KLAIPĖDOS UNIVERSITETAS
JŪROS TECHNOLOGIJŲ IR GAMTOS MOKSLŲ FAKULTETAS
INFORMATIKOS IR STATISTIKOS KATEDRA

SIMONA NOSOVA

VERSLO TAISYKLIŲ AUTOMATIZUOTOS IR ATSEKAMOS REALIZACIJOS INFORMACINĖSE SISTEMOSE METODAS

Magistro baigiamasis darbas
Informacijos sistemų studijų programa: 621I20002

KLAIPĖDA, 2016

MAGISTRO BAIGIAMOJO DARBO LYDRAŠTIS

Simona Nosova

(magistrinio darbo autoriaus vardas, pavardė)

VERSLO TAISYKLIŲ AUTOMATIZUOTOS IR ATSEKAMOS REALIZACIJOS INFORMACINĖSE SISTEMOSE METODAS

(magistro baigiamojo darbo pavadinimas lietuvių kalba)

Patvirtinu, kad magistro baigiamasis darbas parašytas savarankiškai, nepažeidžiant kitiems asmenims priklausančių autorių teisių, visas baigiamasis magistro darbas ar jo dalis nebuvo panaudotas Klaipėdos universitete ir kitose aukštosiose mokyklose.

Simona Nosova

(magistro baigiamojo darbo autoriaus vardas, pavardė ir parašas)

Sutinku, kad magistro baigiamasis darbas būtų naudojamas neatlygintinai 5 m. Klaipėdos universiteto studijų procese.

Simona Nosova

(magistro baigiamojo darbo autoriaus vardas, pavardė ir parašas)

Pildo magistro baigiamojo darbo vadovas

Magistro baigiamąjį darbą ginti .....

(įrašyti – leidžiu arba neleidžiu)

(data) (magistro baigiamojo darbo vadovo vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studijų programą, administratorius (sekretorius)

Baigiamasis darbas įregistruotas katedroje .....

(data)

(katedros sekretorės vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studijų programą, vedėjas

Magistro baigiamąjį darbą ginti .....

(įrašyti – leidžiu arba neleidžiu)

(data) (katedros vedėjo vardas, pavardė ir parašas)

Recenzentu(-ais) skiriu

.....

(įrašyti recenzento(ų) vardą, pavardę)

(data) (katedros vedėjo vardas, pavardė ir parašas)

Klaipėdos universitetas

Jūros technologijų ir gamtos mokslų fakultetas

Informatikos ir statistikos katedra

Baigiamasis magistro darbas

Verslo taisyklių automatizuotos ir atsekamos realizacijos informacinėse sistemose metodas

Simona Nosova

Anotacija

Naujos taisyklėmis grindžiamos informacinės sistemos kūrimas ir tokių sistemų palaikymas, yra ištekliams imlus procesas. Taip yra dėl to, kad jis negali būti atliktas be informacinių sistemų modernizacijos specialistų pagalbos ir dėl to, kad, pasikeitus verslo kontekstui, paprastai keičiasi verslo taisyklės ir jų realizacija programų sistemoje. Šiuo metu nėra universalaus įrankio pritaikyto automatizuotam informacinės sistemos kūrimui ir palaikymui pasikeitus verslo taisyklėms, kuris užtikrintų taisyklių realizacijos atsekamumą, be kurio modernizacijos procesas yra sudėtingas.

Šiame darbe nagrinėjami verslo taisyklių transformacija į realizacijos lygmenį ir tokią transformaciją realizuojantis programinis kodas. Be to, nagrinėjamas šiame procese naudojamas taisyklių atsekamumas. Darbe pristatomas verslo taisyklių, užrašytų naudojant SBVR standartą, automatizuotos ir atsekamos realizacijos informacinėse sistemose metodas bei pateikiama sukurto metodo iliustracija. Metodas patikrinamas pritaikant jį pasirinktos dalykinės srities verslo taisyklių rinkiniui.

Pagrindiniai žodžiai: verslo taisyklės, automatizuota verslo taisyklių realizacija, taisyklių atsekamumas, SBVR

Klaipeda University

Faculty of marine engineering and natural sciences

Department of Informatics and Statistics

Master final thesis

Method for automated and traceable implementation of business rules in information systems

Simona Nosova

Abstract

Development and maintenance of an information system based on business rules is a resource-consuming process since it cannot be performed without the help of information system professionals. Also, when business context changes, business rules and their implementation in program system has to be changed as a result. A versatile tool allowing automated development and maintenance of information system when business rules change and supporting assurance of business rules implementation traceability is not created yet.

This paper analyzes transformation of business rules, their implementation in the source code and business rules traceability in this process. A method for automated and traceable implementation of business rules, expressed in Semantic of Business Vocabulary and Rules (SBVR), is proposed and illustration of it is presented. The method is validated for a selected set of business rules.

Keywords: business rules, SBVR, automated implementation of business rules, traceability of business rules

TERMINŲ IR SANTRUMPŲ SĄRAŠAS

ADBVS (angl. *Active Database Management System* (ADBMS)) – aktyvioji duomenų bazių valdymo sistema.

DDL- Duomenų apibrėžimo kalba (angl. Data Definition Language)

DML- Duomenų manipuliavimo kalba (angl. Data Manipulation Language)

ECA- Įvykis – sąlyga – veiksmas taisyklė (angl. Event-condition-action rules)

MDA- Modeliais paremta architektūra (angl. Model-driven architecture)

OCL- Deklaratyvi apribojimų kalba (angl. Object Constraint Language)

OMG- Object Management Group

SBVR- Verslo žodyno ir verslo taisyklių standartas (angl. The Semantics of Business Vocabulary and Business Rules) - OMG grupės standartas, kuris apima verslo žodyno sudarymą, ir verslo taisyklių, užrašomų ribota natūralia kalba, formavimą.

SQL- struktūrizuota užklausų kalba (angl. Structured Query Language), skirta kurti, modifikuoti ir valdyti reliacinių duomenų bazių duomenis.

UML- Unifikuota modeliavimo kalba (angl. Unified Modeling Language), kuria modeliuojami verslo procesai ir duomenų struktūros, programų struktūra, elgsena ir architektūra.

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

TURINYS

ĮVADAS

Tiriamoji problema

Šiuolaikinio verslo sėkmę didžiąja dalimi lemia sugebėjimas operatyviai reaguoti į besikeičiančią aplinką. Verslo sprendimai arba aplinkos pokyčiai įtakoja verslo taisyklių sistemą, todėl šių verslo ribojimų valdymas, kai dauguma verslo procesų palaiko informacinės sistemos, yra neabejotinai svarbus. Naujos taisyklėmis grindžiamos informacinės sistemos kūrimas ir keitimas, pasikeitus verslo kontekstui, yra brangus ir ilgai trunkantis, nes negali būti atliktas be informacinių sistemų specialistų pagalbos.

Dar nėra universalaus įrankio pritaikyto automatizuotam taisyklėmis grindžiamos informacinės sistemos duomenų bazės projektavimui ir verslo taisyklių realizavimui duomenų bazėje. Kadangi duomenų bazės schema yra kuriama verslo taisyklių, pagrindu, galima daryti prielaidą, jog pakankamai formaliai užrašytas verslo taisykles, pavyzdžiui taikant SBVR standartą, būtų galima automatizuotai įgyvendinti duomenų bazės schemoje arba naudojant kitus duomenų bazės elementus (pavyzdžiui, trigerius, saugomas procedūras, rodinius).

Tokiam automatizuotam verslo taisyklių įgyvendinimui svarbus atsekamumo ryšių tarp konkrečios taisyklės ir ją realizuojančio kodo valdymas. Atsekamumas yra apibrėžiamas kaip gebėjimas identifikuoti ryšį tarp įvairių artefaktų sistemos kūrimo procese. Nekaupiant atsekamumo ryšių informacijos būtų sudėtinga lokalizuoti, kuriuose programų sistemos elementuose yra realizuota konkreti verslo taisyklė ir tai apsunkintų jos palaikymo procesą.

Darbo aktualumas ir naujumas

Pagrindinis uždavinys, kurį šiuo metu stengiasi išspręsti modernios informacinių sistemų ir programų sistemų inžinerijos mokslo tyrėjai, yra programinio produkto kūrimo automatizavimas. Siekiama, kad programinis produktas būtų sukuriamas ne tik greitai, kokybiškai ir kuo mažesnėmis sąnaudomis. Nors jau yra automatiškai programinį kodą generuojančios programinės įrangos, ji nepakankamai universali. Šiai dienai uždavinys, kad verslo poreikių specifikacija, užrašyta neformalia kalba, be didelių žmogiškųjų išteklių naudojimo, virstų galutiniu programinių produktu, pilnai tenkinančiu keliamus reikalavimus, artimiausiu metu dar nebus išspręstas. Tačiau tyrimai šia kryptimi nuolat vykdomi.

Programinio produkto kūrimo procese nepakankamai išvystytas verslo taisyklių naudojimo požiūris, o būtent verslo taisyklės įtakoja sistemos reikalavimus ir turi didžiausią tendenciją keistis viso produkto gyvavimo ciklo metu. Informacinių sistemų ir programų sistemų inžinerijoje naudojami reikalavimų atsekamumo metodai nėra pritaikyti verslo taisyklėms.

Darbo tikslas

Išplėsti verslo taisyklių įgyvendinimo informacinėse sistemose metodus papildant juos taisyklių atsekamumu iki realizacijos programų sistemos komponentų lygmenyje.

Sprendžiami uždaviniai

1. Išanalizuoti verslo taisyklių formalumo lygmenis bei taisyklių klasifikaciją ir jų transformacijos metodus;
2. Išanalizuoti sistemų reikalavimų atsekamumo metodus bei jų taikymo galimybes verslo taisyklių atsekamumui įgyvendinti.
3. Pasiūlyti ir specifikuoti verslo taisyklių automatizuotos ir atsekamos realizacijos informacinėse sistemose metodą.
4. Sukurti pasiūlytą metodą realizuojančią iliustraciją ir aprašyti jos veikimo principus.
5. Atlikti tyrimą pritaikant pasiūlytą metodą įvairioms verslo taisyklėms ir taip įrodyti, kad pasiūlytą metodą galima įgyvendinti.

Tyrimo objektas

Darbo tyrimo objektas yra verslo taisyklių transformacija ir ją realizuojantis programinis kodas bei šiame procese naudojamų taisyklių atsekamumas.

Darbo struktūra

Darbą sudaro įvadas, trys skyriai ir išvados bei tolimesni darbai.

Tyrimo metodai

Darbe tyrimo metu naudojami metodai: literatūros apžvalga ir analizė, lyginamoji ir eksperimentinė analizė.

1 VERSLO TAISYKLĖS IR JŲ VAIDMUO INFORMACINIŲ IR PROGRAMŲ SISTEMŲ INŽINERIJOJE

1.1 VERSLO TAISYKLĖS IR JŲ KLASIFIKAVIMAS

1.1.1 Verslo taisyklių apibrėžimas ir jų sąryšis su sistemos reikalavimais

Verslo taisyklės literatūros šaltiniuose yra apibrėžiamos labai įvairiai. Kaip teigiama šaltinyje [1], verslo taisyklė yra direktyva, skirta daryti įtaką verslui ar jį valdyti, taip realizuojant jo politiką, kurią sąlygoja galimybės vystytis bei išorinės ar vidinės grėsmės. Tai yra bet kokie apribojimai ar nurodymai, kurie reguliuoja verslą, jo funkcionavimą bei struktūrą. Knygoje [2] verslo taisyklės įvardijamos ir kaip vienas iš pirminių sistemos reikalavimų šaltinių, apimančių bendrąsias verslo bei verslą reguliuojančių teisės aktų nuostatas, verslo naudojamų standartų nuostatas, verslo objektų ribojimus. Verslo taisyklės iš tikrųjų yra informacinės sistemos funkcinių reikalavimų pagrindas [1]. Šaltinio [3] autoriai verslo taisykles apibrėžia kaip „atominį“ (nedalomą) teiginį, kuris apibrėžia arba riboja kokį nors verslo aspektą.

Verslo pokyčiai paprastai yra sąlygojami arba vidinių sprendimų, kuriuos priima verslo savininkai arba išorinių veiksnių, tokių kaip įstatymų pasikeitimai. Šie pokyčiai paprastai reikalauja egzistuojančių verslo procesų keitimo. Tokiu atveju būtent verslo taisyklės ir jų realizacija keičiasi verslo procesuose ir tai reikalauja jų peržiūros ir pritaikymo naujiems verslo tikslams. Kaip teigiama šaltinyje [2] kompiuterizuotos sistemos kuriamos ne tam, kad įšaldyti esamas verslo taisykles, o tam, kad tobulinti esamą verslą, nuolat keičiant esamas verslo taisykles.

Verslo taisyklių išgavimas nėra paprasta užduotis, nes dauguma jų yra sunkiai identifikuojamos. Dažnai versle jau taikomos taisyklės tiesiog neturi aiškiai apibrėžtos formuluotės. Šią taisyklių specifikavimo užduotį dar labiau apsunkina ir tai, jog verslo taisyklės turi būti suprantamos ir verslui, ir informacinės sistemos kūrėjams. Verslininkai dažnai nėra susipažinę su formaliąja kalba, tuo tarpu būtent ji yra tiesiog būtina, kuomet programuotojams keliama užduotis tas taisykles įgyvendinti programos kodu. Čia ir slypi pagrindinė problema - rasti tokią optimalią kalbos išraišką, kuri tenkintų abi suinteresuotąsias puses - būtų suprantama verslo atstovams ir pakankamai aiški bei nepaliekanti vietos dviprasmybėms informacinės sistemos kūrėjams.

1.1.2 Verslo taisyklių klasifikavimas

Kaip teigiama [3] dokumente, kiekvieną iš verslo taisyklės teiginių galima priskirti vienai iš šių trijų klasių:

- **Struktūriniai ribojimai** (angl. *Structural assertions*)- terminai ir apibrėžimai, faktai apjungiantys terminus.

Svarbiausias verslo taisyklės elementas yra jai išreikšti naudojama kalba, todėl kiekviena dalykinėje srityje naudojama sąvoka kartu su jos apibrėžimu yra viena iš verslo taisyklių sudedamųjų dalių, apibūdinanti, kaip žmonės suvokia konkrečius verslo dalykus. Terminas yra žodis arba frazė, turinti specifinę reikšmę verslui. Tradiciškai terminai, išreiškiantys veiklos sąvokas, dokumentuojami dalykinės srities žodynuose arba kaip esybės esybių-ryšių modelyje. Terminai, apibrėžimai ir faktai visgi gali būti laikomi sąvokomis, o ne taisyklėmis [4].

Faktais yra išreiškiami teiginiai apie dalykinėje srityje naudojamas sąvokas. Tokie teiginiai nusako ryšius tarp terminais išreikštų sąvokų. Faktų pagrindu yra kuriamos verslo taisyklės, t.y. taisyklės riboja ir remia faktus. Kaip teigia [5] autoriai, faktai yra tai, kas vaizduojama duomenų modelyje- tarpusavyje susiję esybių tipai, jų atributai, bei potipiai. Faktai gali būti dokumentuojami tiek natūralios kalbos sakiniais, tiek grafiniais modeliais [3].

- **Veiksmų ribojimo taisyklės** (angl. *constraints, action assertions*)

Ribojimo taisyklė yra taisyklė, siejama su dinaminiu verslo aspektu. Tokia taisyklė apriboja rezultatus kuriuos gali sąlygoti tam tikri veiksmai [3]. Ši taisyklių klasė gali būti išskaidyta į tris smulkesnes taisyklių klases- sąlygos (angl. C*ondition*), vientisumo ribojimo (angl. *Integrity constraint*) bei autorizavimo (angl. *Authorization*). Sąlygos taisyklių klasei priskirtinos ir ECA (angl. *Event Condition Action*) taisyklės, susidedančios iš tam tikros sąlygos tikrinimo, bei tą sąlygą tenkinant konkrečios kitos taisyklės vykdymo. Vientisumo ribojimo taisyklėms priskiriamos tokios taisyklės, kurios griežtai apriboja tam tikros sąlygos privalomą tenkinimą- t.y., draudžiami bet kokie veiksmai, kurie sąlygotų, kad po jų įvykdymo sąlyga nebebus tenkinama.

- **Išvedimo taisyklės** (angl. *derivation*)

Išvedimo taisyklė gali būti dviejų rūšių - matematiniai skaičiavimai arba loginis išvedimas (angl. *inference*) [3] [5]. Matematinio skaičiavimo rezultatas yra išvedimo faktas (angl. *derived*

fact) gautas remiantis tam tikru matematiniu algoritmu. Loginio išvedimo rezultatas yra taip pat išvedimo faktas, tačiau gautas remiantis loginės indukcijos ir dedukcijos principais [3].

Šiek tiek kitokia taisyklių klasifikacija aprašyta straipsnyje [6]. Čia išskiriamos penkios taisyklių klasės- 1) vientisumo taisyklės, aprašančios kardinalumo ryšius tarp esybių (pvz. 1 su 1 arba 1 su daug); 2) išvedimo taisyklės; 3) reagavimo (angl. *reaction*) arba ECA taisyklės; 4) sąlygos- veiksmo taisyklės (angl. *production rules*); 5) transformacijos taisyklės, ribojančios objektų būsenų pasikeitimus.

Dar labiau apibendrinta verslo taisyklių klasifikacija pasiūlyta [4] ir [7] autorių- siūloma verslo taisykles klasifikuoti į struktūrines (terminai, apibrėžtys, faktai ir vientisumo ribojimai) arba statines, bei dinamines (ECA taisyklės).

1.1.3 Verslo abstrakcijos ir taisyklių formalumo lygmenys

Remiantis šaltiniu [5], galima išskirti tris verslo sistemos abstrakcijos lygmenis—verslo sistemos, informacinės sistemos ir programų sistemos. Kaip pavaizduota 1 lentelėje, kiekvienas lygmuo turi jį atitinkančią sistemą, objektus bei taisykles:

1 lentelė. ***Abstrakcijos lygmenys, juos atitinkančios sistemos, objektai bei taisyklės***

Abstrakcijos lygmuo	**Sistema**	**Objektas**	**Taisyklė**	**Išraiška**
Verslo sistemos lygmuo	Verslo sistema	Verslo objektas	Verslo taisyklė	Natūralia kalba
Informacinės sistemos lygmuo	Informacinė sistema	Informacinis objektas	Informacijos apdorojimo taisyklė	ER/ORM/UML klasių diagrama (objektams) / specifikavimo kalba (taisyklėms)
Programų sistemos lygmuo	Programų sistema	Skaitmeninis objektas	Vykdomoji taisyklė	Programinis kodas, pvz., ADBVS SQL trigeriai

Pagal tris verslo abstrakcijos lygmenis gali būti išskiriami keturi verslo taisyklių formalumo lygmenys [5]. Pirmajam taisyklių formalumo lygmeniui priskiriamos taisyklės, kurios yra išreikštos neformalia verslo kalba, ir joms netaikomi griežti specifikavimo reikalavimai. Antrajam lygmeniui priskirtinos taisyklės taip pat yra išreikštos neformalia kalba, tačiau jos jau turėtų tiksliai atspindėti įmonės tikslus, ir būti sudarytos naudojant verslo terminus bei tam tikras

šio lygmens taisyklių specifikavimo normas. Pažymėtina, jog šio lygmens taisyklės turi būti atominės (nedalomos) bei vis dar aiškiai suprantamos ir verslo atstovams, nes jie turi patvirtinti jų teisingumą. Trečiojo formalumo lygmens taisyklės jau turėtų būti užrašytos formalia taisyklių specifikavimo kalba. Šio lygmens taisyklė aprašo ką reikia daryti, tačiau neaprašo kaip- tai jau ketvirtojo formalumo lygmens taisyklės paskirtis. Taisyklių formalumo lygmenys ir jų savybių privalomumas kiekviename lygmenyje pavaizduotas 2 lentelėje:

2 lentelė. ***Reikalavimai taisyklėms pagal abstrakcijos lygmenis***

	Verslo sistema		**Informacinė sistema**	**Programų sistema**
Formalumo lygmenys / **Taisyklės savybės**	**Veiklos pokalbio dalis**	**Taisyklė (natūralia kalba)**	**Taisyklė (specifikavimo kalba)**	**Įgyvendinta taisyklė**
Tiesiogiai susijusi su tam tikru objektu	Neprivaloma	Privaloma	Privaloma	Vykdomoji Gali būti procedūrinė
Atominė	Neprivaloma	Privaloma	Privaloma	
Deklaratyvi	Neprivaloma	Privaloma	Privaloma	
Tiksli	Neprivaloma	Neprivaloma	Privaloma	
Pilna	Neprivaloma	Neprivaloma	Privaloma	
Patikima	Neprivaloma	Privaloma	Privaloma	
Autentiška	Neprivaloma	Neprivaloma	Privaloma	
Unikali (neperteklinė)	Neprivaloma	Neprivaloma	Privaloma	
Neprieštaringa	Neprivaloma	Neprivaloma	Privaloma	

Panašiai taisyklių formalumo lygmenys apibrėžiami ir straipsnyje [8], kur taisyklės Gali būti neformalios, išreikštos natūralia kalba, pusiau formalios ir formalios. Taisyklės, priklausomai nuo to, kuriam abstrakcijos lygmeniui jos priklauso, pagal [5] gali būti apibrėžiamos taip:

Verslo sistemos lygmenyje verslo taisyklė – tai teiginys, kuris apibrėžia arba apriboja tam tikrus verslo aspektus.

