VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

Tautvydas Šoblinskas

ONTOLOGIJA GRINDŽIAMŲ VERSLO TAISYKLIŲ TRANSFORMACIJŲ ĮGYVENDINIMAS INFORMACINĖSE SISTEMOSE

(IMPLEMENTATIONS OF ONTOLOGY BASED BUSINESS RULES TRANSFORMATIONS IN THE INFORMATION SYSTEMS)

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P111

Informacinių sistemų specializacija

Informatikos mokslo kryptis

Vilnius, 2009

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas

_____
(Parašas)
prof. habil. dr. P. G. Adomėnas
(Vardas, pavardė)

_____
(Data)

Tautvydas Šoblinskas

ONTOLOGIJA GRINDŽIAMŲ VERSLO TAISYKLIŲ TRANSFORMACIJŲ ĮGYVENDINIMAS INFORMACINĖSE SISTEMOSE

(IMPLEMENTATIONS OF ONTOLOGY BASED BUSINESS RULES TRANSFORMATIONS IN THE INFORMATION SYSTEMS)

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P111

Informacinių sistemų specializacija

Informatikos mokslo kryptis

Vadovas Prof. dr. Olegas Vasilecas _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas Dr. Diana Kalibatienė _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas Doc. dr. Angelė Kaulakienė _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Vilnius, 2009

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

Technologijos mokslai mokslo sritis

Informatikos mokslo kryptis

Informatikos studijų kryptis

Inžinerinės informatikos studijų programa, valstybinis kodas **62409P111**

Informacinių sistemų specializacija

PAŽYMA
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ
…….....Nr. .....
Vilnius

Studentas Tautvydas Šoblinskas
(Vardas, pavardė)

Studento studijų svertinis įvertinimų vidurkis.....….....…...……….....…...balo.

Baigiamojo darbo tema: **Ontologija grindžiamų verslo taisyklių transformacijų įgyvendinimas informacinėse sistemose.**

Baigiamasis darbas peržiūrėtas ir studentui .....……..…….....….....
leidžiama ginti šį baigiamąjį darbą magistro laipsnio suteikimo komisijoje.

Katedros vedėjas .....…..... Prof. habil. dr. Petras Gailutis Adomėnas
(Parašas) (Moksl. laipsnis, vardas, pavardė)

VADOVO ATSILIEPIMAS
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ
……

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....……..
.....
……

Vadovas .....…….....
(Parašas)
Prof. dr. Olegas Vasilecas
(Moksl. laipsnis, vardas, pavardė)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

Technologijos mokslų mokslo sritis

Informatikos mokslo kryptis

Informatikos studijų kryptis

Inžinerinės informatikos studijų programa, valstybinis kodas **62409P111**

Informacinių sistemų specializacija

BAIGIAMOJO MAGISTRO DARBO UŽDUOTIS

…….....Nr. .....
Vilnius

Studentui Tautvydui Šoblinskui
(Vardas, pavardė)

Baigiamojo darbo tema: Ontologija grindžiamų verslo taisyklių transformacijų įgyvendinimas informacinėse sistemose
patvirtinta 200…m. ……..…… d. dekano potvarkiu Nr. …….

Baigiamojo darbo užbaigimo terminas 200…m. ……..…… d.

BAIGIAMOJO DARBO UŽDUOTIS:
PROBLEMA: Sudėtingas informacijos apdorojimo ir modeliavimo taisyklių rinkinio formavimas informacinėms sistemoms kurti.
TIKSLAS: Patobulinti informacijos apdorojimo ir modeliavimo taisyklių rinkinio formavimą informacinėms sistemoms kurti.
UŽDAVINIAI:

1. Atlikti transformacijų sampratos informacinių sistemų kūrimo metu analizę, išnagrinėti transformacijų tipus ir kalbas.
2. Atlikti taisyklių transformacijos metodų analizę.
3. Nustatyti pagrindines problemas, susijusias su verslo taisyklių transformacijomis informacinėms sistemoms kurti.
4. Patobulinti transformacijos metodą, kaip dalykinės srities taisyklės galėtų būti transformuotos ir panaudotos informacinei sistemai kurti.
5. Sukurti transformacijos metodą realizuojančios programų sistemos prototipą ir atlikti testavimą.

Baigiamojo darbo rengimo konsultantai:
Dr. Diana Kalibatienė
Doc. Angelė Kaulakienė
(Moksl. laipsnis, vardas, pavardė)

Vadovas ..... Prof., dr. Olegas Vasilecas
(Parašas) (Moksl. laipsnis, vardas, pavardė)

Užduotį gavau

……..
(Parašas)
Tautvydas Šoblinskas
(Vardas, pavardė)
……..……...
(Data)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

ATSILIEPIMAS
APIE BAIGIAMĄJĮ MAGISTRO DARBĄ

…….....Nr. .....

Vilnius

Studentas Tautvydas Šoblinskas parengė baigiamąjį magistro darbą tema: **Ontologija grindžiamų verslo taisyklių transformacijų įgyvendinimas informacinėse sistemose.**

RECENZIJA

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Magistro laipsnio suteikimo komisijos narys

…… .....

(Parašas) (Moksl. laipsnis, vardas, pavardė)

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių sistemų katedra

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

Inžinerinės informatikos studijų programos baigiamasis magistro darbas

Ontologija grindžiamų verslo taisyklių transformacijų įgyvendinimas informacinėse sistemose

Autorius **Tautvydas Šoblinskas** Vadovas prof. dr. **Olegas Vasilecas**

Kalba: lietuvių

Anotacija

Darbe nagrinėjamas verslo taisyklių panaudojimas ir transformacijos, kurių realizacijomis sprendžiamos problemos, susijusios su verslo taisyklių sistemos formavimu ir modeliavimu.

Darbe buvo aprašyta, kaip verslo taisykles, formaliai aprašytas ontologijoje, galima transformuoti į informacijos apdorojimo ir modeliavimo taisykles. Išnagrinėjus transformacijas aprašančią literatūrą, buvo nustatytos transformacijų kalbos, stiliai, tipai, įvairūs būdai, kaip transformacija gali būti atlikta. Transformacijos metodo realizacijai buvo atlikta verslo taisyklių ir ontologijos analizė. Darbe aprašytas metodas, kaip ontologijoje aprašytos verslo taisyklės, išreikštos aksiomomis, gali būti panaudotos informacinei sistemai formuoti: jos transformuojamos į informacijos apdorojimo taisykles, ADB SQL trigerius ir OCL ribojimus. Buvo atlikta lyginamoji ontologijų kūrimo priemonių analizė, kurios metu pasirinktas Protégé įrankis, kuris panaudotas pademonstruoti metodo efektyvumą ir naudą. Transformacijoms įgyvendinti naudota įrankiu sukurta laikraščio ontologija. Metodo realizacijai buvo sukurti Protégé įrankio plėtiniai „Axiom2SQL“ ir „Axiom2OCL“, kuriais galima atlikti automatizuotą ribojimų aksiomų išgavimą ir transformavimą į SQL trigerių prototipus ar OCL ribojimus.

Darbą sudaro 5 dalys: įvadas, analitinė ir projektinė, išvados, literatūros sąrašas (bei 3 priedai).

Darbo apimtis – 95 p. teksto be priedų, 20 pav., 10 lent., 35 bibliografiniai šaltiniai Atskirai pridedami darbo priedai.

Prasminiai žodžiai: verslo taisyklės, ontologija, informacijos apdorojimo taisyklės, informacinės sistemos formavimas, SQL trigeris, OCL.

Vilnius Gediminas Technical University
Fundamental Science faculty
Information Systems department

ISBN ISSN
Copies No. ……
Date ….-….-….

Engineering Informatics study program Master Thesis. Title**: Implementations of ontology based business rules transformations in the information systems**

Author Tautvydas Šoblinskas Academic supervisor prof. dr. Olegas Vasilecas

Thesis language: Lithuanian

Annotation

This thesis describes usage and transformations to solve problems associated with business rules system development and modeling.

The thesis describes how formally described business rules of the ontology can be transformed into information processing and modeling rules. A result of literature analysis are described transformation languages, styles, types, various ways of transformation how can be made. For described transformation method business rules and ontology were analyzed. Thesis describes method how ontology based business rules, represented as axioms, can be transformed and applied for development of information system: business rules transformed into processing rules, ADBMS SQL triggers, and OCL constraints. After ontology development tools analysis Protégé tool was chosen to demonstrate the efficiency and usefulness of the suggested method. For realization of transformations was used ontology of Newspaper domain. The method is implemented as Protégé plug-ins "Axiom2SQL" and "Axiom2OCL", which automatically performs mining of axioms and transformation into SQL trigger prototypes and OCL constraints.

Structure: 5 parts: introduction, analytic and project, conclusions and references (and 3 appendixes).

Thesis consists of: 95 p. text without appendixes, 20 pictures, 10 tables, 35 bibliographical entries.

Appendixes included.

Keywords: Business rules, ontology, information processing rules, system development, SQL trigger, OCL.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpos

ADBMS SQL – Microsoft SQL Server aktyvi duomenų bazė

DAML (ang. *DARPA Agent Markup Language*) – DARPA agentų žymėjimo kalba

DBVS – duomenų bazių valdymo sistema

ECA (ang. *Event-Condition-Action*) taisyklė – taisyklė, kurios struktūrinės dalys: įvykis, sąlygos, veiksmas

OCL (ang. *Object constraint language*) – deklaratyvi ribojimų kalba, skirta aprašyti taisykles taikomas UML modeliams

OIL (ang. *Ontology Inference Layer*) – ontologijų infrastruktūra semantiniam tinklui

OWL (ang. *Web Ontology Language*) – tinklo ontologijų kalba

PAL (ang. *Protege Axiom Language*) – Protégé aksiomų kalba

RDF (ang.*Resource Description Framework*) – RDF meta duomenų modelis

SQL (ang. *Structured Query Language*) – struktūrizuota užklausų kalba

UML (ang.*Unified Modeling Language*) – vieninga modeliavimo kalba

1. ĮVADAS

Šiuo metu kiekvieno verslo sėkmei svarbus efektyvus informacinės sistemos panaudojimas. Tinkamai panaudojus informacines sistemas taupomi finansiniai ir žmogiškieji ištekliai, sprendžiamos verslui aktualios problemos bei gaunami siekiami verslo procesų rezultatai, kurie užtikriną efektyvų verslo darbą ir plėtrą. Kiekvieno verslo informacinėje sistemoje naudojamos verslo taisyklės, kurios valdo įmonės verslo procesus.

Verslo taisyklėmis apibrėžiamas ar kontroliuojamas tam tikras verslo aspektas. Taisyklės padeda įgyvendinti verslo tikslus ir uždavinius. Tam, kad būtų užtikrintas informacinių sistemų palaikymas, pradėtos kurti žiniomis pagrįstos informacinės sistemos. Jei tai bus ontologijos pagrindu veikianti informacinė sistema, joje aprašoma dalykinės srities ontologija. Taikant dalykinės srities taisyklių transformacijos metodą, pagrįstą verslo taisyklių išgavimu iš ontologijų informacinių sistemų kūrimo metu galima ne tik sugebėti panaudoti ir užrašyti esminę konkrečios verslo srities logiką, bet ir kurti vykdomus objektus tiesiogiai remiantis atliktais verslo taisyklių apibrėžimais.

Dalykinei sričiai atvaizduoti informacinėje sistemų terpėje galima panaudoti ontologijos kūrimo įrankius. Kiekvienoje dalykinė srityje egzistuoja tam tikros verslo taisyklės, todėl jas galima išgauti iš dalykinės srities ontologijos. Ontologija apima nagrinėjamos dalykinės srities žinias, o jos koncepcinis modelis gali būti panaudotas dalykinei sričiai modeliuoti.

Ontologija kūrėjui teikia galimybę pakartotinai panaudoti ir dalintis srities žiniomis, vartojant tą patį žodyną įvairiose programinės įrangos platformose. Sistemos kūrėjas taip pat gali nesijaudinti dėl įgyvendinimo detalių, o koncentruotis į nagrinėjamos srities struktūrą ir užduotis. Ontologija pagrįstos informacinės sistemos naudojimas užtikrins, kad informacinė sistema būtų lanksti ir lengvai pritaikoma prie verslo pokyčių, todėl tenkins dinamiško verslo poreikius.

Taisykles, atvaizduotas ontologijoje ir išreikštas ribojančiomis aksiomomis, galima transformuoti į formalią sintaksę, o turint verslo taisykles, išreikštas tinkama sintakse galima, jas taikyti duomenų bazėse (Šoblinskas, Būgaitė 2008). Tai atlikti rankiniu būdu pakankamai sudėtinga, todėl informacijos apdorojimo ir modeliavimo taisyklių rinkinio formavimą kiek įmanoma labiau automatizavus būtų taupomos programų sistemos kūrimo lėšos.

1.1. Tyrimo objektas

Darbe nagrinėjamas dalykinės srities taisyklių, išreikštų dalykinės srities ontologijos aksiomomis, transformacijų panaudojimas informacinių sistemų kūrimo metu.

1.2. Tikslas ir uždaviniai

Tikslas – patobulinti informacijos apdorojimo ir modeliavimo taisyklių rinkinio formavimą informacinėms sistemoms kurti.

Uždaviniai:

1. Atlikti transformacijų sampratos informacinių sistemų kūrimo metu analizę, išnagrinėti transformacijų tipus ir kalbas.
2. Atlikti taisyklių transformacijos metodų analizę.
3. Nustatyti pagrindines problemas, susijusias su verslo taisyklių transformacijomis informacinėms sistemoms kurti.
4. Patobulinti transformacijos metodą, kaip dalykinės srities taisyklės galėtų būti transformuotos ir panaudotos informacinei sistemai kurti.
5. Sukurti transformacijos metodą realizuojančios programų sistemos prototipą ir atlikti testavimą.

1.3. Temos naujumas

Darbo naujumas – dalykinės srities ontologijos aksiomų išgavimo metodo patobulinimas ir išgautų verslo taisyklių panaudojimas informacijos apdorojimo taisyklėms formuoti. Ontologijų kūrimo įrankiui – sukurti plėtiniai, kuriuo galima išgauti ir lengviau formuoti informacijos apdorojimo ir modeliavimo taisykles.

1.4. Temos aktualumas

Kuriant informacinėms sistemas, grindžiamas ontologijomis, siekiama kuo didesnio sistemos kokybiškumo, aiškumo ir suderinamumo. Verslui reikalinga tokia informacinė sistema, kuri yra lanksti ir gali lengvai prisitaikyti prie kintančių verslo poreikių. Dalykinės srities ontologijoje apibrėžiamos verslo taisyklės, kurias išgavus iš ontologijos galima panaudoti formuoti informacijos apdorojimo taisykles. Darbe aprašoma, kaip gali būti atliktas dalykinės srities taisyklių, ontologijoje užrašytų aksiomomis, išgavimas ir transformavimas į informacijos apdorojimo taisyklių rinkinį, kurį vėliau galima panaudoti informacinės sistemos kūrimui. Atliekant taisyklių formavimą, transformaciją iš vienos kalbos į kitą automatizuotai išvengiama žmogiškų klaidų ir skirtingo taisyklių interpretavimo.

