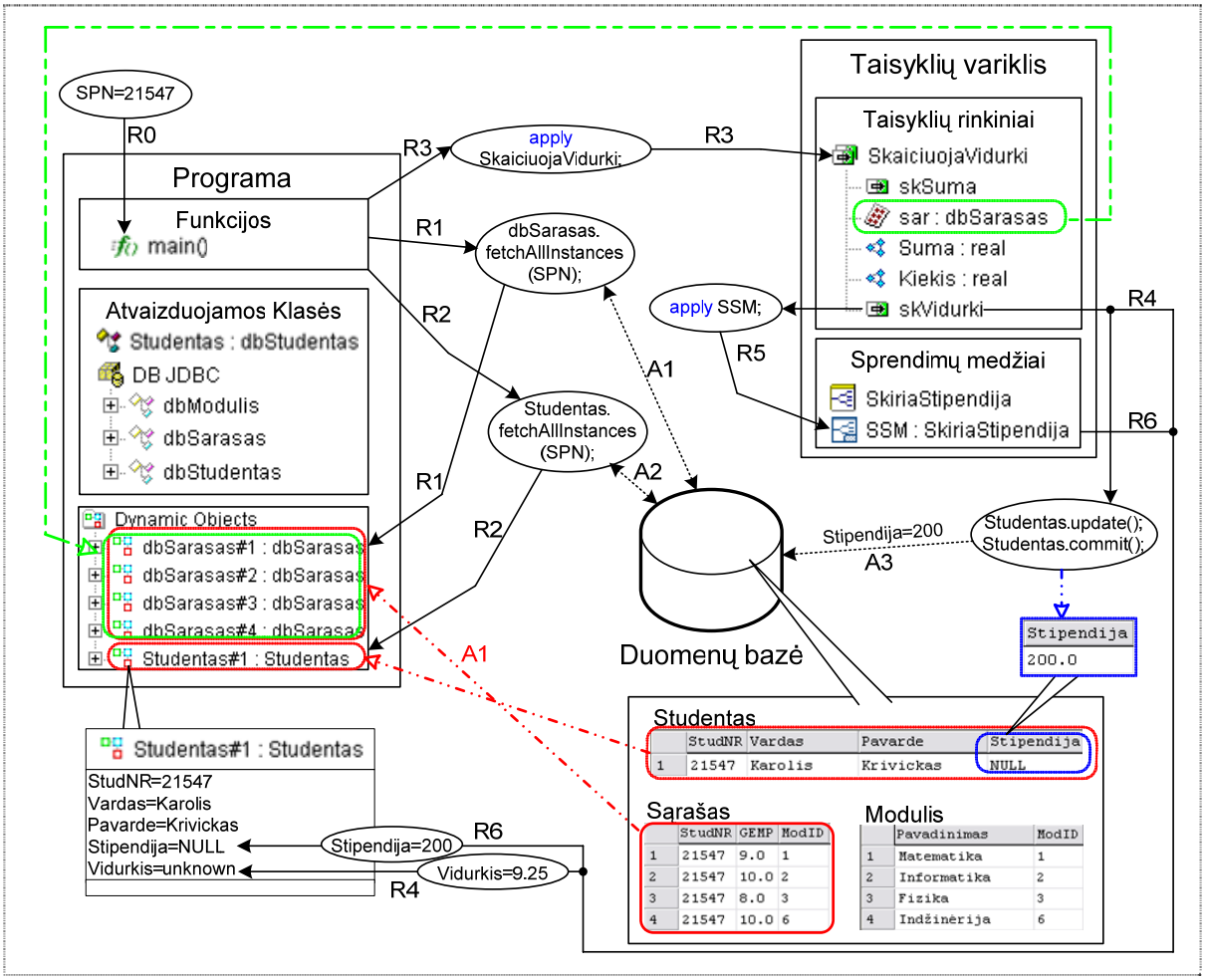
VTVS sistemų kuriamų IS pagrindinis tikslas yra valdyti, redaguoti ir palaikyti VT, kai yra didelis skaičius taisyklių. Šiuo pavyzdžiu yra norima parodyti, tik pagrindinį principą, kaip VT yra maitinamos duomenimis iš DB. Kaip duomenų bazės esantys duomenys

paverčiami informacija ir pagal VT yra atlieka sprendimai. Šiam pavyzdžiui pakanka vieno taisyklių rinkinio su dviem taisyklėm ir vienu sprendimu medžiu. Šiais sprendimais yra parenkama stipendija pagal studento vidurkį.

**4.1 lentelė.** Pagrindinė *main()* funkcija.

```
1.    SPN3 is a dbSarasasParams.
2.    SPN3.SPN = SPN.
3.    dbSarasas.fetchAllInstances(SPN3);
4.    SPN1 is a dbStudentasParams,
5.    SPN1.SPN = SPN,
6.    Studentas.fetchAllInstances(SPN1);
7.    apply SkaiciuojaVidurki;
```

Duomenų filtravimui yra naudojamas standartinis metodas `fetchAllInstances(SPN)` su parametru, kuris kreipiasi į duomenų bazę ir gauna duomenis, kuriuos įrankis atvaizduoja objektinėmis klasėmis. Pagrindinės funkcijos programinis kodas parodytas 4.1 lentelėje. Šis sukurtas pavyzdys iliustruoja minimalią sistemą, kurios pagalba yra parodomas Blaze Advisor sistemos sąveikos esmę, t.y. kaip maitinamos veiklos taisyklės duomenimis.




**4.13 pav.** Galimas ryšio daug-su-daug atvaizdavimas BA įrankiu.

Toliau yra plačiau nagrinėjamas 4.13 paveikslas. Pagal studento pažymėjimo numerį sistema filtruoja duomenis iš DB (R0, R1, R2). Kaip parodyta A1 ir A2 ryšiuose metodai (A tipo; Skyriaus 3.3.2. Duomenų filtravimas, 3.5 lentelė) siunčią užklausą duomenų bazei ir gautus įrašus verčia dinaminiais atvaizduojamų klasių objektais (schemoje pavaizduota

raudona linija). Tokių atvaizduojamų klasių laukai programos vykdymo metu (kai keičiamos reikšmės) – lengvai atnaujinami duomenų bazėje. Pasinaudojus standartiniais A tipo metodais: *uptdate(), insert()*.

Kaip buvo nagrinėjami *P* šablonai (skyrelyje 3.4. VT šablonų naudojimas grindžiamas transformuotomis DS), tai šiame pavyzdyje akivaizdžiai matomas tokių naudojamų šablonų privalumas. Kaip matome iškvietus A1 ryšiu užklausą iš DB – programoje yra sukuriami 4 dinaminiai sąrašo objektai (pažymėti raudona linija). Šių objektų laukai yra susieti su DB sąrašo lentele ir jos eilutėmis. Iškvietus taisyklių rinkinį (R3) šablonas *sar* apjungia dinaminius objektus (4.13 paveiksle pavaizduota žalia linija), ir suveikus taisyklei skaičiuoja sumą (skSuma) – suskaičiuojami visų keturių dinaminių objektų informacija (šiuo atvejų visų pažymių suma ir kiekis). Taip sutaupomas taisyklių skaičius, nes užrašoma tik viena bendra taisyklė visiems objektams apibrėžtiems *sar* šablonu. Kitokiu atveju reikėtų užrašyti taisykles kiekvienam objektui atskyrai. Kai suskaičiuojama pažymių suma ir kiekis – iškviečiama taisyklė skaičiuojanti vidurkį (skVidurki) ir išsaugoma reikšmė objekto *Studentas* lauke. Kai apskaičiuojamas vidurkis – iškviečiamas sprendimų medis skaičiuojantis stipendija, pagal tam tikras nustatytas taisykles. Kadangi šis pavyzdys yra tik mokomasis, taigi nėra duomenų bazėje įvykių ribojančių vientisumą (kai vaikinės eilutės negali egzistuoti be tėvinių, ypač kai ribojimas atnaujinimas pirminio rakto). Kaip jau buvo minėta anksčiau atnaujinimo metodas *update()* – klasės laukus atvaizduojančius stulpelius atnaujina visus kartu – vienu metu. Taigi atnaujinama ir pirminio rakto reikšmė, nors ir vykdymo metu nebuvo pakeista. Ir jeigu būtu veikiantis toks trigeris, tai metodo *update()* tokiu atveju negalėtume naudoti. Šiame pavyzdyje šis metodas sėkmingai atnaujina reikšmes lentelėje Studentas (A3 ryšiu, schemoje pavaizduota mėlyna linija).

Toliau panagrinėsime kaip iš dviejų lentelių (*Studentas* ir *Fakultetas*) išgaunama informacija visais galimais *Blaze Advisor* filtravimo būdais. Tam atlikti buvo parašytas programinis kodas (žr. 4.2 lentelę), kurio rezultatas buvo identiškas visiem skirtingiems variantams. Nors visais atvejais yra gaunamas toks pat atsakymas, tačiau kiekvienas iš būdų turi savo privalumų ir trūkumų. Duomenų įtraukimo kodas yra užrašytas 4.3 lentelėje.

**4.2 lentelė.** Duomenų filtravimas.

**1) Standartiniu metodu *fetchAllInstances()***

```
1.  call fetchAllInstances on Studentas.
2.  Studentas.fetchAllInstances().
3.  for each Studentas do {
4.     print("Studentas: "Vardas" "Pavarde" "StudNumeris" "Fakultetas). }
```

**2) Standartiniu metodu *getAllInstances()***

```
1.  stud is some fixed array of Studentas.
2.  set stud to the outcome of getAllInstances on Studentas.
3.  stud = Studentas.getAllInstances().
4.  for each Studentas in stud do {
5.     print("Studentas: "Vardas" "Pavarde" "StudNumeris" "Fakultetas). }
```

**3) Užrašant SQL užklausas arba panaudojus priskirtąsias procedūras**

```
1.   Studentas.connect().
2.   gautas is a boolean. strSQL is a string. rez is some DbResultSet.
3.   strSQL = "SELECT STUDENTAS.Vardas, STUDENTAS.Pavarde, ",
4.   strSQL = strSQL "STUDENTAS.StudNR, FAKULTETAS.Pavadinimas ",
5.   strSQL = strSQL "FROM FAKULTETAS FAKULTETAS, STUDENTAS STUDENTAS ",
6.   strSQL = strSQL "WHERE FAKULTETAS.FakID  =  STUDENTAS.FakID",
7.   set rez to the outcome of executeQuery on Studentas using strSQL.
8.   strSQL = "gautiStudentas",
9.   rez = Studentas.executeCall(strSQL).
10.  set gautas to the outcome of next on rez.
11.  while gautas do {
12.     print("Studentas: "rez.getString(1)" "rez.getString(2)" "),
13.     print(rez.getInt(3)" "rez.getString(4)).
14.     set gautas to the outcome of next on rezultatas. }
15.  Studentas.disconnect().
```

Pirmas ir antras duomenų filtravimo būdai yra išties artimi vienas kitam, pagrindinis skirtumas yra tai, kad pirmuoju būdu rezultatas yra atvaizduojamas dinaminiais objektais, o antruoju yra gražinamas tam tikro ilgio masyvas su duomenimis. Pirmo būdo privalumas yra tas, kad su šiuo metodu galima naudoti „pattern“ šablonus. Trečias būdas yra apjungtas į SQL užklausų vykdymą ir priskirtųjų procedūrų iškvietimą. Naudojant trečią būdą vienas iš minusu yra tai, kad susidaro ganėtinai ilgas programinis kodas, kai išvystomos ne viena, o šimtai užklausų. Tačiau tiek pirmais, tiek trečiu būdu galima gauti tokį patį rezultatą, taigi visi filtravimo būdai yra tinkami naudojimui. Elementarus pavyzdys kaip įtraukiami nauji studento duomenys įgyvendintas standartiniais Blaze Advisor metodais, pavaizduotas 4.3 lentelėje.

**4.3 lentelė.** Duomenų įtraukimas.

```
1.   set NaujasStudentas to the outcome of newInstance on Studentas,
2.   set NaujasStudentoPK's StudNumeris to NaujasSudNumeris.
3.   call insert on NaujasStudentas using NaujasStudentoPK.
4.   call update on Studentas,
5.   call commit on Studentas.
```

Taigi eksperimento metu buvo išsiaiškinta, kaip Blaze Advisor įgyvendinta IS manipuliuoja duomenimis iš duomenų bazės, tai yra kaip duomenų bazės struktūra yra transformuoja į VT sistemą. Kaip filtruojami duomenys ir formuojami į informaciją, pagal kuriuos yra priimami tam tikri sprendimai. Duomenų filtravimas yra įvairus, programuotojas pats pasirenka kokiu keliu „eiti“. Taip pat renkasi ir kaip vystyti veiklos taisykles komponentais.

Atliktas eksperimentas tarp Blaze Advisor įrankio ir duomenų bazės, kurio metu buvo nustatyta kaip transformuojami duomenų bazės struktūros elementai į VT sistemą ir toliau filtruojami arba saugomi duomenis duomenų bazėje. Iš duomenų bazės duomenų yra formuojama informacija sistemos struktūros elementais ir pasinaudojus sprendimų komponentais yra priimami sprendimai ir gaunamas norimas rezultatas.