Informacinės sistemos lygmens taisyklė – tai teiginys, apibrėžiantis informacijos apdorojimo taisykles ir užrašytas tam tikra taisyklių kalba.

Programų sistemų lygmenyje taisyklės – tai teiginiai, paversti vykdomosiomis taisyklėmis (angl. *executable rules*), pavyzdžiui, ADBVS SQL trigeriai.

Jei šie trys verslo abstrakcijos lygmenys (verslo sistemos, informacinės sistemos ir programų sistemos) yra įgyvendinti teisingai, visi pokyčiai aukštesniame lygmenyje reflektuojami į žemesnį lygmenį. Žemesnio lygmens sistema yra ribojama aukštesnio lygmens sistemos, taigi visos veiklos turėtų būti modeliuojamos laipsniškai iš verslo sistemos lygmens į programų sistemos lygmenį [9].

1.2 VERSLO TAISYKLIŲ TRANSFORMACIJOS METODŲ APŽVALGA

Verslo atstovai, analitikai ir sistemos projektuotojai dažnai gali naudoti skirtingas sąvokas įvairiems objektams apibrėžti. Tiek nepakankamai tikslus taisyklių apibrėžimas verslo sistemos lygmenyje, tiek skirtingų terminų naudojimas taisyklių transformavimo etapuose gali sąlygoti taisyklių iškraipymą ir neteisingą realizavimą programų sistemos lygmenyje.

Verslo taisyklių transformacija yra atliekama dviem etapais- natūralia kalba išreikštos verslo taisyklės pirma transformuojamos iš verslo sistemos lygmens į informacinių sistemų lygmenį, ir antruoju etapu iš informacinių sistemų lygmens jau formalia kalba išreikštos taisyklės transformuojamos į vykdomąsias taisykles programų sistemų lygmenyje [1], [5]. Pirmoji taisyklių transformacija iš verslo lygmens nėra paprasta, nes susiduriama su tokiomis problemomis kaip jų netikslumas, neišsamumas, prieštaringumas, neautentiškumas [5]. Informacijos apdorojimo taisyklėms išreikšti gali būti naudojama natūrali kalba, struktūrizuota anglų kalba, objektų ribojimų kalba (OCL), Ronaldo Rosso notacija ir kt.

Kaip teigia [10], [1] ir [11] autoriai, verslo taisyklės turi būti saugomos centralizuotai- verslo taisyklių saugykloje. Šiuo metu naudojami labai įvairūs verslo taisyklių valdymo bei jų realizavimo įrankiai. Dažniausiai jų pagrindinės funkcijos yra trys [11]:

1) taisyklių įvedimas ir keitimas- taisyklių redaktorius (angl. *rule editor*);
2) taisyklių saugojimas- taisyklių saugykla (angl. *rules repository*);
3) taisyklių vykdymas- taisyklių varikliai (angl. *rule engine*).

Išskiriami du taisyklių saugojimo saugyklose būdai [10]- formalia kalba išreikštų taisyklių įvedimas į saugyklą, kas dėl formalios kalbos specifikos riboja verslo atstovų galimybes tas taisykles keisti, ir antrasis, natūralia kalba išreikštų taisyklių saugojimas, kuris deja nelei džia tokių taisyklių apdoroti automatiškai. Yra ir dar vienas, iš dalies šią problemą išsprendžiantis būdas, kuomet taisyklės užrašomos natūralia ribojimų kalba (angl. *controlled natural language*), tačiau čia susiduriama su kitomis problemomis- tokiu būdu gali būti išreikštos tik gana paprastos verslo taisyklės. Be to, dauguma šią kalbą palaikančių įrankių yra pritaikytos tik anglų kalbai [10].

1.2.1 Natūralia kalba išreikštų verslo taisyklių formalizavimo metodai

Natūralia kalba išreikštų verslo specifikacijų, tuo pačiu ir verslo taisyklių, transformavimo į formalius modelius problema yra labai aktuali, ir kol kas optimalus sprendimas jai dar nėra rastas. Yra gana didelis atotrūkis tarp natūralios kalbos, kuriai pirmenybę teikia veiklos ekspertai bei formalaus modelio reikalingo sistemos kūrėjams. Ši transformacija reikalauja daug „rankinio“ darbo- reikalavimų analizės, darnos tikrinimo ir panašiai. Juo labiau, kad sukurtas formalus modelis vėliau vis tiek turi būti patikrintas dalykinės srities ekspertų, kurie labai tikėtina nėra susipažinę su formalia notacija ir turi būti jos apmokomi [12].

Tobulinant natūralios kalbos taisyklių transformacijos metodus siekiama, kad taisyklių valdymas, tuo pačiu ir jų formalizavimas reikalautų kuo mažesnio informacinių technologijų specialistų įsikišimo, ir būtų paliktas tik dalykinės srities ekspertams. Be to, siekiama sumažinti žmogiškųjų išteklių pastangas reikalingas šiam formalizavimui atlikti. Kaip teigia [4] autoriai, modeliuojant verslo taisykles, susiduriama su tokiomis problemomis, kaip sunkus jų identifikavimas dalykinėje srityje, nes jos neturi formalios išraiškos ir dažnai yra įvairių su taisyklėmis nesusijusių verslo tekstų dalis. Kita problema yra vieningos taisyklių klasifikavimo sistemos nebuvimas, o dėl taisyklių įvairovės ne visada yra aišku kaip tos taisyklės turi būti įgyvendinamos.

Kaip teigiama [12], galima išskirti tris skirtingas požiūrių šiai problemai apie natūralios kalbos transformavimą į formalius modelius spręsti kategorijas:

- metodai, kuomet pilnai išanalizuojama natūralios kalbos specifikacijos gramatika;
- metodai teksto analizei naudojantys informacijos išgavimo technikas;
- metodai naudojantys ir formalios gramatinės analizės ir informacijos išgavimo technikas.

Tam, kad natūralia kalba išreikštas verslo taisykles būtų galima apdoroti automatiškai, jas būtina užrašyti laikantis tam tikrų standartų, t.y. naudojant natūralios kalbos poaibį, apribotą leksine, sintaksine ir semantine prasme. Tokia kalba pakankamai išraiškinga, kad ją galėtų naudoti ne tik IT specialistai, ir pakankamai formali, kad tiktų kompiuterizavimui [13]. OMG (angl. *Object Management Group*) sukūrė standartą SBVR (angl. *Semantics of Business Vocabulary and Business Rules*) - verslo žodyno ir verslo taisyklių semantiką [14], kuri leidžia neformalia kalba išreikštas taisykles aprašyti formaliai ir standartizuotai.

Kaip teigiama knygoje [15] taisyklių išgavimo metodika remiasi natūralia kalba užrašyto taisyklių šaltinio teksto progresyvia transformacija į laikantis SBVR standarto suformuluotas taisykles. Šios knygos autoriai siūlo padalinti transformacijos procesą į keturias veiklas- šaltinio teksto leksikos normalizaciją, aktualių teksto fragmentų išgavimą, šių fragmentų sintaksinę normalizaciją bei semantinę transformaciją kontekstinei informacija atkurti. Šio proceso galutinis rezultatas yra SBVR ribojimų kalba užrašytų taisyklių rinkinys.

Panašus metodas pasiūlytas ir straipsnio [16] autorių- iš natūralia kalba užrašytų taisyklių tekstinio dokumento gali būti atlikta transformacija į SBVR automatiniu būdu. Tam pirmiausia angliškas natūralia kalba užrašytas taisyklių tekstas yra lingvistiškai išanalizuojamas, sužymimos kalbos dalys (angl. *Parts-Of-Speach*), iš kurių tam tikros programinės įrangos pagalba išgaunami SBVR elementai- daiktavardinės sąvokos, faktų tipai ir t.t. SBVR žodynui išgauti kitame etape yra naudojamas UML modelis. SBVR taisyklės iš UML klasių modelio ir natūralia kalba išreikštų taisyklių yra išgaunamos naudojant UML, NLP ir SBVR modulius, kurių kiekvienas atlieka tam tikrą transformaciją- UML modulis išgauna visas klases, objektus, ir jų atitinkamus atributus bei operacijas ir susieja jas su SBVR žodynu. Vėliau SBVR žodynas susiejamas su NLP moduliu išgautais SBVR elementais. SBVR modulis remdamasis tam tikromis taisyklėmis sugeneruoja išbaigtą SBVR taisyklę.

1.2.2 SBVR standartas

Verslo žodynas

Pagrindinė SBVR paskirtis yra sukurti centralizuotą verslo terminologijos ir taisyklių rinkinį, kuris būtų skirtas ne tik atskiriems projektams ar keliems įmonės padaliniams, bet būtų pakartotinai naudojamas daugelyje projektų ir įvairiose situacijose. Be to, toks taisyklių užrašymas struktūrizuota kalba leidžia vėliau jas automatizuotai transformuoti į kitus standartus [17].

SBVR verslo žodynas naudojamas verslo taisyklėms formuoti, taigi jis turi būti apibrėžtas prieš verslo taisyklių konstravimą [17]. Pagrindiniai SBVR elementai yra daiktavardinės sąvokos (angl. *Noun concept*), dar vadinami terminais, ir faktų tipai (angl. *Fact types*), dar vadinami faktais. Faktai yra sudaromi iš terminų, juos jungiant veiksmažodžiais ir raktiniais žodžiais.

Daiktavardinės sąvokos kaip aprašyta [17] skirstomos į tris rūšis:

- Objekto tipas (angl. *Object type*) – tai yra daiktavardiniai žodžiai ar jų grupės, kurios paprastai nusako verslo subjektus (esybes). Pačiuose SBVR standartuose [14] ši sąvokų rūšis vadinama *bendra sąvoka* (angl. *General concept*), arba kaip aprašoma [18], dar vadinama *terminu;*
- Vaidmuo (angl. *Role*)- daiktavardinės sąvokos susijusios su objekto tipu, priklausomai nuo atliekamo vaidmens, pavyzdžiui, „pirkėjas“ gali būti vaidmuo objekto tipui „klientas“.
- Individualus konceptas (sąvoka) (angl. Individual concept)- tai pavadinimas nusakantis vienetinį esybės egzempliorių.

Fakto tipas- tai sakinys, sudarytas iš veiksmažodžio ir vienos arba dviejų daiktavardinių sąvokų (terminų), nusakantis ryšį tarp konceptų [17], [18]. Faktas yra atominis SBVR vienetas, paremtas veiksmažodine forma, tačiau neturintis jokios loginės operacijos. Šiuo požiūriu taisyklė ir verslo taisyklė yra aukštesnio lygio faktas [14].

Verslo taisyklių konstravimas pagal SBVR

Verslo taisyklės konstruojamos apjungiant faktų tipus su raktiniais žodžiais [17], kurie skirstomi į kvantorius (pvz., kiekvienas (angl. *each*), bent vienas (angl. *at least one*) ir kt.), modalumo raktinius žodžius, nusakančius būtinybę (pvz., būtina kad (angl. *it is necessary that*), draudžiama kad (angl. *it is prohibited that*)), loginius operatorius (ir, arba, jei...tai (angl. *and, or, if...then*), ir kitus jungiamuosius žodžius (pvz., kitas, kuris (angl. *another, who*)) [14], [17]. Modalumo raktiniai žodžiai yra skirstomi į būtinybės (angl. *necessity*), įsipareigojimo arba prievolės (angl. *obligation*), leidžiamumo (angl. *permissibility*) ir galimybės (angl. *possibility*). Taigi, taisyklės struktūros šablonas, priklausomai nuo taisyklės sudėtingumo, galėtų būti toks:

Modalumo raktinis žodis + kvantorius + terminas + veiksmažodinis konceptas + kvantorius + objekto tipas

Pavyzdžiui:

It is obligatory that each student *prepare* at least one presentation

Pažymėtina, jog SBVR specifikacija apibrėžia ir šių taisyklės struktūrinių elementų vaizdavimo šrifto stilių (spalvą, pabraukimą, kursyvą). Naudojant SBVR verslas gali apibrėžti savo verslo taisykles verslo žodyno pagrindu [19]. Tačiau SBVR neaprašo, kada tos taisyklės yra iššaukiamos (angl. *triggered*) ir kada jos turi būti vykdomos. Paprastos duomenų apdorojimo

taisyklės, tokios kaip duomenų vientisumo ribojimai (angl. *integrity constraints*) gali būti įgyvendinamos duomenų bazės schemos ribojimais. Ilgalaikėms taisyklėms, kurios tikėtina nesikeis, tinka OCL (angl. *Object Constraint Language*) ribojimai. Visgi sudėtingoms taisyklėms, ir taisyklėms kurios nuolat keičiasi reikia kitokio metodo. Kaip pasiūlė straipsnio [19] autoriai, tokias sudėtingas SBVR taisykles galima būtų transformuoti į įvykiais grindžiamas (angl. *event-driven*) vykdymo taisykles. Tokią SBVR taisyklių transformaciją į vieną arba kelias įvykis-sąlyga-veiksmas (ECA) taisykles siūloma atlikti naudojant sukurtus taisyklių šablonus. Šablonų naudojimas apriboja galimas taisyklės formuluotes, tačiau leidžia paprastai iš verslo taisyklės išgauti reikalingą informaciją, tokią kaip verslo taisyklės rūšis, taisyklėje naudojami terminai arba kardinalumo ryšiai [19], [20]. Tokia informacija reikalinga transformacijai iš SBVR į ECA taisykles atlikti. Kiekvienas šablonas susiejamas su vienu arba daugiau ECA taisyklių. Šablonas šio atveju naudojamas kaip transformavimo funkcija.

1.2.3 Taisyklių transformacijos į programų sistemos lygmenį metodai

Literatūroje yra aprašyta nemažai metodų verslo taisyklėms išgauti, modeliuoti ir transformuoti iš verslo sistemos lygmens į programų sistemos lygmenį. Visgi, nei vienas iš esamų metodų nėra pripažintas standartu [4]. Siekiama sukurti metodą, kuris leistų automatizuoti verslo taisyklių įgyvendinimą.

Informacinės sistemos kūrimui ir funkcionavimui reikalingos dalykinės srities žinos yra aprašomos ontologijų kalbomis (šiuo metu OWL 2, [13]). Ontologijų kalba yra sudėtinga, todėl ontologijos, t.y. formalūs veiklos ar programinės įrangos veikimo modeliai, yra kuriami informacinių technologijų specialistų. Tokių IT srities žinovų sukurtų modelių dėl ontologijų kalbos sudėtingumo dažnai negali patikrinti net dalykinės srities žinovai.

Disertacijos [13] autoriaus sukurtos SBVR veiklos žodynų ir taisyklių transformavimo į OWL 2 ontologijas taisyklės ir algoritmai bei sukurtas įrankio, leidžiančio vykdyti SBVR metamodelį atitinkančių veiklos žodynų ir taisyklių transformacijas į OWL 2 ontologijas, prototipas parodė, kad šios transformacijos yra įmanomos ir veiksmingos. Disertacijos autoriaus eksperimentas su 9 veiklos žodynais ir taisyklių aibėmis patvirtino, kad transformacijos ir reikalavimai, apibrėžti veiklos žodynams ir taisyklėms kurti, leidžia: 1.“*transformuoti numatytą SBVR poaibį ir gauti taisyklingas OWL 2 DL ontologijas“*; 2. *„patikrinti SBVR veiklos žodyno ir taisyklių*

neprieštaringumą, panaudojant ontologijų tikrinimo įrankius“; 3. *„pritaikyti transformacijas lietuvių, anglų ir kitoms panašioms natūralioms kalboms“*.

Straipsnio [4] autoriai pasiūlė ontologija grindžiamą metodą verslo taisyklėms modeliuoti, kuris naudoja ontologijoje apibrėžtas dalykinės srities žinias ir leidžia automatizuoti verslo taisyklių įgyvendinimą naudojant aktyvių duomenų bazių valdymo sistemų trigerius. Kadangi ontologija vaizduoja ne tik verslo konceptus, bet ir struktūrą ribojančias taisykles (aksiomas), kurios paprastai turi formalią išraišką, straipsnio autoriai teigia, jog būtent tai leidžia jas transformuoti į verslo taisykles automatiškai. Autorių pasiūlytu metodu ontologijos aksiomas siūloma transformuoti į atitinkamas vykdymo (aktyviąsias arba dinamines) ECA taisykles.

Aktyvioji taisyklė- tai sintaksinė išraiška, aprašanti sistemos reakciją į tam tikrą įvykį. Tokių taisyklių struktūra- įvykis-sąlyga-veiksmas (angl. *Event Condition Action*):

Įvykis- sistema stebi ar įvyksta taisyklę galintis aktyvuoti įvykis, pavyzdžiui signalas, informuojantis, kad duomenų bazėje pasikeitė duomenys.

Sąlyga- deklaratyvi formulė, kuri turi būti tenkinama, kad būtų vykdomas taisyklės veiksmas, ji vertinama įvykus su taisykle susijusiam įvykiui;

Veiksmas- nurodymai kas turi būti vykdoma, jei taisyklė aktyvuojama įvykiu, ir jos sąlyga yra tenkinama.

Aktyviose duomenų bazių valdymo sistemose (ADBVS) tokios taisyklės paprastai realizuojamos trigeriais [5]. Jei taisyklėje yra neapibrėžtas ją aktyvuojantis įvykis, tai tokia taisyklė vadinama produkcine taisykle, t.y. sąlyga-veiksmas taisyklė. Bendrąja prasme, trigeriai turi tas pačias savybes kaip ir ECA taisyklės, ir dėl to šios dvi sąvokos gali būti laikomos sinonimais [21].

Šaltinyje [4] aprašytas metodas transformuoti ontologijos aksiomas į vykdomąsias taisykles. Juo siūloma iš pradžių patikrinti, ar pasirinkta ontologija turi aksiomas, ir jei jų nėra- jas apibrėžti. Kiekviena ontologijos aksioma transformuojama į atitinkamą ECA taisyklę- apibrėžiamas duomenų bazės įvykis (insert, update, delete), sąlyga ir veiksmas. Jei aksioma turi veiksmą, ji transformuojama į ECA taisyklę. Jei aksioma veiksmo neturi, veiksmas apibrėžiamas kaip leidimas būsenos pasikeitimui jei sąlyga yra tenkinama ir draudimas būsenos pasikeitimui, jei sąlyga netenkinama.

1.3 SISTEMOS REIKALAVIMŲ ATSEKAMUMAS

1.3.1 Atsekamumo apibrėžimas ir atsekamumo ryšių tipai

Reikalavimų atsekamumas yra dinamiška tyrimų sritis, o reikalavimų atsekamumo reikšmė ypač išaugo programinę įrangą kuriant iteraciniu būdu, kuomet vis nauja sistemos versija su didesnėmis ar mažesnėmis modifikacijomis pristatoma vartotojui. [22]. Terminas reikalavimų atsekamumas apibrėžiamas kaip galimybė aprašyti bei sekti reikalavimo gyvavimo ciklą dvejomis kryptimis. [23] Kiekvienam sistemos reikalavimui turi būti sukonstruoti tokie abipusiai ryšiai (trasos), kuriais judant būtų galima pasiekti ir to reikalavimo pirminius šaltinus, ir, judant priešinga kryptimi- projektavimo sprendimus, nustatančius, kaip tą reikalavimą reikia įgyvendinti, tuos sprendimus įgyvendinantį kodą bei to kodo teisingumui tikrinti skirtus testus [2]. Kaip parodė šaltinyje [24] aprašyto tyrimo rezultatai, atsekamumas labiausiai yra naudojamas reikalavimų inžinerijoje ir valdyme, projektų valdyme, tačiau mažiau naudojamas projektavimo ir įgyvendinimo bei sistemos palaikymo etapuose.

Net ir paprasti sistemos reikalavimų pakeitimai gali sąlygoti tai, kad jų įgyvendinimui teks modifikuoti gana daug sistemos komponentų. Atsekamumo informacija palengvina poveikio analizės atlikimą. Pasikeitus verslo poreikiams, o tuo pačiu ir verslo taisyklėms, vienos verslo taisyklės pasikeitimas gali iššaukti kitų taisyklių pakeitimus [25]. Tam, kad teisingai įvertinti siūlomo pakeitimo įtaką, reikia identifikuoti su šiuo pakeitimu susijusius komponentus, o tai gali būti labai sudėtingas uždavinys. Šio uždavinio sprendimui palengvinti reikėtų naudoti veiklos grafiką (angl. *roadmap*), kuris parodo, kur kiekvienas sistemos reikalavimas arba verslo taisyklė yra įgyvendinta programų sistemos lygmenyje. Projektų vadovai sekdami atsekamumo ryšius gali gana greitai įvertinti, kiek artefaktų būtų paveikta siūlomu sistemos pakeitimu, ir priimti sprendimą atsižvelgiant į numatomus pakeitimo kaštus ir su juo susijusias rizikas [26]. Kaip teigiama šaltinyje [27] reikalavimų atsekamumas dokumentuoja priklausomybės ir loginius ryšius tarp funkcinių reikalavimų ir kitų sistemos elementų. Šie elementai- tai kiti sistemos reikalavimai, verslo taisyklės, architektūros komponentai, programinio kodo moduliai, testavimo atvejai ar kita pagalbinė dokumentacija. Atsekamumas ne tik padaro sudėtingus ryšius ir priklausomybes tarp programos kūrime sugeneruojamų objektų lengviau suprantamais, bet ir leidžia sistemos kūrėjams lengviau sekti visą sistemos vystymo procesą- nuo reikalavimų iki jų realizacijos [28].