1.5. Tyrimo metodika

Analitinėje darbo dalyje atliktas bibliotekinis tyrimas, apimantis transformacijas, verslo taisykles ir ontologijas. Naudojant lyginamąjį metodą atrinktas tinkamiausias ontologijų kūrimo įrankis.

1.6. Mokslinė darbo vertė

Darbe yra išanalizuotos transformacijos, jų metodai, tipai, verslo taisyklės ir jų aspektai, ontologijos, atlikta įrankių analizė, išrinktas darbui tinkamiausias. Patobulintas informacijos apdorojimų taisyklių formavimas atlikus pasirinkto metodo patobulinimą – dvi realizacijas. Realizacijomis atliktos transformacijos iš verslo taisyklių aprašytų PAL aksiomomis ontologijoje į informacijos apdorojimo (ir modeliavimo) taisykles SQL ir OCL.

1.7. Darbo rezultatai

- Atlikus transformacijos sąvokos ir jų tipų analizę, nustatyta, kad verslo taisyklėmis grindžiamą sistemą galima kurti net tada, kai galutinės įgyvendinimo sąlygos nėra aiškios.
- Nustatyta, kad metodo realizacijos transformacijai galima panaudoti atvaizdavimo transformacijos tipą.
- Buvo nustatyta, kad Protege ontologijos aksiomų struktūra didžiąja dalimi atitinka SQL vykdomųjų taisyklių struktūrą. PAL-SQL transformacijos realizacija įrodo, kad įmanoma atlikti transformaciją, kuri palengvintų sistemos kūrimą.
- Gauti Protege ontologijos aksiomų transformacijos rezultatai įrodo, kad įmanoma atvaizduoti į OCL ribojimus.
- Palyginus ontologijos aksiomas ir verslo taisyklės, išreikštas ECA taisyklių forma gauta, kad aksiomose yra apibrėžiamos sąlygos ir/arba veiksmai, kurie gali būti transformuoti į ECA taisyklės sąlygas ir veiksmus naudojant įvairias realizacijas.
- Sukurtos dvi transformacijos metodo realizacijos. Pirmoji realizacija atlieka ontologijos aksiomų transformaciją į informacijos apdorojimo taisykles, SQL trigerių prototipus. Antroji realizacija atlieka ontologijos aksiomų transformaciją į informacijos modeliavimo ir apdorojimo taisykles, išreikštas OCL ribojimais.

1.8. Darbo struktūra

Darbą sudaro įvadas, analitinė ir projektinė dalys, išvados, literatūros sąrašas. Analitinėje dalyje nagrinėjamos ir analizuojamos: transformacijos, jų tipai, metodai, verslo taisyklės, jų aspektai, naudojimas ir tam reikalingos sąlygos; ontologijos sąvoka, struktūra, naudojimas. Atliekama ontologijų kūrimo įrankių analizė. Aprašomas metodo realizacijai naudojamas ontologijų kūrimo įrankis Protege. Analitinės dalies pabaigoje pateiktas dalies apibendrinimas. Projektinę dalį sudaro metodo ir jo transformacijų ir jų realizacijų aprašymai, testavimai, gauti rezultatai bei dalies apibendrinimas.

1.9. Darbo aprobacija

Buvo publikuotas su darbu susijęs straipsnis, aprašantis transformacijos metodą ir jo realizaciją, – T.Šoblinskas, D. Būgaitė. „Verslo taisyklių transformacijos kuriant informacines sistemas.“ 11-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“ medžiaga. 2008, p. 79-86.

Dalyvaujama aukštųjų technologijų plėtros programos projekte “Veiklos taisyklių sprendimai informacinių sistemų kūrimui (VETIS)”, kuriame naudojama darbo medžiaga.

2. ANALITINĖ DALIS

2.1. Transformacijų samprata informacinių sistemų kūrimo metu

2.1.1. Transformacijos apibrėžimas

Transformacija – objekto formos ar kokybės keitimas. Tai galima apibūdinti kaip transformacija iš vienos pradinės kalbos, aplinkos ar modelio į siekiamą, taip gaunama skirtingų sistemų integracija.

Palyginimui galima panaudoti vertimą iš vienos kalbos į kitą, kuriam reikalingas tam tikras minimalus sintaksės transformacijos taisyklių rinkinys, kuriuo atliekamas lygiagretus palyginimas ir apibrėžiamas kalbos elementų transformavimas (vertimas) (Galley, Hopkins et al. 2003).

Automatinis transformavimas (vertimas) iš vienos kalbos į kitą atliekamas naudojant tam tikrą lingvistiškai pagrįstą išrikiuotų pradinės kalbos žodžių medį. Transformavimas arba vertimas, pvz., iš anglų kalbos į prancūzų, yra pakankamai lengvas kai kalbos, objektai struktūriškai artimi, tačiau verčiant iš anglų į lietuvių kalbą atsiranda alternatyvų pasirinkimo problemos, frazių vertimo ir kitos problemos (Galley, Hopkins et al. 2003). Todėl tinkamai transformacijai kaip ir bet kokiam vertimui reikalingas 1 su 1 dvikryptis ryšys, tada bet koks semantinis kalbos vertimas į kitą kalbą bus nuoseklus, kai transformacija bus atliekama iš vieno žinomo objekto į kitą. Naudojant vienareikšmišką dvikryptį ryšį taip pat užtikrinamas grįžtamasis ryšys.

Tai užtikrinama atlikus transformuojamos kalbos ar modelio semantikos analizę, išnagrinėjus tarpusavio objektų ryšius ir jų atvaizdus siekiamoje kalboje ar modelyje. Kiekviena transformacija reikalauja aiškaus sintaksės ir semantikos supratimo tiek pradinio šaltinio (kalbos ar modelio), tiek siekiamos tikslo (kalbos ar modelio). Tai gali būti pasiekta naudojant metamodelio technikas, kuriomis galima apibrėžti abstrakčios sintaksės modelius ir ryšius tarp modelių elementų bei modelių transformacijų taisyklių. Taip automatiniais transformavimo įrankiais pasiekiamas aukštesnis automatizacijos lygmuo ir tai įgyvendinama greitai ir efektyviai, be to, padeda pagerinti programinės įrangos kūrėjų produktyvumą ir sistemos kokybę (Gomez-Perez, Gernandez-Lopez, Corcho 2004).

Transformacija atliekama naudojant transformacijos taisyklių rinkinį, kuri sudaro galimų transformacijų atvejų aibė. Ši aibė aprašoma tam tikroje bibliotekoje, tačiau pačios transformacijos gali būti skirtingų tipų arba formų. Tačiau transformacijų metu naudojamas panašus principas, kai transformacijos taisyklėmis aprašoma, kaip kiekvienas pradinės modelio (kalbos) elementas yra adaptuojamas (pakeistas, ištrintas, sukurtas). Modeliuotojas (arba programų sistema) pasirenka tinkamas taisykles iš taisyklių bibliotekos ir nustato jų veikimo seką tai atliekama remiantis

turimomis žiniomis apie transformuojamą objektą (modelį) (Chisholm 2004) bei žinomus būsimo modelio kriterijus.

Naudojant transformacijos taisykles, galima transformuoti verslą palaikančių informacinių sistemų komponentą (formaliai užrašytas) verslo taisykles į informacijos apdorojimo taisyklės, kurios galėtų būti naudojamas kuriant sistemas. Tam, kad užtikrintume informacinių sistemų palaikymą, pradėtos kurti žiniomis grindžiamos informacinės sistemos. Vienas iš būdų vaizduoti žinias yra dalykinės srities ontologijos. Ontologijose dalykinės srities žinios yra atvaizduojamos dalykinės srities terminais, ryšiais tarp jų ir aksiomomis, kurios apibrėžia arba riboja terminų interpretaciją. Ontologijoms kurti yra naudojami tinkami ontologijų kūrimo ir palaikymo įrankiai.

Dalykinės srities žinių dalį – verslo taisykles, – vaizduojamą ontologijos aksiomomis, galima transformuoti į duomenų bazių ribojimus (Šoblinskas, Būgaitė 2008). Detaliau tai bus aptarta po transformacijų, verslo taisyklių ir ontologijų analizės.

2.1.2. Transformacijų formos

Po transformacijų analizės buvo išskirtos tokios transformacijos formos (arba tipai):

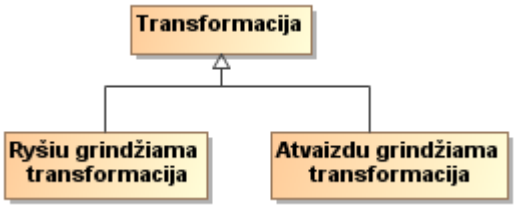
- modelio transformacija,
- konteksto transformacija,
- duomenų transformacija.

Iš šių trijų išskirtų formų kiek plačiau aptariama modelio ir duomenų transformacija, kadangi jos aktualiausios būsimai transformacijai.

2.1.3. Modelio transformacija

Be transformacijų tipų, galima išskirti ir jų taikomus stilius, kadangi bendra transformacijų modelio struktūra yra tokia, kad būtų galima naudoti kelis skirtingus transformavimo stilius vienam tipui (Appukuttan, Clark, Reddy, Tratt, Venkatesh 2003). Pagal transformacijos modelio struktūrą transformavimo stiliai gali kisti sistemos gyvavimo cikle. Galima išskirti į du transformacijos modelio struktūros tipus: naudojant ryšius ir naudojant atvaizdus.

Transformacija yra klasė, apibendrinanti ryšį ir atvaizdą. Kai kalbama apie transformacijos modelį, turimas omenyje ryšys arba atvaizdas (jų transformacijas).



1 pav. Transformacijos klasė, apibendrinanti ryšio ir atvaizdo transformacijas (Appukuttan, Clark,

Reddy, Tratt, Venkatesh 2003).

Ryšiai yra daugiadimenės transformacijų specifikacijos. Ryšiai nėra vykdomi, kdangi jie negali kurti ar pakeisti modelio, tačiau gali patikrinti dviejų modelių tarpusavio neprieštaringumą. Tradiciškai ryšiai naudojami sistemos kūrimo specifikavimo stadijose ar patikrinti atvaizdo teisingumą.

Atvaizdai yra transformacijų įgyvendinimai. Skirtingai nei ryšiai, atvaizdai yra potencialiai vienkrypčiai ir gali turėti grįžtamąsias reikšmes. Atvaizdais galima pakeisti bet kokį kiekį ryšių, tačiau atvaizdas turi atitikti pakeičiamus ryšius.

Ryšių ir atvaizdų transformacijų identifikavimas ir atskyrimas vieno nuo kito, teikia modelio kūrėjui keletą privalumų. Modeliuotojas tada gali (Appukuttan, Clark, Reddy, Tratt, Venkatesh 2003):

- naudoti ryšius pradinėse sistemos kūrimo stadijose, kai įgyvendinimo sąlygos dar nėra galutinai aiškios (tai leidžia konkretizuoti transformacijas prieš nusprendžiant įgyvendinimo detales);
- sukurti daug atvaizdų, kurie padaryti naudojant visiškai skirtingas programavimo kalbas, ir atitinka ryšius;
- sukonstruoti ryšį iš jau egzistuojančio atvaizdo ir išgauti detales, kurios galėtų vėliau būti prarastos mišinyje.

2.1.4. Duomenų transformacija

Duomenų ir duomenų bazių požiūriu transformavimas viena iš esminių duomenų atvaizdų kūrimo operacijų. Duomenų transformavimas yra procesas, kurio metu sukuriami atitikmenys tarp įrašų (objektų) ir šaltinio schemos atributų dažniausiai skirtingiems įrašams ir atributams tikslo schemai. Duomenų transformavimo pavyzdys pristatymo ir sąskaitų adresų informaciją iš pirkimo užsakymo pavaizduoti į sąskaitą faktūrą. Duomenų transformacija taip pat gali apimti tokias operacijas kaip (BizTalk Server Developer Center 2004):

- duomenų vidurkio radimas ir įdėjimas į konkretų atributą,
- vieno tipo duomenų formato konvertavimas į kito tipo formatą (pvz., tekstinio tipo duomenų transformacija į ASCII formatą),
- duomenų pridėjimas ar atėmimas iš vieno ar daugiau įrašų ir jų užrašymas viename tikslo schemos atribute.

Duomenų transformavimas apima iš šaltinio paimtų duomenų apdorojimą ir išsaugojimą tikslo schemos atribute, duomenų konvertavimą iš vieno tipo apdorotų šaltinių duomenų užrašymą tikslo schemos atribute.

2.1.5. Transformavimo kalbos

Tam, kad būtų galima suprasti taikomas transformacijas ir mechanizmus, būtina aptarti konkrečias transformavimui vartojamas kalbas.

Viena iš transformavimo kalbų yra Atlaso transformavimo kalba (ang. *Atlas Transformation Language* (toliau – ATL)). ATL turi deklaratyviąją dalį, kuri grindžiama sutapimų taisyklėmis ir jų žymėjimo. Tokia taisyklė sudaroma šaltinio struktūrai ir siekiamai struktūrai, kurioje aprašomas (sukuriamas) kiekvienas modelių sutapimas. Atsekamumo nuorodos yra sukuriamos automatiškai. Taikomas taisyklių paveldėjimas ir polimorfiškos taisyklių nuorodos. Žvalga vykdoma naudojant OCL išraiškas. ATL siūlo naudoti du imperatyvius konstruktus: vadinamus iškviečiamąja taisykle ir veiksmų bloku. Iškviečiamoji taisyklė yra iškviečiama nurodžius detales, panašiai kaip procedūra, tačiau jos pagrindas sudarytas iš deklaratyvios siekiamos struktūros. Sutampančios ir iškviečiamosios taisyklės gali būti naudojamos kartu vienoje transformacijos programoje. Veiksmų blokai yra nuoseklios imperatyvios instrukcijos, kurios gali būti panaudotos sutampančioms arba iškviečiamosioms taisyklėms nustatyti. Rekomenduojamas stilius yra deklaratyvus (t.y. nenaudojamos iškviečiamosios taisyklės ir veiksmų blokai). Imperatyvus stilius turėtų būti naudojamas tik tada, kai deklaratyvios kalbos konstruktai teikia galimybes reikalingas konkrečiam atvejui.