# 5. PASIŪLYTOS METODIKOS ĮVERTINIMAS

Šiuolaikinėse informacinėse sistemose būtinas veiklos taisyklės manipuliavimo mechanizmas, kuris kaupia ir leidžia modifikuoti taisykles. Veiklos taisyklės tūri būti įgyvendinamos tiek vartotojų veiksmuose, tiek taikomojoje programoje ar duomenų bazėje.

Veiklos taisyklės, kurios yra interpretuojamos manipuliavimo mechanizmo, turi atitikti tokius kriterijus:

- formalizavimas;
- užrašymas mechanizme naudojama kalba;
- ir tik tada saugojimas ir palaikymas.

Analizuojant veiklos taisyklių valdymo sistemas ir testuojant Blaze Advisor įrankį, buvo nustatyta, kad metodika turi užtikrinti tokius projektavimo ir realizavimo kriterijus:

- Duomenų struktūrų projektavimas.
- Sistemingas informacinės sistemos vystymas.
- Duomenų struktūrų transformavimo algoritmas į veiklos taisyklių valdymo sistemą – be struktūrinių praradimų.
- Veiklos taisyklių paruošimas komponentais.

Pasiūlytoje metodikoje pagrindinis akcentas yra nukreiptas į duomenų bazės ir VTVS tarpusavio sąveiką, taigi nagrinėjamas duomenų bazės projektavimas. Pateikiama strategijos koncepcija, kaip transformuojama iš objektinio modelio į reliacinę duomenų bazę. Sistemos kūrimo vientisumui ir sistemingumui ir nubraižytos užduočių ir veiklos diagramos atspindinčius pagrindinius vystymosi aspektus. Išskiriami keli veiklos taisyklių paruošimo būdai išvystomi VTVS komponentais arba duomenų bazės trigeriais. Ir sudarytas algoritmas kaip duomenų bazės struktūros elementai yra transformuojami į *Blaze Advisor* duomenų struktūrą. Viso to pasakoje, laikantis nustatytos metodikos projektavimo ir realizavimo metu yra išvystoma informacinė sistema, kurios gyvavimo raidos ciklas yra ganėtinai ilgas.

# 6. IŠVADOS

1) Išanalizavus veiklos taisyklių sąvoką, jos integracijos galimybę organizacijos veiklos procesų architektūroje bei egzistuojančių darbų srautų valdymo priemones (Oracle BPEL, BizTalk), nustatyta, kad šiuolaikinėse informacinėse sistemose būtina naudoti VT manipuliavimo mechanizmus. Jie užtikrina efektyvų veiklos taisyklių (sprendimų) atskyrimą nuo programinio kodo. Taip pat veiklos taisykles būtina užrašyti formalizuota ir vykdymo mechanizmams tinkama interpretavimo kalba, nes tokiu atveju atliekamas greitesnis ir efektyvesnis sprendimų priėmimas.

2) Išanalizuotos veiklos taisyklių valdymo sistemos (VTVS – Blaze Advisor, HaleyAuthority, Ilog JRules), identifikuota jų vidinė veikimo schema, vidiniai komponentai, aktoriai ir jų veiklos. Nustatyta, kad tokiose sistemose veiklos taisyklės yra saugomos saugykloje, o jas iškviečia ir interpretuoja taisyklių variklis. Efektyvų VT valdymą, operatyvų atnaujinimą ir vykdymą užtikrina lanksti VTVS saugykla ir adaptyvi VTVS variklio architektūra.

3) Detaliai išnagrinėta VTVS (Blaze Advisor*)* architektūra, VT kūrimo komponentai, VT specifikavimo kalba. Nustatyta, kad įrankyje yra pilnai įgyvendinti VTVS sistemoms reikalingi VT kūrimo elementai, tokie kaip: taisyklių rinkiniai, sprendimų medžiai, lentelės ir kiti. Šių komponentų pagalba galima sudaryti įvairaus sudėtingumo veiklos taisykles, kontroliuojančias organizacijos veiklos procesus. Vis dėlto norint realizuoti sudėtingą veiklos taisyklių logiką, būtina turėti tinkamą duomenų struktūrą, kuri turi būti gaunama iš organizacijoje naudojamų duomenų bazių. Šiuo metu nėra nurodymų (metodikos) užtikrinančios tinkamą perėjimą iš duomenų bazių struktūrų į veiklos taisyklių kūrimo aplinką.

4) Darbe yra pasiūlyta duomenų struktūrų (DS) transformavimo į veiklos taisyklių kūrimo aplinką metodika, kurioje akcentuojami pagrindiniai kūrimo principai: duomenų struktūrų transformavimas ir VT vystymas komponentais.

5) Pagrindiniai pasiūlytos metodikos ypatumai yra šie:

- Sudarytos rekomendacijos, kurios nurodo DS įtaką ir ypatumus informacinių sistemų kūrimo etapuose: projektavimo, realizavimo ir palaikymo. Pateikiamas duomenų bazės projektavimo strategijos pasirinkimas. Tokiu būdu užtikrinamas tinkamas duomenų struktūrų parengimas transformavimui.
- Sudarytas algoritmas nusako nuoseklų duomenų bazės struktūros transformavimą į VT valdymo sistemos duomenų struktūrą. Aprašyti vis galimų duomenų struktūrų transformavimo rezultatai VTVS aplinkoje. Tokiu būdu užtikrinamas ryšys tarp VTVS ir duomenų bazės bei neprarandami struktūriniai elementai. Nurodymai skirti duomenų

įtraukimui, atnaujinimui ir šalinimui bei jų ryšių kardinalumui leidžia išvengti galimų anomalijų duomenyse bei duomenų praradimo.

- Sudarytos rekomendacijos veiklos taisyklių kūrimui (taisyklių rinkiniams, šablonams, sprendimo medžiams, lentelėms) grindžiamomis transformuotomis duomenų struktūromis. Tai suteikia galimybę sudaryti aukšto sudėtingumo lygio veiklos taisykles ir užtikrinti tinkamą jų interpretavimą.
- Sudarytos konceptualios schemos vaizduojančios duomenų srautus vykdymo metu informacinėje sistemoje: nuo duomenų bazės iki sprendimo. Tokiu būdu užtikrinama, kad bus tinkamai pasirinkta veiklos taisyklių integracija informacinėje sistemoje.

6) Siūlomos metodikos efektyvumui nustatyti, atliktas tyrimas, kurio metu buvo vykdoma duomenų struktūrų transformacija į VTVS aplinką pasirinktai dalykinei sričiai (studentų stipendijų administravimas). Nustatyta, kad laikantis metodikos yra neprarandami DB struktūros elementai ir užtikrinamas pilnas VTVS manipuliavimas duomenimis. Tokiu būdu yra išlaikomas informacinės sistemos vientisumas ją projektuojant ir neprarandamas ryšys tarp DB ir VTVS. Priešingu atveju, transformavimo metu praradus struktūrinius elementus, galimos anomalijos sąveikoje tarp duomenų ir VTVS, t.y. jei prarandami duomenys, tai prarandama informacija, iš to seka, kad gali nepilnai arba išvis neveikti veiklos taisyklės.

# 7. LITERATŪRA


[1] AGOSTA, L. Business Rules Meet the Business User [interaktyvus]. 2005 [žiūrėta 2008-05-12]. *Prieiga per internetą:* <http://www.intelligententerprise.com/showArticle.jhtml?articleID=167100318>

[2] AMBER S. W. Agile database techniques [interaktyvus]. Wiley, 2003 [žiūrėta 2009-02-02]. *Prieiga per internetą:* <http://www.ambysoft.com/essays/mappingObjects.html>

[3] BRAYE, Lucie, et al. State of the Art Business Rules Languages [interaktyvus]. Research project with SWIFT Standards Department, 2006 [žiūrėta 2009-02-02]. *Prieiga per internetą:* <http://efficient.citi.tudor.lu/cms/efficient/content.nsf/id/research>

[4] BUSINESS RULES GROUP. Veiklos taisyklių manifestas, 2003 [žiūrėta 2008-05-12]. *Prieiga per internetą:* <www.businessrulesgroup.org/brmanifesto/BRManifestLithuanian(v1.0).pdf>

[5] CHAPPELL, D. Understanding BizTalk Server 2006 [interaktyvus]. 2005 [žiūrėta 2009-02-10]. *Prieiga per internetą:* <http://download.microsoft.com/documents/australia/windowsserversystem/biztalk2006/Understanding_BTS06.pdf>

[6] DEVANATHAN, N. BRMS and Business Users [interaktyvus]. 2007 [žiūrėta 2008-05-12]. *Prieiga per internetą:* <http://hosteddocs.ittoolbox.com/ND061807.pdf>

[7] ESWARAN, Sharanya, et al. Adapting and Evaluating Commercial Workflow Engines for e-Science. E-Science and Grid Computing, 2006. E-Science '06. Second IEEE International Conference 2006. 20-28psl.

[8] FAIR ISAAC. The Blaze Advisor Business Rules Management System: How It Works [interaktyvus]. 2008 [žiūrėta 2008-05-12]. White Paper. *Prieiga per internetą:*<http://researchlibrary.theserverside.net/viewer/viewDocument.do?accessId=7637554>

[9] FENG, X.; SUBRAMANIAN, M. Incorporating Business Rule Engine Technology in Control Center Applications-Toward Adaptive IT solutions. Energy 2030 Conference, 2008. ENERGY 2008. 1-5 psl.

[10] GRAHAM, I. Service Oriented Business Rules Management Systems. [interaktyvus] 2005 [žiūrėta 2008-05-20]. *Prieiga per internetą:*<http://researchlibrary.theserverside.net/viewer/viewDocument.do?accessId=7637562>

[11] HARMON, P. Business Rules: An Introduction. Business Process Trends. [interaktyvus] Newsletter 2003 [žiūrėta 2009-02-10]. *Prieiga per internetą:* <http://www.bptrends.com/publicationfiles/07-03 NL Business Rules Intro111.pdf>

[12] YUAN-CHWEN, Y. A Business-oriented Architecture Compliant with Semantic SOA. 2007 [žiūrėta 2008-05-20]. *Prieiga per internetą:*<http://dspace.lib.fcu.edu.tw/bitstream/2377/10815/1/CE07NCS002007000138.pdf>

[13] KENNEDY, M. Oracle BPEL Process Manager [interaktyvus]. 2005 [žiūrėta 2009-02-10]. *Prieiga per internetą:* <http://download.oracle.com/otndocs/products/bpel/quickstart.pdf>

[14] LAMMEL U. Business Rules make Business more flexible [interaktyvus]. 3rd Conference on Baltic Business and Socio-Economic Development BBSED 2007 [žiūrėta 2009-02-10]. *Prieiga per internetą:* <www.wi.hs-wismar.de/~laemmel/Forschung/Docs/laemmel_BBSED07.pdf>

[15] PANT, S; HSU, C. Strategic Information Systems Planning: A Review. 1995 Information Resources Management Association International Conference. 1995 [žiūrėta 2009-02-02]. *Prieiga per internetą:* <http://viu.eng.rpi.edu/publications/strpaper.pdf>

[16] ROSCA, D.; D'ATTILIO, J. Business Rules Specification, Enforcement and Distribution for Heterogeneous Environments. Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International. 2001. 3-9 psl.

[17] ROSENBERG, F.; DUSTDAR, S. Towards a distributed service-oriented business rules system. Web Services, 2005. ECOWS 2005. Third IEEE European Conference on 2005. 1-11 psl.

[18] VINCENT, P.; FAIR ISAAC. Structured Rules Language - a commercial rules representation [interaktyvus]. W3C Workshop on Rule Languages for Interoperability, 2005 [žiūrėta 2009-02-10]. *Prieiga per internetą:* <http://www.w3.org/2004/12/rules-ws/paper/55/>

# 8. TERMINŲ IR SANTRAUKŲ ŽODYNAS

| Sutrumpinimas | Angliškai | Reikšmė |
|---|---|---|
| **BA** | Blaze Advisor | VTVS sistema |
| **BPEL** | BPEL | verslo procesų vykdymo kalba |
| **DB** | DB | duomenų bazė |
| **DBOM** | DBOM | duomenų bazės objektų modelis |
| **IDE** | IDE | integruota programų kūrimo aplinka |
| **IS** | IS | informacinės sistemos |
| **IT** | IT | informacinės technologijos |
| **LDAP** | LDAP | serveris |
| **ODBC** | ODBC | tvarkyklė |
| ***P*** | – | pattern šablonai |
| **SQL** | SQL | kalbos standartas skirtas prisijungimui ir manipuliavimui su duomenų baze |
| **SRL** | SRL | struktūrinė taisyklių kalba |
| **TVP** | RMA | taisyklių valdymo programa |
| **VT** | BR | veiklos taisyklė |
| **VTVS** | BRMS | veiklos taisyklių valdymo sistema |
| **XML** | XML | duomenų struktūra |

# 9. PRIEDAI

## 1 priedas. EKSPERIMENTAS

***Darbo tikslas.*** Ištirti *Blaze Advisor* įrankio ir pasirinktos duomenų bazės tarpusavio sąveikos funkcionalumą, ir paruošti mokomąją metodinę medžiagą.