Atsekamumo ryšiai leidžia sekti reikalavimo gyvavimo ciklą abejomis kryptimis- nuo reikalavimo šaltinio iki jo realizacijos programų sistemos lygmenyje ir atvirkščiai- nuo realizacijos iki jo šaltinio. Tokio atsekamumo įgyvendinimui reikalinga, kad kiekvienas reikalavimas turėtų unikalią ir nekintančią žymę (angl. *label*), kuri leistų nedviprasmiškai ja remtis viso projekto gyvavimo ciklo metu. Suinteresuotųjų šalių požiūris ir poreikiai įtakoja kokius atsekamumo ryšius tikslinga naudoti konkrečiame programinės įrangos kūrimo projekte.

Tradiciniai sistemų inžinerijos procesai nemini reikalavimų atsekamumo kaip labai svarbios produkto kūrimo veiklos. Atsekamumo informacija yra surenkama, ir didžiąją laiko dalį pasyviai saugoma įvairiuose dokumentuose. Tai sudaro įspūdį, kad reikalavimų atsekamumas kuriant programinės įrangos produktą yra beprasmė veikla. Kaip teigia šaltinio [29] autorius, kol kas projektuose naudojami atsekamumo metodai turi trūkumų, nes jie reikalauja per daug pastangų, t.y. atsekamumo informacijos kaupimas ir priežiūra dar nėra automatizuoti.

Gali būti, kad atsekamumo informacijos kaupimas ir sisteminimas bus daug brangesnis nei visos tos sukauptos informacijos nauda. Visgi, ypač dideliems projektams, atsekamumo informacija tiesiog būtina- ji ne tik padeda aptikti verslo reikalavimus, kurie dar nėra įgyvendinti programų sistemos lygmenyje, bet ir leidžia tiksliau įvertinti planuojamo sistemos pakeitimo įtaką, taip pat nustatyti kurie testai reikalingi atliktos modifikacijos patikrinimui (validavimui).

Kaip pasiūlyta Jarke, 1998 (cituojama šaltinyje [27]), pagal atsekamumo ryšių (angl. *trace links*) kryptį sistemos reikalavimų atžvilgiu atsekamumas gali būti keturių tipų:

Nuoseklus atsekamumas į reikalavimus (angl. *Forward to requirements*)- kliento poreikiai, kurie paprastai apima verslo tikslus, rinkos poreikius bei vartotojo reikalavimus, yra trasuojami į sistemos reikalavimus. Toks atsekamumas reikalingas tam, kad būtu galima nustatyti kurie reikalavimai bus paveikti, jei kliento poreikiai pasikeis programinio produkto kūrimo ar palaikymo procese. Be to, išbaigtas tokių atsekamumo ryšių rinkinys taip pat parodo ir tai, jog kiekvienas kliento poreikis buvo transformuotas į reikalavimus.

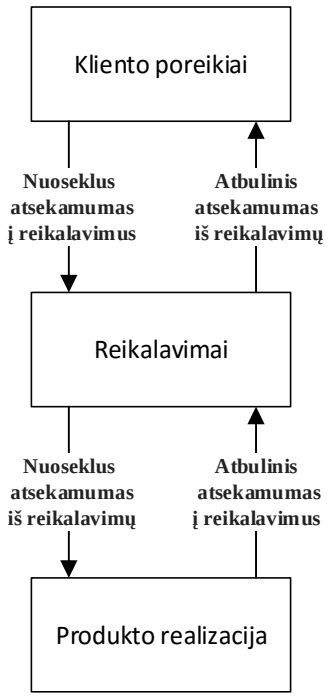
Nuoseklus atsekamumas iš reikalavimų (angl. *Forward from requirements*)- tai atsekamumo ryšio tipas, kuris leidžia nustatyti, kad visi reikalavimai yra įgyvendinti, nes žinoma, kuriuose programinio kodo elementuose jie yra realizuoti, t.y. šie atsekamumo ryšiai susieja

individualius funkcinius ir nefunkcinius reikalavimus su konkrečiais programų sistemos elementais.

Atbulinis atsekamumas į reikalavimus (angl. *Backward to requirements*)- šio atsekamumo ryšiai parodo kodėl kiekvienas produkto elementas (pvz. kodo modulis) buvo suskurtas, t.y. kurį ar kuriuos reikalavimus jis realizuoja.

Atbulinis atsekamumas iš reikalavimų (angl. *Backward from requirements*)- šis atsekamumo ryšio tipas leidžia nustatyti kiekvieno reikalavimo kilmę, t.y. parodo, iš kurio kliento poreikio kilo konkretus reikalavimas.

Kaip pavaizduota 1 paveikslėlyje, paruoštame pagal [27] reikalavimai turi būti atsekami abejomis kryptimis- tiek iš/į jų šaltinio, tiek iš/į jų realizacijos:

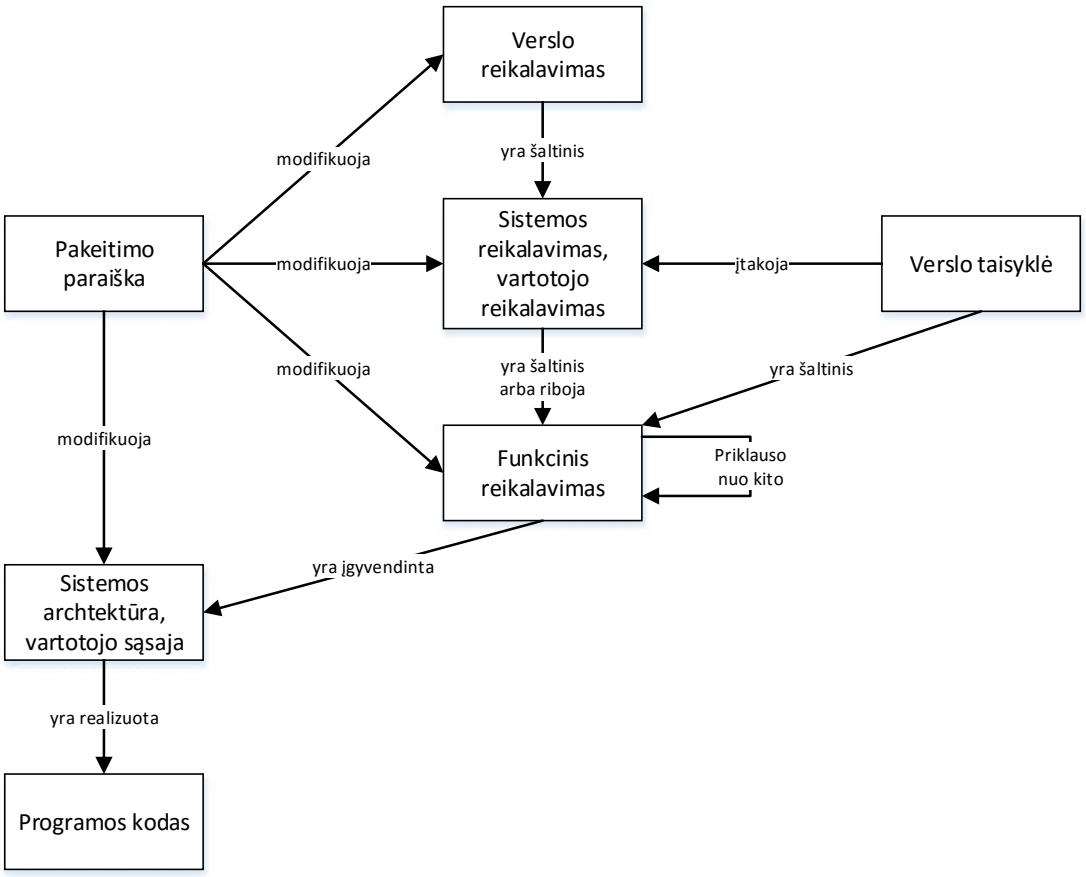


1 pav. ***Reikalavimų atsekamumo tipai***

Knygos [27] autoriaus teigimu, tikrai nebūtina įgyvendinti visų tipų atsekamumą. Ryšių, susiejančių reikalavimų šaltinius ir jų realizacijas programų sistemos lygmenyje parinkimas turi priklausyti ne tik nuo kuriamos programos apimties, bet ir kitų aspektų- kaip dažnai kuriamą sistemą reikės modifikuoti, ir kokie būtent atsekamumo ryšiai gali naudingi atliekant pakeitimus.

Svarbiausias kriterijus priimant sprendimą dėl konkretaus atsekamumo modelio įgyvendinimo turėtų būti atsekamumo informacijos nauda, lyginant su jos surinkimo kaštais, t.y. prieš renkant atsekamumo informaciją būtina įvertinti, ar tai tikrai bus naudinga ateityje. Suinteresuotosios šalys, turinčios skirtingus interesus ir tikslus priklausomai nuo poreikio įtakoja konkrečių atsekamumo ryšių kaupimo pasirinkimą [30].

Kaip pavaizduota 2 paveikslėlyje, parengtame pagal [27] autoriaus pateiktą schemą, atsekamumo ryšiai gali būti labai įvairūs. Šiame darbe didesnis dėmesys skiriamas ne reikalavimams, o verslo taisyklėms. Kaip matome 2 paveikslėlyje, verslo taisyklė gali įtakoti sistemos reikalavimus, vartotojo reikalavimus bei būti funkcinių reikalavimų šaltinis arba ribojimas.



2 pav. ***Galimi atsekamumo ryšiai***

Knygos [27] autoriaus pasiūlyta galimų naudoti atsekamumo ryšių schema nedetalizuoja verslo taisyklių atsekamumo ryšių su jų galima realizacija duomenų bazių valdymo sistemoje. Toks atsekamumo ryšių modelis, apimantis atsekamumo ryšius iš verslo taisyklių į jų realizacijas

įvairiuose ADBVS komponentuose galėtų būti naudingas, nes didžioji verslo taisyklių dalis ir yra susijusi būtent su duomenimis kuriamoje sistemoje- duomenų įvedimo/modifikavimo ribojimais, duomenų apsaugos (pvz. prieinamumo tik konkretiems vartotojams) taisyklėmis ir pan.

Ne visi atsekamumo ryšiai yra naudingi ir dar nėra efektyvaus būdo nustatyti kaupiamų atsekamumo ryšių vertę bei skirtingo lygio grūdėtumo kūrimo ir palaikymo įtaką [31]. Dažniausiai kuomet reikalavimai jau yra realizuoti programų sistemos lygmenyje, atsekamumo ryšys tarp reikalavimo ir jo realizacijos susilpnėja, ypač produkto palaikymo etape [32].

1.3.2 Reikalavimų lokalizavimo sąvoka

Verslo taisyklių transformacijos iš verslo sistemos lygmens į programų sistemos lygmenį sampratai yra svarbi ir lokalizavimo sąvoka. Knygoje [2] pateiktas reikalavimų lokalizavimo (angl. *requirements allocation*) apibrėžimas, kad tai yra *„jų skaidymo į tam tikras prasminiu požiūriu susietų reikalavimų grupes, kuriose kai kurie reikalavimai gali kartotis, ir tų grupių susiejimas su joms priklausančius reikalavimus realizuojančiais kuriamos sistemos komponentais*“ . Tuo tarpu knygos [27] autorius reikalavimų lokalizavimą apibrėžia kaip “*reikalavimų paskirstymo tarp įvairių architektūrinių posistemių ir komponentų procesą*“. Kitais žodžiais tariant, reikalavimų lokalizavimo proceso metu reikalavimai, apibrėžti viename sistemos lygmenyje, yra priskiriami konkrečioms žemesnio sistemos architektūros lygmens komponentams. Jei lyginsime su reikalavimų atsekamumu, t.y. galimybe susiejimo ryšiais (angl. *linkage*) atsekti (angl. *trace*) žemesnio lygmens reikalavimą atgal į jo šaltinį, t.y. aukštesnio lygmens reikalavimą, arba priešinga kryptimi, šios dvi sąvokos yra panašios, ypač jei į atsekamumą žvelgsime ne tik kaip į procesą padedantį atsekti bet kurio reikalavimo istoriją, bet platesne prasme- kuomet atsekamumo ryšiai veda ir prie tų reikalavimų realizacijos, pvz. programinio kodo.

Tam, kad būtų galima lokalizuoti sistemos reikalavimus, jau turi būti parinkta sistemos architektūra, t.y. turi būti žinomi tos sistemos komponentai [2]. Žemesnio lygmens komponentuose reikalavimai pradedami lokalizuoti tik po to, kai jie buvo lokalizuoti ir transformuoti (nuleisti žemyn) į atitinkamo (einamojo) lygmens komponentus.

Nors minėtuose šaltiniuose šios sąvokos taikomos sistemos reikalavimams, visi jomis apibrėžti procesai yra aktualūs ir kalbant apie šiame darbe nagrinėjamas verslo taisykles bei jų transformaciją.

Vienas pagrindinių iššūkių, su kuriuo susiduria organizacijos, kurių veikla vykdoma su informacinių sistemų pagalba, yra užtikrinti, kad sistemos būtų vystomos besikeičiant verslo poreikiams. Tai ypač svarbu besikeičiant verslo taisyklėms, nes jos paprastai realizuojamas mažuose programinio kodo fragmentuose, ir jų pakeitimas yra sudėtingas procesas [33]. Kaip teigia [34] autorius, bet kokie verslo taisyklių pasikeitimai reikalauja, kad verslo taisyklės turėtų aiškų susiejimą su jas įgyvendinančiais programų sistemos komponentais.

1.3.3 Atsekamumo metodai

Reikalavimų atsekamumo metodai skiriasi kaupiamos informacijos apimtimi ir pobūdžiu. Nėra universalaus metodo tinkančio visiems atvejams- metodo parinkimas priklauso nuo to, ką tiksliai norima pasiekti jį įgyvendinus. Šaltinio [2] autorius kaip pagrindinius atsekamumo metodus įvardija kryžminių nuorodų sistemas, trasavimo matricas, integruojančius dokumentus.

Kryžminių nuorodų sistemos tai metodas, kuomet kiekvienam reikalavimui priskiriamos nuorodos tiek į aukštesnio lygmens reikalavimus, tiek į žemesnio lygmens reikalavimus ar kitokio pobūdžio informacinius įrašus. Kryžminės nuorodos gali būti saugomos atskiroje duomenų bazės lentelėje.

Integruojančių dokumentų metodas leidžia susieti skirtingų lygmenų reikalavimų specifikacijų dokumentus. Dokumento fragmentai susiejami su žemesniojo lygmens teksto dokumento fragmentais, o atlikus pakeitimus aukštesnio lygmens dokumente, atitinkamai pasikeičia ir susieti fragmentai žemesniame lygmenyje.

Kaip teigiama [2] [27], populiariausias metodas reikalavimų trasavimui yra **reikalavimų lokalizavimo matrica** (angl. *requirements mapping matrix*) bei **reikalavimų ryšių matrica** (angl. *requirements trace matrix*). Reikalavimų lokalizavimo matricoje kiekvienas reikalavimas susiejamas su vienu ar daugiau sistemos komponentų (tai gali būti ir architektūrinis sistemos komponentas, lentelė sąryšiniame duomenų modelyje (angl. *relational data model*), arba objekto klasė). Susiejimas matricoje galimas ir su konkrečiu programinio kodo elementu, pavyzdžiui, klasės metodu, saugoma procedūra, failu arba moduliu. Tai yra, atsekamumo ryšys gali „vesti“ prie labai konkretaus reikalavimo realizacijos elemento, kad ir koks smulkus jis būtų.

Reikalavimų ryšių matrica skirta atvaizduoti ryšius tarp įvairių sistemos elementų porų, pavyzdžiui, vienos rūšies reikalavimų susiejimas su kitos rūšies reikalavimais, arba tiesiog tos pačios rūšies reikalavimų porų susiejimas.

Deja, nei vienas atsekamumo metodas dar neleidžia šio proceso pilnai automatizuoti. Tačiau kai kurie įrankiai, su juose pritaikytais atsekamumo metodais gerokai palengvina ne tik atsekamumo procesą, bet ir visą sistemos reikalavimų valdymą. Būtų tiesiog neįmanoma įgyvendinti reikalavimų atsekamumą rankiniu būdu, nebent kuriama labai nedidelės apimties sistema. Visgi, ir pilnas šio proceso automatizavimas kol kas nėra pasiektas, nes didžioji dalis tokios atsekamumo ryšių informacijos išgaunama iš konkrečių sistemą kuriančių ar reikalavimus formuluojančių žmonių, t.y. atsekamumo ryšius dažniausiai gali identifikuoti tik sistemą kuriantys specialistai, o atsekamumo įrankiai tik palengvina šios informacijos valdymą. Visgi, automatizuoto reikalavimų atsekamumo ryšių sudarymo srityje jau dabar yra pasiekimų. Kaip pasiūlyta straipsnio [35] autorių, atsekamumo ryšiai gali būti sugeneruojami automatiškai, tikrinant ar reikalavimai, užrašyti naudojant LTL formulę (angl. *Linear temporal formula*), yra realizuoti sistemos architektūroje. Tačiau pasiūlyto metodo trūkumas yra tai, jog jis reikalauja labai specifinių žinių tokiam reikalavimų formalizavimui, ir kaip nurodo minimo šaltinio autoriai, jis galėtų būti praplėstas automatizuota laikantis SBVR standartų užrašytų reikalavimų transformacija į LTL formules. Šaltinio [36] autorių teigimu, šiuo metu automatizavimas susiduria su dviem pagrindiniais iššūkiais- vertės subjektyvumas ir prieštaraujantys kliento poreikiai, o esamiems įrankiams dar trūksta atsekamumo reikalingo tokioms problemoms spręsti.

Šiuo metu atsekamumo tyrimus atliekantys mokslininkai siekia sukurti ar patobulinti atsekamumo ryšių išgavimo technikas. T.y., siekiama, kad reikėtų beveik nulinių suinteresuotųjų šalių pastangų tam, kad išgauti ir panaudoti atsekamumo informaciją, o tai reiškia siekiama kad atsekamumo informacija būtų surenkama ir vėliau naudojama visiškai automatiškai [37]. Didžiausias su šio proceso automatizavimu susijęs iššūkis yra tai, jog visi artefaktai programinės įrangos kūrime yra skirtingo formato bei skirtingo abstrakcijos lygmens. Visgi, viena savybė jungia visus šiuos artefaktus- tai tekstinė informacija todėl šiuo metu procesui tobulinti yra naudojamos informacijos išgavimo technikos [38]. Kaip parodė šaltinio [39] autorių atliktas tyrimas, šiuo metu pagrindinės problemos su kuriomis susiduriama šioje srityje yra bendrų žinių

ir supratimo apie reikalavimų įgyvendinimo atsekamumą ir atsekamumo išlaikymą reikalavimams kintant, trūkumas

2 VERSLO TAISYKLIŲ AUTOMATIZUOTOS IR ATSEKAMOS REALIZACIJOS IS METODAS

2.1 Metodo motyvacija

Šiuo metu pagrindinis siekis verslo taisyklių požiūriu (angl. *business rule approach*) kuriamos programinės įrangos procesui pagerinti yra koncentruojamas į būdų ir priemonių, leidžiančių automatiškai realizuoti verslo aplinkos pokyčius programinės įrangos sistemose, radimą [40].

Verslo taisyklės įtakojančios ir ribojančios verslo objektus dažniausiai yra įgyvendinamos sistemos duomenų bazėje, kuri turi greičiausią prieigą prie tam tikrus reikalavimus turinčių atitikti duomenų. Duomenų bazių valdymo sistemose verslo taisyklės gali būti įgyvendinamos duomenų bazės schemos ribojimais, saugomomis procedūromis ir trigeriais.

Pati duomenų bazės schema yra sudaroma verslo taisyklių pagrindu. Verslo taisyklės apibrėžia ne tik kiekvieną duomenų bazės lentelę, jos atributus, bet ir ryšius tarp įvairių duomenų objektų bei tų ryšių kardinalumą, reikalavimus ir ribojimus patiems duomenų bazėje saugomiems duomenims ir jų tipams.

Iš visų dalykinės srities verslo taisyklių rinkinio atrinkus tas, kurios aprašo duomenų bazės schemos elementus, taikant įvairaus tipo taisyklių šablonus būtų galima atlikti automatizuotą taisyklės transformaciją į konkretų ją realizuojantį duomenų bazės elementą. Tokiam taisyklės automatizavimui būtų būtina, jog taisyklė būtų užrašyta pakankamai formalia kalba, pavyzdžiui SBVR.