Transformacijos programos parašytos ATL yra nepaveldimos tiesiogiai. Šaltinių modelius tik skaitomi, o tikslo modeliai yra tik rašomi. Abu modelių tipai turi būti aiškiai identifikuojami kūrimo metu. Egzistuoja du būdai, kuriais ATL deklaratyvioji programos dalis gali veikti: standartinė ir įmantresnė (ang. *refining*). Standartiniu būdu elementai yra tik sukuriami, kai taisyklė sutampa. Tačiau modeliai negali būti transformuojami tuo pat metu (šaltinio modelius galima tik skaityti), o transformacijos tik modifikuoja mažas modelio dalis ir palieka visa kita nepakeista, šiuo būdu sudėtinga rašyti. Turi būti bent viena kopijavimo (perrašymo) taisyklė kiekvienam tipui, kuris apibrėžtas metamodelyje. To nereikalaujama įmantresniu būdu, kurio metu tie elementai, kurių atitikmenys nerandami, automatiškai perkopijuomi. Dažniausiai kūrėjai keičia šaltinio modelį, teigdami, kad jie iš tikro modifikuoja šaltinio modelį skirtingai, o kiekviena žvalgos išraiška veikia originalų šaltinio modelį (Jouault, Kurtevr 2005).

Kita kalba (ir tuo pačiu metamodelis) yra QVT, kurios metamodelį sudaro 3 transformacijos kalbos. Pati QVT yra apibrėžiama kaip MOF 2.0 metamodelis (kuris bus aptartas vėliau). OCL 2.0 naudojamas užklausoms atlikti. Minėtos trys QVT kalbos sudaro vieną hibridinę transformacijos kalbą, kurioje naudojami deklaratyvūs ir imperatyvūs konstruktai. Kalbos vadinamos ryšiai *(Relations)*, branduolys (*Core*) ir veiksmų atvaizdai *(Operational Mappings)*. Šios kalbos sudaro sluoksninę struktūrą. Ryšių ir branduolio kalbos yra deklaratyviosios dviejuose skirtinguose abstrakcijos

lygmenyse. Specifikacijos dokumentas apibrėžia konkrečią tekstinę sintaksę ir abstrakčią sintaksę. Veiksmų branduolio kalba yra imperatyvinė kalba, kuri praplečia ryšių ir branduolio kalbas.

Ryšių kalba teikia galimybes specifikuoti transformacijas kaip ryšių rinkinius modeliuose. Ryšiai sudaryti iš objektų struktūrų rinkinių. Šie rinkiniai gali būti sulyginti su egzistuojančiais modeliais. Kalba automatiškai tvarko atsekamumo manipuliacijas ir paslepia detales nuo kūrėjo.

Branduolio kalba, priešinga nei ryšių kalba, yra deklaratyvi. Transformavimo apibrėžtys išlieka ilgiau nei analogiški aprašymai ryšių kalba. Atsekamumo nuorodos dažniausiai laikomos modelio elementais. Kūrėjas yra atsakingas už nuorodų sukūrimą ir naudojimą.

Vienas iš branduolio kalbos tikslų yra teikti pamatinę ryšių kalbai dalį. Tai atliekama aprašius ryšių kalbos semantiką ir pateikiama kaip ryšių į branduolio transformacija. Transformacija gali būti užrašyta ryšių kalba. Tačiau kartais sprendžiamai transformacijos problemai sudėtinga pateikti išsamų deklaratyvų sprendimą. Todėl QVT siūlomi du mechanizmai, praplečiantys deklaratyvias ryšių ir branduolio kalbas: trečioji kalba, tai operacijų atvaizdai bei įtrauktas transformacijos funkcionalumas įgyvendinimas papildomoje kalboje (Juodos dėžės principu).

Ryšių kalbą praplečia operacijų atvaizdų kalba imperatyviais konstruktais ir OCL konstruktais su tam tikrais pašaliniais efektais. Ši idėja grindžiama tuo, kad naudojant imperatyvius konstruktus objekto struktūrai, kuri nurodoma ryšyje, sukuriamas atitinkamas egzempliorius. Tokiu būdu deklaratyviai aprašyti ryšiai yra imperatyviai įgyvendinti kalboje. Operacijų atvaizdų kalbos sintaksė teikia konstruktus, kurie dažniausiai naudojami imperatyvinėse kalbose (ciklai, sąlygos ir t.t.). Juodosios dėžės mechanizmas leidžia transformacijos metu prijungti išorinį vykdomąjį kodą. Šis mechanizmas leidžia sudėtingus algoritmus įgyvendinti bet kokioje kalboje ir pakartotinai naudoti egzistuojančias bibliotekas. Tačiau gali kilti nenumatytų rezultatų, kadangi pačios transformacijos nevykdys ir nekontroliuos transformacijos mašina.

2.1.6. Transformacijų metodai

2.1.6.1. Metamodeliais grindžiamos transformacijos

Programinės įrangos inžinerijoje metamodeliavimas yra dažnai naudojama technika, kuri leidžia aprašyti pagrindines abstrakcijas, apibrėžti modelius ir jų ryšius (Neil, Pons 2004). Tam gali būti naudojama *Meta Object Facility* darbo aplinka, palaikanti įvairius metaduomenų tipus ir aprašanti skirtingus informacijos modelius. Šios ypatybės leidžia projektuotojams apibrėžti skirtingus pradinio modelio požiūrius ir detalumus. Šiame kontekste MOF yra metametamodelis, kadangi jis naudojamas kitiems metamodeliams, kaip UML kalba, apibrėžti. MOF naudojama apibrėžti modelių struktūrą ir semantiką specifinėms ir bendroms dalykinėms sritims. MOF yra objektiškai orientuotas modelis, kuris

taip pat tinkamas apibrėžti objektiškai orientuotus ar bendrus modelius. MOF gali būti panaudota specifiniams modelių transformavimo metamodeliams apibrėžti (A. D'Ambrogio 2005).

Transformavimo metodiką sudaro algoritmo taikymas, kuris turi laikiną esybių-ryšių modelį (tai yra įeigą) ir laikiną daugiadimensį modelį (tai yra išeiga). Rekursiniam algoritmui taikyti galima esybių-ryšių diagramą paversti esybių-ryšių modelį (Neil, Pons 2004). Atributai, turintys daug reikšmių, pavirs į silpnas esybes su laikinais ryšiais ir laikinas ryšys pavirs į esybę, turinčią dvigubą ryšį, kuris yra susijęs su dalyvaujančiomis esybėmis. Norint išsaugoti būsimą hierarchiją, siūloma išlaikyti abu ryšius (esamąjį ir laikinąjį).

Transformacijos išeigos daugiadimensė schema sudaryta iš faktų, matavimų, dimensijų ir hierarchijų. Faktu išreiškiamas domėjimosi taškas, pagal kurį atliekamas sprendimo priėmimo procesas, bei modeliuojami įvykiai, kurie įvyko nagrinėjamoje srityje, pvz., kompanijoje. Matavimai yra atributai, kurie išreiškia skirtingus požiūrius. Dimensija nustatoma pagal suskaidymo/skirtingumo mąstą atvaizduojamiems faktams, o hierarchijos dimensijose nustato kaip egzemplioriai gali būti agreguoti ir pasirinkti sprendimo priėmimo procesui. Laikinas daugiadimensiškumas apima laikinos schemos, kuri nepriklauso hierarchijai dimensijose, pagrindinių faktų analizę. Schemos registruos tam tikrų ryšių atributų variacijas, kurios kinta su laiku. Tai atlikus gaunama koncepcinė schema, viename modelyje apjungianti daugiadimensiškumą ir laikiną schemą, naudojama registruoti ir analizuoti laikinas variacijas bei užklausas.

2.1.6.2. Metamodelio nauda

Kiekviena transformacija reikalauja aiškaus sintaksės ir semantikos supratimo, tai gali būti pasiekiama naudojant metamodelio technikas, kurios naudojamos apibrėžti abstrakčios sintaksės modelius ir ryšius tarp modelių elementų ir modelių transformacijų taisyklių. Automatiniais transformavimo įrankiais galima pasiekti aukštesnį automatizacijos lygmenį, kuris yra greitas ir efektyvus, tokiu būdu padidinamas programinės įrangos kūrėjų produktyvumas ir sistemos kokybė (Gomez-Perez, Gernandez-Lopez, Corcho 2004).

Žvelgiant vien tik iš transformacijos perspektyvos ryšys yra dvikryptė transformacijos specifikacija, kuri nurodo susijusius šaltinio ir tikslo metamodelio elementus. Tačiau jis nėra vykdomas, kai tuo tarpu atvaizdas yra nekryptingos transformacijos įvykdymas, kuris naudojamas kurti ar keisti modelius. Atvaizdas gali būti vieno ar daugiau ryšių papildymas. Tuo atveju atvaizdas turi derėti su papildomais ryšiais (apibūdinamais) (Gomez-Perez, Gernandez-Lopez, Corcho 2004).

2.1.7. Schema grindžiamos transformacijos

Bene paprasčiausia atlikti duomenų transformaciją iš vieno formato į kitą, tam sukūrus atitinkamą taikomąją programą. Tai dažniausiai ne triviali užduotis, kurią dažnai komplikuoja įvairūs

techniniai aspektai, susiję su specifiniais duomenų šaltiniais, kurie gali pilnai neatitikti konvertavimo proceso. Tai išsprendžiama panaudojus bendresnę struktūrą, kurį leidžia lanksčiau atlikti transformaciją tarp įvairių modelių. Šaltinio ir būsimi duomenys atvaizduojami naudojant įprastą duomenų modelį ir pervedimo (transformacijos) kalbą. Transformacijos kalba leidžia atlikti transformacijos užduoties specifikaciją ir derinimą. Tokiu būdu naujos transformacijos lengviau „prigis“, tačiau tai gali pareikalauti daugiau programavimo darbo siekiant apibrėžti naują transformaciją.

Dažnai siekiama sukurti tokį mechanizmą, kuris supaprastintų transformacijos specifikavimą (Murshed, Singh 2005). Buvo nustatyta, kad daugumos šaltinio duomenų struktūra yra panaši, į tas kurios gaunamos po (duomenų) transformacijos, bei dauguma transliuojamos struktūros pakitimų atliekamų transformavimo procese yra standartiniai bei priklauso tik nuo schemų skirtumų. Schemos terminas naudojamas apibūdinti duomenų modelio naudojimą modeliuoti jo duomenis. Pavyzdžiui, duomenų bazės naudoja schemas modeliuoti duomenų bazių egzempliorius; struktūrizuoti dokumentai dažnai turi ta pačią gramatiką (pvz., HTML), kituose modeliuose šie apibrėžimai gali būti taikomi tik dalinai (pvz., pusiau struktūrizuoti duomenys). Todėl, kad dauguma transformacijų, siekiamos gauti sistemos schema yra artimai susijusi su šaltinio sistemos schema, kadangi abi schemos siekia išreikšti tuos pačius duomenis. Todėl galima daryti išvadą, kad nemaža dalis vertimo (naudojant dvi schemas) iš vieno formato į kitą (transformacijos) gali būti padaryta automatiškai, grindžiant (dažnai standartiniais) ryšiais, tokiu būdu transliavimo sistemos naudotojui užtenka tik nurodyti nestandartines transformacijos dalis.

Todėl pakankamai nesunku sukurti transformavimo sistemą, kuri įgyvendintų šią idėją. Transformavimo sistemai duodamos šaltino ir siekiamoji tikslo schema. Sistema nustato skirtumus tarp schemų. Tai atliekama remiantis taisyklėmis pagrįstu metodu, kur kiekviena taisyklės apibrėžtų galimus sutapimus tarp dviejų schemų komponentų. Tada būtų sukurta taisyklė-aprašymas kaip vienos schemos egzempliorius, kuris būdų transformuotas į kitą schemą. Sistema naudodama taisykles kiekviena kartą stengiasi nustatyti unikalų labiausiai tinkantį komponentą tikslo schemoje ar nustatyti komponentą, kurio neturėtų likti siekiamoje tikslo schemoje. Jei procesas vyksta sėkmingai, duomenų transformacija gali būti atliktas automatiškai naudojant atrankos (atitikimo) taisyklių biblioteką. Tačiau egzistuoja du atvejai, kai procesas gali vykti nesėkmingai. Pirmasis atvejis, kai šaltinio schemos komponentas negali būti nustatytas tikslo schemoje, tačiau ir atitikimo procesą vykdančios taisyklės negali nustatyti, kad jis turėtų būti praleistas. Antras atvejis, kai šaltinio schemos komponentas atitinka kelis komponentus tikslo schemoje, o sistema negali nustatyti geriausio atitikmens tikslo schemoje. Pirmuoju atveju sistemos naudotojai gali pridėti atitinkamų taisyklių, kad sistema veiktų tinkamai specifiniu atveju. Antruoju atveju sprendimas būtų, kad sistema užklaustų naudotojo kuris iš variantų tinkamesnis. (Milo, Zohar 1998)

2.2. Verslo taisyklių transformacijos

2.2.1. Verslo taisyklės sąvoka

Taisyklių transformacijai buvo pasirinktos verslo taisyklės, todėl pirmiausia apibrėškime verslo taisykles ir kodėl naudinga naudoti būtent verslo taisyklių metodą.

Verslo taisyklė – sakinys, apibrėžiantis ar ribojantis tam tikrą verslo aspektą (Business Rules Group 2000). Verslo taisyklės organizacijoje naudojamos apibrėžti politiką, direktyvas ar procedūras, numatančia atitinkamus sprendimus, lemiančius verslo veiklą. Jas galima nusakyti kaip teiginius ar ribojimo sakinius (toliau ribojimus), suvaržančius tam tikrą verslo sritį. Verslo taisyklė išreiškia gryną verslo logiką (Ross 2003: 185), apibrėžia verslo struktūrą ir kontroliuoja įmonės verslo procesus. Verslo taisyklės įvedimu siekiama labiau kontroliuoti ir įtakoti verslą, jo duomenis. Verslo taisyklė apibrėžia verslo struktūrą ir kontroliuoja įmonės verslo procesus. Jos gali kilti iš išorinių šaltinių – priimti nutarimai vyriausybės ar kitos organizacijos, kuriai priklauso įmonė, tiek iš pačios organizacijos. Trumpai apžvelkime verslo taisyklių sakinių kategorijas bei kas yra formalios taisyklės. Galima apibrėžti tokias verslo taisyklių sakinių kategorijas (Business Rules Group 2000):

Verslo terminų apibrėžimas. Pagrindinis verslo taisyklės elementas – kalba, kuria nusakoma taisyklė. Pats terminas yra verslo taisyklė, aprašanti kaip žmonės turi suvokti dalykus.

Faktai, susiję su terminais ir tarpusavyje. Veikiančios organizacijos struktūra gali būti aprašyta faktais, kurie susieja terminus tarpusavyje. Pvz., verslo taisyklė – klientas gali užsakyti prekes. Faktai gali būti įforminti (dokumentuoti) aprašančia kalba ar ryšiais, atributais ir apibendrintomis struktūromis, kurios grafiškai išreikštos modeliu.