***Darbo atlikimo metodika.*** Šiam tikslui įgyvendinti bus panaudota instrukcija „žingsnis po žingsnio“, kuri parodys eigą kaip dirbant su įrankiu yra gaunas norimas rezultatas.

Šiam darbui įgyvendinti yra iškelti uždaviniai:

- aprašyti uždavinį natūralia kalba;
- suformuluoti užduoti;
- nubraižyti duomenų bazės schemą;
- nubraižyti sprendimų medį;
- suformuluotą uždavinį įgyvendinti įrankiu.

Darbo eiga parodyta sekančiuose punktuose:

*2 punktas.* Uždavinio aprašymas natūralia kalba: kiekvienas žmogus turi vieną ar kitą savo mėgstama hobį, taigi mes norime sukurti tokią sistemą, kuri padėtų atrinkti žmones į tam tikrus klubus, kurie turi tam tikrą grupes statusą.

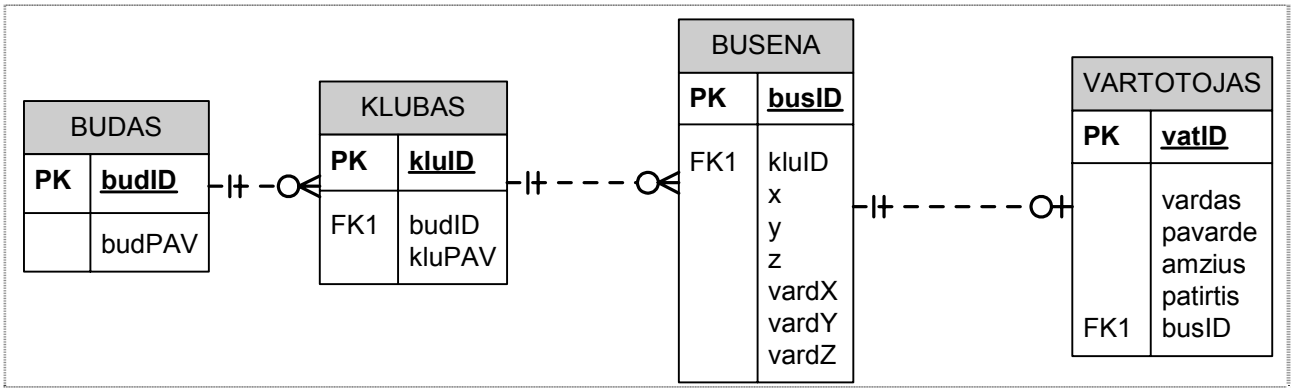
*3 punktas.* Uždavinio formulavimas: sukurti sistemą, kuri įvedus vartotojo duomenis pagal juos atrinktu į tam tikrus klubus, ir atsižvelgiant į amžių ir hobio užsiėmimo laikotarpį parinktų statusą grupėje.

*4 punktas.* Norint nubraižyti duomenų bazės schemą pirmiausia reikia nustatyti kokie duomenys bus naudojami. Išanalizavus kokie duomenis yra reikalingi suprojektuojama duomenų bazė, kuri susideda iš 4 lentelių: *BUDAS, KLUBAS, BUSENA, VARTOTOJAS*. *BUDAS* lentelėje yra renkama informacija apie žmonių mėgstamus praleidimo būdus. *KLUBAS* lentelėje būtu galima rinti informacija apie klubus, šiuo atveju renkamas klubo pavadinimas ir susijęs su juo mėgiamiausio būdo identifikatorius. *BUSENA* lentelėje yra renkami duomenis apie žmonių statusą grupėje-klube t.y. amžių grupės pagal kurias nustatomas statusas (pvz.: yra x y z, kur pirma grupė nuo x yra pradinis amžius iki y amžiaus, sekanti grupė nuo y iki z, paskutinė grupė nuo z). *VARTOTOJAS* lentelėje gali būti renkama informacija apie patį vartotoją, šiuo atveju yra renkama tik minimali informacija, tokia kaip vardas, pavarė, amžius ir patirtis užsiimant mėgstamu hobiu priklausančiam klubui. Ši duomenų bazė yra tik minimaliam projektui įgyvendinti, nes yra labiau mokomoji nei plataus taikymo.

Duomenų bazės struktūra ir duomenų bazės grafinė struktūra pavaizduota atitinkamai 1.1 lentelėje ir 1.1 paveiksle.

**1.1. lentelė.** Duomenų bazės struktūra.

| Lentelės pavadinimas | Lauko pavadinimas | Lauko tipas | Apibūdinimas |
|---|---|---|---|
| *BUDAS* | budID | INT | Būdo identifikatorius |
| | budPAV | CHAR(30) | Būdo pavadinimas |
| *KLUBAS* | kluID | INT | Klubo identifikatorius |
| | budID | INT | Būdo identifikatorius |
| | kluPAV | CHAR(30) | Klubo pavadinimas |
| *BUSENA* | busID | INT | Būsenos identifikatorius |
| | kluID | INT | Klubo identifikatorius |
| | X | INT | Nuo amžius |
| | Y | INT | Nuo iki amžius |
| | Z | INT | Nuo iki amžius |
| | vardX | CHAR(30) | Pirmo grupės vartotojo vardas |
| | vardY | CHAR(30) | Antros grupės vartotojo vardas |
| | vardZ | CHAR(30) | Trečios grupės vartotojo vardas |
| *VARTOTOJAS* | vatID | INT | Vartotojo identifikatorius |
| | busID | INT | Būsenos identifikatorius |
| | Vardas | CHAR(30) | Vartotojo vardas |
| | Pavarde | CHAR(30) | Vartotojo pavardė |
| | Amzius | INT | Vartotojo amžius |
| | Patirtis | INT | Vartotojo hobio patirtis |




**1.1 pav.** Duomenų bazės grafinė struktūra.

*5 punktas.* Pasirinkti duomenų bazės serverį: šiam projektui įgyvendinti buvo pasirinkta *Microsoft SQL Server 2000* (toliau MS SQL). Į pasirinktą serverį sugeneruoti suprojektuotą duomenų bazės struktūrą. Rekomenduojami lentelių duomenų įrašai pavaizduoti 1.2 – 1.5 lentelėse.

**1.2. lentelė.** *BUSENA* lentelės duomenys.

| busID | kluID | x | y | z | vardX | vardY | vardZ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 14 | 18 | 40 | kartingistas | formulistas | sodininkas |
| 2 | 2 | 16 | 18 | 45 | motoroleristas | britvininkas | cioperistas |
| 3 | 3 | 5 | 18 | 50 | poteris | stalone | sigalas |
| 4 | 4 | 5 | 15 | 30 | stano | gabalis | povilaitis |
| 5 | 5 | 5 | 20 | 50 | eseriukas | karpis | ungurys |
| 6 | 6 | 18 | 30 | 60 | kiskis | vilkas | lokys |

**1.3. lentelė.** *VARTOTOJAS* lentelės duomenys.

| vatID | busID | vardas | pavarde | amzius | patirtis |
|---|---|---|---|---|---|
| 1 | 1 | Arturas | Korsakas | 23 | 5 |

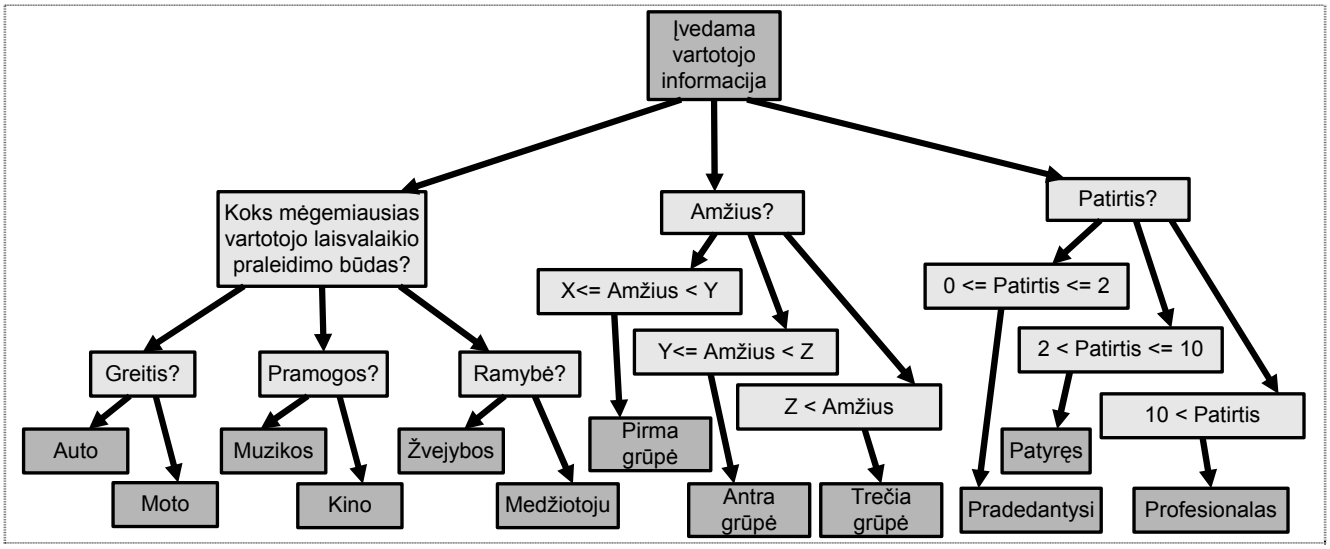
**1.4. lentelė.** *KLUBAS* lentelės duomenys.

| kluID | kluPAV | budID |
|---|---|---|
| 1 | auto | 1 |
| 2 | moto | 1 |
| 3 | kino | 2 |
| 4 | muzikos | 2 |
| 5 | zveju | 3 |
| 6 | medziotoju | 3 |

**1.5. lentelė.** *BUDAS* lentelės duomenys.

| budID | budPAV |
|---|---|
| 1 | greitis |
| 2 | pramogos |
| 3 | ramybe |

*6 punktas.* Kad galėtume įsivaizduoti pačios sistemos pasirinkimų galimybė - nubraižomas pasirinkimų medis: pagal esamus duomenų bazėje duomenis (kurie yra 1.2 – 1.5 lentelėse) gaunamas 1.3. paveiksle pavaizduotas pasirinkimų medis.



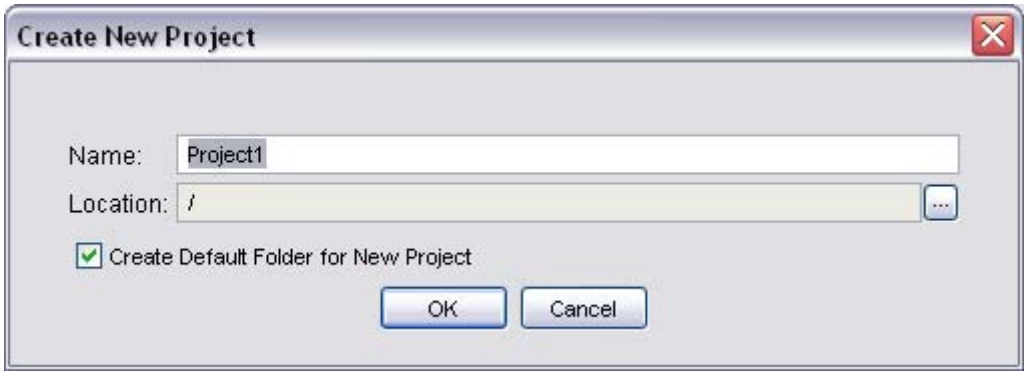

**1.2. pav.** Pasirinkimų medis.

*7 punktas.* Kad galėtume prie „*Blaze Advisor*“ įrankio prijungti duomenų bazės serverį *MS SQL* reikia pirmiausia sukonfigūruoti *ODBC Data Source Administrator* langą (konfigūravimo pavyzdys parodytas [9]).

*8 punktas.* Darbo pradžia su įrankiu: pasileidę *Builder* langą galime importuoti jau esančius projektus ir jais pasinaudoti, tačiau mes sukursime naują projektą.