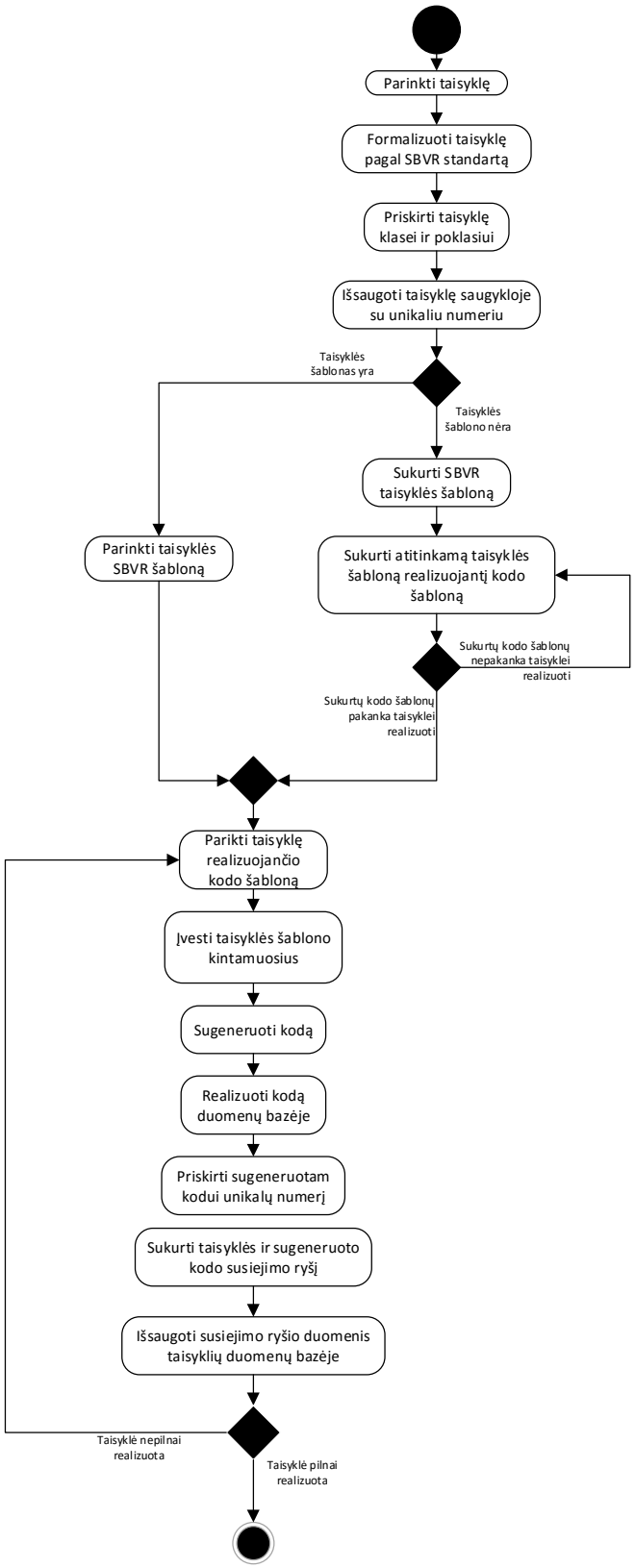
2.2 Metodo aprašymas

Verslo taisyklių automatinio transformavimo į duomenų bazės schemos elementus metodui įgyvendinti būtina išgauti ir susisteminti visą verslo sistemos lygmens taisyklių rinkinį. Šiame darbe bus daroma prielaida, kad toks pasirinktos dalykinės srities taisyklių rinkinys jau yra sukurtas. Metodo pritaikymui turi būti atrinktos tik tam tikros taisyklės- tik tos, kurios gali būti

realizuojamos duomenų bazės elementuose. Toliau pateikiami šio metodo parinktam verslo taisyklių rinkiniui įgyvendinimo etapai:

1. Tinkamo verslo taisyklių klasifikatoriaus parinkimas- šiame darbe bus parinktas taisyklių klasifikatorius pagal [5], t.y. taisyklės bus klasifikuojamos į struktūros ir dinamines taisykles, bei šių klasių poklasius.
2. Verslo taisyklių ir atsekamumo ryšių saugyklos sukūrimas.
3. Verslo taisyklių realizavimas duomenų bazėje pagal automatizavimo schemą (žr. pav. 3)
4. Duomenų bazės schemos sukūrimas.
5. Atsekamumo ryšių ataskaitos sudarymas.

Automatizuoto taisyklės įgyvendinimo duomenų bazėje žingsniai pavaizduoti UML veiklos diagramoje (pav. Nr. 3):



3 pav. ***UML veiklos diagrama- automatizuoto verslo taisyklės įgyvendinimo duomenų bazėje žingsniai***

Kaip pavaizduota veiklos diagramoje (pav.3), pradiniame žingsnyje parenkama konkreti verslo taisyklė (šiame darbe darysime prielaidą, jog jau turime tinkamų realizuoti duomenų bazėje atrinktų verslo taisyklių rinkinį). Parinkta taisyklė yra užrašoma pagal SBVR standartą, priskiriama konkrečiai taisyklių klasei bei poklasiui. Taisyklei suteikiamas unikalus numeris, kurį sudaro klasės kodas, poklasio kodas bei taisyklės numeris. Tuomet tikrinama, ar tokio tipo taisyklei jau yra sukurtas SBVR taisyklės šablonas. Metodui realizuoti bus naudojamas taisyklės šablonas, kuris kaip aprašyta [41] ir [42] apibrėžia tam tikro tipo taisyklės struktūrą. Jei tokio šablono dar nėra- jis sukuriamas. Sukurtas šablonas išsaugomas duomenų bazėje. Pažymėtina, kad vieno poklasio taisyklė gali turėti daug šablonų, todėl šablono pritaikymas kiekvienai taisyklei yra parenkamas atsižvelgiant ne tik į taisyklės tipą, bet ir į jos struktūrą.

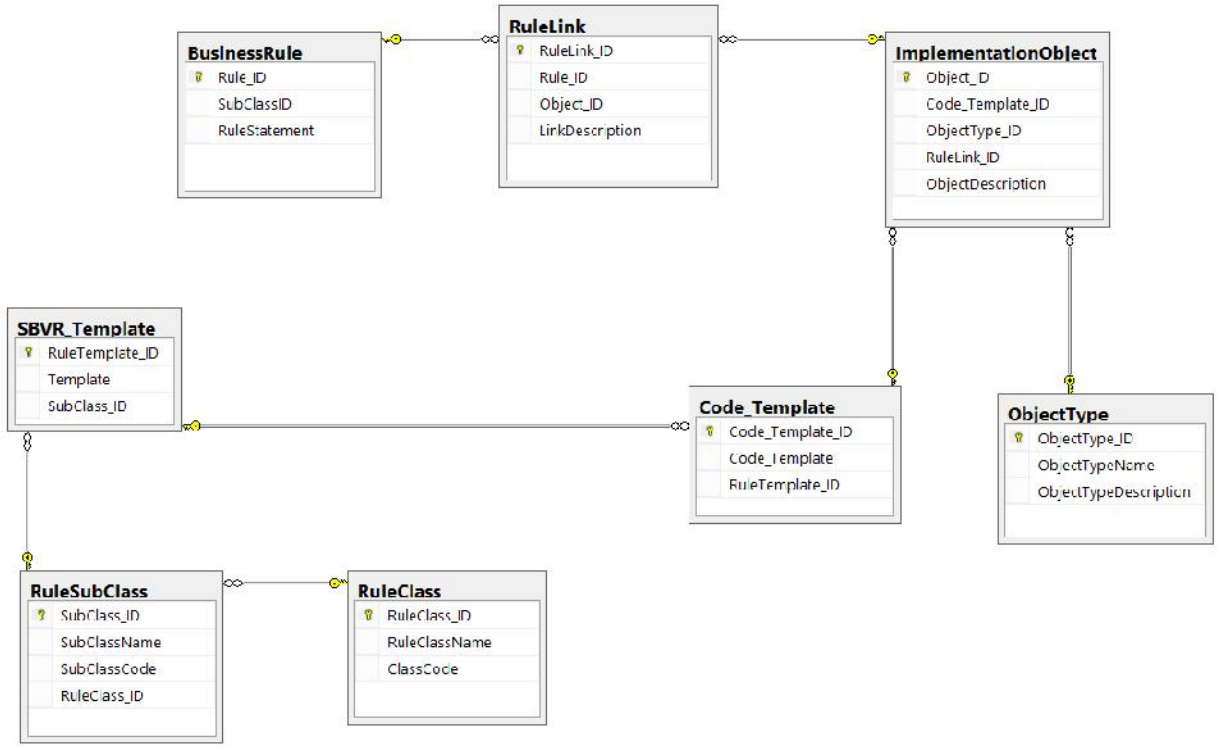
Taisyklės šablonas sudaromas parenkant konkrečius raktinius žodžius bei kintamas šablono dalis. Tokiu būdu, kiekvienas taisyklės šablonas bus sudarytas tiek iš fiksuotų (universalių tam pačiam taisyklės tipui) dalių, tiek iš kintamų, konkrečiai taisyklei parenkamų dalių (pavyzdžiui, terminų, kvantorių ir pan.).

Po taisyklės šablono sukūrimo arba parinkimo į sukurtą šabloną įvedami (iš vartotojo sąsajos) kintami SBVR taisyklės elementai. Pagal konkrečiam SBVR taisyklės šablonui sukurtą DDL kodą sugeneruojamas parinktai taisyklei realizuoti skirtas DDL skriptas, kuris vėliau įgyvendinamas duomenų bazėje.

Tam, kad būtų galima atsekti kur duomenų bazėje yra realizuota konkreti taisyklė, taisyklių saugykloje bus saugoma atsekamumo ryšio informacija- tai yra, kiekviena taisyklė susiejama su ją realizuojančiu kodu, ir tas ryšys, su jam priskirtu unikaliu identifikaciniu numeriu saugomas taisyklių saugykloje.

Taigi, kaip pavaizduota taisyklių ir atsekamumo ryšių saugyklos schemoje (pav. 4), kiekviena verslo taisyklė bus registruojama duomenų bazėje (saugykloje). Kartu su pačia taisykle šioje taisyklių saugykloje bus kaupiama papildoma informacija apie taisyklę:

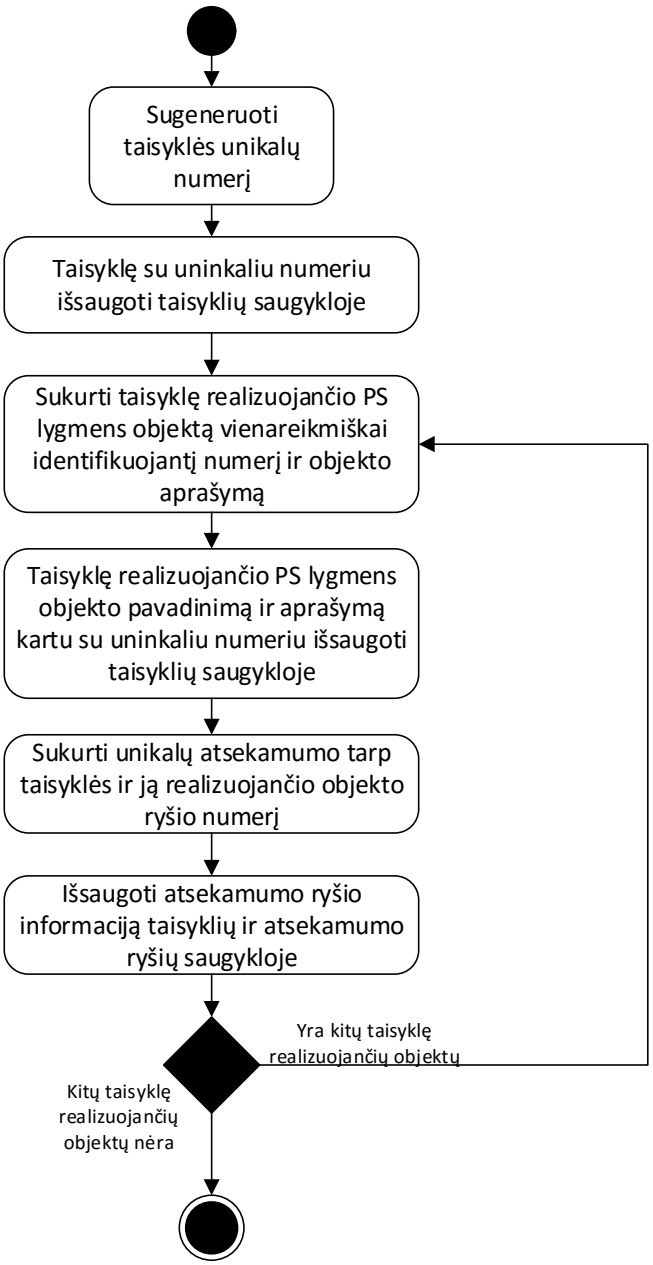
- taisyklės klasė;
- taisyklės poklasis;
- taisyklei parinkto šablono numeris;
- susiejimo su taisyklę realizuojančiu kodu ryšio numeris.



4 pav. ***Verslo taisyklių ir jų atsekamumo ryšių saugyklos schema***

Papildomai šioje taisyklių saugykloje bus saugomi duomenų bazės elementų sukūrimo DDL kodo šablonai (kas apibendrintu metodo atveju galėtų būti tiesiog nuorodos į šablonus). Jie taip pat priskiriami konkrečiam duomenų bazės (DB) elemento tipui. DB elemento tipas šiuo atveju gali būti duomenų bazės lentelė, trigeris, saugoma procedūra, arba dar smulkesni taisykles duomenų bazėje realizuojantys elementai, tokie kaip pirminiai ar išoriniai raktai, lentelės atributų savybės ir pan. Kiekvienas tokio elemento tipas (schemoje vadinama *ObjectType*) gali turėti daug DDL kodo šablonų. Kiekvienas taisyklės šablonas gali turėti vieną arba daugiau tokio tipo taisyklę realizuojančių DDL kodų. Iš to seka, kad kiekviena taisyklė, kuriai pritaikomas vienas SBVR taisyklės šablonas duomenų bazėje gali turėti vieną ar daugiau šią konkrečią taisyklę įgyvendinančių DB elementų (schemoje vadinama *ImplementationObject*), ir vieną arba daugiau susiejimo ryšių (schemoje *RuleLink*) tarp konkrečios taisyklės ir ją realizuojančio duomenų bazės elemento.

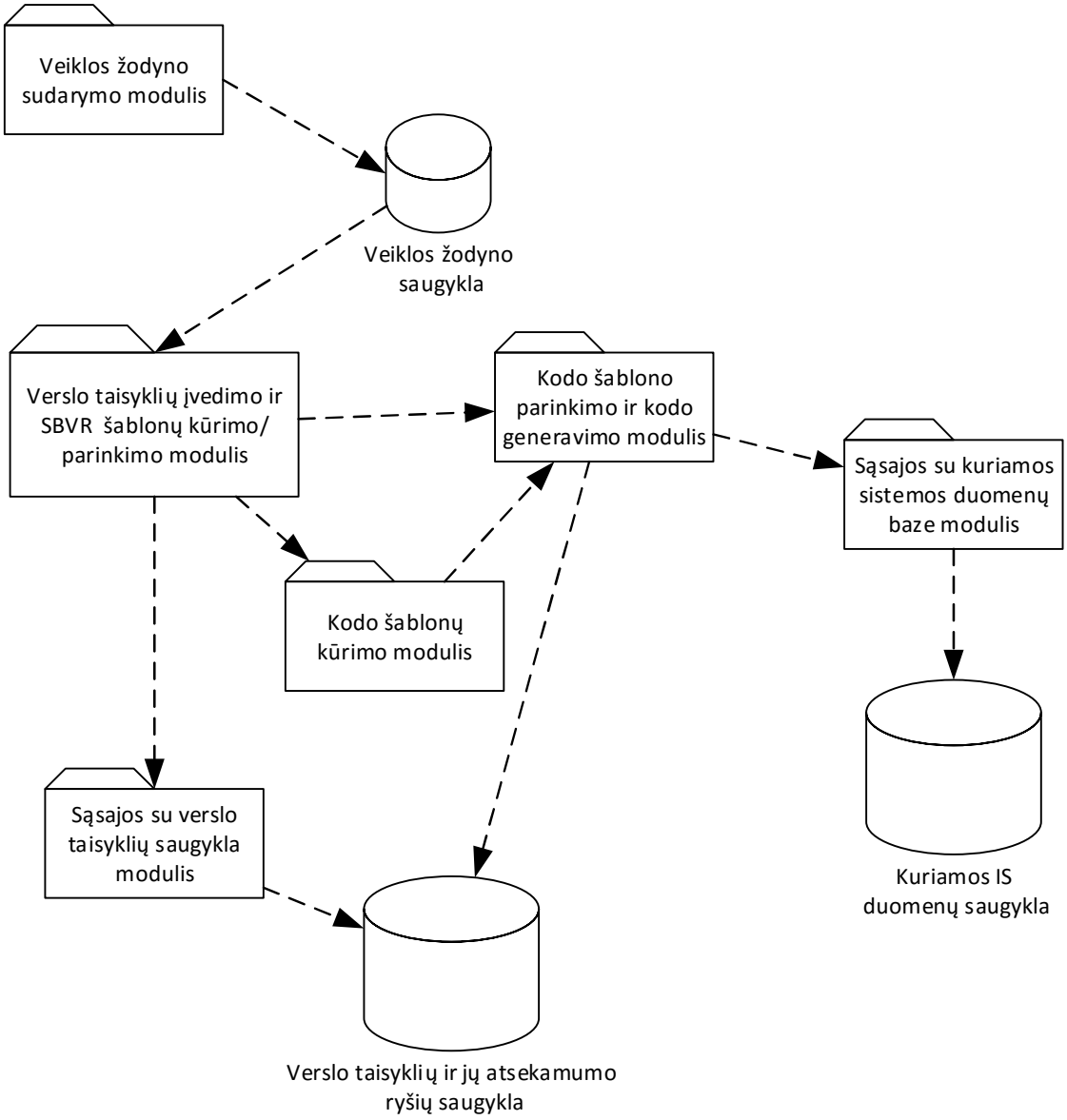
Siūlomas metodas leidžia įgyvendinti vertikalų verslo taisyklės atsekamumą- iš verslo sistemos lygmens į programų sistemos lygmenį. Verslo taisyklės atsekamumo užtikrinimo algoritmas pavaizduotas naudojant UML veiklos diagramą:



5 pav. ***UML veiklos diagrama- atsekamumo užtikrinimo žingsniai***

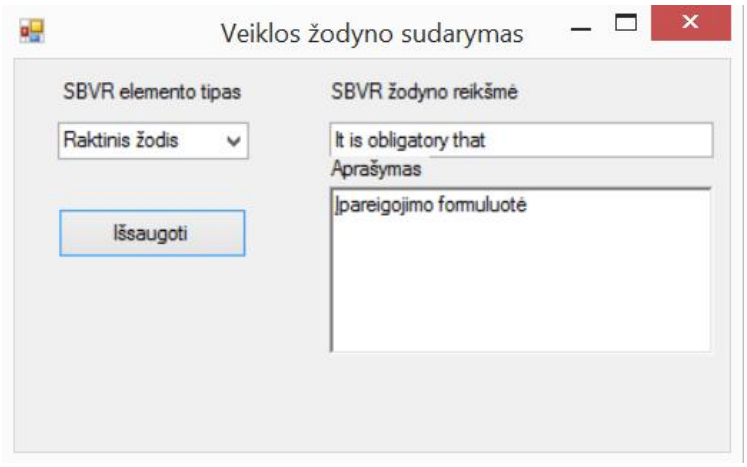
2.3 Metodo iliustracija

Šiame skyriuje aprašomi pasiūlytą metodą galinčios realizuoti programų sistemos komponentai bei programinės įrangos veikimo principai. 6 pav. pavaizduoti programų sistemos komponentai:



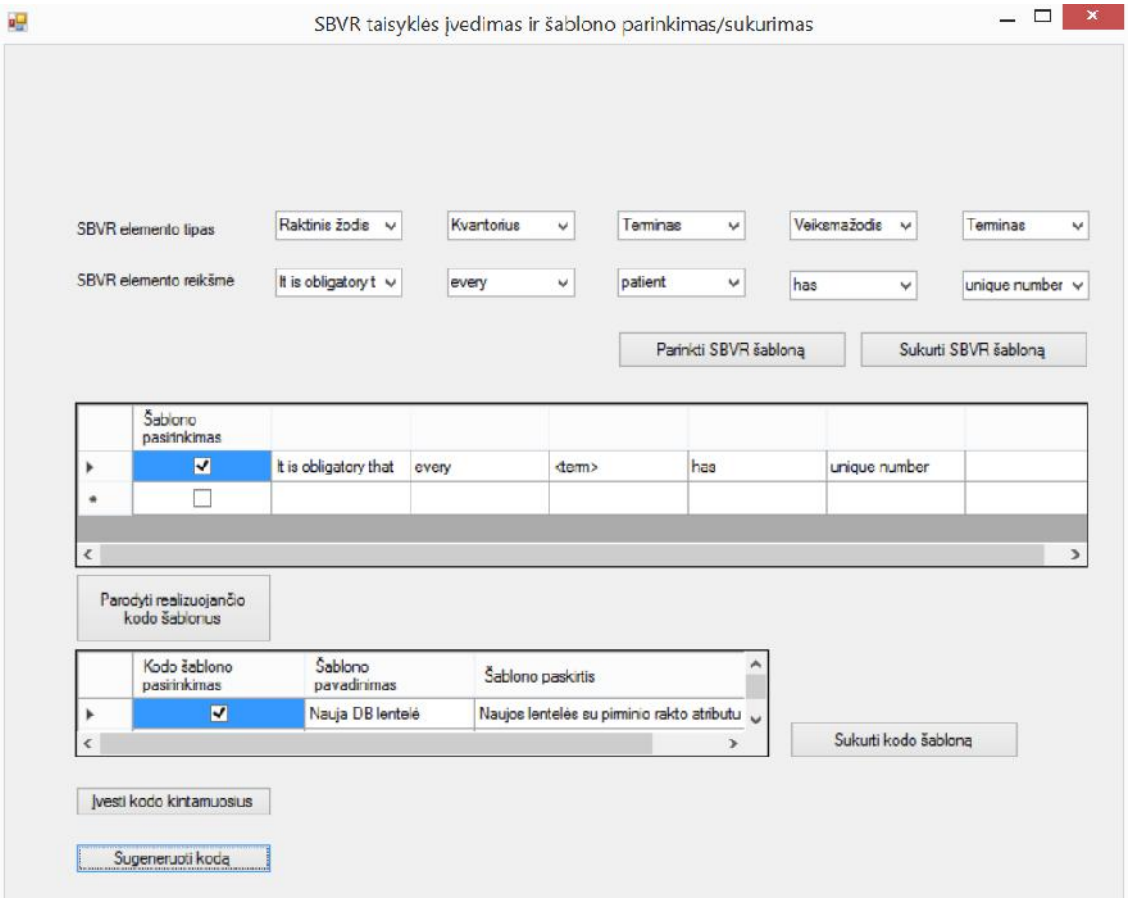
6 pav. ***Siūlomą metodą realizuojančios programų sistemos komponentų schema***