Ribojimai. Kiekvieno verslo ribojimai interpretuojami panašiai ir tai artimai susieta su duomenų ribojimais, kur nurodoma kaip jie gali būti keičiami ir ar išviso gali būti keičiami.

Išvedimai. Verslo taisyklė apibrėžiama kaip žinios, turinčios tam tikrą formą, kurios gali būti transformuotos į kitas žinias.

Be minėtų kategorijų verslo taisyklės gali būti formalios ir neformalios. Formali verslo taisyklė – verslo taisyklės išraiška, kurioje naudojama specifinė formali gramatika. Kiekviena verslo taisyklė gali būti išreikšta vienu ar keliais formalių taisyklių sakiniais, tačiau kiekvienas formalios taisyklės sakinys turi būti daugiau nedaloma verslo taisyklė. Formalios taisyklės sakinys turi priklausyti įsigalėjusiam formalios išraiškos tipui. Formalios išraiškos tipai, turintys savo struktūra, yra English, IDEF1X, Oracle CASE*Method, Object Role Modeling (Business Rules Group 2000) ir kiti.

2.2.2. Požiūris į verslo taisykles

Verslo taisykles galima nagrinėti:

- verslo darbuotojų požiuriu,

- informacinės sistemos kūrėjų požiūriu.

Pirmu atveju, verslo darbuotojų (verslo sistemos lygmenyje) verslo taisyklės yra sakiniai, išreiškiantis verslo politiką deklaratyviu būdu (Business Rules Group 2000). Naudojamos taisyklės yra susijusios su versle dalyvaujančių žmonių veikla. Antru atveju, informacinių sistemų lygmenyje verslo taisyklės yra sakiniai, apibrėžiantys informacijos apdorojimo taisykles (Business Rules Group 2000) ir naudojantys formalią kalbą.

Antru atveju, IS kūrėjų požiūriu (IS lygmenyje) naudojamos taisyklės yra susijusios su reikalingų faktų užrašymu duomenų pavidalu ir jų reikšmių keitimu. Tai daugiausiai klausimai, susiję su darbu naudojant duomenis, juos įrašant, keičiant ar kitaip apdorojant informacinėje sistemoje. Verslo taisyklė gali apibrėžti kada ir kaip atlikti duomenų įrašų kūrimą, atnaujinimą ar pašalinimą informacinėje sistemoje. Paprastas pavyzdys, pirkimo įrašas negali būti įvestas, jei klientas neturi atitinkamos pinigų sumos ir nesumokėjo už prekę. Tai, pavyzdžiui, apibrėžia taisyklė, teigianti: „Negalima atlikti prekės pirkimo, kol nesumokėti pinigai“.

Pirmiausia, žiūrint tik iš informacinės sistemos pusės galima lengvai pastebėti problemą. Tampa įmanoma lengvai suprasti ir modeliuoti verslo taisykles, išreiškiamas kaip duomenų ribojimus (Business Rules Group 2000). Tai daug sunkiau padaryti žiūrint iš verslo pozicijos.

Antra, išvengiama uždavinių, kurie kiltų nagrinėjant žmonių veiklą organizacijoje. Į šią kategoriją patenka:

- taisyklių taikymas žmonėms;
- atsižvelgimas į sistemos komponentus bei jų tarpusavio sąveiką palaikant racionalų verslą;
- darbo eigos ir procesų ryšių su verslo taisyklėmis vaizdavimas;
- taisyklių realizavimas, palaikymas, pritaikymas;
- nustatymas kada taisyklė tampa nereikalinga.

Kūrėjų komandos tikslu tampa projekto vėlesnių uždavinių ir įtakojančių veiksmų analizė bei įgyvendinimas (Business Rules Group 2000).

Taip pat iš informacinių sistemų požiūrio verslo taisyklės gali būti apibrėžtos kaip dažnai besikeičiantis informacinių technologijų įgyvendinimo komponentas. Jo įgyvendinimas pagrįstas faktu, kad verslo taisyklės gali dažnai keistis, o viena iš praktikų yra jas susieti su verslo procesais (Von Halle 2002), todėl, kad verslo taisyklių metodas yra metodika ir net galbūt speciali technologija, kuri leidžia nustatyti, išgauti, panaudoti, parodyti, automatizuoti ir greitai keisti taisykles atsižvelgiant į strategines verslo perspektyvas. Gaunamas rezultatas yra verslo taisyklių sistema – automatizuota sistema ar jos dalis, kurioje taisyklės yra atskirtos, logiškai ir galbūt fiziškai, nuo kitų sistemos aspektų ir bendrų duomenų saugyklų, vartotojų sąsajų ir galbūt nuo taikomųjų programų (Lublinsky, Le Tien 2002).

2.2.3. Verslo taisyklių naudojimo motyvacija

Į klausimą, kodėl transformacijos tikslas yra verslo taisyklių naudojimo palaikymas, galima atsakyti taip:

- Verslo taisyklės – tam tikros dalykinės srities žinios, išreikštos ribojimais ir apibrėžimais (Business Rules Group 2000).
- Verslo taisyklių metodas – sistematinis būdas išsaugoti ir dokumentuoti žinias (Ross 2003, 185), net ir tada kai vyksta darbuotojų kaita.
- Verslo taisyklės – reikalavimai, išreiškiantis gryną verslo logiką (Ross 2003, 185).
- Taikant verslo taisyklių metodą galima lanksčiau ir efektyviau atlikti pakeitimus ir sutrumpinamas projekto kūrimo laikas (von Halle 2002).
- Verslo taisyklės gali būti išreikštos formaliomis kalbomis (Business Rules Group 2000).
- Formaliai užrašytas verslo taisykles galima automatizuotai įdiegti ir panaudoti informacinėje sistemoje.
- Ir kiti privalumai aprašyti skyriuje: Unikalūs verslo taisyklių aspektai (von Halle 2002).

2.2.4. Unikalūs verslo taisyklių naudojimo aspektai

Verslo taisyklės ir jų metodas turi tris unikalius privalumus (von Halle 2002):

- taisyklės atsekamumas;
- į objektus orientuotų, informacijos inžinerijos ir taisyklių formalizmų integracija;
- taisyklių koreliacija iki verslo motyvacijos (strategijos, uždavinių, tikslų ir taktikų).

Greičiausiai pats unikaliausias verslo taisyklių aspektas yra taisyklės atsekamumas. Taisyklių rinkinys, apimantis sąveikas ir duomenis, kur taisyklės yra valdomos, yra atskirtas kaip išorinis loginis komponentas.

Mes galime pasirinkti verslo taisyklių technologiją, kuri specialiai sukurta valdyti taisyklių rinkinio vykdymą. Alternatyviai, mes gali naudoti ne verslo taisyklės technologiją, net tarkim Java programavimo kalbą, tačiau šiuo būdu prarandami verslo taisyklių sistemos konceptai ir privalumai.

Antras unikalus verslo taisyklių metodo aspektas yra tas, kad naudojamas objektinis orientavimas, informacijos inžinerija ir taisyklių formalizmus apimanti integracija.

Trečias unikalus aspektas yra tas, kad taisyklės koreliuoja su daugeliu verslo motyvacijos aspektų, kurie apima tikslus, uždavinius, strategijas ir taktikas. Šiuo būdu, verslas gali įvertinti taisyklių

efektyvumą siekiant trokštamų baigčių. Galima išskirti 10 pagrindinių verslo taisyklių metodo naudojimo privalumų:

- paprastumas,
- teorinė bazė,
- mažesnis skaičius būtinų ne techninių konceptų,
- taisyklių nepriklausomumas,
- taikomųjų programų kūrimo lengvumas,
- taisyklių pakartotinis panaudojimas,
- supaprastintas sistemos projektavimas,
- dinaminės taisyklės,
- našumas,
- lengvas sistemos tobulinimas.

Paprastumas. Verslo taisyklių metodas yra lengvai suprantamas tiek verslo ir ne verslo (techninių specialybių) atstovams. Pats taisyklės konceptas, net ir skirtinga taisyklių klasifikacija, yra pakankamai intuityvi. Verslo atstovai gali būti ne itin susidomėję duomenų modeliais, procesų modeliais ar objektų modeliais. Tačiau jie tikrai susidomės verslo politika ir taisyklėmis, kadangi jomis valdomas verslas.

Teorinė bazė. Taisyklių konceptas kaip duomenų darnos saugojimas turi savo kilmę susieta su reliaciniu modeliu. Papildomi tyrinėjimai buvo atlikti aktyviose duomenų bazių sistemose. Žinių inžinerijoje egzistuoja didelė teorinė ir praktinė patirtis naudojant taisykles.

Mažesnis skaičius būtinų ne techninių konceptų. Verslo taisyklės metodo centre yra pati taisyklė. Egzistuoja tik keletas konceptų supančių taisyklę, tokie kaip sprendimai, taisyklių struktūrų šablonai, taisyklių šeimos, taisyklių sakiniai. Sprendimas yra paprastas loginis taisyklių grupavimas. Atliekant analizę taisyklės gali būti sugrupuojamos kartu į taisyklių šablonus, remiantis taisyklių sakiniais. Taisyklės šablonai gali būti grupuojami kartu į taisyklių šeimas, remiantis panašia taisyklių išeiga, o pati taisyklė sudaryta iš vieno ar daugiau sakinių.

Taisyklių nepriklausomumas. Verslo taisyklių metodas stengiasi išreikšti taisykles tokia sintakse, kuri nepriklauso nuo technologijos (pvz., programavimo kalbos) ar taikomosios aplinkos (programinės įrangos).

Taikomųjų programų kūrimo lengvumas. Atsiradus komercinei taisyklių technologijai, įvairių rūšių taisyklių apdorojimas tapo lengvesnis. Su kai kuriais produktais, taikomosios programos, (vad. taisyklių produktais), kuriose taikomoji programa atlieka sprendimą, remdamasi esminių taisyklių vykdymu. Kiti taisyklių produktai palaiko taisyklių vykdymą ir interpretavimą, kuriuo parodomi

duomenų pakeitimai. Dar kitais atvejais, taisyklių profesionalai apibrėžia, tada kai taisyklės turėtų suveikti, atsižvelgiant į taikomąja programinę įrangą ir dalykinę sritį.

Taisyklių pakartotinis naudojimas. Jei pasirenkamas komercinis produktas, taisyklės valdymo komponente gali būti panaudotos įvairiose transakcijose ir gali sąveikauti su įvairiomis naršyklėmis; duomenų bazių valdymo sistemų (DBVS) produktais ir kitų rūšių programinę įranga. Todėl taisyklės ar jų rinkinio realizacija mes galime apibrėžti tik vieną kartą, tačiau jas vėliau turėti aktyvias įvairiems atvejams ir tikslams. Mes galime taisykles testuoti dar nesukūrus visos taikomosios programos ar užpildžius duomenų bazės. Net jei ir mes nenaudojame komercinio produkto, mes gali suprojektuoti savo sistemą tokiu būdu, pagal kurį taisyklės grupuojamos kartu pagal loginį veikimą ir vėliau jos gali būti dar kartą panaudotos ir padalintos, jei yra poreikis.

Supaprastintas sistemos projektavimas. Verslo taisyklių metodas siekia atskirti esminius procesus nuo taisyklių vykdymo. Sukuriami du atskiri srautai: vienas taisyklėms ir kitas sistemos vykdymo sekai. Verslo taisyklių sistema projektuojama apie esminį intelektualinį proceso srautą. Tai yra koncentruojamasi į taisykles, analitikas gali išskirti tarpusavio sąsajas tarp taisyklių, kurios yra įdomios iš veikimo arba naudotojo pasirinkimo perspektyvos. Tokiu būdu, taisyklių vykdymas gali būti paskirtas specialiai taisyklių technologijai (ar vietoje sukurtom taisyklėms), kai tuo tarpu pagrindinį sistemos srautą (be taisyklių) suprojektuoja sistemos projektuotojas. Todėl tokiu būdu bus mažiau reikalingų kintamųjų ir galbūt mažiau kodo, priklausomai nuo pasirinktos technologijos. Todėl, kad naudojant kai kurias komercinius taisyklių produktus, analitikas apima sistemos darnos ir skaičiavimo taisykles, o sistemos kūrėjai išreiškia jas daugiau deklaratyviomis taisyklėmis nei procedūriniu programiniu kodu. Kai kurie komerciniai produktai valdo minėtas taisykles, jas kompiliuoja ir nustato vykdymo vietą, panašiai kaip reliaciniai produktai valdo duomenis, pasirenka tinkamiausia veikimo (ang. *access*) būdą ir jų vykdo. Tinkamų taisyklių nustatymas gali trukti savaites ar mėnesius, priklausomai nuo verslo žinių prieinamumo. Tačiau kartą nustačius ir dokumentavus taisyklę, mes ją galime įgyvendinti per porą valandų. Per tas valandas mes nustatome kur ir kaip įgyvendinti taisyklę. Jei tai atliekama su kitai ankstesniais procedūriniai metodais, tikėtina, kad taip įgyvendinsime taisyklę, ne tik įgyvendindami procedūrinį kodą, bet dažnai ir įvairias programas ar metodus. Komerciniai taisyklių produktai leidžia mums ne tik įgyvendinti dažnas taisykles vienoje vietoje, bet ir dalintis dažnais pasitaikančiais, naudojamais aprašyti taisykles.

Dinaminės taisyklės. Kai taisykles galima greitai pakeisti, mes galime lengvai pakeisti verslo taisyklių sistemą. Taisyklių kūrėjas gali pakeisti po vieną taisyklę tam tikru metu, arba daug taisyklių ir gaus tai, kad pakeitimas gali įvykti visos atitinkamoms verslo transakcijoms. Tačiau visa tai priklauso nuo pasirinktos technologijos. Šiuo būdu, verslo taisyklių sistema tampa verslo pokyčių platforma. Pačios taisyklės tampa verslo adaptyvumo instrumentu.

Vykdymas. Komerciniai taisyklių produktai yra specialiai suprojektuoti tam, kad valdytų ir vykdytų taisykles. Todėl jie turi vidinę logiką, kur aprašo kaip geriausiai tai atlikit ir pasiekti optimaliausią veikimą.

Sistemos kūrimas dalimis. Verslo taisyklių sistema gali būti sukurta pakankamai lengvai papildant reikalingomis dalimis. Pirmą sistemos dalį sudaro vientisas duomenų pagrindams, kuriame atsižvelgiama į galimą plėtimąsi. Dalimis kuriamos sistemos paleidimas veikti tampa jau prie egzistuojančios infrastruktūros sukuriami atnaujinimai ar papildomi taisyklių rinkiniai.

2.3. Metataisyklės

Taip galima paminėti metataisykles, kurios yra taisyklės apie taisykles. Šio taisyklių rinkinio tikslas yra nusakyti, ko reikia laikytis norint sukurti „gerą taisyklių“ mašiną (Date 2000). Tai daugiau apytikslis planas, kurio reiktų laikytis.