1) Pasirenkame vieną iš galimų variantu, norint pradėti kurti naują projektą:
   - spragtelti **New Project** ikoną;
   - pasirinkti **File > New Project**;
   - ar tiesiog: **Ctrl + N**;

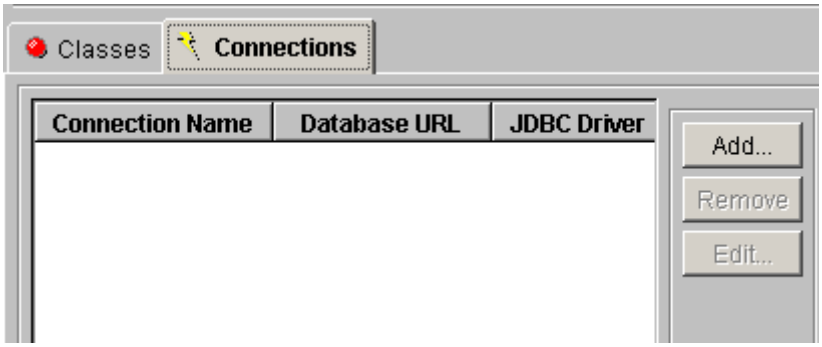
   Pasirodo naujo projekto langas:



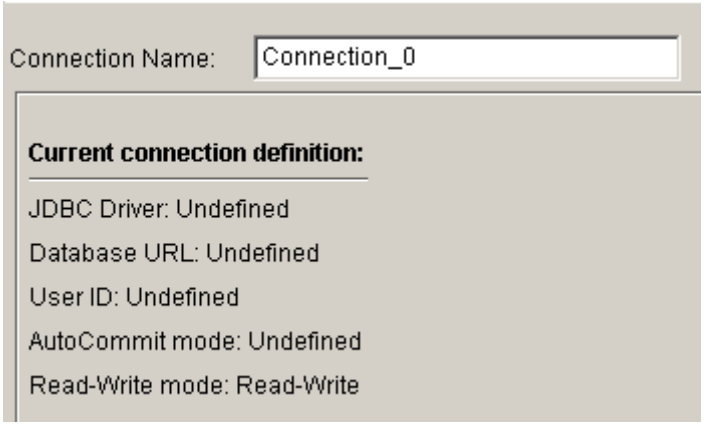

2) Į laukelį *„Name“* įrašome projekto pavadinimą pavyzdžiui: *PasirinkKluba*;
3) Nurodome vietą kurioje norime, kad saugotų projektą.
4) Spaudžiame **OK**.
5) Atsidaro *Builder* langas, toliau išsaugome projektą paspaudę .

*9 punktas.* Duomenų bazės prijungimas prie įrankio:

1) Pasirenkame **Project > Business Object Models > New Database Business Object Model**. Pasirodžiusiame lange pereiname į **Connections** lapą:




2) Paspaude **Add** pasirodo langas, kuriame matomi parametrai:



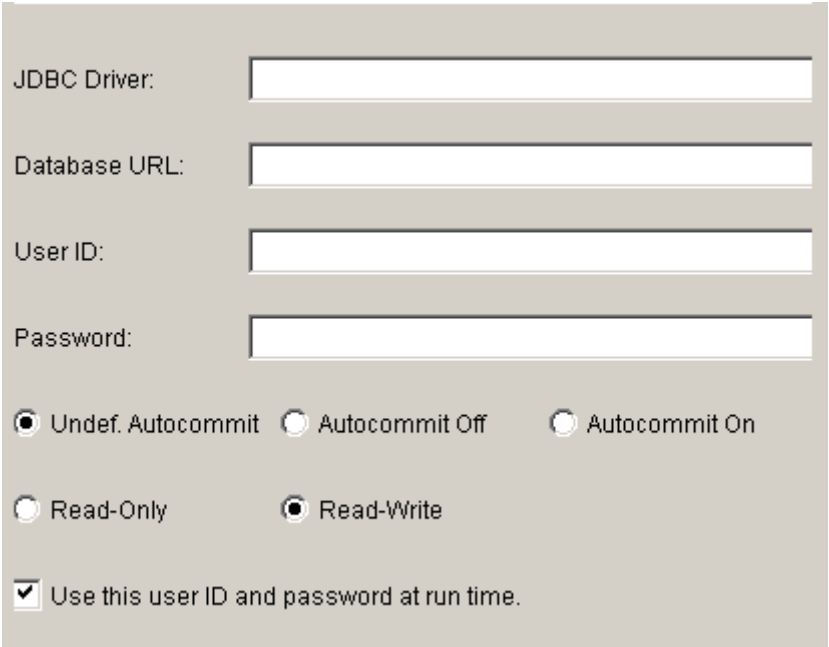

3) Į laukelį **Connection Name** įrašome prisijungimo vardą (būtina be tarpų)(pvz. *dbKlubas*).
4) Spaudžiame **Next**.
5) Atsidariusiame lange įvedame tokius prisijungimo duomenis:
   - JDBC tvarkyklių vardą kuri naudojame:

```
JDBC Drive = sun.jdbc.odbc.JdbcOdbcDriver
```

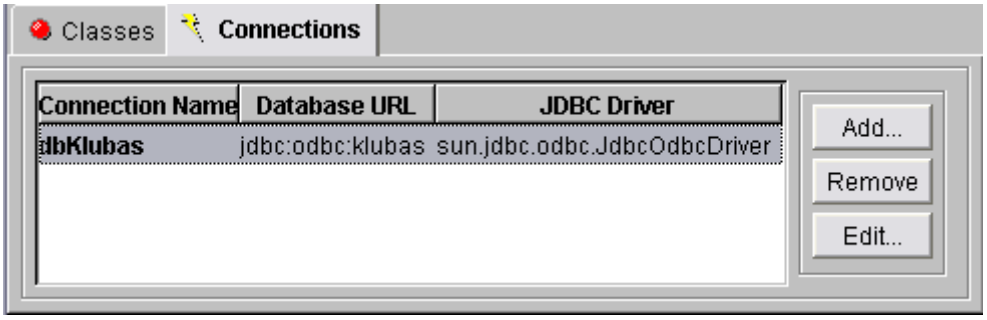
- Kadangi naudojame ODBC tvarkykles tai nurodome sukonfigūruotas su ODBC Data Source Administrator (klubas yra sukurta duomenų bazės pavadinimas serveryje):

```
Database URL = jdbc:odbc:klubas
```

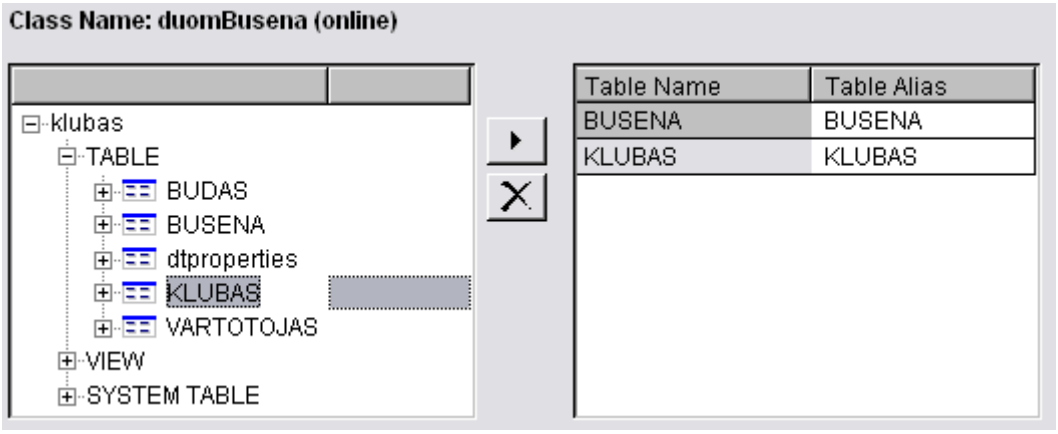
- Jei reikia įvedame vartotojo identifikatorių ir slaptažodį (*User Id* ir *Password*).




6) Spaudžiame **Finish**. Gauname tokį vaizdą:

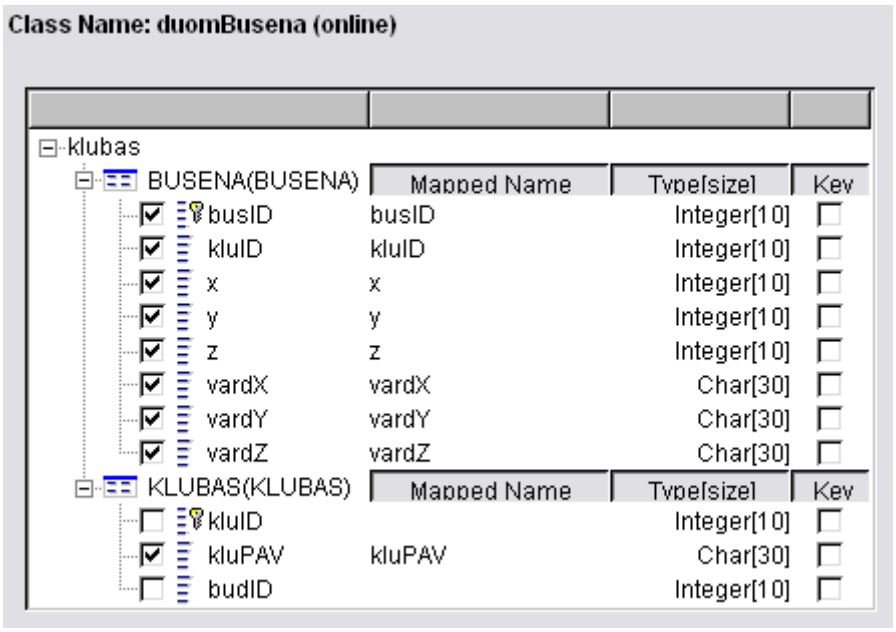



7) Pereiname į **Classes** lapą. Čia galima susikurti reikalingas duomenų klases iš duomenų bazės. Naujos klasės kūrimas prasideda spragtelėjus **Add.** Į laukelį **SRL mapping class name** įrašomas klasės vardas, kuris bus naudojamas ateityje, ir spaudžiame **Next**. Sekančiame lange pasirenkamas prisijungimas prie duomenų bazės, šiuo atveju turėtu matytis vienas prisijungimo vardas *dbKlubas* ir spaudžiame **Next**.
8) Pasirodžiusio lango kairėje pusėje yra parodoma duomenų bazės struktūra. Išsirenkame mums reikalingas lenteles ir mygtuko ▸ pagalbą įsikeliame į dešinę lango pusę. Gaunamas vaizdas, kai pasirenkamos lentelės BUSENA ir KLUBAS:

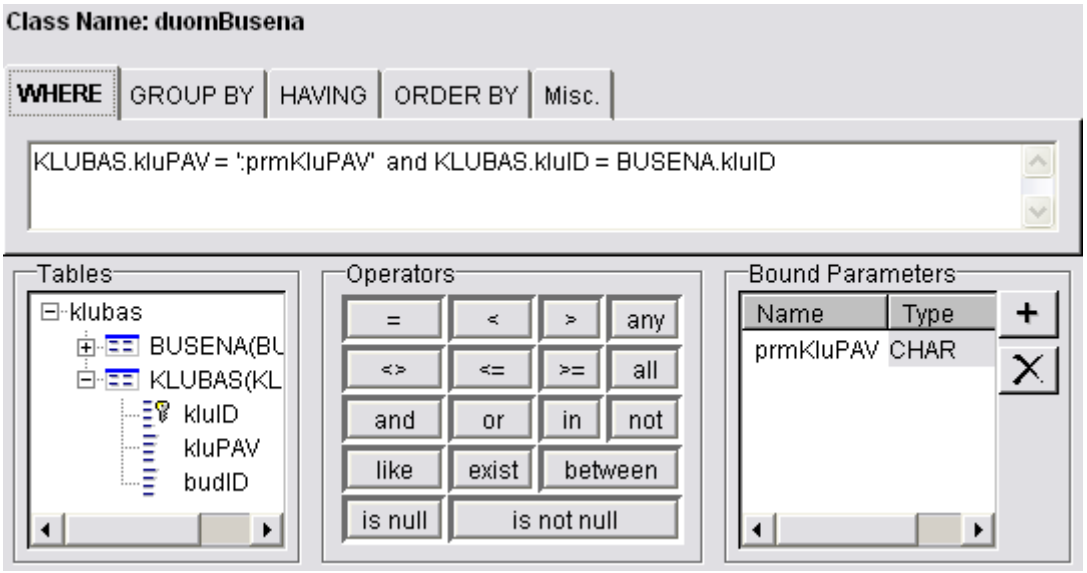



Norint pratęsti darbą reikią spragtelėti mygtuką **Next.**

9) Toliau išsirenkame iš pateiktų lentelių laukus, kuriuos mes filtruosime ir spaudžiame **Next**:




10) Toliau pateikiami SQL komandos *SELECT* ypatumai, tokie kaip *WHERE, GROUP BY, HAVING, ORDER BY*. Šiuo atveju naudosi tik paprasta filtravimą *WHERE*, kaip pateikta lange:

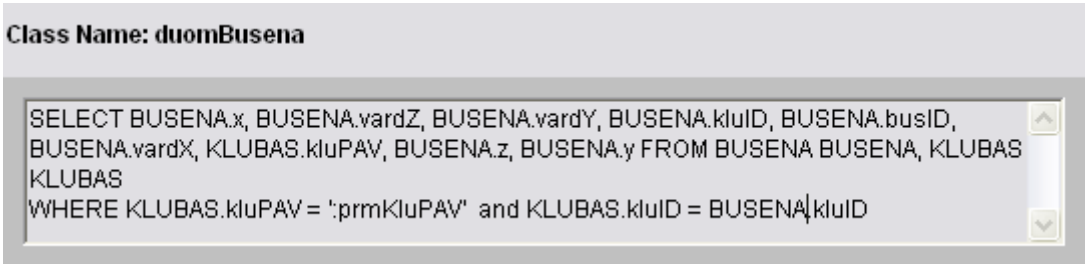


Jeigu reikia filtravimo su apribojimais, kaip ir šiuo atveju, yra įvedami **Bound Parameters** (pridedami mygtuko + pagalba). Parametrų įvedimas į lauką yra galimas dviem būdais:

- spragtelėjus du kartus kairiu pelės klavišu ant parametro pavadinimo;
- arba įrašant parametro pavadinimą tiesiog į lauką, tačiau reikia nepamiršti ji įrašyti tarp viengubu apostrofu ir prieš vardą būtina padėti dvitaški (pvz. *':prmKluPAV'*).