Pirmasis programos modulis būtų skirtas veiklos žodyno sudarymui. Veiklos žodyno saugykloje būtų saugomi SBVR struktūrinių elementų tipai, jų reikšmės ir aprašymai:



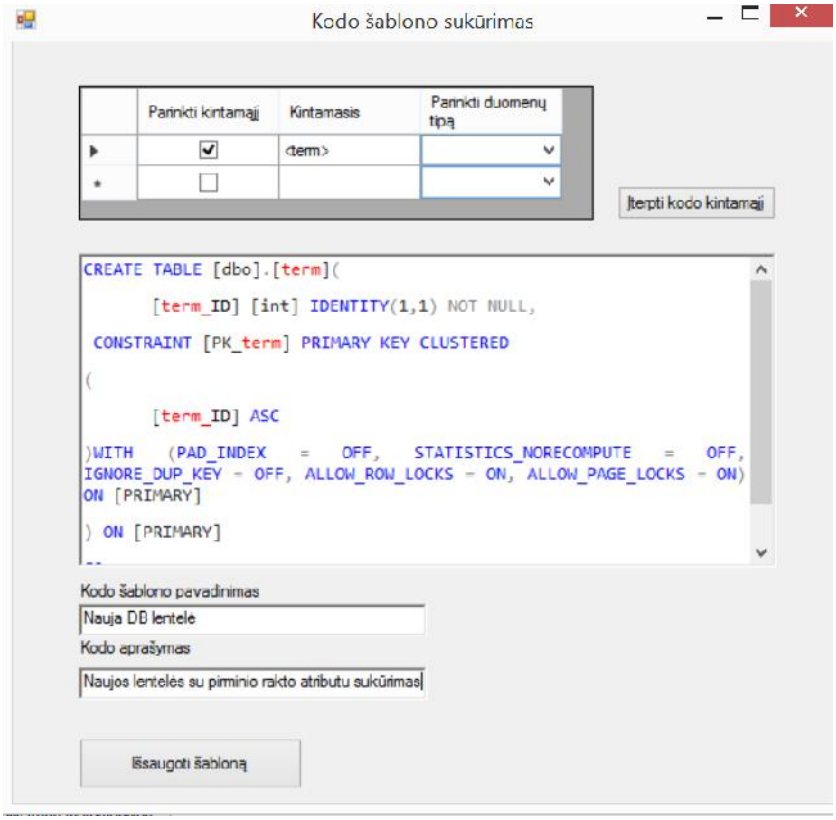
7 pav. ***Veiklos žodyno sudarymo iliustracija***

Pati taisyklė, užrašyta laikantis SBVR standarto, vartotojo sąsajoje būtų suvedama eilės tvarka parenkant atitinkamą SBVR elemento tipą (raktinis žodis, terminas, veiksmažodis ir t.t.), bei jį atitinkantį egzempliorių, t.y. konkrečią dalykinės srities žodyno reikšmę (7 pav.). Programa pagal SBVR taisyklės konstrukcinių elementų (terminų, raktinių žodžių, veiksmažodžių, kvantorių) išdėstymo tvarką randa visus galimus tokio išdėstymo SBVR taisyklės šablonus (t.y., šablonus kuriuose visiškai atitinka SBVR konstrukcinių elementų parinkimo eiliškumas, tačiau kai kurie konstrukciniai elementai turi fiksuotą reikšmę (pvz. konkretų raktinį žodį), o kai kurie tik nusako jo tipą (pvz. terminas), t.y. kintamąjį. Jei tinkamas šablonas nerandamas sistemoje, jis turi būti sukuriamas. Šabloną leidžiama išsaugoti sistemoje tik tada, kai įvedamas bent vienas SBVR taisyklės elemento tipas su kintama reikšme.



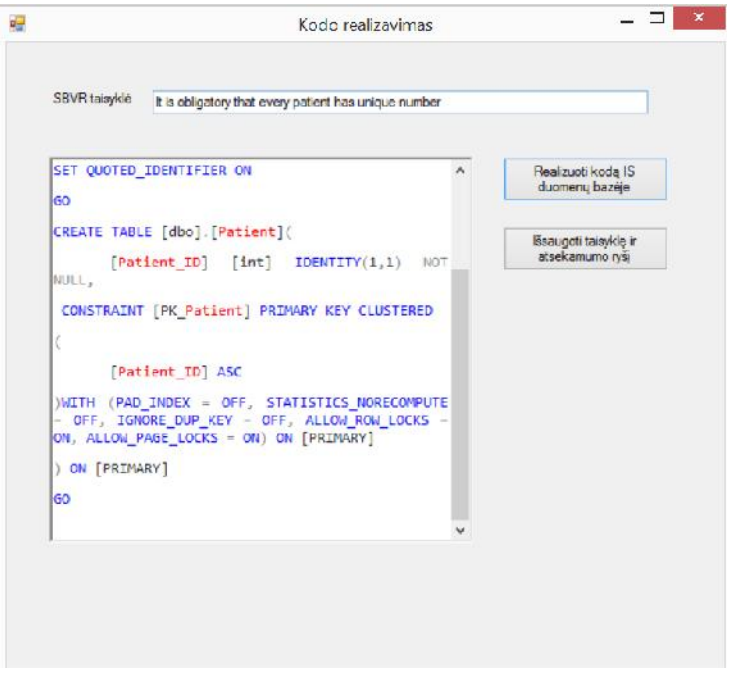
8 pav. ***SBVR taisyklės įvedimo ir šablono parinkimo/sukūrimo iliustracija***

Parenkamas tinkantis SBVR taisyklės šablonas, kuriam parenkamas arba sukuriamas naujas taisyklę realizuojančio kodo šablonas, kuris iš karto susiejamas su parinktu SBVR taisyklės šablonu, t.y. kodo šablonas kuriamas naudojant SBVR taisyklės šablono kintamuosius:



9 pav. ***Kodo šablono sukūrimo iliustracija***

Kitame žingsnyje parinktas kodo šablonas sugeneruojamas pritaikant jį konkrečiai verslo taisyklei įvedant kintamųjų SBVR elementų reikšmes:



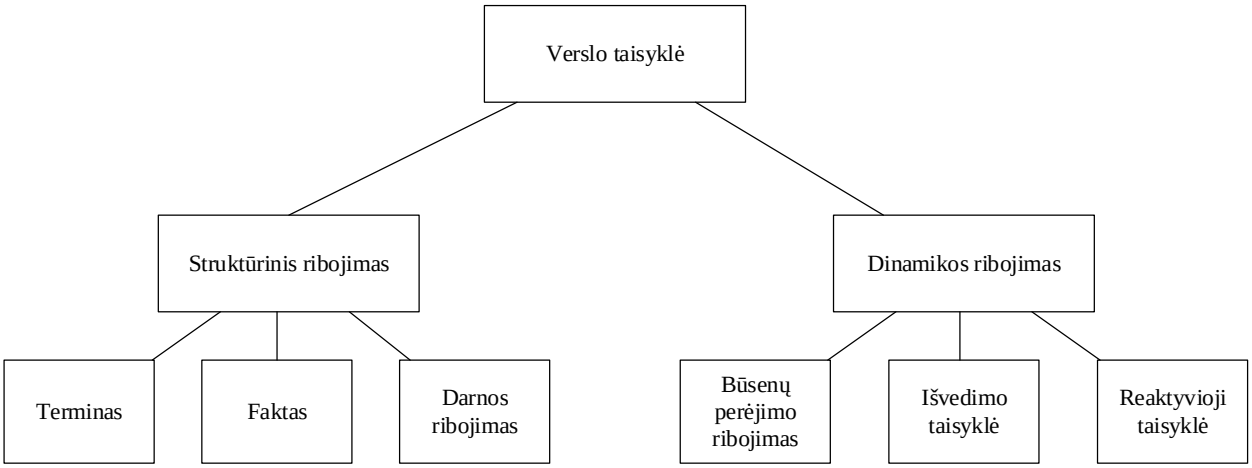
10 pav. ***Kodo realizavimo iliustracija***

Realizuojant taisyklę duomenų bazėje taisyklė yra išsaugoma taisyklių saugykloje, ir jai priskiriamas atsekamumo ryšys su ją realizuojančiu DB objektu.

3 METODO PATIKRINIMAS

3.1 Taisyklių klasifikatoriaus parinkimas

Automatinio verslo taisyklių transformavimo į duomenų bazės elementus metodui patikrinti būtina apibrėžti kaip bus klasifikuojamas pasirinktos dalykinės srities verslo taisyklių rinkinys. Šiame darbe bus remiamasi šaltinyje [5] aprašyta verslo taisyklių klasifikacija, pagal kurią verslo taisyklės skirstomos į dvi pagrindines klases- struktūrinius ribojimus ir dinamikos ribojimus. Šios dvi taisyklių klasės toliau skirstomos į dar tris taisyklių poklasius:



11 pav. ***Metodui parinkta verslo taisyklių klasifikacija***

Kiekviena iš verslo taisyklių rinkinio parinkta taisyklė, kurios įgyvendinimą bus siekiama automatizuoti kokiame nors duomenų bazės schemos elemente, bus priskiriama vienai iš šių dviejų klasių, ir kuriam nors atitinkamos klasės poklasiui. Taisyklei bus suteikiamas unikalus numeris, be to papildomas kodas, identifikuojantis, kuriai klasei ir kuriam poklasiui ši taisyklė priklauso.

3.2 Dalykinės srities ir taisyklių rinkinio parinkimas

Metodo patikrinimui pasirinkta dalykinė sritis- pirminės sveikatos priežiūros paslaugas teikiančios medicinos įstaigos veikla. Verslo taisyklių rinkinys, kuriam bus taikomas pasiūlytas metodas parinktas, toks, kad taisykles būtų galima realizuoti kuriant duomenų bazės schemą.

Kuriama informacinė sistema apima pacientų registraciją vizitui pas tam tikrą gydytoją tam tikru laiku bei medicininių įrašų kaupimą.

Detaliam nagrinėjimui, t.y. įrodymui, kad taisyklės gali būti automatiškai transformuojamos tuo pat metu kaupiant ir jų atsekamumo informaciją, parinkta tik dalis dalykinės srities verslo taisyklių rinkinio (3 lentelėje). Kita informacinės sistemos duomenų bazės schemai sukurti taikyta verslo taisyklių rinkinio dalis šiame darbe detaliai nenagrinėjama, o tik tiesiogiai realizuojama informacinės sistemos duomenų bazėje, ir išsaugoma taisyklių ir atsekamumo ryšių saugykloje (pilnas realizuotas taisyklių rinkinys pateikiamas eksperimento rezultatų skyriuje).

3 lentelė. ***Pasiriktos dalykinės srities verslo taisyklių rinkinio dalis***

Eil. Nr.	Taisyklė
1.	Kiekvienas medicinos įstaigos pacientas turi būti registruotas sistemoje
2.	Kiekvienas pacientas turi turėti unikalų paciento numerį
3.	Pirmą kartą sistemoje registruojamas pacientas privalo nurodyti asmens kodą, vardą ir pavardę
4.	Visi įstaigoje dirbantys gydytojai privalo būti registruoti sistemoje
5.	Kiekvienas gydytojas privalo turėti unikalų gydytojo numerį
6.	Sistemoje registruojamas gydytojas privalo nurodyti vardą, pavardę, specialybę ir vidutinę pas jį besilankančių pacientų vizito trukmę
7.	Kiekvieno gydytojo pamaina turi būti registruojama sistemoje
8.	Kiekviena gydytojo pamaina turi turėti nurodytą darbo pradžios ir darbo pabaigos laiką
9.	Gydytojo pamainos trukmė negali viršyti 12 val. per parą
10.	Kiekvienas vizitas turi turėti unikalų vizito numerį
11.	Vienas vizitas sudaromas tarp vieno gydytojo ir vieno paciento
12.	Kiekvienas paciento vizitas pas gydytoją privalo turėti pradžios ir pabaigos laiką
13.	Vieno vizito pas gydytoją trukmė negali viršyti 1 val.

Taisyklės bus užrašomos pagal SBVR standartą anglų kalba. Anglų kalba pasirinkta dėl to, kad jos gramatika yra paprastesnė, ir leidžia taisykles išreikšti labiau pritaikyta automatizavimui

forma. Be to, jei tokia automatizuoto taisyklių įgyvendinimo informacinėse sistemose programinė įranga būtų kuriama, tikėtina, kad ji būtų pritaikyta būtent anglų kalbai.

3.3 Šablonų sukūrimas

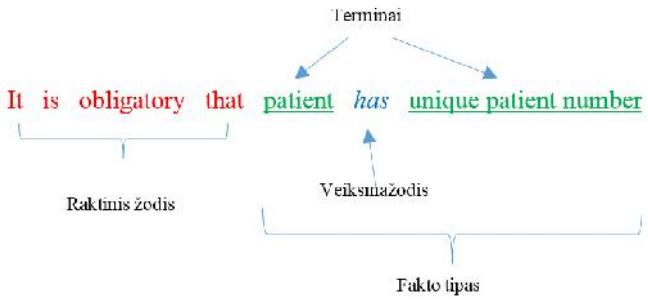
Metodui įgyventi iš pasirinkto taisyklių rinkinio pirmiausiai atrinksime verslo taisykles tiesiogiai susijusias su aiškiausiai šioje dalykinėje srityje išreikšta esybe *Pacientas*:

- Kiekvienas medicinos įstaigos pacientas turi būti registruotas sistemoje;
- Kiekvienas pacientas turi turėti unikalų paciento numerį;
- Pirmą kartą sistemoje registruojamas pacientas privalo nurodyti asmens kodą, vardą ir pavardę;

Projektuojant pasirinktos dalykinės srities informacinės sistemos duomenų bazę šios trys taisyklės aiškiai apibrėžia, jog turėtų būti sukuriama duomenų bazės lentelė su tam tikrais atributais ir tam tikromis jų savybėmis.

Kadangi tai pirmas eksperimento etapas, kuomet nėra sukurtas nei vienas taisyklės ir jos realizavimo duomenų bazėje šablonas, būtina pradėti būtent nuo šių šablonų sukūrimo, kad juos būtų galima pritaikyti jau automatiniam panašių taisyklių realizavimui.

Iš trijų pacientą kaip esybę aprašančių verslo taisyklių išvesime dvi SBVR taisykles, nes antroji ir trečioji taisyklė iš esmės apima pirmąją- t.y. tai jog pacientas turi būti registruotas sistemoje ir reiškia dviejų po to sekančių taisyklių realizavimą- lentelės su tam tikrais atributais ir jų savybėmis sukūrimą duomenų bazėje. Pirma SBVR taisyklė:



Ši taisyklė sudaryta iš raktinio žodžio „It is obligatory that“, bei fakto tipo „patient *has* unique patient number“. Taisyklę sudaro du terminai (šiuo atveju antrasis terminas „unique patient

number“ sudarytas iš trijų žodžių, įskaitant ir jo savybę nusakantį būdvardį „unique“). Kadangi šios taisyklės veiksmažodis „*has*“ kartu su raktiniu žodžiu „It is obligatory that“, nusako priklausomybės ryšio tarp dviejų SBVR terminų „patient“ ir „unique patient number“ privalomumą, sudarydami taisyklės šabloną šias SBVR dalis paliksime fiksuotas. Tuo tarpu abu taisyklės terminai gali keistis, t.y. terminai bus taisyklės kintamieji, kuriuos reikėtų įvesti, naudojant šį sukurtą šabloną kitai panašiai taisyklei. Tokiu būdu pasirinktos SBVR taisyklės šablonas būtų toks:

It is obligatory that <term> has unique <term> number

Šį šabloną išsaugome taisyklių saugykloje ir priskiriame jį darnos ribojimų klasei SubClass_ID= 3. Taisyklės šablonui priskiriamas identifikacinis numeris RuleTemplate_ID=1.

Realizuojant šitą šabloną reikia priskirti reikšmes kintamiesiems term ir unique identifier. Tam galima naudoti vartotojo sąsają, kurioje įvedamos šių kintamųjų reikšmės. Pažymėtina ir tai, jog kuriant duomenų bazės lentelę yra būtina nurodyti atributų duomenų tipus. Šiuos duomenis taip pat galima būtų parinkti konkrečiam dalykinės srities atvejui juos įvedant vartotojo sąsajoje.

Lentelėje 4 pateikiama transformacijos schema, t.y. išanalizuojama į ką konkrečiai bus transformuojamas kiekvienas SBVR taisyklės elementas:

4 lentelė. ***SBVR taisyklės transformacijos schema***

Eil. Nr.	**SBVR elementas (kas transformuojama)**	**DB elementas (į ką transformuojama)**	**Pritaikymas taisyklei It is obligatory that patient *has* unique patient number**
1.	It is obligatory that	Nusako priklausomybės ryšio tarp dviejų SBVR terminų <term> ir <unique identifier> privalomumą. T.y. <unique identifier> yra privalomas lentelės `Term` atributas.	Patient number (toliau pavadinsime `Patient_ID`) yra privalomas lentelės `Patient` atributas, kuris privalo būti unikalus.
2.	*has*		
3.	term	DB lentelės pavadinimas `Term`	Transformuojama į lentelės pavadinimą `Patient`.

4.	unique identifier	DB lentelės `Term` pirminis raktas (atributas) unique identifier	Lentelės `Patient` atributas `Patient_ID` yra pirminis raktas

Pasirinkta SBVR taisyklė "**It is obligatory that patient *has* unique patient number**" priklauso struktūrinių ribojimų klasei ir darnos ribojimų poklasiui, kuris taisyklių saugyklos duomenų bazėje galės būti identifikuojamas pagal naudojamam taisyklės šablonui RuleTemplate_ID=1 priskirtą poklasį RuleSubCalss_ID=3, todėl taisyklių saugykloje šią taisyklę išsaugome su identifikaciniu numeriu Rule_ID=0001, ir priskirtu taisyklės šablonu RuleTemplate_ID=1.

Pagal ankstesniame skyriuje aptartus siūlomo taisyklių automatizavimo duomenų bazėje metodo žingsnius, toliau sukursime pasirinktos taisyklės šablonui tinkamą DDL kodą. Konkrečios taisyklės atveju tai bus duomenų bazės lentelės sukūrimo sakinys. Visgi, čia reikėtų atsižvelgti ir į tai, kad sukūrimo sakinys tokio tipo taisyklei tiks tik tuo atveju, jei taisykle aprašomos esybės lentelė dar nėra sukurta duomenų bazėje. Todėl šiame etape, žiūrint į galimą poreikį ateityje, būtų tikslinga sukurti du šią taisyklę realizuojančius DDL kodo šablonus- vieną lentelės sukūrimui, kitą- jau galimai duomenų bazėje esančios lentelės atributo savybių, konkrečiai pirminio rakto sukūrimui. DDL kodo šablonas lentelės sukūrimui:

```
USE [Database_Name]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[term](
     [term_ID] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_term] PRIMARY KEY CLUSTERED
(
     [term_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Šį šabloną išsaugome taisyklių saugykloje, priskirdami jį ObjectType_ID=1, kur DB objekto tipas yra „lentelė". DDL kodo šablonui suteikiamas identifikacinis numeris

CodeTemplate_ID=1, ir jis išoriniu raktu susiejamas su konkrečiu anksčiau taisyklių saugykloje išsaugotu SBVR taisyklės šablonu.

Kuriant naujos duomenų bazės lentelės sukūrimo DDL kodo šabloną daroma prielaida, kad pati duomenų bazė `Database_Name` jau yra sukurta. Kaip buvo aprašyta taisyklės transformavimo lentelėje, pirmasis terminas taisyklėje bus transformuojamas į duomenų bazės lentelės pavadinimą, o antrasis į tos lentelės atributą kuriam sukuriamas pirminis raktas.

Kaip minėta anksčiau, gali būti, kad tokio tipo taisyklė bus įgyvendinama jau sukūrus esybės <term> lentelę duomenų bazėje pagal kokią nors kitą dalykinės srities verslo taisyklę. Tokiu atveju galima būtų parinkti kitą, tai pačiai taisyklei priskirtą DDL kodo šabloną, skirtą lentelės keitimui:

```
ALTER TABLE [dbo].[term] ADD term_ID datatype
CONSTRAINT term_ID_PrimaryKey PRIMARY KEY (term_ID);
```

Šį šabloną taip pat išsaugome taisyklių saugykloje, priskirdami jį ObjectType_ID=1, kur DB objekto tipas yra „lentelė“. DDL kodo šablonui suteikiamas identifikacinis numeris CodeTemplate_ID=2, ir jis susiejamas su tuo pačiu anksčiau taisyklių saugykloje išsaugotu SBVR taisyklės šablonu.

Taigi, mūsų konkrečios taisyklės apie pacientą ir jo unikalų numerį atveju, taikysime naujo duomenų bazės objekto sukūrimo DDL kodo šabloną CodeTemplate_ID=1. Įvedus kintamųjų reikšmes (tai būtų atliekama metodo realizacijai pritaikytos programos pagalba, tai yra, kintamuosius nuskaitant iš atitinkamų vartotojo sąsajoje suvestų SBVR taisyklės elementų), DDL kodas lentelės `Patient` sukūrimui būtų toks:

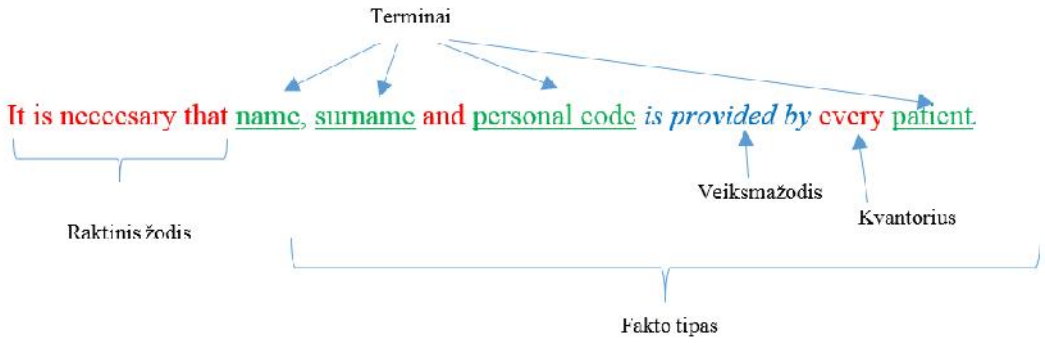
```
CREATE TABLE [dbo].[Patient](
     [Patient_ID] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_Patient] PRIMARY KEY CLUSTERED
(
     [Patient_ID] ASC
)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Šį kodą panaudojame naujai duomenų bazės lentelei `Patient` sukurti. Kol kas, kaip ir buvo apibrėžta pirmoje išanalizuotoje taisyklėje “**It is obligatory that patient *has* unique patient number**“ lentelė `Patient` turi tik vieną atributą `Patient_ID`, kuris yra ir pirminis raktas.