Taisykles C. J. Date suskirstė į tris kategorijas:

- vaizdavimo taisyklės,
- duomenų bazių taisyklės,
- taikomosios taisyklės.

Pirmajai kategorijai galime priskirti tokias taisykles, kurios taikomos naudotojui bendraujant su taikomąja programa (pvz. klaidų pranešimai ir pan.). Antrajai kategorijai, kurios apibrėžia duomenis duomenų bazėse. Pvz., šiomis taisyklėmis nustatomas duomenų formatą, jų atnaujinimą ir išgavimą. Paskutinės kategorijos taikomųjų programų taisyklės, tai tokios taisyklės, kurios naudojamos programos veikimui versle.

2.4. Verslo taisyklių naudojimas informacinių sistemų gyvavimo cikle

Sistemos gyvavimo ciklas egzistuoja nuo tada kai buvo sukurtos kompiuterinės taikomosios programos. Jis iš esmės sudarytas iš reikalavimų programai, verslo srities analizavimo, programos projektavimo, kūrimo, testavimo ir derinimo bei programos įdiegimo veikti produkcijoje (Navigli, Velardi, Gangemi 2003). Egzistuoja nemažai variacijų šia tema. Tačiau atrodo, kad sistemos gyvavimo ciklas atliekamas netiksliai lyginus su kitais inžineriniai darbais, pvz., lėktuvų projektavimas. Lėktuvų projektuotojai tikrai laikosi reikalavimų ir atlieka projektavimo darbą, tačiau jie kuria ypatingai detales specifikacijas, kurios bus naudojamos kurti objektams tokiu būdu, kuris atrodo niekados nebus įgyvendintas programinės įrangos kūrime. Programinės įrangos kūrimas niekada nepasiekia tokio inžinerijos lygio, nors ir naudojamos naujos metodologijos, kurios sukuriamos kas keli metai. Šios metodologijos dažnai žada, kad jos pakeis tradicinius sistemų gyvavimo ciklus, tačiau visada pabaigoje lieka tik viena iš variacijų. Gali kilti klausimas ar verslo taisyklių metodas nėra dar viena variacija bandanti sąveikauti su verslo logiką, ar ji tikrai tinka sistemos gyvavimo ciklui.

Verslo taisyklių metodas turi aktualus tuo, kad juo galima panaudoti ir užrašyti esminę konkrečios verslo srities logiką, galima remiantis atliktais verslo taisyklių apibrėžimais kurti vykdomus objektus. Jei mūsų valdymo praktikos pagrįstos sistemos gyvavimo ciklu, jos taip pat turi keistis, jei mes daugiau atsisakysime rankomis kuriamo programinio kodo. Tačiau susiduriama su tam tikrais sunkumais, pvz., valdymo praktikos sąveika su verslo metodu produkcijos aplinkoje. Tą galėtų išspręsti sukurta verslo taisyklių mašina, kuri leidžia verslui greitai reaguoti į pakitusias sąlygas. O pastarosios žinių bazės kūrimą , taisyklių rinkinių formavimą, padėtų įgyvendinti ontologijos kūrimo įrankiu aprašyta dalykinė sritis.

Grįžkime prie verslo pokyčių kurie gali būti pavyzdžiui, pasikeitę finansiniai kontrakto parametrai, pvz., pakitusi PVM lengvata tam tikroms prekėms, kurio skaičiavimui naudojami prekių duomenis saugomi duomenų bazėje. Tam greičiausiai prireiks surasti programiniame kode aprašytą ir taisyklėje naudojamą minėtą kintamąjį bei ji pakeisti. Tai daug lengviau padaryti jei tai būtų atlikta išorinėje taisyklių žinių bazėje, o vėliau panaudoti taisyklių mašinoje. Tačiau tradiciniu būdu tai atlieka duomenų bazės administratorius tuo metu, kai produkcijos sistema nedirba ir negalima sutrikdyti jos veiklos. Tai dažniausiai atliekama programuojant rankiniu būdu. Dažnai programuotojai atlieka neadekvačius testavimus, administratoriai turi užtikrinti tinkamą testavimo aplinką, bei vykdomos kitos biurokratinės procedūros, kad sistema tinkamai veiktų produkcijoje.

Toks biurokratinis ir pakankamai nepatogus mechanizmas gali būti reikalingas kaip tradicinių sistemų gyvavimo ciklo dalis, tačiau jo galima atsisakyti naudojant verslo taisyklių metodą, kadangi taisyklės užrašomos paprasta taisyklių kalba ir valdomos atskirai nuo sisteminio kodo. Verslo taisyklių metodas pasižymi didesniu lankstumu, galima lengvai derinti verslo taisykles (privalumus apžvelgsime vėliau). Tiesiogiai remtis užrašytais verslo taisyklių apibrėžimais. Todėl verslo taisyklių metodas gali padėti padidinti įmonės darbo efektyvumą.

Akivaizdu, kad valdymo praktikos, kurios buvo sukurtos sistemos gyvavimo ciklui, nebus tokios pat, kai bus nutolta nuo rankiniu būdu rašomo programinio kodo (Navigli, Velardi, Gangemi 2003). Tikėtina, kad egzistavusios valdymo praktikos prieštaraus verslo taisyklių metodui, todėl sistemos gyvavimo ciklas bus kitoks.

2.4.1. Verslo taisyklių rinkinys ir jo formavimas

Verslo taisyklių rinkinys – grupė susijusių taisyklių, pritaikomų verslo taisyklių mašinoje. Prieš kuriant verslo mašiną susiduriama su nemažai problemų ir klausimų kuriuos reikia išspręsti.

Kiekvienas verslo taisyklių mašinų projektuotojas ir kūrėjas turi daug ką apmąstyti iš techninės pusės. Bendrame po nustatymų galutiniame sistemos veikimo procese bus įtraukta daug kitų šalių, kurios dalyvaus įgyvendinant verslo taisyklių mašinos darbą po jos sukūrimo. Svarbiausia, kad projektuotojas suprastų visą bendrą vaizdą. Projektuotojui ypač reikia apgalvoti tai, kas apibrėš verslo

taisykles ir kada tai bus daroma, tam būtina parinkti tinkamus kandidatus ir laiką. Akivaizdu, kad šie susitarimai ypatingai stipriai įtakoja taisyklių verslo mašinos kūrimą. Yra daugybė skirtingų aktorių (darbuotojų tipų), kurie gali būti susiję su tuo, kaip bus apibrėžtos taisyklės, tačiau ir taisykles galima apibrėžti skirtingose verslo taisyklių mašinos įgyvendinimo ciklo stadijose (kaip ir realiame verslo cikle). Taisyklių mašina turi atitikti savo veikimo aplinką, kad aktoriai prasmingai ja naudotųsi.

2.4.1.1. Aktorių tipai

Egzistuoja skirtingų tipų aktoriai, galintys įvairiai įtakoti verslo taisyklių apibrėžimą, todėl svarbu pasirinkti geriausius aktorius verslo taisyklėms apibrėžti. Aktoriai turi skirtingus sugebėjimus ir nuo to priklausys jų galima įtaka projektuojant, todėl į tai reikia atsižvelgti. Pavyzdžiui, viena aktorių grupė mažiau išmano nagrinėjamą verslo sritį, kurioje veikia taisyklių mašina, todėl negalės patikimai naudotis ekranais konkretizuojant taisykles, naudojančiais nemažai verslo žargono, kadangi jo nesupranta, tai gali būti rimta problema. Šiems naudotojams būtinai reikės papildomų pagalbinių funkcijų, kurios paaiškintų jiems neaiškius taisykles apibrėžiančius ekrano laukus.

Pasitaiko situacijų, kai prieinama iki tokio lygio, kai taisyklių mašinos projektuotojas nebegali kompensuoti naudotojų personalo sugebėjimų trūkumo įgyvendinant projekto galimybes. Verslo taisyklių mašina paprastai yra naudojama verslo sričių komplekse. Tam būtina turėti tinkamus žmogiškuosius resursus, kadangi dėl jų trūkumo taisyklių mašinos įgyvendinimas gali būti nesėkmingas. Akivaizdu, kad niekas nėra suinteresuotas, kad būtų iš naujo pakartotinai grįžtama prie tų pačių sprendimų priėmimo.

Nors verslo taisyklių mašina yra lyg ir nauja koncepcija, tačiau ji gali identifikuoti keletą aktorių kategorijų, kurie yra arba gali būti susiję su verslo taisyklių apibrėžimu ir tuo pačiu vėlesniu įgyvendinimu sistemoje. Šiuo metu yra įmanoma suprasti kai kurias aktorių vaidmenų perspektyvas ir jų galimą įtaka verslo taisyklių projektavime.

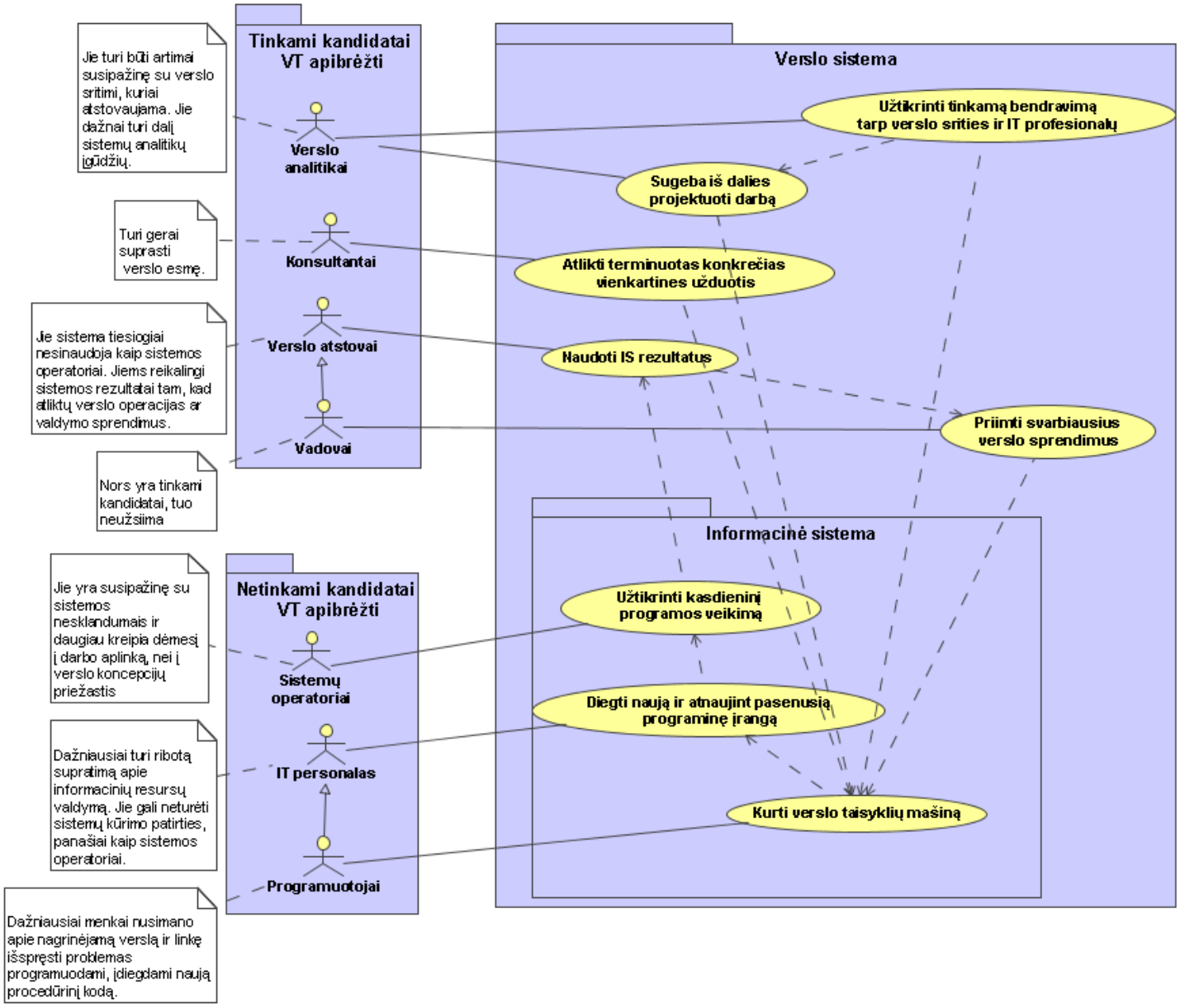
Kiekviena iš šių aktorių kategorijų yra aprašoma iš aktorių funkcijų perspektyvos, tikrieji pareigybių pavadinimai gali kisti priklausomai nuo organizacijos. Tačiau verslo mašinos projektavimas ir kūrimas yra taisyklių mašinos projektuotojo atsakomybė, tuo tarpu verslo dalyviai ir jų valdymas turi savas atsakomybes. Kiekvienas galvojantis apie verslo taisyklių mašinos įgyvendinimą, turi apmąstyti ar jie turi tinkamą personalą, ar jie gali pasisamdyti reikalingus darbuotojus (konsultantus), kurie turės pakankamai sugebėjimų apibrėžti verslo taisykles.

Pagrindinės išskiriamos aktorių grupės, grafiškai pavaizduotos 2 paveiksle.

- Verslo analitikai – puikūs kandidatai.
- Sistemų operatoriai – netinkami kandidatai.
- Konsultantai – geri kandidatai.
- Verslo atstovai – geri kandidatai, jei dirba katu su verslo analitikais ar konsultantais.

- Vadovai – nors kaip verslo atstovai tinkami kandidatai, jie tuo neužsiima.

- IT personalas – tiesiogiai neturi dalyvauti, tačiau būtina su jais derinti.
- Programuotojai, IT personalo tipas – netinkami kandidatai.



2 pav. Aktoriai, apibrėžiantys verslo taisykles, pavaizduoti užduočių diagramoje

Aptarsime tuos tipus, kurie yra svarbiausi apibrėžti verslo taisyklių rinkinio taisykles. Visų pirma tai, **verslo analitikai**, kurie yra puikūs kandidatai verslo reikalavimus paversti į verslo taisykles. Todėl, kad jie dažniausiai yra profesionalai, kurių vienintelė funkcija užtikrinti tinkamą bendravimą tarp verslo srities ir IT profesionalų. Kitais atvejais jų vaidmenį atlieka savanoriai ar komandiruoti verslo naudotojai, kurie siekia darnaus bendravimo specifinėje sistemoje ar projekte. Verslo analitikai turėtų būti ypatingai protingi, komunikabilūs ir artimai susipažinę su verslo sritimi, kuriai atstovaujama. Jie dažnai turi dalį sistemų analitikų įgūdžių. Jie taip pat gali sugebėti skaidyti problemas į sudedamąsias

dalis ir apibrėžti aukštos kokybės verslo reikalavimus. Kartais jie gali sugebėti tam tikrame lygyje projektuoti darbą, kuris naudingas kuriant prototipus.