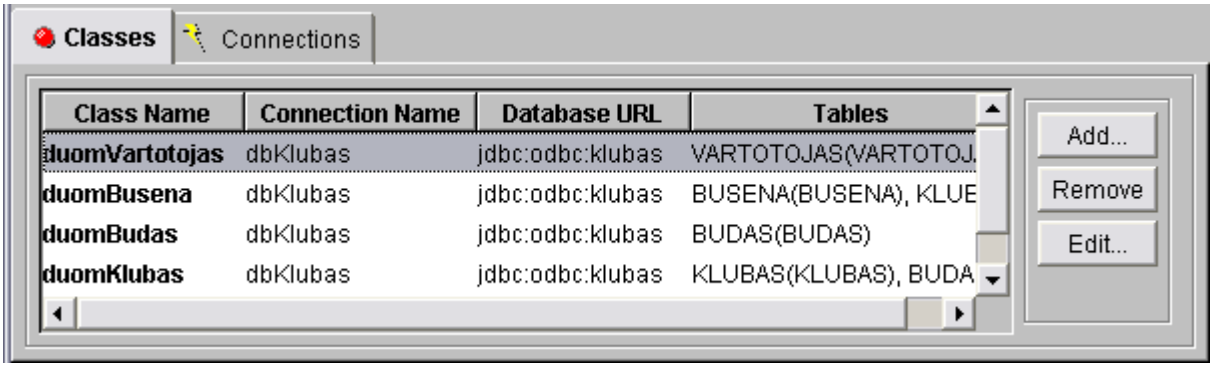
Spaudžiame **Next** mygtuką.

11) Pasitikrinimui parodomas langas su pilnai sukombinuotu *SELECT* sakiniu:



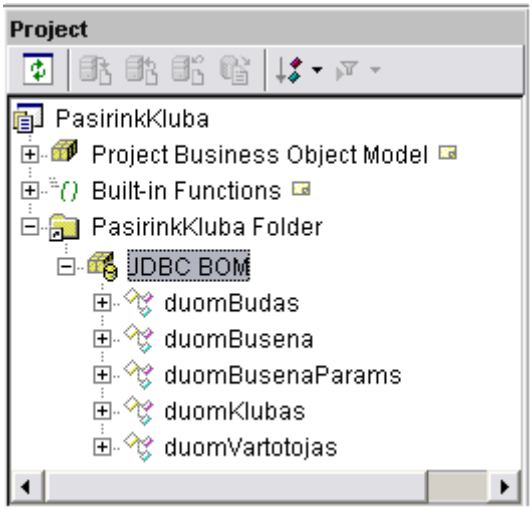

Patikrinus ar viskas yra suvesta teisingai spaudžiame **Finish**.

12) Analogiškai sukuriamos ir kitos lentelių duomenų klasės pakartojant 7-11 žingsnius. Klasės *duomBudas* duomenys bus naudojami vartotojui užklausos atrinkti jo mėgstamiausia laisvalaikio praleidimo būdą. Pagal būda bus atrinkti atitinkami sukurti klubai, klasė *duomKlubai*. O taip pat vartotojo klasė *duomVartotojas*. Gaunamas toks vaizdas:



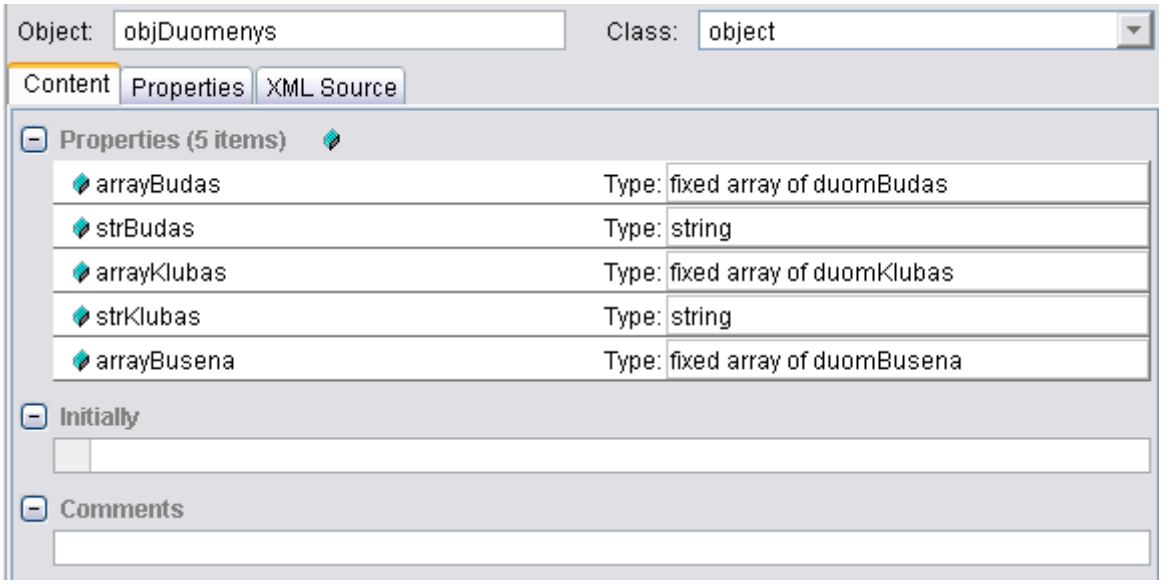

Duomenų bazė prijungta ir sukurtos duomenų klasės, darbas baigtas, spaudžiame **Ok**.

13) Projekto meniu atsiranda naujas „*JDBC Business Object Model*“ punktas ir modelyje sukurtos keturios klasės (*duomBudas, duomBusena, duomKlubas, duomVartotojas*) ir viena automatiškai sukurta *duomBusenaParams*, kuri šiuo atveju apibrėžia *duomBusenos* klasės apribojimų parametrus. Atlikus visus žingsnius tūrėtų būti matomas toks projekto menių:




*10 punktas.* Kadangi jau turime susikūrę klases iš duomenų bazės reikia susikurti klasių objektus, su kuriais atliksime veiksmus. Naują objektą galime susikurti spragtelėjus **New Object** ikoną arba **Project > Object Model Entities > New Object.**

Yra reikalingi keturi objektai su kuriais manipuliuosime: bendras objektas *objDuomenys* kuriame rinksime visą informaciją gautą iš duomenų bazės, *objBusena* objektas reikalingas palyginti vartotojo duomenis su esančiais duomenimis duomenų bazėje, ir papildomas objektas *objBusParams* reikalingas filtravimui kur nurodamas parametras pagal kuri apribojamas filtravimas iš duomenų . Objektai ir jos savybės parodytos sekančiuose paveiksluose:



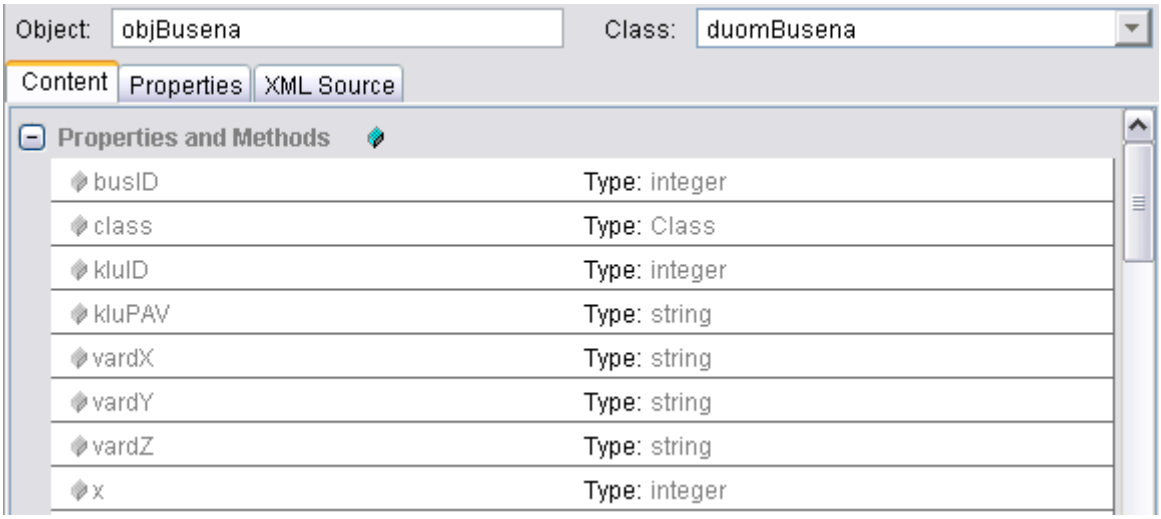

*objDuomenys* objektas nepaveldi jokiu duomenų, taigi **Class** laukelyje reikia nurodyti, kad tai yra *object* tipo. Savybės įtraukiamos spragtelėjus **New Property** ikoną.

Savybės ir jų reikšmė:

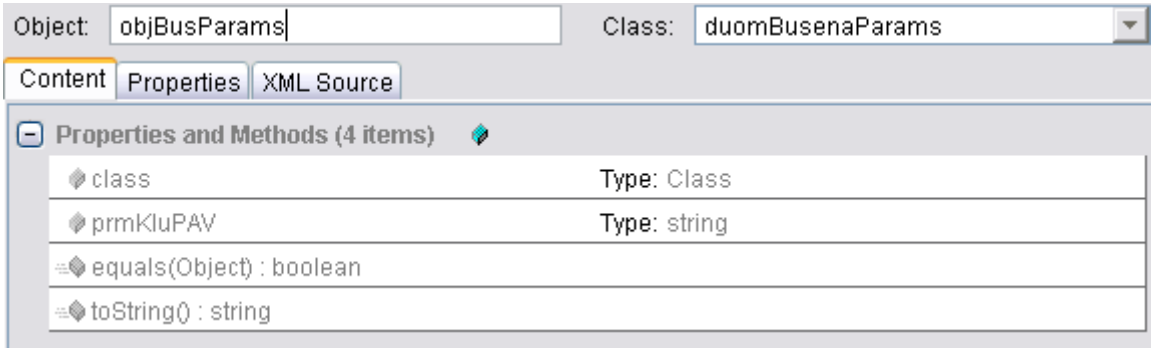
- *arrayBudas:* ***fixed array of duomBudas*** – fiksuoto ilgio masyvas gautas iš klasės *duomBudas*;
- *arrayKlubas:* ***fixed array of duomKlubas*** – fiksuoto ilgio masyvas gautas iš klasės *duomKlubas*;

- *arrayBusena:* ***fixed array of duomBusena*** – fiksuoto ilgio masyvas gautas iš klasės duomBusena;
- *strBudas*: ***string*** – eilutė reikalinga duomenų vaizdavimui;
- *strKlubas*: ***string*** – eilutė reikalinga duomenų vaizdavimui.