Šiame etape, pagal metodo įgyvendinimo žingsnių seką, priskiriame mūsų konkrečią taisyklę realizuojančiam objektui unikalų identifikacinį numerį, ir išsaugome duomenis apie šį objektą taisyklių saugyklos lentelėje ImplementationObject (Object_ID=0001, ObjectType_ID=1 (*objekto tipas „lentelė“*)).

Taigi, taikant metodą konkrečiai taisyklei Rule_ID=0001, ją realizavome sukūrę DB objektą Object_ID=0001. Taisyklės ir ją realizuojančio DB objekto susiejimui sukuriame ir taisyklių saugykloje išsaugome ryšį RuleLink=0001, kuris nusako jog taisyklė Rule_ID=0001 yra realizuota objekte Object_ID=0001. Realizuojantis objektas taisyklių saugykloje išsaugomas su jo aprašymu, t.y. vienareikšmiška nuoroda, kuri leistų lokalizuoti kur tiksliai įgyvendinta taisyklė.

Toliau metodą taikysime antrajai pasirinktos dalykinės srities taisyklių rinkinio taisyklei „Pirmą kartą sistemoje registruojamas pacientas privalo nurodyti asmens kodą, vardą ir pavardę“. Užrašome šią taisyklę pagal SBVR:



Pasirinkta taisyklė sudaryta iš būtinumą nusakančio raktinio žodžio „It is neccesary that“, kuris kartu su veiksmažodžiu „*is provided by*“ apibrėžia, kad kiekvienas termino „patient“ egzempliorius, turi pateikti atributų, kuriuos čia nusako kiti terminai „name, surname and personal code“ reikšmes.

Šiuo atveju, kuriant tokio tipo taisyklės šabloną, kintamuosius sudarys terminai, kurie priklausomai nuo jų vietos taisyklės teiginyje, bus transformuojami į lentelę ir jos privalomus atributus. Taigi SBVR taisyklės šablonas bus toks:

It is neccesary that <term1>, <term2>.....and <termN> *is provided by* every <term>

Lentelėje 5 pateikiama transformacijos schema, t.y. išanalizuojama į ką konkrečiai bus transformuojamas kiekvienas SBVR taisyklės elementas:

5 lentelė. ***SBVR taisyklės transformacijos aprašymas***

Eil. Nr.	**SBVR elementas (kas transformuojama)**	**DB elementas (į ką transformuojama)**	**Pritaikymas taisyklei „ It is neccesary that name, surname and personal code *is provided by* every patient“**
1.	It is necessary that … *is provided by*	Atributų reikšmių ribojimas NOT NULL	Nusako, kad visi taisyklėje išvardinti lentelės Patient atributai (Name, Surname ir Personal_code) privalo būti užpildyti
2.	name, surname and personal code	Lentelės `<term>` atributai `<term1>`, `<term2>...` ir `<termN>`	Lentelės `Patient` atributai `Name`, `Surname` ir `Personal_code`
3.	every	Ribojimas, nusakantis, kad ši taisyklė taikoma kiekvienam lentelės `<term>` įrašui.	Ribojimas, nusakantis, kad ši taisyklė taikoma kiekvienam lentelės `Patient` įrašui.
4.	patient	DB lentelė `<term>`	DB lentelė `Patient`

Pasirinkta SBVR taisyklė “ It is neccesary that name, surname and personal code *is provided by* every patient“ priklauso struktūrinių ribojimų klasei ir darnos ribojimų poklasiui, kuris taisyklių saugyklos duomenų bazėje galės būti identifikuojamas pagal naudojamam taisyklės šablonui RuleTemplate_ID=2 priskirtą poklasį RuleSubCalss_ID=3, todėl taisyklių saugykloje šią taisyklę išsaugome su identifikaciniu numeriu Rule_ID=0002, ir priskirtu taisyklės šablonu RuleTemplate_ID=2.

Pagal ankstesniame skyriuje aptartus siūlomo taisyklių automatizavimo duomenų bazėje metodo žingsnius, toliau sukursime pasirinktos taisyklės šablonui tinkamą DDL kodą. Šios

konkrečios taisyklės atveju duomenų bazės lentelės sukūrimo sakinys jau netinka (lentelę Patient jau sukūrėme duomenų bazėje realizuodami pirmąją taisyklę), todėl šablono sukūrimui naudosime ne naujo objekto sukūrimo, o jau esamo objekto modifikavimo DDL kodo sintaksę. Visgi, kaip jau buvo aptarta analizuojant pirmąją taisyklę, gali būti taip, kad tokio tipo taisyklė bus analizuojama dar neturint sukurtos lentelės, todėl reikės jos sukūrimo DDL kodo. Todėl kaip ir pirmosios taisyklės ir jai pritaikyto šablono atveju, ir šiai taisyklei sukursime du šią taisyklę realizuojančius DDL kodo šablonus- vieną lentelės sukūrimui, kitą- esamos lentelės modifikavimui- lentelės atributų bei jų NOT NULL ribojimų pridėjimui.

DDL kodo šablonas lentelės sukūrimui:

```
CREATE TABLE [dbo].[term](
[term1] [datatype] NOT NULL,
[term2] [datatype] NOT NULL,
.....................................
[termN] [datatype] NOT NULL,
) ON [PRIMARY]

GO
```

Šis DDL kodo šablonas yra pritaikytas bet kokiam terminų skaičiui taisyklėje. Jei objektas, kuris šiuo konkrečiu atveju yra lentelė pavadinimu „term“ jau yra sukurtas, reikėtų naudoti kitą šabloną, skirtą lentelės atributų keitimui:

```
ALTER TABLE [dbo].[term]
ADD
[term1] [datatype] NOT NULL,
[term2] [datatype] NOT NULL,
.....................................
[termN] [datatype] NOT NULL
GO
```

Šiuos šablonus išsaugome taisyklių saugykloje, priskirdami jiems ObjectType_ID=1, kur DB objekto tipas, kaip ir pirmosios taisyklės atveju, yra „lentelė“. DDL kodo šablonams suteikiame identifikacinius numerius CodeTemplate_ID=3, ir CodeTemplate_ID=4 ir jie išoriniu raktu RuleTemplate_ID=2 susiejami su konkrečiu anksčiau taisyklių saugykloje išsaugotu SBVR taisyklės šablonu.

Taikome kodo šabloną CodeTemplate_ID=4 konkrečiai taisyklei “ It is neccesary that name, surname and personal code *is provided by* every patient“ įrašydami kintamuosius bei duomenų tipus:

```
USE [Database_Name]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TABLE [dbo].[Patient]
ADD
[Name] nvarchar(50) NOT NULL,
[Surname] nvarchar(50) NOT NULL,
[Personal_code] nvarchar(50) NOT NULL
GO
```

Šiame etape, pagal metodo įgyvendinimo žingsnių seką, priskiriame mūsų konkrečią taisyklę realizuojančiam DDL kodui unikalų identifikacinį numerį, ir išsaugome duomenis apie šį objektą taisyklių saugyklos lentelėje `ImplementationObject` (Object_ID=0002, ObjectType_ID=1 (*objekto tipas „lentelė“)*).

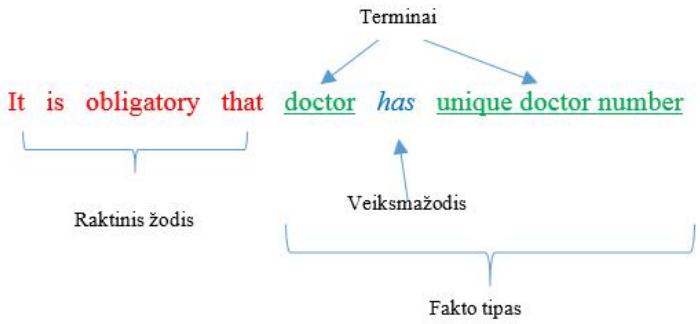
Taigi, taikant metodą konkrečiai taisyklei Rule_ID=0002, ją realizavome sukūrę DB objektą Object_ID=0002. Taisyklės ir ją realizuojančio DB objekto susiejimui sukuriame ir taisyklių saugykloje išsaugome ryšį RuleLink=0002, kuris nusako jog taisyklė Rule_ID=0002 yra realizuota objekte Object_ID=0002.

3.4 Sukurtų šablonų taikymas

Sukurtiems šablonams patikrinti taikysime juos kitoms pasirinktos dalykinės srities verslo taisyklių rinkinio taisyklėms. Pradžioje sukurtus šablonus bandysime pritaikyti su esybe „Gydytojas“ susijusioms taisyklėms:

- Visi įstaigoje dirbantys gydytojai privalo būti registruoti sistemoje
- Kiekvienas gydytojas privalo turėti unikalų gydytojo numerį
- Gydytojas privalo nurodyti vidutinę pas jį besilankančių pacientų vizito trukmę

Šias tris taisykles, panašiai kaip ir pirmųjų, su pacientu susijusių, taisyklių atveju, galime jas formalizuoti transformuodami į dvi SBVR taisykles:



Šiai taisyklei pritaikėme SBVR taisyklės šabloną RuleTemplate=1:

It is obligatory that <term> has unique <term> number

Taisyklės realizavimui duomenų bazėje taikysime DDL kodo šabloną CodeTemplate_ID=1 (parenkame iš dviejų pirmajam taisyklės šablonui sukurtų DDL kodo šablonų).

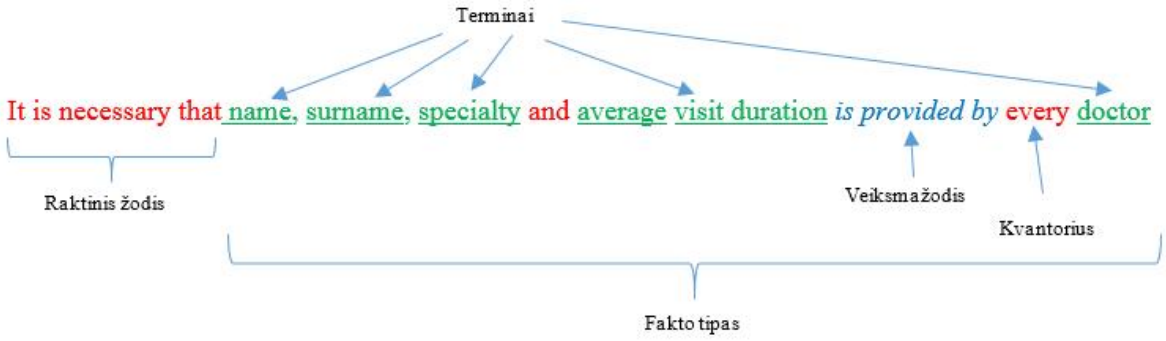
Metodą realizuojančioje programoje taisyklės kintamieji, suvesti vartotojo sąsajoje būtų panaudojami sugeneruoti kodui, kurį galima realizuoti kuriamos IS duomenų bazėje:

```
CREATE TABLE [dbo].[Doctor](
	[Doctor_ID] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_Doctor] PRIMARY KEY CLUSTERED
(
	[Doctor_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Antrąją taisyklę, kuria teigiama, kad kiekvienas gydytojas turi nurodyti vardą, pavardę, specialybę ir vidutinę vizito trukmę, užrašome formaliai naudojant anksčiau sukurtą šabloną RuleTemplate=2.

It is neccesary that <term1>, <term2>.....and <termN> *is provided by* every <term>

T.y. vardas, pavardė, specialybė ir vidutinė vizito trukmė bus vienas iš jau sukurtos duomenų bazės lentelės `Doctor` atributų.



Kadangi realizuodami ankstesnę taisyklę apie unikalų kiekvieno gydytojo numerį jau sukūrėme lentelę `Doctor`, šios taisyklės įgyvendinimui duomenų bazėje naudosime DDL kodo šabloną, skirtą lentelės keitimui, t.y. CodeTemplate=4:

```
ALTER TABLE [dbo].[Doctor]
ADD
[Name] nvarchar(50) NOT NULL,
[Surname] nvarchar(50) NOT NULL,
[Specialty] nvarchar(50) NOT NULL
[average_visit_duration] int NOT NULL
GO
```

Duomenų bazėje realizavus šį DDL kodą lentelė `Doctor` papildoma dar vienu atributu `average_visit_duration`. Su savybe NOT NULL, neleidžiančia, kad ši reikšmė būtų nenurodoma sistemoje registruojant naują įstaigos gydytoją.

Kitos taisyklių rinkinio taisyklės, susijusios su trečia esybe „Pamaina“:

- Kiekvieno gydytojo pamaina turi būti registruojama sistemoje
- Kiekviena gydytojo pamaina turi turėti nurodytą darbo pradžios ir darbo pabaigos laiką
- Gydytojo pamainos trukmė negali viršyti 12 val. per parą

Šioms taisyklėms jau sukurtų šablonų parinkimas nėra labai akivaizdus. Visgi, tinkamai jas transformavus į formalią SBVR išraišką, bent dvejoms iš jų galime pritaikyti jau turimus taisyklių šablonus:

It is obligatory that shift *has* unique shift number

Šiai taisyklei pritaikėme SBVR taisyklės šabloną RuleTemplate=1:

It is obligatory that <term> has unique <term> number

Taisyklės realizavimui duomenų bazėje taikysime DDL kodo šabloną CodeTemplate_ID=1 (parenkame iš dviejų pirmajam taisyklės šablonui sukurtų DDL kodo šablonų).

Iš atitinkamų SBVR taisyklės šablone įvestų kintamųjų sugeneruotą kodą realizuojame duomenų bazėje:

```
CREATE TABLE [dbo].[Shift](
     [Shift_ID] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_Shift] PRIMARY KEY CLUSTERED
(
     [Shift_ID] ASC
)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Šis kodas duomenų bazėje sukuria naują lentelę `Shift`, su kol kas vieninteliu atributu `Shift_ID`.

Pagal SBVR standartą užrašius antrąją šiame etape nagrinėjamą taisyklę „Kiekviena gydytojo pamaina turi turėti nurodytą darbo pradžios ir darbo pabaigos laiką“, galime įžvelgti, jog ji apibrėžia ne tik naujai sukurtos lentelės `Shift` atributus, bet ir ryšį tarp šios lentelės ir anksčiau sukurtos lentelės `Doctor`:

6 lentelė. ***SBVR taisyklės elementų transformacija į DB schemos elementus***

Eil. Nr.	SBVR elementas (kas transformuojama)	DB elementas (į ką transformuojama)
1.	Each	Nusako, kad taisyklės ribojimai taikomi kiekvienam lentelės Shift įrašui.
2.	shift	DB lentelė `Shift`, turinti nuorodą į tėvo lentelę `Doctor` per išorinio rakto atributą `Doctor_ID`
3.	of a	Išorinis raktas `Doctor_ID` lentelėje `Shift`. Nnusako tėvo-vaiko ryšį tarp lentelių `Doctor` ir `Shift`.
4.	doctor	Nuoroda į tėvo lentelę `Doctor (REFERENCES Doctor (Doctor_ID)`
5.	must *have*	Atributų reikšmių ribojimas NOT NULL
6.	begin time and end time	Lentelės `Shift` atributai `begin_time` ir `end_time`

Tokiai taisyklei sukuriame šabloną, kuriame kintamaisiais bus visi šios taisyklės terminai (juos sunumeruojame):

Each <term1>of a <term2> must have <term3> and <term4>

Tokiam SBVR šablonui realizuoti sukuriame kodo šabloną:

```
ALTER TABLE [dbo].[term1]
ADD
```

```
[term3] [datatype] NOT NULL,
[term4] [datatype] NOT NULL,
[term2_ID] [datatype] NOT NULL

GO
ALTER TABLE [dbo].[term1]  WITH CHECK ADD  CONSTRAINT [FK_term1_term2] FOREIGN
KEY([term2_ID])
REFERENCES [dbo].[term2] ([term2_ID])
GO

ALTER TABLE [dbo].[term1] CHECK CONSTRAINT [FK_term1_term2]
GO
```

Nagrinėjamai taisyklei pritaikome kodo šabloną įrašius taisyklės kintamuosius, t.y. terminus `shift`, `doctor`, `begin_time` ir `end_time`:

```
ALTER TABLE [dbo].[Shift]
ADD
[begin_time] [datetime] NOT NULL,
[end_time] [datetime] NOT NULL,
[Doctor_ID] [int] NOT NULL

GO
ALTER TABLE [dbo].[Shift]  WITH CHECK ADD  CONSTRAINT [FK_Shift_Doctor] FOREIGN
KEY([Doctor_ID])
REFERENCES [dbo].[Doctor] ([Doctor_ID])
GO

ALTER TABLE [dbo].[Shift] CHECK CONSTRAINT [FK_Shift_Doctor]
GO
```

Šios taisyklės realizacija duomenų bazėje yra lentelė `Shift` su išoriniu raktu nurodančiu į lentelę `Doctor`.

Kita nagrinėjama taisyklė „Gydytojo pamainos trukmė negali viršyti 12 val. per parą“ taip pat dar neturi sukurto šablono. Ši taisyklė jau visai kitokio pobūdžio nei ankstesnės- ji operuoja ne duomenų bazės objektais, o jau pačiais duomenimis. Todėl šablonas, kurį sukursime šiai taisyklei realizuoti jau bus pritaikytas konkrečiai dalykinei sričiai, ir bus skirtas ne naujo duomenų bazės objekto sukūrimui ar keitimui, o duomenų ribojimams įgyvendinti ir esant poreikiui, tai yra pasikeitus taisyklei, juos atitinkamai koreguoti. Taikant SBVR struktūrizuotą anglų kalbą, šią taisyklę užrašome taip:

It is permitted that duration of a shift *is* at most 12 hours

Šiai taisyklei įgyvendinti turime tiksliai apibrėžti terminą „duration“- kaip ir kitus pasirinktos dalykinės srities terminus. Kadangi tai nėra šio darbo pagrindinis tikslas, darysime prielaidą, kad toks žodynas su dalykinės srities terminais ir jų nedviprasmiškais apibrėžimais jau

yra sudarytas. O šioje ir kitose taisyklėse naudojamas terminas „duration“ (trukmė), bus vartojimas kaip terminas nusakantis skirtumą tarp jo pradžios ir pabaigos laiko. Taigi, nagrinėjamos taisyklės atveju, tai bus skirtumas tarp jau sukurtos lentelės `Shift` atributų `begin_time` ir `end_time`. Tam kad tokią taisyklę ateityje galima būtų modifikuoti, sukursime šabloną, kuriame kintamieji bus valandų skaičius (kvantorius), ir terminai - tokiu atveju taisyklę bus galima automatiškai modifikuoti pasikeitus nustatytam darbo valandų skaičiui:

It is permitted that <term1> of a <term2> is at most <quantification> hours

Taisyklė duomenų bazėje bus realizuojama trigeriu, kuris tikrins, kad skirtumas tarp `begin_time` ir `end_time` neviršytų nustatyto valandų skaičiaus:

```
CREATE trigger [dbo].[term1_duration]
on [dbo].[term1]
for insert
as
begin
 if exists(
 select *
 from inserted where datediff(hour,begin_time,end_time)>quantification)
 begin
 raiserror (Duration of a <term1> can not exceed <quantification> hours!', 16,
1);
 rollback transaction;
 return
 end;
end
GO
```

Įrašius taisyklės kintamuosius, sukuriame trigerį `shift_duration`, kuris tikrins, kad įvedamos pamainos trukmė neviršytų nustatyto valandų skaičiaus (t.y. 12 val.).