Jiems taisyklių apibrėžimo ekranas turi būti specifinis ir pritaikytas būtent pagal jų analizuojamą verslą. Verslo analitikas taip pat turi gerai suvokti detalų vaizdą. Tai naudinga, kai reikia įgyvendinti mažiausias taisykles, kurios neišvengiamai paveiks daugiau taisyklių apibrėžimų nei pradžioje galėjo pasirodyti. Verslo analitikai tikėsis turėti įrankių, užtikrinančių taisyklės apibrėžimą. Pavyzdžiui, kelios valdymo ataskaitų rodo, kad yra tenkinama taisyklių apibrėžimų seka ir jos suveiks. Tai įvyksta tik kartą, kuriant verslo taisyklių mašiną, kai taisyklės apibrėžimas bus patenkintas ir greitai po to kai mašina bus įgyvendinta. Projektuotojams naudinga atsižvelgti į verslo analitikų įtraukimą, tada kai kuriami taisyklių apibrėžimo ekranai.

Sistemų operatoriai nėra tinkami kandidatai padėti apibrėžti taisykles. Paprastai jie įdarbinami tam, kad užtikrintų kasdieninį programos veikimą. Iš esmės dažniausiai jie yra susipažinę su sistemos nesklandumais ir daugiau kreipia dėmesį į darbo aplinką, nei į verslo koncepcijų priežastis. IT profesionalai dažnai klaidingai galvoja, jog sistemų operatoriai geriausiai padės suprasti verslą, tačiau tai gali sukelti tik papildomų problemų.

Organizacijos, norinčios įgyvendinti programą, turinčią verslo taisyklių mašiną, dažnai projektams samdosi konsultantus. **Konsultantas** yra pašalinis asmuo, nepriklausantis organizacijai, kuris samdomas atlikti specifinę užduotį. Tuo konsultantas skiriasi nuo verslo analitikų ir sistemų operatorių, kurie yra organizacijos darbuotojai. Konsultantai turi gerai suprasti verslo esmę, bent jau tokiu lygiu kaip verslo analitikai. Tiesa, dažniausiai, bet ne visada, yra sunku surasti kvalifikuotus konsultantus. Net ir tada jei konsultantai turi geras verslo žinias, jie dažniausiai mažiau susipažinę su organizacijos struktūra ir išorine politika, nei savi darbuotojai, kurie yra verslo analitikai.

Tačiau, konsultantai pranašesni tuo, kad yra labiau motyvuoti įvykdyti savo užduotis ir jais galima pasitikėti, kai reikia tinkamai atlikti darbą. Esantį žinių trūkumą jie kompensuoja išsiaiškindami, ką jie turi žinoti ir retai drovisi uždavinėti klausimus, kadangi nėra tiesioginiai organizacijos darbuotojai. Didelis pranašumas naudotis konsultantais yra tas, kad jie gali atlikti terminuotas konkrečias vienkartines užduotis. Kadangi gali būti taip, kad neįmanoma pervesti darbuotojus iš kitų pareigų atlikti tokias užduotis, kaip kad, taisyklės apibrėžimas, jei jie jau pilnai dirba savo pareigose. Kitas konsultantų naudojimo pranašumas, kad jie gali būti bendrai susipažinę su taisyklių mašina, kas menkai tikėtina iš darbuotojų. Todėl galima tikėtis, kad jie supras taisyklės apibrėžimo procesą, o tai papildomas pranašumas prieš nuolatinius darbuotojus.

Todėl konsultantai yra labai geri kandidatai verslo taisyklėms apibrėžti. Net ir tokiose situacijose, kai organizacija kuria savo pačios taisyklių mašiną, konsultantai gali būti reikalingi apibrėžti taisyklėms. Tai ypač pasitvirtina, jei taisyklių mašiną reikia dislokuoti skirtingose vietose per trumpą laikotarpį.

Verslo atstovai – verslą išmanantys darbuotojai, kurie paprastai užima aukštesnes pareigybes nei sistemos operatoriai. Jie nesinaudoja sistema tiesiogiai kaip sistemos operatoriai. Jie naudojasi gaunamais sistemos rezultatais, kad atliktų verslo operacijas ar priimtų sprendimus.

Mąstas, kuriuo verslo atstovai nori ar gali padėti taisyklių apibrėžimo procese, yra skirtingas. Tam tikromis sąlygomis verslą išmanantys darbuotojai yra pernelyg užsiėmę ir negali skirti tam laiko. Kitais atvejais jie gali nenorėti „priartėti“ prie technologijų. Jei tokie darbuotojai gali ir nori būti įtraukti į šį procesą, tai jie yra geri kandidatai padėti apibrėžti taisykles. Tačiau jiems kartais trūksta metodiško gilumo ir dėmesio detalėms, to daugiau tikimasi iš verslo analitiko.

Verslo atstovai turi svarbias darbo žinias apie savo verslo sritį įtakojančias priežastis bei taisykles. Geriausia, kad konsultantas ar verslo analitikas sužinotų iš verslo atstovų reikalingas žinias ir vėliau jas pritaikytų apibrėždami taisykles, negu tai atliktų pats verslo atstovas.

Vadovai priima svarbiausius sprendimus, jie galutinai sprendžia, kurioms konkrečioms verslo sritims reikalinga verslo mašina. Jie priima patarimus iš kitų organizacijos narių, gali vetuoti taisyklių mašinos įsigijimą ar kūrimą. Tačiau negalima tikėtis, kad vadovai bus įtraukti į taisyklių apibrėžimo detalių svarstymą.

Kai organizacija nusprendžia kurti verslo taisyklių mašina, gali būti ginčytinų klausimų, kuriuos reikia spręsti bendradarbiaujant su IT personalu. Šiuo metu, IT personalu laikomi organizacijos techninės ir programinės įrangos resursų administratoriai, kurie dažniausiai turi ribotą supratimą apie informacinių resursų valdymą. Jie gali neturėti sistemų kūrimo patirties, panašiai kaip sistemos operatoriai. Dauguma jų užduočių orientuotos į naujos programinės įrangos įdiegimą ir atnaujinimą.

Nors IT personalas tiesiogiai nedalyvauja apibrėžiant taisykles, tačiau dažnai tenka su jais dirbti, kad taisyklių apibrėžimas vyktų tvarkingai ir sėkmingai. Verslo taisyklių mašina gali netikti esamai struktūrai, ypač tuo metu, kai taisyklės yra apibrėžiamos. Priklausomai nuo taisyklių įgyvendinimo strategijos, taisyklių apibrėžimas gali sukelti pokyčius aplinkoje, kurioje bus realizuota taisyklių mašina. Pavyzdžiui, taisyklių mašina gali generuoti naujus failus, turinčius programinį kodą, nurodantį kada naujos taisyklės yra apibrėžiamos. Jei IT personalas paliko pernelyg daug suvaržymų būsimoje įgyvendinimo aplinkoje, gali kilti problemų, pavyzdžiui, gali būti neįmanoma kurti reikalingų failų. Todėl būtina bent iš darbą dalies derinti su IT personalu.

2.4.1.2. Kada apibrėžiamos verslo taisyklės?

Buvo aptarti taisyklių apibrėžimo aktoriai, tačiau kitas ne mažiau svarbus parametras yra taisyklių apibrėžimo laikas. Kai žmonės bendrai galvoja apie taisyklių mašinas, jie kartais įsivaizduoja, kad taisyklės yra apibrėžiamos visą analizuojamos programos veikimo laiką. Iš tikrųjų yra nemažai skirtingų variantų, kuriuose nurodoma taisyklės, kada gali būti apibrėžiamos.

2.4.1.3. Iš anksto žinomos taisyklės

Kartais, jei taisyklės žinomos iš anksto, taisyklių mašina gali būti iškviesta kuriant visą sistemą ar jos didelį komponentą. Tokiu būdu, taisyklės gali būti įdiegtos prieš visos sistemos galutinį išdėstymą. Sistema yra įdiegiama ir patenka į produkciją, vėliau taisyklės nebeapibrėžiamos. Tai panašu į tradicinį sistemos kūrimą, išskyrus tai, kad taisyklių mašina yra naudojama vietoje pačios programavimo aplinkos. Galbūt didžiausias pranašumas šiuo atveju tai, kad padidinama galima proceso užbaigimo sparta.

2.4.1.4. Taisyklių apibrėžimas įgyvendinimo metu

Verslo taisyklių būdai yra ypač tinkami tada, kai verslo taisyklės yra nežinomos iki programos įgyvendinimo. Tai vienas iš verslo taisyklių metodo privalumų.

Tai vyksta gana skirtinga forma nuo tradicinių sistemų kūrimo ciklo, kuris apima projektavimą, programavimą, vienetų testavimą, kokybės garantijos testą ir galiausiai produkciją. Tie, kurie daugiau susipažinę su šiuo kūrimo ciklu, galvoja, kad taikomosios programos yra arba kuriamos šiuo būdu, arba nuperkamos, ir skubiai, be jokių pokyčių įgyvendinamos produkcijoje. Gali būti ganėtinai sunku su tokiais asmenimis. Jei jie mato programoje padarytus pokyčius, tokius kaip taisyklės apibrėžimas, jie gali kartais panorėti vystymo, testavimo, kokybės garantijos ir produkcijos sąlygų paruošimo. Tai - nereikalinga, jei didžioji dalis taisyklės apibrėžimo yra pradėta atskiroje produkcijos fazėje. Viskas, ką reikia atlikti, tai įdiegti programinę įranga, apibrėžti taisykles, atlikti testavimą realių sąlygų aplinkoje ir paleisti veikti.

Tokiu būdu, jei kas nors kuria verslo taisyklių mašiną kaip programinės įrangos dalį, kur taisyklės yra apibrėžiamos įdiegimo metu, privalo gerai apgalvoti taisyklės apibrėžimo metodiką, kad naudotojai iš anksto suprastų kaip programinė įranga bus naudojama produkcijoje. Tai leis išvengti nereikalingų nesusipratimų.

2.4.1.5. Verslo pokyčiai produkcijos sistemose

Kai programa pradeda veikti produkcijoje, verslo taisyklių metodas leidžia, kai to prireikia, įdiegti versle vykstančius pokyčius. Taisyklių mašinos kūrėjai turėtų būti atsargus ir nepersistengti, kadangi gali prisidaryti tik papildomų sunkumų. Verslas dažniausiai sparčiai nekinta, todėl akivaizdu, kad tai gali sukelti kūrėjui patikimumo problemų (Chisholm 2004). Tai taip pat gali sudaryti įspūdį, kad taisyklių mašina yra sukurta atitikti reikalavimus, kurie palyginus retai pateikiami. Egzistuoja ribojimai, kuriais nurodoma, ką galima atlikti su verslo taisyklių mašina pagal verslo pokyčių valdymo sąlygas.

Visiems kuriantiems taisyklių mašiną ir atidžiai galvojantiems apie tai yra svarbu, ką naujos taisyklės gali įnešti po įdiegimo į produkciją. Apibendrinant taisyklių mašiną gali susitvarkyti su šiais pokyčių tipais po jos įdiegimo (Chisholm 2004):

Naujos taisyklės yra galimos, jei jos naudoja egzistuojančią duomenų bazę arba yra galimas egzistuojančios duomenų bazės išplėtimas.

Duomenų bazė gali būti saugiai išplėsta pridėjus naujus ne pirminių raktų stulpelius prie egzistuojančių lentelių. Pridėjus naujas lenteles arba naujus sąryšius tarp lentelių dažniausiai viršijamos daugumos verslo mašinų galimybės.

Kai kurios komercinės verslo taisyklų mašinos, kurios yra bendros ir lanksčios, gali sugebėti prisitaikyti prie radikalių verslo pokyčių, tačiau viskam yra ribos. Kiekvienas, kuriantis verslo mašiną, turėtų žinoti minėtus ribojimus, tačiau juos nėra sunku viršyti siekiant atitikti specifinius reikalavimus. Pavyzdžiui, taisyklių mašina galėtų būti sukurta taip, kad leistų naujų lentelių įvedimą prie esamos duomenų bazės tam, kad kategorizuotų kitas lenteles. Nepaisant to, taisyklių mašinos kūrėjas turi neklaidinti sistemos potencialių naudotojų apie tai kas gali būti pakeista sistemoje po jos įdiegimo produkcijoje.

Jei verslo vartotojams nereikia įvesti ar pakeisti taisyklių, kokių atsargumo priemonių turėtų būti imtasi? Žvelgiant iš vienos pusės, aplinkos kokybės garantija gali būti nustatyta imitavus produkciją ir atlikus testavimą, kur būtų išbandytos naujos įdiegtos taisyklės. Šioje simuliacijoje būtų patikrinta ar jos veikia teisingai. Iš kitos pusės, naujosios taisyklės gali būti paprasčiausiai įdiegtos produkcijoje, tačiau šis variantas rizikingesnis. Taisyklių mašina turėtų pateikti ataskaitas, kurios būtų matyti kaip naujoji taisyklė sąveikauja su kitomis taisyklėmis ir duomenimis. Pagal ataskaitų rezultatus būtų sąlyginai galima nustatyti riziką bei pataisyti įdiegimo kryptį produkcijoje. Be to, naujoji taisyklė gali būti visiškai izoliuota nuo kitų duomenų ir taisyklių, pavyzdžiui, ji paprasčiausiai skaičiuoja naują stulpelį, kuris yra naudojamas tik ataskaitose ar yra papildomas ribojimas tam tikram duomenų laukui (Chisholm 2004). Tokiu būdu, taisyklių mašina padaroma saugesne įdiegimui produkcijoje.

Žinoma, jei visas procesas turės savo perrašytas taisykles, vienintelis saugus būdas pokyčius vykdyti kokybiškai ir patikimose sąlygose, tai yra ten atlikti jų testavimą. Tačiau reikmė perrašyti taisykles visam verslo procesui turėtų būti palyginus retas. Jei verslas radikaliai pasikeitė, greičiausiai reikės iš pagrindų pakeisti analizuojamą duomenų bazę, kuri nebeatitinka to, kam taisyklių mašina buvo suprojektuota. Todėl sugebėjimas pertvarkyti ištisus taisyklių rinkinius yra nerealus reikalavimas taisyklių mašinai ir projektuotojai turėtų apsvarstyti ar verta laikytis šio reikalavimo.

2.4.1.6. Kokios rūšies organizacijoms reikia verslo taisyklių metodo?

Svarbu žinoti, kokioms organizacijoms pirmiausia reikės verslo taisyklių metodo. Kadangi egzistuoja vienos organizacijų rūšys, kurioms labai tinkamas verslo taisyklių metodas ir kitos, kurioms abejotina ar jis suteiks naudos.