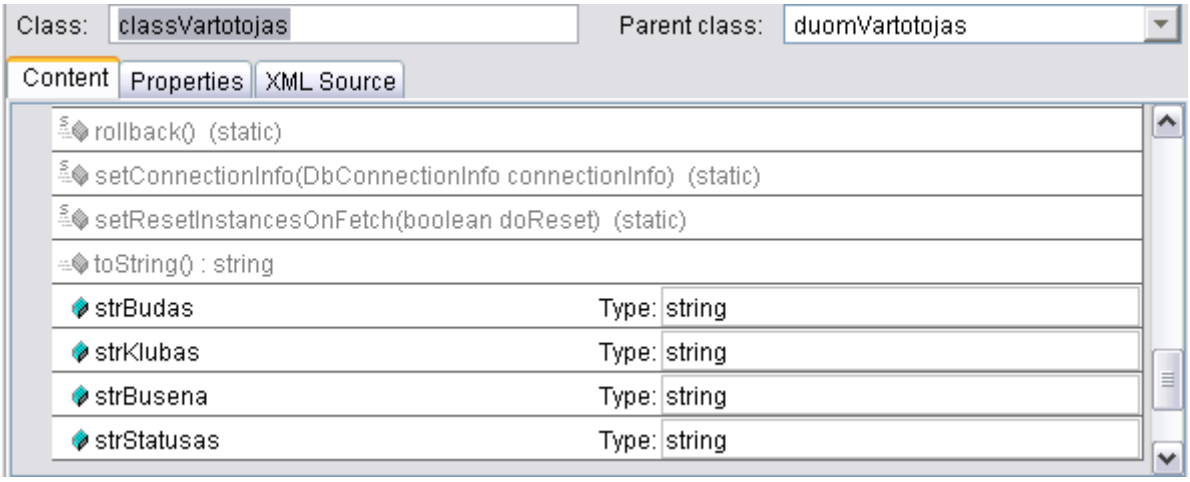
Objektas *objBusena* priklauso klasei *duomBusena*, todėl **Class** laukelyje būtina nurodyti *duomBusenos* pavadinimą papildomų savybių nereikia įvedinėti:




Kadangi *objBusena* objekto filtravimas yra su ribojimais, taigi reikia sukurti ir papildomą objektą *objBusParams* priklausantį klasei *duomBusParams*:



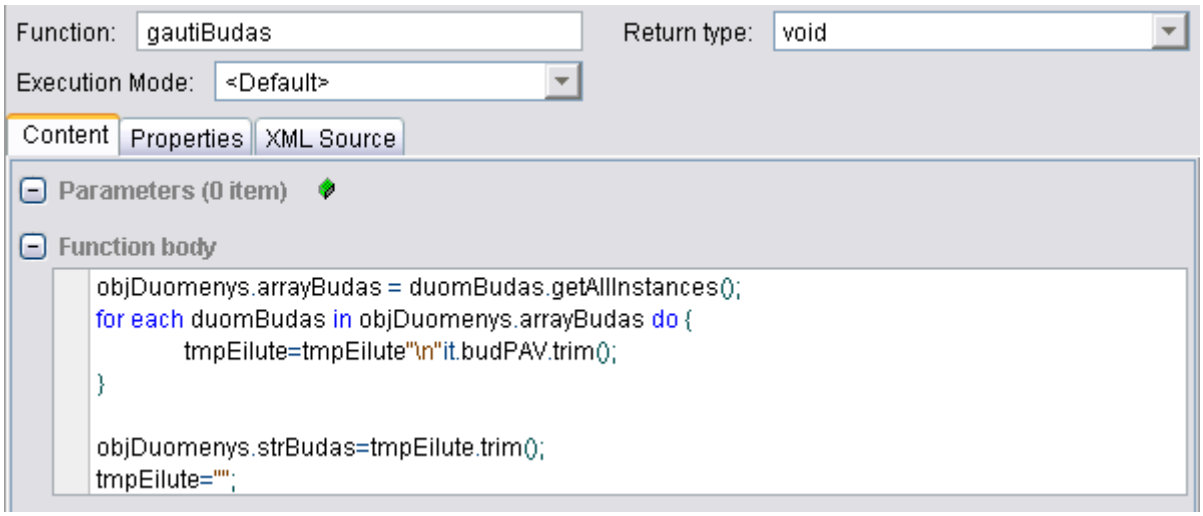

Paskutiniam objektui reikės sukurti naują klasę su papildytomis savybėmis. Tam atlikti galime dviem būdais spragtelėjus **New Class** ikoną arba pasirinkus **Project > Object Model Entities > New Class.** Toliau įrašyti klasės vardą *classVartotojas* ir parinkti klasę *duomVartotojas.* Ir įtraukti keturias naujas savybes *string* tipo: *strBudas*, *strKlubas*, *strBusena* ir *strStatusas:*



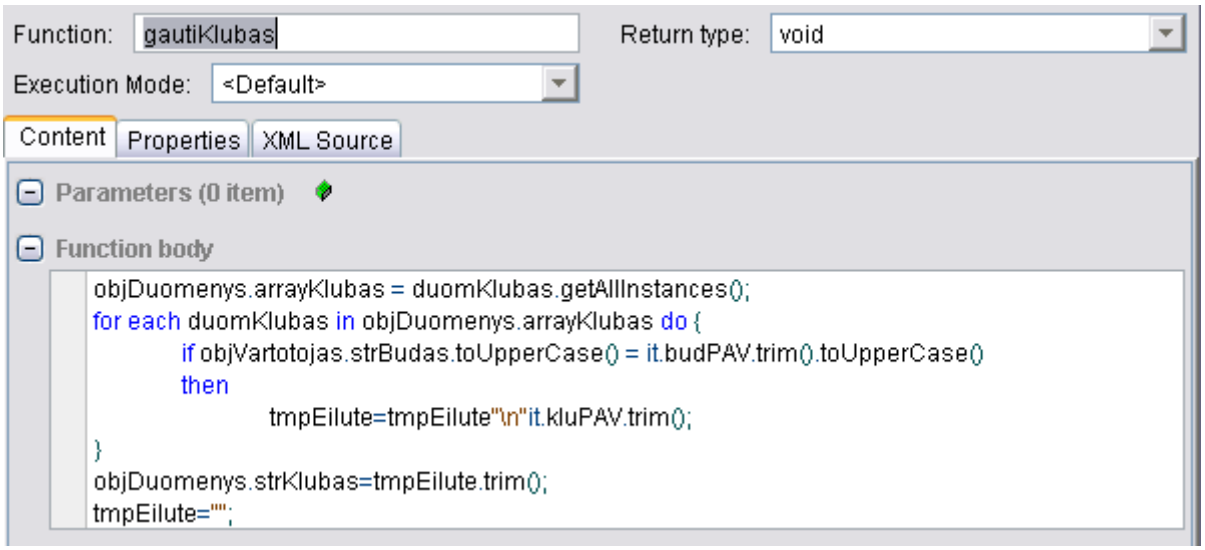

Šiai klasei sukuriame objektą *objVartotojas* ir parenkame klasę *classVartotojas*.

*11 punktas.* Funkcijos kūrimas: **New Function** arba **Project > Decision Entities > New Function.** Sudarytos funkcijos:

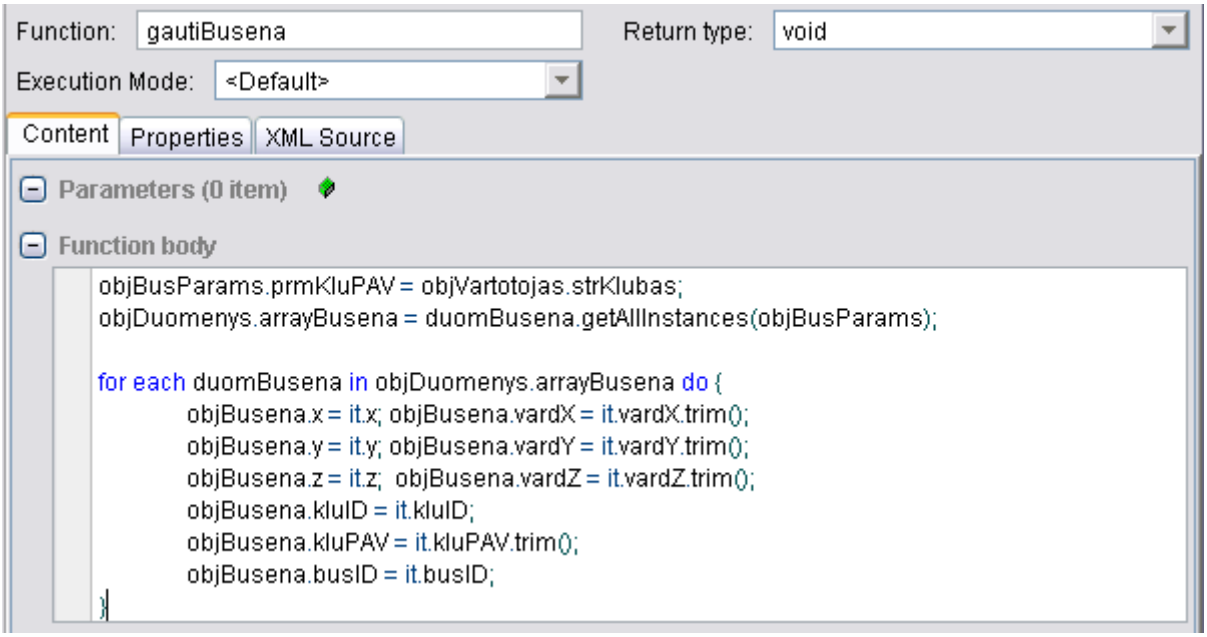
1) Funkcija: gautiBudas():



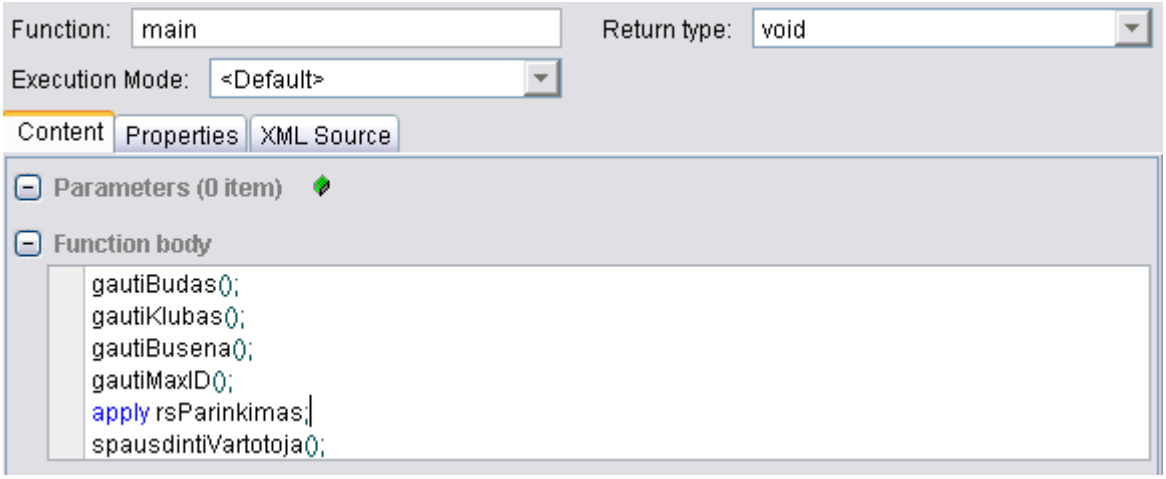

2) Funkcija: gautiKlubas():



3) Funkcija: gautiBusena():




4) Pagrindinė funkcija, ji visada turi toki pavadinimą: main():




*12 punktas.* Taisyklių paruošimas. Pirmiausiai reikia susikurti taisyklių rinkti tai galima padaryti spragtelėjus ikoną **New Ruleset** arba **Project > Decision Entities > New Ruleset**. Atsiradus menių taisyklių rinkinio ikona, ant jos paspaudus kuriame naują taisyklę spragtelėjus ikoną **New Rule.**

1) Taisyklės: rlPirmaGrupe, rlAntraGrupe, rlTreciaGrupe, rlBlogaGrupe:

rlPirmaGrupe

```
if objBusena.x <= objVartotojas.intAmzius and objVartotojas.intAmzius < objBusena.y
then
{
        objVartotojas.strBusena = objBusena.vardX;
        objVartotojas.busenaID = objBusena.busID;
}
```

rlAntraGrupe

```
if objBusena.y <= objVartotojas.intAmzius and objVartotojas.intAmzius < objBusena.z
then
{
        objVartotojas.strBusena = objBusena.vardY;
        objVartotojas.busenaID = objBusena.busID;
}
```

rlTreciaGrupe

```
if objBusena.z <= objVartotojas.intAmzius
then
{
        objVartotojas.strBusena = objBusena.vardZ;
        objVartotojas.busenaID = objBusena.busID;
}
```

rlBlogaGrupe

```
if objBusena.x > objVartotojas.intAmzius and objBusena.x > 0
then
        objVartotojas.strBusena = "Nepatenka į grupes!"
```

2) Kitos taisyklės:

rlParenkaPradedantysis

```
if objVartotojas.intPatirtis <= 2 and objVartotojas.intPatirtis >= 0
then
        objVartotojas.strStatusas = "pradedantysis"
```

rlParenkaPatyres

```
if objVartotojas.intPatirtis > 2 and objVartotojas.intPatirtis <= 10
then
        objVartotojas.strStatusas = "patyręs"
```

rlParenkaProfesionalas

```
if objVartotojas.intPatirtis > 10 and objVartotojas.intPatirtis <= 150
then
        objVartotojas.strStatusas = "profesionalas"
```

*13 punktas.* Gaunamiems duomenims iš vartotojo galime sukurti įvyki, spragtelėję ikoną **New Event Rule.** Tokios taisyklės yra taikomos norimai informacijai gauti, inicializavimo arba kai ji yra reikalinga.

*14 punktas.* Įvykdžius visus dvylika punktu gaunama minimali sistema, kuri parenką vartotojui pramogų praleidimo klubą ir nustato jo statusą grupėje.