Nagrinėjame likusias, dar nerealizuotas, pasirinkto taisyklių rinkinio taisykles:

- Kiekvienas vizitas turi turėti unikalų vizito numerį
- Vienas vizitas sudaromas tarp vieno gydytojo ir vieno paciento
- Kiekvienas paciento vizitas pas gydytoją privalo turėti pradžios ir pabaigos laiką
- Vieno vizito pas gydytoją trukmė negali viršyti 1 val.

Iš šių taisyklių galime spręsti, kad „Vizitas“ turės būti dar viena nauja duomenų bazės lentelė, nes ji, be tėvo-vaiko ryšio jau sukurtų lentelių `Patient` ir `Doctor` atžvilgiu, turi ir savo atributų- unikalų numerį, bei pradžios ir pabaigos laiką. Užrašome taisykles pagal SBVR standartą, jau taikydami sukurtus SBVR taisyklių šablonus:

It is obligatory that visit *has* unique visit number

Taikytas SBVR šablonas **“It is obligatory that <term> has unique <term> number”**

Each visit of a patient must *have* begin time and end time

Taikytas SBVR šablonas **“Each <term1>of a <term2> must have <term3> and <term4>“**

It is permitted that duration of a visit *is* at most 1 hour

Taikytas SBVR šablonas **“It is permitted that <term1> of a <term2> is at most <quantification> hours”**

Parinkus atitinkamą SBVR taisyklės šablono kodų šabloną, sukuriame lentelę `Visit`, su unikaliu numeriu `Visit_ID`:

```
CREATE TABLE [dbo].[Visit](
     [Visit_ID] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_Visit] PRIMARY KEY CLUSTERED
(
     [Visit_ID] ASC
)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Pritaikius kitą taisyklę realizuojantį kodo šabloną lentelę papildome atributais `begin_time` ir `end_time` bei išoriniu raktu Patient_ID:

```
ALTER TABLE [dbo].[Visit]
ADD
[begin_time] [datetime] NOT NULL,
[end_time] [datetime] NOT NULL,
[Patient_ID] [int] NOT NULL

GO
ALTER TABLE [dbo].[Visit]  WITH CHECK ADD  CONSTRAINT [FK_Visit_Patient] FOREIGN
KEY([Patient_ID])
REFERENCES [dbo].[Patient] ([Patient_ID])
GO

ALTER TABLE [dbo].[Visit] CHECK CONSTRAINT [FK_Visit_Patient]
GO
```

Trigeriu `Visit_duration`, tikrinančiu, kad įvedamų vizito pradžios ir pabaigos laikų skirtumas neviršytų vienos valandos relaizuojame ir trečiąją taisyklę“ It is permitted that duration of a visit *is* at most 1 hour”:

```
CREATE trigger [dbo].[visit_duration]
on [dbo].[visit]
for insert
as
begin
 if exists(
 select *
 from inserted where datediff(hour,begin_time,end_time)>1)
 begin
 raiserror (Duration of a visit can not exceed 1 hour!', 16, 1);
 rollback transaction;
 return
 end;
end
GO
```

Iš viso šiame skyriuje nagrinėjamo taisyklių rinkinio liko nerealizuota viena taisyklė „Vienas vizitas sudaromas tarp vieno gydytojo ir vieno paciento“. Šios taisyklės realizacija duomenų bazėje yra du išoriniai raktai `Doctor_ID` ir `Patient_ID` lentelėje `Visit`, nurodantys į atitinkamas lenteles `Doctor` ir `Patient`. Ši taisyklė dar neturi sukurto SBVR taisyklės ir ją realizuojančio kodo šablonų. Be to, vienas iš minėtų išorinių raktų duomenų bazėje jau sukurtas realizavus anksčiau nagrinėtą taisyklę „Each visit of a patient must have begin time and end time”. Kadangi dalis taisyklės jau realizuota, galime perfrazuoti nagrinėjamą taisyklę į SBVR taisyklę apibrėžiančią tik vieną, trūkstamą išorinio rakto nuorodą į lentelę `Doctor`.:

It is permitted that one visit *is made with* only one doctor

Šios taisyklės šablonas, kaip kintamuosius naudojant terminus, būtų toks:

It is permitted that one <term1> is made with only one <term2>

O jį realizuoti, kai abiejų terminų (kintamųjų) lentelės jau yra sukurtos, galime pagal tokį kodo šabloną:

```
ALTER TABLE [dbo].[term1]
ADD
 [term2_ID] [int] NOT NULL
GO
ALTER TABLE [dbo].[term1]  WITH CHECK ADD  CONSTRAINT [FK_term1_term2] FOREIGN
KEY([term2_ID])
REFERENCES [dbo].[term2] ([term2_ID])
GO