2.4.1.7. Invariantinės operacijos

Daugumos organizacijų veiklos struktūra yra paprasta. Pavyzdžiui, bendrovė gamina vieną ar kelis produktus. Kiekvieno produkto pardavimui taisyklės bus žinomos iš anksto ir niekada nekis. Iš tikrųjų niekas šioje organizacijoje gali ir nepagalvoti, kad naudojamos taisyklės pardavimo procese. Šios rūšies operacijos yra invariantinės – jas sudaro vienas taisyklių rinkinys, kuris taikomas visiems operacijų tipams (ar klasėms) naudojamiems įmonėje (Chisholm 2004). Taisyklių rinkinys bus gerai žinomas organizacijai prieš bet kokios operacijos atlikimą, todėl nėra priežasties tikėtis, kad taisyklių rinkinys pakis kuriai nors operacijai. Tačiau išlieka galimybė, kad taisyklių rinkinys laikui einant gali nežymiai pasikeisti, pavyzdžiui, jei pakeičiami pardavimų mokesčiai, bet tai nutinka gana retai.

Invariantinės operacijos nėra geros verslo taisyklių mašinoms, kurios yra programos dalis. Kadangi vieną kartą sukūrus programą, kuri gali apdoroti invariantines operacijas, taisyklės retai būtų keičiamos. Todėl tikėtina, kad geresnis būdas sistemą sukurti tradiciniu būdu. Svarbi išimtis yra programinės įrangos paketai, kurie gali būti įdiegiami įmonėse, naudojančiose invariantines operacijas, kur kiekviena įmonė yra skirtinga.

2.4.1.8. Programinės įrangos paketai

Jau ankščiau buvo minėti programinės įrangos paketai yra geri kandidatai turėti taisyklių mašinos elementų. Jie ne tik gali būti programos, sukurtos pardavimui bendroje rinkoje, bet ir organizacijos sukurtos programos, skirtos naudoti skirtingose savo organizacijos vietose. Paketai tradiciškai teikia papildomą funkcionalumą specifiniai verslo sričiai ir yra suprojektuoti įdiegti tose vietose, kur mažai ar visai nieko nežinoma apie jo kūrimą. Kai paketas yra įdiegiamas naujoje vietoje, kūrėjas susiduria su šiais dviem klausimais:

Kokios yra verslo esminės variacijos, kas yra išskirtinio šiam klientui ir į ką nenukreiptas paketas? Kaip paketas integruosis į būsimas IT sąlygas? Kai kurie paketai yra atskiro tipo (ang. stand-alone), tačiau daugumai reikalinga tam tikro lygio integracijos.

Taisyklių mašinos vartotojai gali susidurti su programa, kuri, kaip jie mano, yra valdoma invariantinių operacijų. Tačiau paketo kūrėjai iš dalies atsižvelgia į daugybę skirtingų verslo sričių versijų kiekvienoje vietoje, kur bus paketas įdiegtas. Todėl paketai turi turėti bendresnius gebėjimus, kurie gali būti patikslinti įdiegimo metu. Dėl šios priežasties programinės įrangos paketai, projektuojami diegimui į kelias vietas, yra idealūs kandidatai verslo taisyklių metodui.

2.4.1.9. Sutarties verslas

Daugelio organizacijų verslas naudoja invariantines operacijas. Tačiau modernioje ekonomikoje vis daugiau verslo naudoja daug įvairių kontraktų su skirtingomis klientais. Paslaugų tiekėjams, kaip, pvz., transportavimo bendrovės, gali tekti derėtis dėl skirtingų kontraktų su įvairų rūšių klientais.

Pavyzdžiui, sutartis, transportuoti gaiviuosius gėrimus, skirsis nuo sutarties, pagal kurį reikia pergabenti medicinines atliekas. Todėl kiekviena sutartis gali turėti savo skirtingus uždavinius ir sąlygas. Šiuo atveju visas verslas organizacijos yra pagrįstas šių sutarčių rinkiniu ir tikrai aišku, kad greičiausiai nebus dviejų identiškų sutarčių.

Rankinis metodas naudojamas programoje apibūdinti kiekvienai sutarčiai yra tinkamas tik tokiam verslui, kuris turi tik kelias panašias sutartis. Išeitis gali būti bandyti klientus priverstinai priimti identiškus ar labai panašius sutartis, o tai ribos verslo plėtrą, kadangi dabartinėje modernioje ekonomikoje siekiama kuo didesnio lankstumo. Todėl abu metodai apribos verslo plėtrą.

Šio pavyzdžio vienintelis sprendimas yra taisyklėmis pagrįstas metodas. Sutarties verslas yra įdomi sritis verslo taisyklių mašinoms, kadangi jos yra infrastruktūros pamatinė dalis, kuri gali realiai įtakoti verslo augimą ir pokyčius.

2.4.1.10. Individuali rinkodara

Didžioji dalis individualios rinkodaros susijusi su komponentais, kurie niekuo nesusiję su verslo taisyklėmis, kaip sandėliavimo duomenys ir klientų santykių vadyba. Tačiau dažnai šis individualios rinkodaros terminas reiškia daug daugiau nei tiesiog rinkodara. Jis gali apimti didelę santykių dalį tarp organizacijos ir klientų. Pvz., labai turtingiems ir svarbiems banko klientams sandėriuose dažnai leidžiama savo nuožiūra apibrėžti konkrečias sąlygas, kaip jų vertybinių popierių rinkinys turėtų būti valdomas. Pavyzdžiui, konkretus klientas gali nenorėti investuoti į alkoholinius gėrimus ar gali norėti, kad būtų nedaugiau 50% paketo visada investuota, ir gali nurodyti, kad akcijos būtų investuotos tik Lietuvos ar kokios kitos šalies bendrovėse.

Ateityje daugumai bendrovių reikės judėti panašia linkme, jei jos nori išlikti konkurencingos. Verslo taisyklių mašina yra vienintelis būdas suderinti šiuos reikalavimus ir taisyklių mašinos turi potencialą atlikti perversmą organizacijos bendravime su klientais.

2.4.1.11. Veiklos, pagrįstos intensyviomis taisyklėms

Egzistuoja tam tikros veiklos, kurios beveik visiškai pagrįstos verslo taisyklėmis. Galima išskirti sąskaitybą ir kreditingumo įvertinimą.

Sąskaityba yra ideali aplinka kiekvienam norinčiam naudoti taisyklėmis pagrįstą metodą, kadangi beveik visa sritis yra vien taisyklės. Pavyzdžiui, organizacija nori parduoti kuo daugiau produktų ir tikriausiai turi ne vieną sistemą valdančią pardavimus. Tokiu atveju, finansų srautas eina per šias sistemas, kurios turi atpažinti kas yra kas ir kokioms sąskaitoms srautai yra skirti. Apskaitos programa projektuojama tokiu būdu, kad suprastų duomenų bazes į kurias kreipiamasi ir turėtų taisykles, kurios jomis operuoja atliekant apskaitos funkcijas. Dauguma komercinių apskaitos paketų

veikia tokiu principu ir reikalauja, kad jiems taisyklių rašytojai (dažniausiai konsultantai) terminuotai apibrėžtų apskaitos taisykles kiekvienam klientui.

Kreditingumo įvertinimas yra kita verslo sritis, kuri pagrįsta taisyklėmis. Kai asmuo ar organizacija kreipiasi paskolos, yra iškviečiamas taisyklių rinkinys nustatyti ar prašymas turi būti patenkintas. Taisyklių grandinė, suveikusi vienam paskolos prašytojui, gali būti visai skirtinga kitam klientui. Ir ne tik kreditingumo įvertinimas, bet ir kitos veiklos rūšys, kur „dalykai“ yra pasirenkami iš tam tikrų sąrašų, reikalauja verslo taisyklių metodo. Tokiu būdu, yra keli pritaikymai, kurie yra iš prigimties pagrįsti taisyklių naudojimu ir kuriuose negali būti alternatyvų verslo mašinose įdiegimui.

2.5. Taisyklių išraiška ir realizacija

Transformacijos objektas, verslo taisyklės išreiškiamos tam tikros srities specifine taisyklių kalba, kuria taip pat aprašomi taisyklių kintamieji (konstruktai – pamatiniai kintamieji). Šie kintamieji labai priklauso nuo verslo reikalavimų, todėl gali būti įvairūs variantai nuo tekstinio aprašymo (naudojant specifinę kalbą ar pvz., net gryną anglų k.) iki sprendimų lentelių ar sprendimų medžių. Tam tikrais atvejais, galima grafiškai nustatyti verslo taisyklių vykdymo tvarką, naudojant taisyklių srautą.

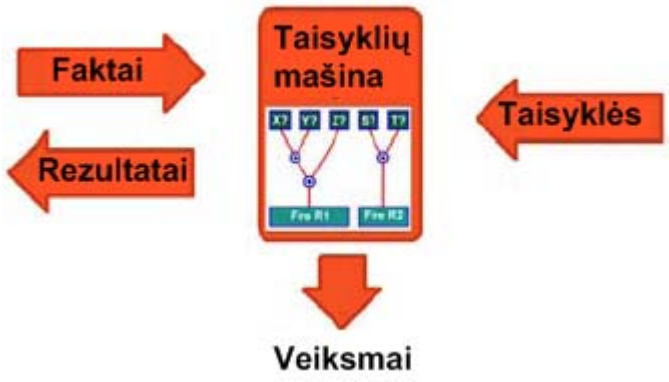
Verslo taisyklių mašina palaiko taisyklių vertinimą, kurios išreikštos tam tikra kalbą. Jų pagrindinis funkcionalumas yra valdyti faktų rinkinius ir vertinti taisyklių rinktinius, kuriuos sudaro vienas ar keli predikatai. Didesnio faktų skaičiaus ir efektyvus predikatų vertinimas yra vienas pagrindinių taisyklių vertinimų iššūkių. Verslo taisyklės pagrįstos verslo taisyklių apibrėžimais, išreikštos verslo taisyklių kalba. Verslo taisyklių mašinos tradiciškai teikia tam tikrą atvaizdo palaikymo mechanizmą tam, kad būtų galima generuoti žemesnio lygio vykdomąjį kodą, paprastai bendresnio pobūdžio kalbos klases. Šios klasės gali būti naudojamos kaip dalinis didesnio verslo komponento įgyvendinimas. Todėl naudojant tokį būdą gaunama tai, kad verslo taisyklės įgyvendinimo gyvavimo ciklas yra susietas su tuo komponento, kuriam priklauso, gyvavimo ciklu.

Verslo taisyklių įgyvendinamumo lankstumas ir lengvas keitimas yra pasiekiamas taisyklių palikimui naudojant dinaminius pokyčius (naudojant originalią taisyklių kalbą). Tokia savybė teikia patogų būdą dinamiškai keisti verslo taisykles, kai nereikia iš naujo sukurti ir įdiegti įgyvendinimo, kad prisitaikytume prie pasikeitusios verslo aplinkos.

Verslo taisyklės įgyvendinamos naudojant deklaratyvų programavimą, aprašant kokioms sąlygoms esant taisyklė yra tenkinama ir kokiomis netenkinama (Lublinsky, Le Tien 2002). Priklausomai nuo aprašytų verslo taisyklės sąlygų suveikia taisyklė, atliekamas aprašytas veiksmas.

Verslo taisyklės gali būti įgyvendintos naudojant paplitusi duomenų bazėse naudojama SQL ir verslo taisyklių mašinas. Šiuo atveju, verslo taisyklės yra paprastos IF <Sąlyga> THEN <veiksmai> deklaracijos. <Veiksmai> gali būti bet kas nuo paprastos reikšmės grąžinimo iki duomenų bazės

atnaujinimo ar tarpinės procesų žinutės siuntimo. Taisyklių mašinos naudoja taisykles analizuoti faktus, kurie yra informacija apie pasaulį (pvz., klientų informacija, darbuotojų informacija ir t.t.).



3 pav. Taisyklių mašina

Atlikus faktų analizę, taisyklių mašina grąžina rezultatą ir/arba inicijuoja veiksmo vykdymą. Rezultatai gali būti paprastos duomenų reikšmės arba sudėtingos duomenų struktūros. Veiksmai gali būti bet kas net ir el. laiško išsiuntimas, duomenų bazės įrašų atnaujinimas ar įspėjamųjų pranešimų iškvietimas (Oracle 2005). Taisyklių mašina yra apibrėžiama kaip esybė, vykdanti taisykles. Kai taisyklių mašinai pateikiamos taisyklės, mašina gali analizuoti jai patiektus faktus ir vykdyti numatytus veiksmus (Oracle 2005).

Taisyklė taip pat gali būti apibūdinama kaip paprastas IF-THEN (JEI-TAI) sakinys, kuriame:

- IF dalis sudaro taisyklės sąlygą;
- THEN dalis sudaro taisyklės veiksmą;
- Veiksmas vykdomas, kai sąlyga tenkinama (tačiau gali būti ir atvirkštinis veikimas, vykdoma kai netenkinama aksioma).

Sąlygos yra loginė išraiška, apimanti faktus. Faktai išreiškia informaciją apie realų pasaulį. Faktai gali pvz., žmogaus amžius, verslo forma, užsakymas ir t.t. Veiksmas yra bet koks veiksmas, ar rezultato grąžinimas, įspėjimo išvedimas ir t.t. (pvz. išvesti tam tikrą pranešimą). Rezultatai – duomenų struktūros.

2.6. Ontologijos

Verslo taisyklės yra susietos su faktais, t.y. prie duomenų modelio. Tai ypač pasireiškia kai verslo taisyklės naudojamos kaip ribojimai (Zimbrão, Miranda, de Souza, Estolano 2002). Šiuo būdu gali išreikšti verslo taisyklės duomenų bazėje arba jos gali būti išreikštos per teiginius (ang.

Assertions), kurie būtų pakankamai bendri. Duomenų bazės teiginys patikrinantis ribojimas nėra skirtas vienam lentelei (esybei) ir gali būti panaudotas kelioms lentelėms.

Tačiau net ir šis būdas nėra tinkamas, kadangi dauguma komercinių duomenų bazių sistemų neįgyvendina teiginių kaip apibrėžti SQL 92/99 standarte. Todėl vienas iš sprendimų kurti „rankinius“ trigerius. Tačiau tai tuo pačiu ir problema, kadangi net ir patys programuotojai kartais nesupranta verslo taisyklių išreikštų SQL. Nors pakankamai lengva suprasti ką SQL kodas daro, tačiau nelengva ką jis reiškia verslo kontekste (Zimbrão, Miranda, de Souza, Estolano 2002).

Tai viena iš pagrindinių priežasčių kodėl darbui pasirinkta dalykinės srities ontologija. Naudojant ontologiją taip pat išsprendžiamos ankščiau minėtos taisyklių rinkinio formavimo problemos. Todėl ją galima panaudoti verslo taisyklių sistemos (ir taisyklių rinkinio) formavimui ir modeliavimui, kuris gali būti įvykdytas transformavus verslo taisykles, ontologijoje aprašytas aksiomų pavidalu. Toliau aptarkime ontologijų sąvoką ir naudojimą.