*Informacija:*

Question: Koks jusu vardas?
Answer: Arturas

Question: Kokia jusu pavarde?
Answer: Korsakas

Question: Koks jusu amzius?
Answer: 24

Question: Įveskite vieną iš galimų variantų: unknown
greitis
pramogos
ramybe
Answer: greitis

Question: Įveskite vieną iš galimų variantų:
auto
moto
Answer: auto

Question: Kokia Jūsų patirtis užsiimant šiuo hobiu?
Answer: 6

*Rezultatas:*

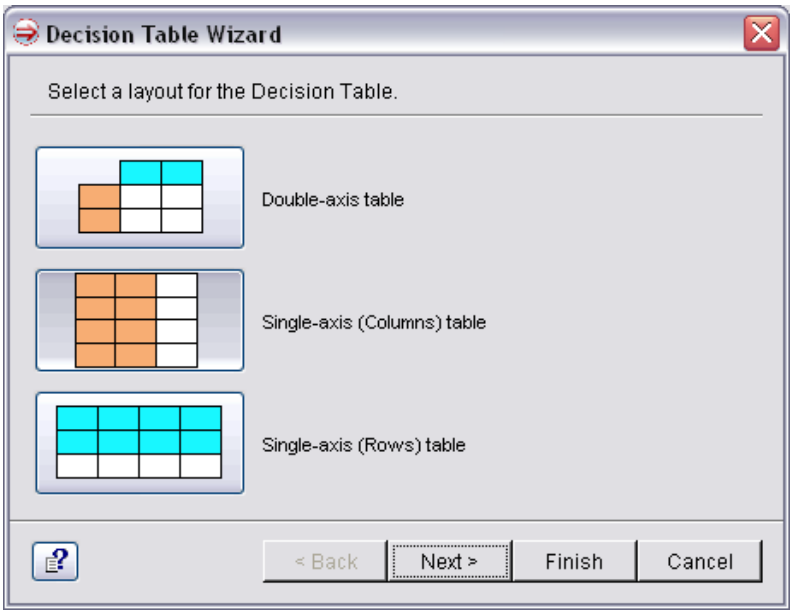
```
Vartotojas (Vardas, Pavardė): Arturas Korsakas
 yra kviečiamas į 'auto' klubą,
 pradinė būsena yra formulistas.
 ir statusas grupėje yra patyres.

Execution complete.
```

*15 punktas.* Sekančiuose punktuose bus nagrinėjamos kitos įrankio komponentės, tokios kaip sprendimų lentelės ir medžiai. Šiame punkte aptarsime sprendimų lentelių kūrimą.

Sprendimų lentelių kūrimas prasideda spragtelėjus ikoną arba **Project > Decision Entities > New Decision Table...**. Sukursime dvi elementarias lenteles, kurių pagalba gausime toki patį rezultatą, kaip gavome panaudojus taisyklių rinkinius. Sukursime dvi lenteles: parinkti statusą pagal patirtį, ir parinkti būseną pagal amžių, grupėje.

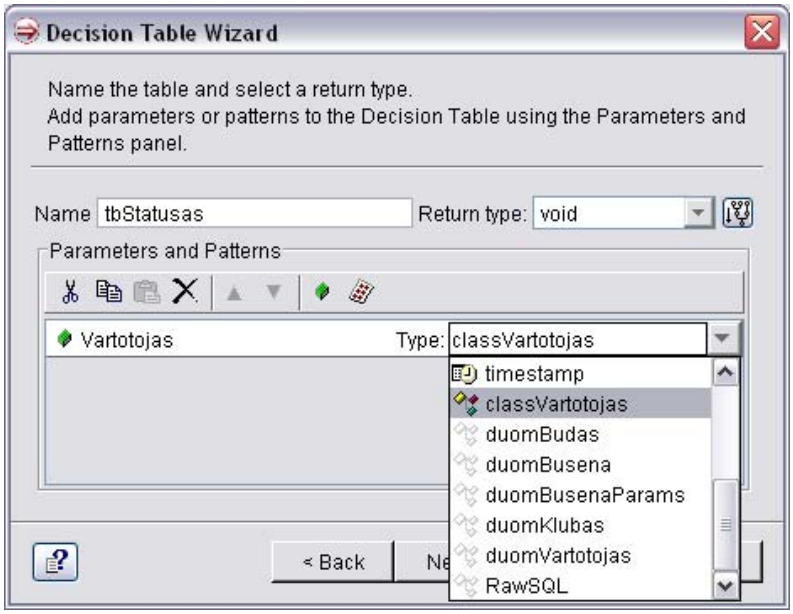
Kūrimo žingsniai:

1) Pirmiausia parenkame lentelių tipą, šioje dalyje parinksime *Single-axis* tipą, kaip parodyta (pasirinkus spragtelti *Next*):

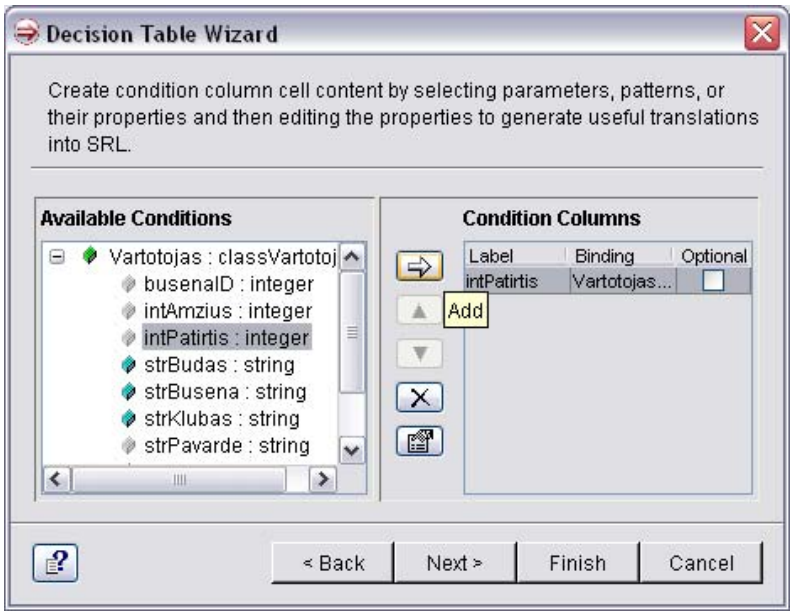



2) Pasirodžiusiame lange įvedami parametrai, su kuriais duomenimis manipuliuosime – formuosime lenteles. Taip pat parenkamas lentelės pavadinimas ir rezultato gražinimo

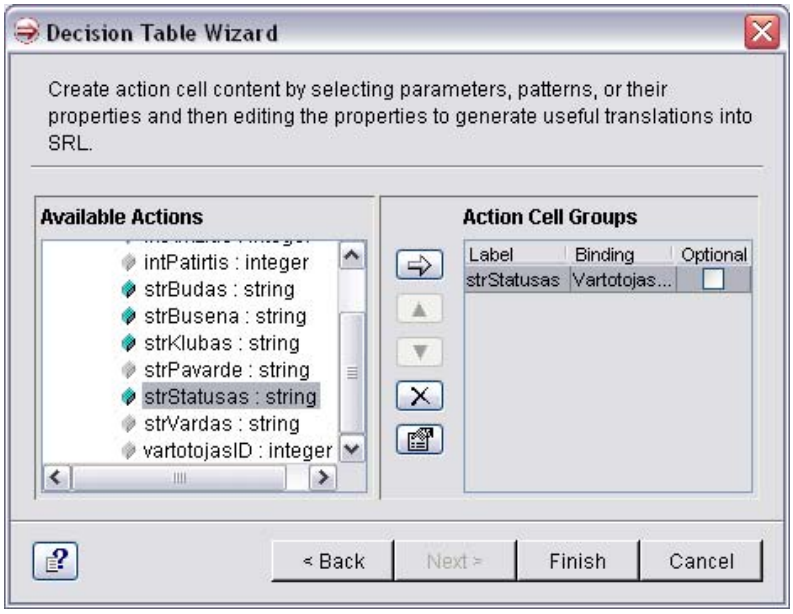
tipas. Įvedamas vienas parametras *classVartotojas* tipo, pagal vartotojo patirti bus parinktas statusas grupėje. Kaip parodyta paveiksle (toliau *Next*):



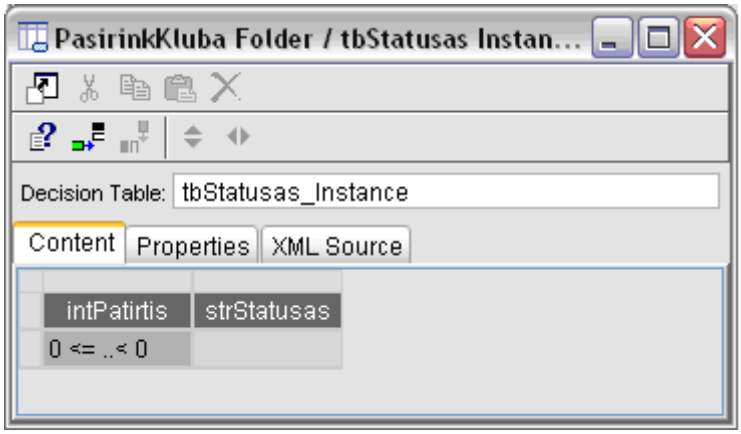

3) Šioje kūrimo dalyje parenkamos lentelės stulpelių(*Condition Columns*) reikšmės, šiuo atveju pakanka parinkti tik patirtį, kaip parodyta pasinaudojus mygtuku  (toliau *Next*):



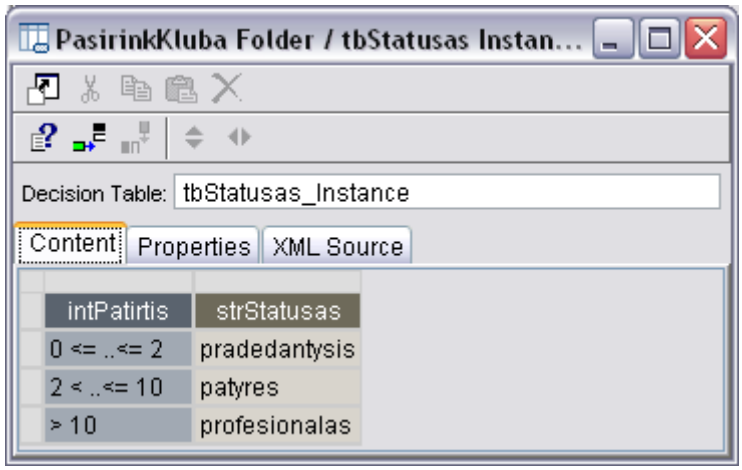

4) Jeigu būtume pasirinkę lentelės tipą *Double-axis* toliau būtų parodytas langas kuriame reikėtų įvesti sąlygas eilutėms (*Condition Rows*). Kadangi buvo pasirinktas *Single-axis* tipas šis langas praleidžiamas ir parodomas veiksmų langas (*Action Cell Groups*). Šiame lange reikia įvesti parametrą kuriame bus įvedamos reikšmės atitinkamai sąlygai, šiuo atveju vartotojo statusą, kaip parodytą (*toliau Finish*):



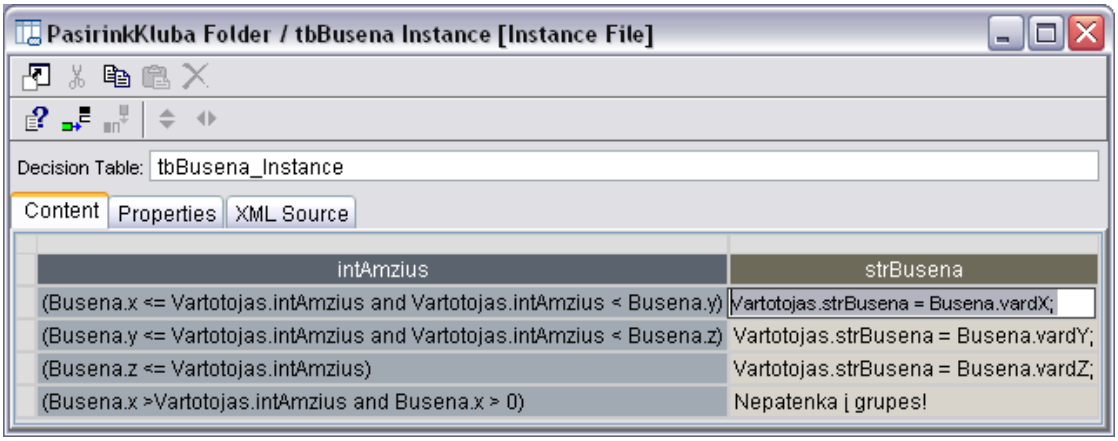

5) Baigę kurti lentelę, įrankis sukuria lentelės egzempliorių, kurio pagalba pildysime lentelėje dujomis. Standartiškai egzemplioriui suteikiamas pavadinimas prie lentelės vardo pridėjus *Instance*, galima ir persivadinti pagal poreikį. Šiuo atveju yra pavadinta *tbStatusas_Instance* kaip parodyta:



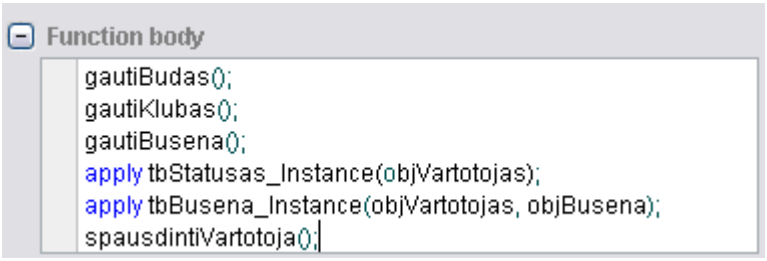

6) Kadangi buvo pasirinktas *Single-axis* tipas, tai įrankyje yra leidžiamas tik naujų eilučių įterpimas paspaudus mygtuką . Šiame darbe pakaks trijų sąlygos eilučių, sąlygos tipą galima keisti spragtelėjus dešinį klavišą ant sąlygos ir išsirinkti iš *Format Cell* tipą. Atliekame veiksmus taip, kad galutinė lentelė atrodytų taip:



7) Antros sprendimų lentelės kūrimas yra analogiškas pirmajai lentelei, aptarsime pagrindinius aspektus. Pasirenkamas taip pat *Single-axis* tipas, parametrai yra šiuo atveju yra du: Vartotojas iš *classVartotojas* ir Būsena iš *duomBusena* klasių. Į stulpelių formavimą įtraukiamas vartotojo amžius, tačiau prieš einant į kitą langą būtiną spragtelėjus mygtuką  ir uždėti varnelę ant *SRL* eilutės, kad galėtume naudoti išraiškas. Ta pati atliekame veiksmų lange įkelti vartotojo būsena pažymėti *SRL* eilutę. Baigus kurti sprendimų lentelę sukuriamas egzempliorius *tbBusena_Instance*. Egzemplioriuje naudosime 4 amžiaus sąlygas, taigi įkėlus ant kiekvienos paspaudus dešinį klavišą parinkti *Format Cell* tipą: *SRL*. Tuo tarpu veiksmų stulpelyje kur yra vartotojo būsenos, kur yra imami duomenys iš objekto, būtina nurodyti *SRL* tipą ir būtina nurodyti kur įrašyti duomenis pvz.: *Vartotojas.strBusena = Busena.vardX;*, o kur imamas statinė *string* tipo eilutė, tiesiog įrašoma reikšmė. Atlikus tokią veiksmų seką įrankyje turime gauti tokią galutinę lentelę:




8) Kad galėtume vykdyti sprendimų lenteles reikia pagrindinėje komandoje jas iškviesti kaip parodyta sekančiame paveiksle:




9) Įvykdžius šiuos visus žingsnius ir paleidus programą, su sprendimo lentelėmis, išvedą rezultatą identišką rezultatui panaudojant taisyklių rinkinius (žr. 13 punktą).

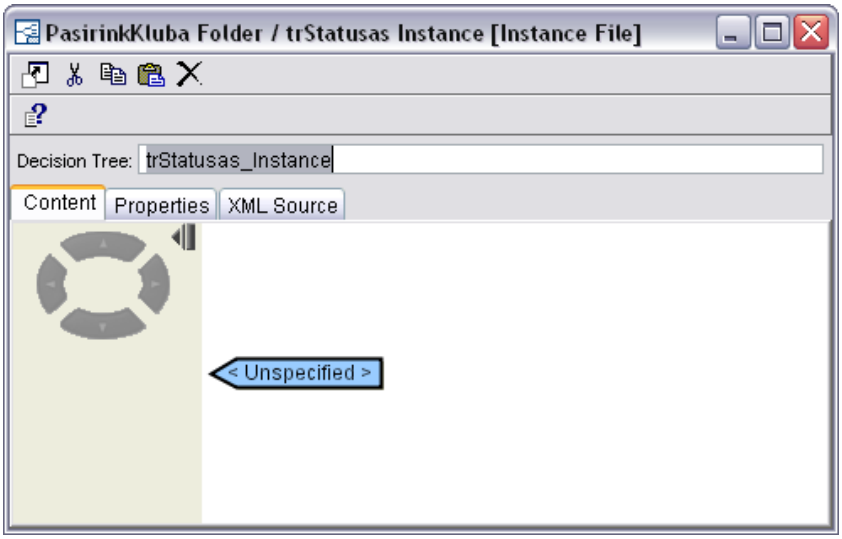
*16 punktas.* Šiame punkte aptarsime sprendimų medžių kūrimą. Kūrimas prasideda spragtelėjus ikoną  arba pasirinkus **Project > Decision Entities > New Decision Tree...**.

Panaudojimui sukuriami du elementarus sprendimų medžiai, kad būtų platesnis panaudojimas, iš vieno medžio bus kviečiamas kitas sprendimų medis. Kūrimo žingsniai yra tokie:

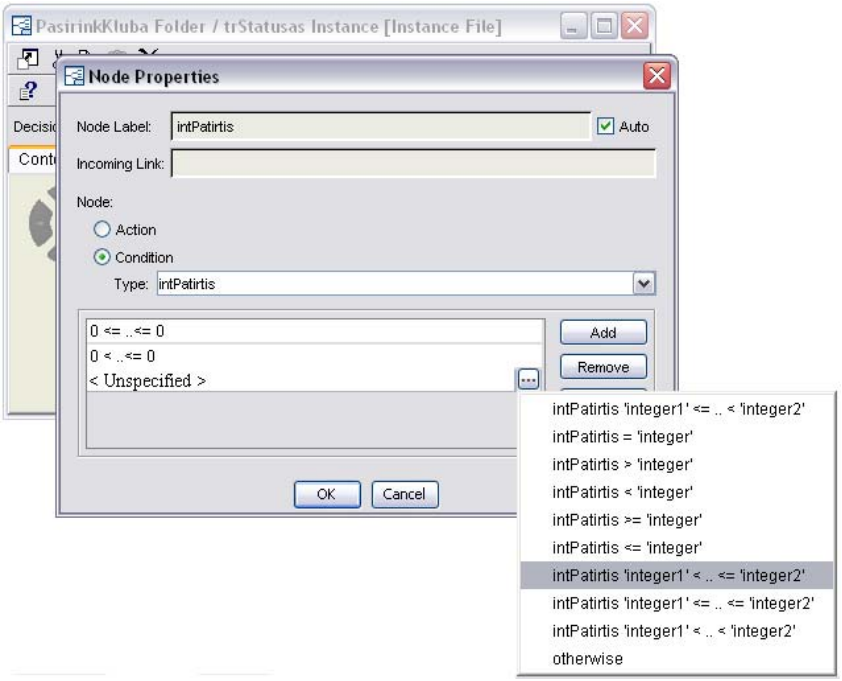
1) Pasirodžiusiame lange įvedame parametrus, kurie bus naudojami medžiams formuoti. Kadangi kūrimo principas yra labai panašus į sprendimo medžių kūrimą, tai nevisi

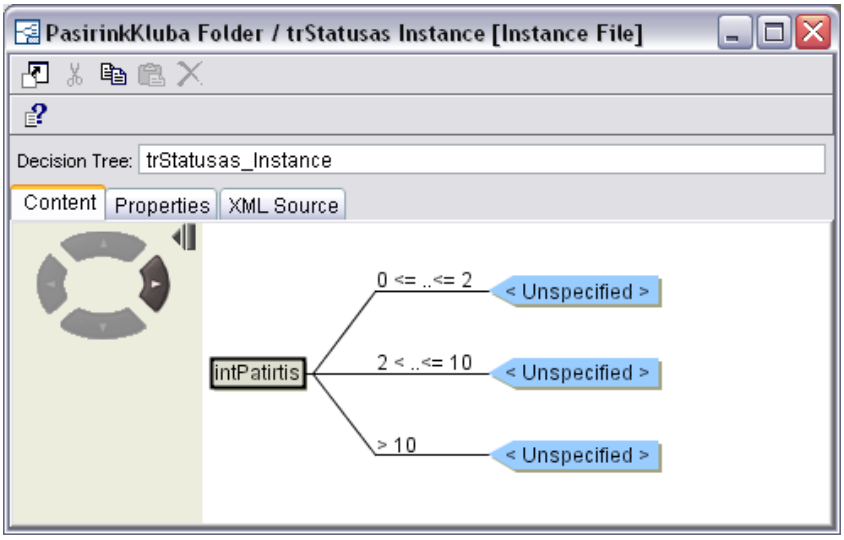
akcentai bus pavaizduoti paveikeliais. Taigi sukuriamas naujas parametras su *classVartotojas* tipu ir toliau spaudžiame *Next*.

2) Sekančiame lange reikia parinkti kintamuosius (parametrus) sąlygų formavimui. Įtraukiame tolesniam naudojimui vartotojo patirtį ir spaudžiame *Next*.
3) Toliau naudojame rezultatų (veiksmų) formavimui išrenkami parametrai. Čia įtraukiame vartotojo statusą grupėje, toliau spaudžiame *Finish*.
4) Toliau sukuriamas sprendimo medžio egzempliorius *trStatusas_Instance* ir pateikiamas langas medžio formavimui, kur kairės pusės viršutinėje dalyje pateikta navigacija, kurios pagalba galima apeiti visą medį:

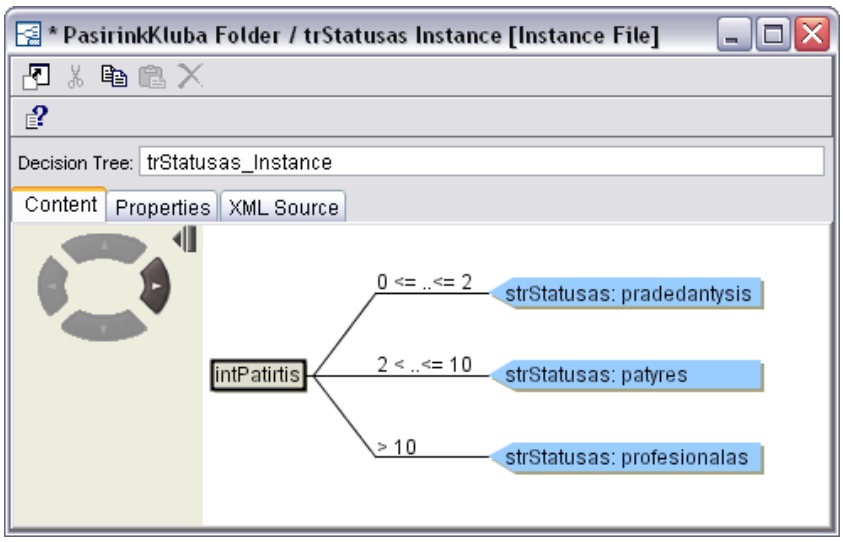



5) Spragtelėjus du kartus kairiu pelės klavišu ant medžio mazgo *Unspecified* pasirodo langas, kuriame galime įvesti sąlygas (išsišakojimus) ar veiksmus (rezultatus). Šiuo atveju pažymime *Condition* ir laukelyje *Type* nurodome, kad tikrinsime vartotojo patirtį. Toliau spaudžiame *Add* mygtuką, spaudžiame ant naujai atsiradusios eilutės  ir išsirenkame tipą. Baigę darbą su šiuo mazgu spaudžiame *OK*.






6) Sukuriamos trys medžio šakos su trimis skirtingais mazgais. Ant kiekvieno naujo mazgo *Unspecified* spragtelėjus du kartus kairiu pelės klavišu pasirodo toks pat langas, kaip ir medžio viršūnėje. Taip galima tęsti medžio struktūra praplečiant ir pereinant visas galimybes, kadangi ši dalis yra nedidele, toliau medis baigiasi. Kad galėtume gauti rezultatą pasirenkamas *Action* pasirinktį. Toliau spaudžiame *Add* ir mygtuką ir pasirenkame *strStatusas = 'string'* ir įrašome reikšmes. Tapati atlieka ir su kitais medžio galutiniais mazgais. Gauname tokį sprendimų medį:




7) Antro sprendimų medžio kūrimas yra analogiškas pirmam medžiui. Pirmiausia sukuriami du parametrai: Vartotojas iš *classVartotojas* ir Būsena iš *duomBusena* klasių. Į sąlygų formavimą įtraukiamas vartotojo amžius, tačiau prieš einant į kitą langą būtiną spragtelėjus mygtuką ir uždėti varnelę ant *SRL* eilutės, kaip ir atlikome sprendimų lentelėse. Ta pati atliekame veiksmų (rezultatų) lange, įkelti vartotojo būsena ir pažymėti taip pat *SRL* eilutę. Baigus kurti sprendimų lentelę sukuriamas egzempliorius *trBusena_Instance*. Toliau parodomas medžio formavimo langas, su šio medžio mazgais atliekame analogiškus veiksmus kaip ir pirmo medžio struktūros kūrime. Skiriasi tik tuo, kad sąlygų ir rezultatų tipai užsirašo *SRL* eilute kaip parodyta sekančiame paveikslėlyje:


8) Pirmose trijuose mazguose, netik parenkama vartotojo būsena, tačiau dar papildomai iškviečiamas pirmasis spendimų medis, kuris parenka vartotojo statusą. Kad galėtume iškviesti sprendimą pasinaudojant medžius pagrindinėje funkcijoje reikia įšviesti pasinaudojus komandą: *apply trBusena_Instance(objVartotojas, objBusena)*.
9) Įvykdžius šiuos visus žingsnius ir paleidus programą, su sprendimų medžiais išvedami rezultatai yra identiški rezultatams panaudojant taisyklių rinkinius ar sprendimų lenteles (žr. 13 punktą).