ALTER TABLE [dbo].[term1] CHECK CONSTRAINT [FK_ term1_term2]
GO
```

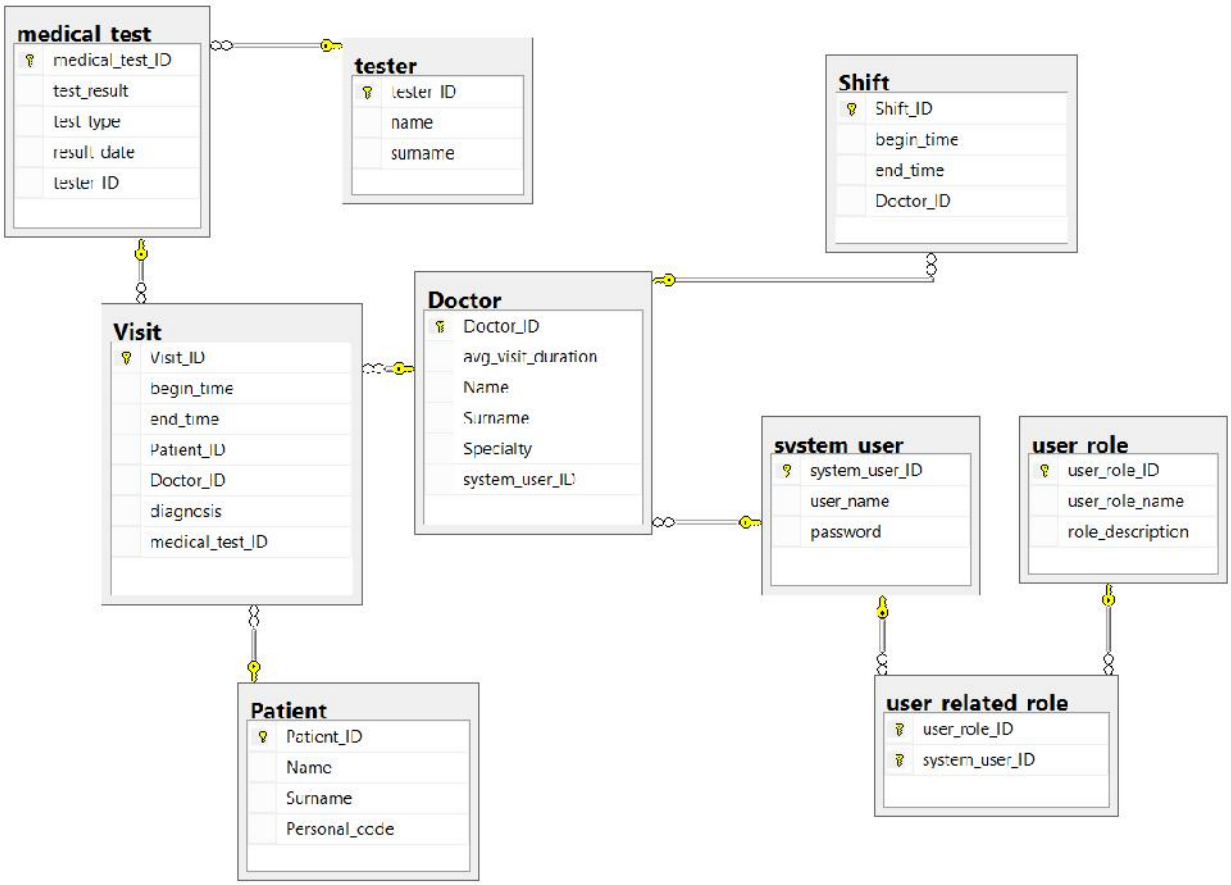
3.5 Rezultatai

Duomenų bazės schema

Pažingsniui taikant siūlomą metodą verslo taisyklėms automatizuoti duomenų bazėje buvo realizuotas pasirinktos dalykinės srities verslo taisyklių rinkinys. Dauguma parinktų verslo taisyklių buvo skirtos duomenų bazės schemai ir jos ribojimams aprašyti. Taisyklės pirmiausia buvo užrašomos struktūrizuota anglų kalba. Vėliau, atsižvelgiant į taisyklės teiginio struktūrą buvo sudaromas SBVR taisyklės šablonas- tai yra, tam tikros taisyklės teiginio dalys fiksuojamos, o kitos pakeičiamos tam tikrais kintamaisiais. Taisyklių rinkinys buvo nagrinėjamas po kelias, atrenkant taisykles su vartojamais tais pačiais terminais, kas dažnai reiškia tą patį duomenų bazės objektą. Kai kurioms taisyklėms buvo kuriama po du realizuojančio kodo šablonus- vienas naujo DB objekto sukūrimui, kitas- jau esamo objekto keitimui. Tokiu atveju šablonas realizuojant ar esant poreikiui keičiant kitas tokios pačios struktūros taisykles būtų parenkamas priklausomai nuo to, ar reikiamas duomenų bazės objektas jau buvo sukurtas anksčiau.

Realizavus detaliai nagrinėjamą taisyklių rinkinį, buvo sukurtos keturios duomenų bazės lentelės. Taip pat, su išoriniais raktais buvo sukurti ryšiai tarp jų. Dauguma taisyklių buvo realizuota taikant DDL kodo šablonus. Visgi, tokiu pačiu principu taisykles galima realizuoti ir DML kodu. Buvo sukurti duomenų vientisumą ribojantys trigeriai `visit_duration` ir `shift_duration`.

Likusių pasirinktos dalykinės srities taisyklių rinkinio taisyklių transformacija pateikiama Priede Nr. 1.



12 pav. ***Kuriamos IS duomenų bazės schema***

Atsekamumo lentelė

Verslo taisyklių atsekamumui užtikrinti kiekviena taisyklė užrašyta laikantis SBVR standarto buvo išsaugota su unikaliu numeriu taisyklių ir atsekamumo ryšių saugykloje, kurios schema buvo pateikta 4 pav. Taisyklė priskiriama konkrečiam poklasiui. SBVR šablonas taisyklių saugyklos duomenų bazėje nėra susiejamas su konkrečia taisykle, nes toks susiejimas net ir keičiantis taisyklei nelabai turėtų prasmės. Taisyklė susiejama tik su konkrečiu ją realizuojančio kodo šablonu (jei ateityje ją reikėtų modifikuoti) ir su konkrečiu ją realizuojančiu duomenų bazės objektu. Toks susiejimas, kadangi formaliai taisyklė gali būti realizuojama daugiau nei viename duomenų bazės objekte, yra sukuriamas per saugyklos elementą `RuleLink`- tai yra atsekamumo ryšys tarp konkrečios taisyklės ir ją realizuojančio duomenų bazės objekto. Iš tokios duomenų saugyklos galime ataskaitų forma galime gauti visą reikiamą informaciją tiek apie visą sistemos

taisyklių rinkinį, tiek apie juos realizuojančius objektus. T.y. bet kada bus galima pasakyti kuriame objekte (-uose), konkrečiu atlikto eksperimento atveju, kokiame duomenų bazės schemos elemente yra realizuota konkreti verslo taisyklė, ir priešinga kryptimi, kurias verslo taisykles realizuoja konkretus programų sistemos lygmens objektas. Taigi, pasirinktos dalykinės srities verslo taisyklių, užrašytų laikantis SBVR standarto rinkinys buvo išsaugotas priskiriant kiekvienai taisyklei unikalų taisyklės numerį:

7 lentelė. ***SBVR taisyklių rinkinys taisyklių ir atsekamumo ryšių saugykloje***

Taisyklės numeris	**Taisyklė**
0001	It is obligatory that patient has unique patient number
0002	It is necessary that name, surname and personal code is provided by every patient
0003	It is obligatory that doctor has unique doctor number
0004	It is necessary that name, surname, specialty and average visit duration is provided by every doctor
0005	It is obligatory that shift has unique shift number
0006	Each shift of a doctor must have begin time and end time
0007	It is permitted that duration of a shift is at most 12 hours
0008	It is obligatory that visit has unique visit number
0009	Each visit of a patient must have begin time and end time
0010	It is permitted that duration of a visit is at most 1 hour
0011	Every visit must have diagnosis
0012	Every doctor must be system user
0012	Every doctor must be system user
0013	Each system user must be related to only one user role
0013	Each system user must be related to only one user role
0013	Each system user must be related to only one user role
0014	It is obligatory that system user has unique system user number
0015	It is obligatory that user role has unique user role number
0016	It is obligatory that medical test has unique medical test number

0017	It is obligatory that tester has unique tester number
0018	Patient can have medical test taken on each visit
0018	Patient can have medical test taken on each visit
0019	Every medical test must be performed by only one tester
0019	Every medical test must be performed by only one tester
0020	Every system user must have user name and password
0021	Every user role must have user role name and role description
0022	Every medical test must have test result, test type and result date
0023	Every tester must have name and surname

Taisykles realizuojantys objektai taisyklių saugykloje taip pat turi unikalų numerį, aprašymą bei priskirtą objekto tipą:

8 lentelė. ***Taisykles duomenų bazėje realizuojantys objektai***

Realizuojančio objekto Nr.	**Objekto aprašymas**	**Objekto tipas**
0001	Lentele Patient su piminiu raktu Patient_ID	Lentele
0002	Lenteles Patient papildymas atributais Name,Surname ir Personal_code	Lentele
0003	Lentele Doctor su pirminiu raktu Doctor_ID	Lentele
0004	Lenteles Doctor papildymas atributais Name,Surname, Speciality ir Average_visit_duration	Lentele
0005	Lentele Shift su pirminiu raktu Shift_ID	Lentele
0006	Preidedami papildomi atributai begin_time,end_time ir Doctor_ID lenteleje Shift, Doctor_ID išrorinis raktas	Lentele su išoriniu raktu
0007	Trigeris tikrinantis kad pamainos shift trukme neviršytu nustatyto valandu skaiciaus	DML trigeris
0008	Lentelė Visit su pirminiu raktu Visit_ID	Lentelė
0009	Lentelė Visit su atributais begin_time, end_time, ir išoriniu raktu Patient_ID	Lentelė su išoriniu raktu
0010	Trigeris, ribojantis vizito trukme iki nustatyto valandu skaiciaus	DML trigeris
0011	Pridedamas lentelės Visit atributas diagnosis	DB lentelės atributas
0012	Pridedamas išorinis raktas lentelėje Doctor su nuoroda į lentelę system_user	DB lentelės išorinis raktas
0013	Pridedamas lentelės atributas system_user_ID	DB lentelės atributas

0014	Lentelės user_role su sudėtiniu pirminiu raktu sukūrimas	Lentelė
0015	Išorinio rakto sukūrimas lentelėje user_roles-nuoroda į kitą lentelę system_user	DB lentelės išorinis raktas
0016	Išorinio rakto sukūrimas lentelėje user_roles-nuoroda į kitą lentelę user_role	DB lentelės išorinis raktas
0017	Lentelė system_user su piminiu raktu system_user_ID	Lentelė
0018	Lentelė user_role su piminiu raktu user_role_ID	Lentelė
0019	Lentelė medical_test su piminiu raktu medical_test_ID	Lentelė
0020	Lentelė tester su piminiu raktu tester_ID	Lentelė
0023	Papildomas lentelės Visit atributas medical_test	DB lentelės atributas
0024	Išorinio rakto sukūrimas lentelėje Visit-nuoroda į kitą lentelę medical_test	DB lentelės išorinis raktas
0025	Papildomas lentelės medical_test atributas tester_ID	DB lentelės atributas
0026	Išorinio rakto sukūrimas lentelėje medica_test-nuoroda į kitą lentelę tester	DB lentelės išorinis raktas
0027	Lentelės system_user sukūrimas su atributais user_name ir password	Lentelė
0028	Papildomi atributai lentelėje user_role-user_role_name ir role_description	DB lentelės atributas
0029	Papildomi lentelės medical_test atributai test_result, test_type and result_date	DB lentelės atributas
0030	Papildomi lentelės tester atribubtai name ir surname	DB lentelės atributas

Tarp kiekvienos taisyklės ir ją realizuojančių objektų sukuriamas unikalų numerį turintis ryšys:

9 lentelė. ***Atsekamumo ryšių lentelė***

Atsekamumo ryšio Nr.	**Taisykles numeris**	**Realizuojančio objekto Nr.**
0001	0001	0001
0002	0002	0002
0003	0003	0003
0004	0004	0004
0005	0005	0005
0006	0006	0006
0007	0007	0007
0008	0008	0008
0009	0009	0009
0010	0010	0010
0011	0011	0011
0012	0012	0012
0013	0012	0013
0014	0013	0014

0015	0013	0015
0016	0013	0016
0017	0014	0017
0018	0015	0018
0019	0016	0019
0020	0017	0020
0021	0018	0023
0022	0018	0024
0023	0019	0025
0024	0019	0026
0025	0020	0027
0026	0021	0028
0027	0022	0029
0028	0023	0030

Pilna verslo taisyklių atsekamumo ryšių lentelė pateikta Priede Nr. 2.

IŠVADOS IR TOLIMESNI DARBAI

Šiuolaikiniam verslui, kuris yra palaikomas įvairiomis žiniomis, kurios tame tarpe išreikštos verslo taisyklėmis, grindžiamomis informacinėmis sistemomis (IS), būtina turėti galimybę lanksčiai pritaikyti savo IS prie besikeičiančių verslo aplinkos reikalavimų. Atlikus darbą, galima padaryti šias išvadas:

1. Verslo taisyklių vaidmuo IS ir jų programų sistemų inžinerijoje yra reikšmingas, tačiau dar nėra pakankamai ištirtas. IS kūrimas grindžiamas verslo taisyklėmis dar neturi universalaus ir lankstaus sprendimo, leidžiančio kurti naujas ar modifikuoti esamas sistemas be IS specialistų pagalbos.
2. Verslo taisyklėms, kaip ir sistemos reikalavimams gali būti taikomi jų atsekamumą užtikrinantys metodai, leidžiantys vienareikšmiškai lokalizuoti, kur programų sistemos (PS) lygmenyje yra įgyvendinta konkreti taisyklė, ir priešinga kryptimi - kokią verslo taisyklę realizuoja tam tikras PS kodo fragmentas arba objektas.
3. Darbe pasiūlytas ir pasirinktam dalykinės srities verslo taisyklių rinkiniui pritaikytas metodas automatiškai realizuoti verslo taisykles, išlaikant jų atsekamumą, parodė, jog toks sprendimas gali būti pritaikomas sukūrus PS, kuri verslo atstovams (ne programuotojams) leistų pagal sukurtus šablonus keisti verslo taisyklių realizaciją IS duomenų bazėje. Verslo taisyklės gali būti realizuojamos automatiškai, jei joms yra pritaikomi iš anksto sukurti SBVR taisyklių bei jas realizuojančio kodo šablonai.
4. Pasiūlytas metodas gali būti išplėstas ir taisyklių realizavimui ne tik duomenų bazėje, kaip buvo patikrinta šiame darbe, bet ir verslo taisykles realizuojant dalykinėje programoje.
5. Kuriant pasiūlytą metodą galinčią įgyvendinti programinę įrangą detalesnės analizės reikalautų verslo taisyklių tarpusavio sąveikos aspektas. Tokia automatiškai verslo taisykles realizuojanti sistema be vertikalaus atsekamumo iš verslo sistemos lygmens į programų sistemos lygmenį turėtų apimti ir horizontalų verslo taisyklių susiejimą, kuris leistų išvengti verslo taisyklių prieštaravimų.

Literatūros sąrašas

[1] V. Avdejenkov, „Verslo taisyklių valdymo sistemų taikymo įmonėse tyrimas,“ *Daktaro disertacija,VGTU*, 2009.

[2] A. Čaplinskas, Reikalavimų inžinerija, Vilnius, 2007.

[3] Business Rules Group, „Guide Business Rules Project Final Report,“ 2000.

[4] D. Kalibatienė ir O. Vasilecas, „Ontology axioms for the implementation of business rules,“ *Technological and Economic Development of Economy*, t. 16, pp. 471-486, 2010.

[5] D. Kalibatienė ir O. Vasilecas, Šiuolaikinės duomenu bazės, 2008.

[6] M. Muehlen ir M. Indulska, „Modeling languages for business processes and business rules: A representational analysis,“ *Information systems*, t. 35, pp. 379-390, 2010.

[7] N. Karami ir J. Ilijima, „A Logical Approach for Implementing Dynamic Business Rules,“ *Contemporary Management Research*, t. 6, pp. 29-52, 2010.

[8] O. Vasilecas ir D. Kalibatienė, „Ontology-Based Application for Domain Rules Development,“ *Computer Science and Information Technologies*, t. 756, pp. 9-32, 2010.

[9] O. Vasilecas ir R. Dubauskaitė, „An open issues in business rules – based information system development,“ *Innovative infotechnologies for science, business and education*, t. 1, pp. 3.1-3.5, 2009.

[10] S. Sosunovas, „User defined templates for the specification and transformation of business rules,“ *Doctoral Dissertation*, 2008.

[11] I. Valatkaitė ir O. Vasilecas, „Automatic enforcement of business rules as ADBMS triggers from conceptual graphs model,“ *Informacinės technologijos ir valdymas*, t. 2, pp. 36-42, 2004.

[12] M. Selway, G. Grossmann, W. Mayer ir M. Stumptner, „Formalising natural language specifications using a cognitive linguistic/configuration based approach,“ *Information systems*, t. 54, pp. 191-208, 2015.

[13] J. Karpovič, „Veiklos žodyno ir veiklos taisyklių semantikos vaizdavimas semantinio tinklo ontologijų kalba,“ *Daktaro disertacijos santrauka*, 2015.

[14] Object Management Group, „Semantics of Business Vocabulary and Business Rules,“ OMG, 2013. [Internete]. Available: http://www.omg.org/spec/SBVR/1.3/index.htm. [Žiūrėta 2016-01-10].

[15] L. Morgenstern, P. Stefaneas, F. Levy, A. Wyner ir A. Paschke, „Theory, Practice and Applications of Rules on the Web,“ *7th International Symposium, RuleML 2013, Seattle, WA, USA*, pp. 23-32, 2013.

[16] S. B. Imran, M. G. Lee ir B. Bordbar, „SBVR Business Rules Generation from Natural Language Specification,“ *Artificial Intelligence for Business Agility — Papers from the AAAI 2011 Spring Symposium*, pp. 2-8, 2011.

[17] E. Pardede, M. Bonais ir W. Rahayu, „Integrating Information Systems Business Rules into a Design Model,“ *15th International Conference on Network-Based Information Systems, IEEE*, pp. 104-111, 2012.

[18] T. Nadeem, „Automated Translation of SBVR to SQL Queries,“ *International Journal of Emerging Sciences*, t. 4, pp. 38-51, 2014.

[19] W. Deroover ir J. Vanthienen, „A Transformation from SBVR Business rules into Event Coordinated Rules by means of SBVR Patterns,“ *Towards a Service-Based Internet*, pp. 172-179, 2010.

[20] W. De Roover ir J. Vanthienen, „Unified Patterns to transform business rules into an event coordination mechanis," *Business Process Management Workshops,* t. 66, pp. 730-742, 2010.

[21] C. Arevalo, M. T. Gomez Lopez, A. R. Quintero ir I. Ramos, „An Architecture to Infer Business Rules from Event Condition Action Rules Implemented in the Persistence Layer," *Uncovering Essential Software Artifacts through Business Process Archeology,* pp. 201-223, 2013.

[22] H. Saleem, Z. Ali Khan ir S. Afzal, „Towards Identification and Recognition of Trace Associations in Software Requirements Traceability," *IJCSI International Journal of Computer Science Issues,* t. 9, pp. 257-263, 2012.

[23] T. Sunil ir M. Kurian, „A methodology to evaluate object oriented software systems using change requirement traceability based on impact analysis," *International Journal of Software Engineering & Applications* , t. 5, pp. 48-60, 2014.

[24] E. Bouillon, P. Mader ir I. Philippow, „A Survey on Usage Scenarios for Requirements Traceability in Practice," *Requirements Engineering: Foundation for Software Quality,* pp. 158-173, 2013.

[25] K. V. Rai ir C. Anantaram, „Structuring business rules interactions," *Electroninc Commerce Research and Applications,* pp. 54-73, 2004.

[26] A. Kannenberg ir H. Saiedian, „Why Software Requirements Traceability Remains a Challenge," *The Journal of Defense Software Engineering,* pp. 14-19, 2009.

[27] K. Wiegers ir J. Beatty, Software Requirements, Third Edition., Microsoft Press, 2013.

[28] N. A. Azram ir R. Atan, „Traceability Method for Software Engineering Documentation," *IJCSI International Journal of Computer Science Issues,* t. 9, pp. 216-220, 2012.

[29] S. Pavalkis, „Išvestinėmis savybėmis grindžiamas modelių atsekamumas,“ *Daktaro disertacijos santrauka*, 2013.

[30] G. Spanoudakis ir A. Zisman, „Software Traceability: A Roadmap,“ *Handbook of Software Engineering and Knowledge Engineering*, 2004.

[31] O. Gotel, J. Cleland-Huang, H. Huffman, A. Zisman, A. Egyed, P. Grunbacher ir G. Antoniol, „The Quest for Ubiquity: A Roadmap for Software and Systems Traceability Research,“ *IEEE*, pp. 71-80, 2012.

[32] A. Hindle, C. Bird, T. Zimmermann ir N. Nagappan, „Do Topics Make Sense to Managers and Developers?,“ *Empirical Software Engineering*, pp. 479-515, 2015.

[33] L. Lin, S. Embury ir B. Warboys, „Tool Support to Implementing Business Rules in Database Applications,“ *Computer Software and Applications Conference, IEEE*, t. 1, pp. 157-162, 2007.

[34] W. Wan-Kadir ir P. Loucopoulos, „Relating evolving business rules to software design,“ *Journal of Systems Architecture*, t. 50, pp. 367-382, 2004.

[35] A. Goknil, I. Kurtev ir K. Van Der Berg, „Generation and validation of traces between requirements and architecture based on formal trace semantics,“ *The Journal oj Systems and Sofware*, t. 88, pp. 112-137, 2014.

[36] D. Selva, C. Bruce ir E. Crawley, „A RuleBased Method for Scalable and Traceable Evaluation of System Architectures,“ *Research in Engineering Design*, t. 25, pp. 325-349, 2014.

[37] M. Wieloch, S. Amornborvornwong ir J. Cleland-Huang, „Trace-by-Classification: A Machine Learning Approach to Generate Trace Links for Frequently Occurring Software Artifacts,“ *Traceability in Emerging Forms of Software Engineering (TEFSE), IEEE*, pp. 110-114, 2013.

[38] A. De Lucia, A. Marcus, R. Oliveto ir D. Poshyvanyk, „Information Retrieval Methods for Automated Traceability Recovery,“ *Software and Systems Traceability*, pp. 71-98, 2011.

[39] S. Nair, J. L. de la Vara ir S. Sen, „A Review of Traceability Research at the Requirements Engineering Conference,“ *IEEE*, pp. 222-229, 2013.

[40] A. I. Andreescu ir M. Mircea, „Perspectives on the Role of Business Rules in Database Design,“ *Database Systems Journal*, t. III, nr. 1, pp. 59-67, 2012.

[41] O. Vasilecas, V. Avdejenkov ir S. Sosunov, „The framework for the implementation of business rules in ERP,“ *Informacijos mokslai*, pp. 146-157, 2009.

[42] O. Vasilecas ir S. Sosunovas, „Preparing business rules templates and object role modelling for transformations,“ *International Conference on Computer Systems and Technologies - CompSysTech'*, t. 42, pp. II.2-1-II.2-6, 2005.

PRIEDAI

1 Priedas. Darbe detaliai nenagrinėtos taisyklių rinkinio dalies transformacija ir pritaikyti šablonai

Taisyklės numeris	Taisyklė	SBVR šablonas	SBVR šablono Nr.	Kodo šablono nr.	Kodo šablonas	Kodas
0011	Every visit must have diagnosis	Every <term> must have <term1>,<term2>.... and <termN>	0005	0008	ALTER TABLE [dbo].[term] ADD [term1] [datatype] NOT NULL, [term2] [datatype] NOT NULL, ... [termN] [datatype] NOT NULL, GO	ALTER TABLE [dbo].[Visit] ADD [diagnosis] nvarchar(max) GO
0012	Every doctor must be system user	Every <term1> must be <term2>	0006	0010	ALTER TABLE [dbo].[term1] ADD [term2_ID] [datatype] NOT NULL GO	ALTER TABLE [dbo].[doctor] ADD [system_user_ID] int NOT NULL GO
0012	Every doctor must be system user	Every <term1> must be <term2>	0006	0009	ALTER TABLE [dbo].[term1] WITH CHECK ADD CONSTRAINT [FK_term1_term2] FOREIGN KEY([term2_ID]) REFERENCES [dbo].[term2] ([term2_ID]) GO ALTER TABLE [dbo].[term1] CHECK CONSTRAINT [FK_term1_term2] GO	ALTER TABLE [dbo].[doctor] WITH CHECK ADD CONSTRAINT [FK_doctor_system_user] FOREIGN KEY([system_user_ID]) REFERENCES [dbo].[system_user] ([system_user_ID]) GO ALTER TABLE [dbo].[doctor] CHECK CONSTRAINT [FK_doctor_system_user] GO

0020	Every system user must have user name and password	Every <term> must have <term1>,<term2>.... and <termN>	0005	0007	CREATE TABLE [dbo].[term]([term1] [datatype] NOT NULL, [term2] [datatype] NOT NULL, ... [termN] [datatype] NOT NULL,) ON [PRIMARY] GO	CREATE TABLE [dbo].[system_user]([user_name] nvarchar(10) NOT NULL, [password] nvarchar(50) NOT NULL,) ON [PRIMARY] GO
0014	It is obligatory that system user has unique system user number	It is obligatory that <term> has unique <term> number	0001	0002	ALTER TABLE [dbo].[term] ADD term_ID int CONSTRAINT term_ID_PrimaryKey PRIMARY KEY (term_ID);	ALTER TABLE [dbo].[system_user] ADD system_user_ID int CONSTRAINT system_user_ID_PrimaryKey PRIMARY KEY (system_user_ID);
0015	It is obligatory that user role has unique user role number	It is obligatory that <term> has unique <term> number	0001	0001	CREATE TABLE [dbo].[term]([term_ID] [int] IDENTITY(1,1) NOT NULL, CONSTRAINT [PK_term] PRIMARY KEY CLUSTERED ([term_ID] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO	CREATE TABLE [dbo].[user_role]([user_role_ID] [int] IDENTITY(1,1) NOT NULL, CONSTRAINT [PK_user_role] PRIMARY KEY CLUSTERED ([user_role_ID] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO
0021	Every user role must have user role name and role description	Every <term> must have <term1>,<term2>.... and <termN>	0005	0008	ALTER TABLE [dbo].[term] ADD [term1] [datatype] NOT NULL, [term2] [datatype] NOT NULL, ... [termN] [datatype] NOT NULL, GO	ALTER TABLE [dbo].[user_role] ADD [user_role_name] nvarchar(50) , [role_description] nvarchar(50) GO

0013	Each system user must be related to only one user role	Each <term1> is related to only one <term2>	0007	0011	CREATE TABLE [dbo].[term1_related_term2]([term2_ID] [int] NOT NULL, [term1_ID] [int] NOT NULL, CONSTRAINT [PK_term1_related_term2] PRIMARY KEY CLUSTERED ([term2_ID] ASC, [term1_ID] ASC) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO	CREATE TABLE [dbo].[user_related_role]([user_role_ID] [int] NOT NULL, [system_user_ID] [int] NOT NULL, CONSTRAINT [PK_user_related_role] PRIMARY KEY CLUSTERED ([user_role_ID] ASC, [system_user_ID] ASC) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO
0013	Each system user must be related to only one user role	Each <term1> is related to only one <term2>	0007	0012	ALTER TABLE [dbo].[term1_related_term2] WITH CHECK ADD CONSTRAINT [FK_term1_related_term2_term2] FOREIGN KEY([term2_ID]) REFERENCES [dbo].[term2] ([term2_ID]) GO ALTER TABLE [dbo].[term1_related_term2] CHECK CONSTRAINT [FK_term1_related_term2_term2] GO	ALTER TABLE [dbo].[user_related_role] WITH CHECK ADD CONSTRAINT [FK_user_related_role_role] FOREIGN KEY([user_role_ID]) REFERENCES [dbo].[user_role] ([user_role_ID]) GO ALTER TABLE [dbo].[user_related_role] CHECK CONSTRAINT [FK_user_related_role_role] GO
0013	Each system user must be related to only one user role	Each <term1> is related to only one <term2>	0007	0013	ALTER TABLE [dbo].[term1_related_term2] WITH CHECK ADD CONSTRAINT [FK_term1_related_term2_term1] FOREIGN KEY([term1_ID]) REFERENCES [dbo].[term1] ([term1_ID]) GO ALTER TABLE [dbo].[term1_related_term2] CHECK CONSTRAINT [FK_term1_related_term2_term1] GO	ALTER TABLE [dbo].[user_related_role] WITH CHECK ADD CONSTRAINT [FK_user_related_role_user] FOREIGN KEY([system_user_ID]) REFERENCES [dbo].[system_user] ([system_user_ID]) GO ALTER TABLE [dbo].[user_related_role] CHECK CONSTRAINT [FK_user_related_role_user] GO

0016	It is obligatory that medical test has unique medical test number	It is obligatory that <term> has unique <term> number	0001	0001	CREATE TABLE [dbo].[term]([term_ID] [int] IDENTITY(1,1) NOT NULL, CONSTRAINT [PK_term] PRIMARY KEY CLUSTERED ([term_ID] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO	CREATE TABLE [dbo].[medical_test]([medical_test_ID] [int] IDENTITY(1,1) NOT NULL, CONSTRAINT [PK_medical_test] PRIMARY KEY CLUSTERED ([medical_test_ID] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO
0018	Patient can have medical test taken on each visit	<term1> can have <term2> taken on each <term3>	0008	0014	ALTER TABLE [dbo].[term3] ADD [term2_ID] int GO	ALTER TABLE [dbo].[Visit] ADD [medical_test_ID] int GO
0018	Patient can have medical test taken on each visit	<term1> can have <term2> taken on each <term3>	0008	0015	ALTER TABLE [dbo].[term3] WITH CHECK ADD CONSTRAINT [FK_term3_term2] FOREIGN KEY([term2_ID]) REFERENCES [dbo].[term2] ([term2_ID]) GO	ALTER TABLE [dbo].[Visit] WITH CHECK ADD CONSTRAINT [FK_Visit_medical_test] FOREIGN KEY([medical_test_ID]) REFERENCES [dbo].[medical_test] ([medical_test_ID]) GO
0022	Every medical test must have test result, test type and result date	Every <term> must have <term1>,<term2>.... and <termN>	0005	0008	ALTER TABLE [dbo].[term] ADD [term1] [datatype] NOT NULL, [term2] [datatype] NOT NULL, ... [termN] [datatype] NOT NULL, GO	ALTER TABLE [dbo].[medical_test] ADD [test_result] nvarchar(MAX) , [test_type] nvarchar(50), [result_date] datetime GO
0019	Every medical test must be performed by only one tester	Every <term1> must be performed by only one <term2>	0009	0016	ALTER TABLE [dbo].[term1] ADD [term2_ID] [int] NOT NULL GO	ALTER TABLE [dbo].[medical_test] ADD [tester_ID] [int] NOT NULL GO

0019	Every medical test must be performed by only one tester	Every <term1> must be performed by only one <term2>	0009	0017	ALTER TABLE [dbo].[term1] WITH CHECK ADD CONSTRAINT [FK_term1_term2] FOREIGN KEY([term2_ID]) REFERENCES [dbo].[term2] ([term2_ID]) GO ALTER TABLE [dbo].[term1] CHECK CONSTRAINT [FK_ term1_term2] GO	ALTER TABLE [dbo].[medical_test] WITH CHECK ADD CONSTRAINT [FK_medical_test_tester] FOREIGN KEY([tester_ID]) REFERENCES [dbo].[tester] ([tester_ID]) GO ALTER TABLE [dbo].[tester] CHECK CONSTRAINT [FK_ medical_test_tester] GO
0017	It is obligatory that tester has unique tester number	It is obligatory that <term> has unique <term> number	0001	0001	CREATE TABLE [dbo].[term]([term_ID] [int] IDENTITY(1,1) NOT NULL, CONSTRAINT [PK_term] PRIMARY KEY CLUSTERED ([term_ID] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO	CREATE TABLE [dbo].[tester]([tester_ID] [int] IDENTITY(1,1) NOT NULL, CONSTRAINT [PK_tester] PRIMARY KEY CLUSTERED ([tester_ID] ASC)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]) ON [PRIMARY] GO
0023	Every tester must have name and surname	Every <term> must have <term1>,<term2>.... and <termN>	0005	0008	ALTER TABLE [dbo].[term] ADD [term1] [datatype] NOT NULL, [term2] [datatype] NOT NULL, ... [termN] [datatype] NOT NULL, GO	ALTER TABLE [dbo].[tester] ADD [name] nvarchar(50) , [surname] nvarchar(50) GO

2 Priedas. Taisyklių atsekamumo lentelė

Taisyklės numeris	Taisyklė	Objekto Nr.	Objekto aprašymas	Objekto tipas	Atsekamumo ryšio Nr.	Naudoto kodo šablono Nr.
0001	It is obligatory that patient has unique patient number	0001	Lentelė Patient su piminiu raktu Patient_ID	Lentelė	0001	0001
0002	It is neccesary that name, surname and personal code is provided by every patient	0002	Lentelės Patient papildymas atributais Name,Surname ir Personal_code	Lentelė	0002	0004
0003	It is obligatory that doctor has unique doctor number	0003	Lentelė Doctor su pirminiu raktu Doctor_ID	Lentelė	0003	0001
0004	It is necessary that name, surname, specialty and average visit duration is provided by every doctor	0004	Lentelės Doctor papildymas atributais Name,Surname, Speciality ir Average_visit_duration	Lentelė	0004	0004
0005	It is obligatory that shift has unique shift number	0005	Lentelė Shift su pirminiu raktu Shift_ID	Lentelė	0005	0001
0006	Each shift of a doctor must have begin time and end time	0006	Preidedami papildomi atributai begin_time,end_time ir Doctor_ID Lentelėje Shift, Doctor_ID išrorinis raktas	Lentelė su išoriniu raktu	0006	0005
0007	It is permitted that duration of a shift is at most 12 hours	0007	Trigeris tikrinantis kad pamainos shift trukme neviršytu nustatyto valandu skaiciaus	DML trigeris	0007	0006
0008	It is obligatory that visit has unique visit number	0008	Lentelė Visit su pirminiu raktu Visit_ID	Lentelė	0008	0001
0009	Each visit of a patient must have begin time and end time	0009	Lentelė Visit su atributais begin_time, end_time, ir išoriniu raktu Patient_ID	Lentelė su išoriniu raktu	0009	0005
0010	It is permitted that duration of a visit is at most 1 hour	0010	Trigeris, ribojantis vizito trukme iki nustatyto valandu skaiciaus	DML trigeris	0010	0006

0011	Every visit must have diagnosis	0011	Pridedamas lentelės Visit atributas diagnosis	DB lentelės atributas	0011	0008
0012	Every doctor must be system user	0012	Pridedamas išorinis raktas lentelėje Doctor su nuoroda į lentelę system_user	DB lentelės išorinis raktas	0012	0009
0012	Every doctor must be system user	0013	Pridedamas lentelės atributas system_user_ID	DB lentelės atributas	0013	0010
0013	Each system user must be related to only one user role	0014	Lentelės user_role su sudėtiniu pirminiu raktu sukūrimas	Lentelė	0014	0011
0013	Each system user must be related to only one user role	0015	Išorinio rakto sukūrimas lentelėje user_roles-nuoroda į kitą lentelę system_user	DB lentelės išorinis raktas	0015	0012
0013	Each system user must be related to only one user role	0016	Išorinio rakto sukūrimas lentelėje user_roles-nuoroda į kitą lentelę user_role	DB lentelės išorinis raktas	0016	0013
0014	It is obligatory that system user has unique system user number	0017	Lentelė system_user su piminiu raktu system_user_ID	Lentelė	0017	0002
0015	It is obligatory that user role has unique user role number	0018	Lentelė user_role su piminiu raktu user_role_ID	Lentelė	0018	0001
0016	It is obligatory that medical test has unique medical test number	0019	Lentelė medical_test su piminiu raktu medical_test_ID	Lentelė	0019	0001
0017	It is obligatory that tester has unique tester number	0020	Lentelė tester su piminiu raktu tester_ID	Lentelė	0020	0001
0018	Patient can have medical test taken on each visit	0023	Papildomas lentelės Visit atributas medical_test	DB lentelės atributas	0021	0014

0018	Patient can have medical test taken on each visit	0024	Išorinio rakto sukūrimas lentelėje Visit-nuoroda į kitą lentelę medical_test	DB lentelės išorinis raktas	0022	0015
0019	Every medical test must be performed by only one tester	0025	Papildomas lentelės medical_test atributas tester_ID	DB lentelės atributas	0023	0016
0019	Every medical test must be performed by only one tester	0026	Išorinio rakto sukūrimas lentelėje medica_test-nuoroda į kitą lentelę tester	DB lentelės išorinis raktas	0024	0017
0020	Every system user must have user name and password	0027	Lentelės system_user sukūrimas su atributais user_name ir password	Lentelė	0025	0007
0021	Every user role must have user role name and role description	0028	Papildomi atributai lentelėje user_role-user_role_name ir role_description	DB lentelės atributas	0026	0008
0022	Every medical test must have test result, test type and result date	0029	Papildomi lentelės medical_test atributai test_result, test_type and result_date	DB lentelės atributas	0027	0008
0023	Every tester must have name and surname	0030	Papildomi lentelės tester atribubtai name ir surname	DB lentelės atributas	0028	0008