2.6.1. Ontologijos sąvoka

Žodis „**ontologija**“ yra paimtas iš filosofijos, kur jis reiškia būties sistematinį paaiškinimą (Gomez-Perez, Gernandez-Lopez, Corcho 2004). Ontologija yra hierarchiškai struktūrizuotas ribojimų rinkinys, aiškiai ir detaliai aprašantis sritį. Šis rinkinys gali būti panaudotas kaip karkasinis pagrindas žinių bazei (Gomez-Perez, Gernandez-Lopez, Corcho 2004). Ontologija taip pat gali būti apibrėžta kaip tekstinis ar grafinis konceptualizacijos aprašymas (Murshed, Singh 2005), kuris gali būti panaudotas dalintis žiniomis, naudojantis panašų žodyną, semantiką ir ryšius tarp konceptų konkrečioje srityje. Taip pat ontologijos yra naudingos praktiškai aiškinant metaduomenų terminus, struktūrizuojant srities žinias, jos naudingos duomenų įprasminimo požiūriu, kadangi formaliu žodynu nurodomas kontekstas naudojamas aprašyti informaciją (Ressler, Dean, Benson, Dorner, Morris 2007). Dalykinės srities ontologija siekia sumažinti ar panaikinti koncepcinę ir terminologinę painiavą tarp bendruomenės narių, kurie naudojasi ir dalinasi tarpusavyje įvairios rūšies informacija. Tai atliekama identifikuojant ir tinkamai apibrėžiant svarbiausius konceptus, kurios naudojamos tam tikroje dalykinėje srityje. Ontologijos specifikuoja bendrą dalykinės srities supratimą.

Naudodami ontologijas kūrėjai gali sukurti geresnes sistemas, kurios aiškiau suprasta viena kitą. Sistemų kūrėjams jos teikia pakartotinį naudojimą, patikimumą ir specifikavimą (Navigli, Velardi, Gangemi 2003). Todėl ontologiją kurti ir išlaikyti apsimoka.

Ontologijos turi skirtingą palaikomą formalumo laipsnį, tačiau jos naudoja tokius metaduomenis kaip konceptai, ryšiai, aksiomos, egzemplioriai. Žvelgiant iš terminologinio taško, ontologija gali atrodyti kaip žodynas, saugantis formalių aprašų rinkinius, kurie sudaryti iš aksiomų. Todėl ontologijos žodynas gali būti naudojamas nusakyti terminų reikšmę ir leidžia nuosekliai atlikti

terminų bei jų tarpusavio ryšių interpretaciją (Navigli, Velardi, Gangemi 2003). Ontologijos sukūrimui reikia, kad dalykinės srities specialistai nuodugniai išanalizuotų dalykinę sritį:

- išnagrinėtų dalykinės srities žodyną, kuris bus naudojamas,
- sukurtų formalius terminų aprašus (formalizuotus į konceptus, ryšius ir jų egzempliorius), kurie bus naudojami šiame žodyne,
- charakterizuotų konceptų ryšius, kurie sieja ir naudoja terminus.

Tam, kad sutaupytume laiką ir kitus išteklius, būtina pasirinkti tinkamą ontologijos kūrimo įrankį. Tai pakankamai nelengva užduotis, kadangi yra nemažai įrankių, tačiau visi turi savų trūkumų ir privalumų (apie pasirinkimą žr. Ontologijų kūrimo įrankių analizė). Prieš pasirenkant tinkamą ontologijų kūrimo įrankį apžvelkime ontologijos modelio struktūrą ir ontologijos naudojimą IS kūrimo metu.

2.6.2. Ontologijos modelio struktūra

Dažniausiai ontologijos modelis yra sudarytas iš šių keturių komponentų:

- **Klasės** – išreiškia konceptus (Gomez-Perez, Gernandez-Lopez, Corcho 2004). Pavyzdžiui, turizmo kelionių organizavimo srityje tai gali būti vietovės (miestai, kaimai ir kitos), statiniai (tiltai, bažnyčios ir kiti) ir transportas (lėktuvai, traukiniai, automobiliai, laivai ir kitos).
- **Ryšiai** – išreiškia jungtis tarp dviejų srities konceptų. Dažniausiai ontologijos turi dvilypius ryšius (Gomez-Perez, Gernandez-Lopez, Corcho 2004).
- **Funkcijos** – specialus ryšių atvejis, kuriame ryšio n-tasis elementas yra unikalus kitiems n-1 ankstesniems elementams (Gomez-Perez, Gernandez-Lopez, Corcho 2004).
- **Formalios aksiomos** – modeliuojami sakiniai, nusakantys tiesą. Jos paprastai naudojamos išreikšti žinias, kurios negali būti formaliai apibrėžtos kitais komponentais (Gomez-Perez, Gernandez-Lopez, Corcho 2004). Taip pat formalios aksiomos yra naudojamos patikrinti pačios ontologijos ar saugomų žinių darną žinių bazėje.

Laikantis kitokio požiūrio į ontologijos modelio struktūrą, kiekvienas ontologijos modelis gali būti apibrėžtas trimis ašimis (Lohse, Valtchanov, Ratchev, Onori, Barata 2005): konceptų ryšių, hierarchijos ir abstrakcijos (konceptų) naudojimu. Modelio ryšiai nurodo kaip koncepcijos susijusios viena su kita. Hierarchija palengvina modelio suprantamumą, kadangi apibrėžus skirtingus detalumo lygius galima lengviau ir aiškiau suprasti sudėtingą modelio struktūrą. Hierarchija sudaroma grupuojant žemesnio lygio konceptus, kurie formuoja aukštesnio lygios konceptus. Abstrakcijos yra apibrėžiamos naudojant aukštesniųjų (ang. *super*) klasių ir poklasių (ang. *sub-class*) struktūrą, kurią konkrečiame

modelyje galima palaipsniui plačiau ir detaliau specifikuoti. Klasių struktūra, esanti skirtingų sričių ontologijose, apibrėžia aiškinamąjį terminų žodyną, kurį sudaro specifinės srities konceptus ir terminologija.

2.6.3. Ontologijos naudojimas kuriant sistemas

IS kūrimo procese galima išskirti du ontologijos naudojimo atvejus. Pirmu variantu, turime rinkinį pakartotinio naudojimo ontologijų, organizuotų ontologijos bibliotekoje, kurią sudaro sritis ir užduočių ontologijos. Pagal antrą scenarijų, pakartotinis naudojimas yra ribotas, kadangi turime tik bendro pobūdžio ontologiją, kurioje egzistuoja apibrėžtos skirtingos pamatinės pasaulio esybės, meta lygio klasės ir ryšių rūšys (Guarino 1998).

Pagal pirmą variantą, semantinis turinys, išreikštas ontologija (ar ontologijomis), pasirinkta iš ontologijos bibliotekos, panaudojamas kuriant informacinės sistemos komponentą. Tokiu būdu sumažinama koncepcinės analizės kaina ir užtikrinama, kad ontologija bus adekvati informacinei sistemai. Jei informacinė sistema kuriama tradiciniu būdu, toks turinys bus tiesiog prijungtas prie standartinių komponentų. Tačiau, jei tai bus ontologijos pagrindu veikianti informacinė sistema, tai kūrimo fazės rezultatas bus atskiras komponentas, kurį galime įvardinti kaip taikomoji ontologija. Šioje taikomojoje ontologijoje apjungiamos tiek srities ontologija, tiek užduočių ontologija.

Lyginant su inžinerijai skirta programine įranga, ypatingai svarbi ontologijos teikiama nauda sistemos kūrime – aukštesnio lygio pakartotinis naudojimas. Taikomas žinių pakartotinis naudojimas vietoje programinės įrangos. Dar daugiau, ontologija teikia kūrėjui lankstaus modifikavimo galimybę, kadangi galima pakartotinai panaudoti ir dalintis tos pačios srities žiniomis, naudojant vieningą žodyną įvairiose programinės įrangos platformose. Sistemos kūrėjas taip pat gali pernelyg nesijaudinti dėl įgyvendinimo detalių, o koncentruotis į nagrinėjamos srities struktūrą ir atliekamą užduotį.

Pagal antrą variantą, ontologijos žinių kiekis gali būti mažesnis, tačiau jis turi būti kokybiškesnis. Turi būti apibrėžti ontologijos elementų fundamentiniai skirtumai, kurie gali padėti projektuotojui atliekant koncepcinę analizę. Praktikoje ontologija naudojama ne kaip statybinės dalys, kurios bus pritaikomos ir pakartotinai naudojamos, tačiau daugiau kaip galingas įrankis. Toks įrankis gali pagerinti analizavimo proceso kokybę, padidindamas aiškumą. Šiuo būdu sukuriama taikomoji ontologija, naudojantis reliatyviai maža pamatine ontologija, todėl nebūtina turėti jau sukurtą srities ontologiją.

Ontologija pagrįstas informacinės sistemos kūrimas gali būti panaudotas ne tik kuriant naują informacinę sistemą, tačiau ir efektyviau pertvarkant esamą informacinę sistemą (atliekant reinžineriją). Reinžinerijos metu bus padidintas kūrime naudojamų priemonių pakartotinis panaudojimas ir valdymas. Šiuo būdu gali būti sumažintas didelis kiekis programinės įrangos investicijų, kurios buvo atliktos praeityje kuriant sistemą (Guarino 1998).

Šiuo atveju mes dalykinės srities ontologiją aprašius kompiuterinėje sistemoje, taip pat nurodyme ir egzistuojančias aksiomas. Vėliau aprašytas aksiomas galime perkelti į kitų sistemos kūrimo priemonių palaikomas kalbas, kurių aksiomas galima naudoti tolesniame modeliavime arba net sistemos kūrimo objektus (pvz., trigerius). Tokiu būdu užtikrinamas pakartotinis panaudojimas keičiant ar kuriant sistemą. Taip pat pasinaudojama ontologijos naudojimo privalumais, kurie palengvina sistemos kūrimą.

2.7. Ontologijų kūrimo įrankių analizė

Šiuo metu egzistuoja daugiau nei 50 (Denny 2004) įvairių: komercinių, atviro kodo ir riboto platinimo ontologijų kūrimo įrankių.

Pasirenkant buvo nagrinėti įvairūs populiariausių įrankių palyginimai, nustatyti jų pagrindiniai trūkumai bei sudarytas pagrindinių įrankių palyginimas pagal konkrečius reikalavimus. Pateikti du atlikti palyginimai. Tas įrankis bus tinkamiausias, kuris turės mažiausiai trūkumų ir/arba tenkins visus iškeltus reikalavimus. Pateikta ontologijų kūrimo įrankių palyginimo 1 lentelė (Su, Ilebrekke, Banks 2002), kurioje palyginti šeši ontologijų kūrimo įrankiai: Ontolingua, WebOnto, WebODE, Protégé, OntoEdit OilEd.

1 lentelė. Šešių įrankių palyginimas

Įrankių įvertinimo lentelė								
Parametrai \ Įrankiai			**Ontoligua**	**WebOnto**	**WebODE**	**Protégé**	**OntoEdit**	**OilEd**
Fizinė kokybė	**Metamodelio adaptacija**	**Išraiška**	Aukšta	Vidutinė+	Vidutinė+	Vidutinė-	Vidutinė-	Vidutinė
	Saugojimas	**Serveryje**	Serveryje	Serveryje	Serveryje	Lokalus	Lokalus	Lokalus
	Prieinamumas	**Tinklinė**	Taip	Taip	Taip	Ne	Ne	Ne
		Eksportavimas	KIF, Loom, OKBC	Nėra	XML, RDF(S), OIL	RDF(S) ir kitas	F-logic, DAML-OIL	OIL, DAML-OIL, OIL, RDF (S)
Empirinė kokybė			Silpna+	Silpna+	Gera-	Silpna+	Gera-	Gera
Sintaksinė kokybė			Klaidų tikrinimas	Klaidų prevencija	Klaidų prevencija	Klaidų prevencija	Klaidų prevencija	Silpna
Semantinė kokybė	**Darnos tikrinimas**		Silpnas+	Silpnas+	Geras-	Silpnas+	Geras-	Geras
	Modelio daugkartinis panaudojimas		Biblioteka ir integravimas	Biblioteka	Biblioteka ir integracija	Integracija	Nėra	Nėra
Pateikiamumas	**Instrukcijos**		Yra	Yra-	Yra-	Yra	Yra-	Yra-
	Pagalbiniai pranešimai		Nėra	Nėra	Yra-	Yra	Yra	Yra
Pragmatinė kokybė	**Vizualizacija**		Silpna	Gera-	Gera	Gera-	Gera-	Gera-
	Filtravimas		Silpnas+	Geras-	Geras-	Geras-	Geras-	Geras-
	Paaiškinimai		Silpnas+	Silpnas+	Geras	Geras	Silpna+	Geras-
Socialinė kokybė			Modelio integr.	Silpna	Modelio integr.	Modelio integr.	Silpna	Silpna

Fizinė kokybė palyginta pagal tris kriterijus: metamodelio adaptavimą, saugojimą ir prieinamumą. Semantinė kokybė lyginta pagal šiuos parametrus: pastovus tikrinimas ir modelio daugkartinis panaudojamas. Pateikiamumo kokybė (suvokiamumas) lyginama pagal dokumentacijos lygį ir įrankio pagalbinius pranešimus. Pragmatinė kokybė buvo vertinta pagal vizualizaciją, filtraciją, paaiškinimą ir vykdymą. Kiti vertinimo parametrai yra empirinė, sintaksinė ir socialinė kokybė.

Fizinė kokybė yra siejama su naudojamas kalbos tinkamumu nagrinėjamai sričiai. Įrankių metamodeliai turi turėti kelias perspektyvas, sietis su sritimi ar būti pakankamai ekspresyvus, kad patenkintų naudotojo poreikius. Ontolingua, WebOnto ir WebODE turi aukštesnį įvertinimą nei kiti trys įrankiai. OilEd praplečia Protégé ir OntoEdit ekspresyvumą. WebODE-e yra įtraukti konstruktoriai agregavimams ir tam tikri žingsniai buvo atlikti OntoEdit-e. Visi įrankiai turi funkcijas, teisingai išreiškiančias agregacijas. Ontolingua, WebOnto ir WebODE leidžia ontologijas saugoti serveryje, kiti trys įrankiais leidžia saugoti vietinėje saugykloje. Ontolingua turi eksportavimą į KIF, OKBC, Loom ir kitas kalbas, bet ne į tinklo kalbas. WebOnto neturi eksportavimo mechanizmo. Kiti įrankiai eksportuoja į kelias tinklo kalbas (Protégé su papildymais eksportuoja į dar daugiau kalbų). Tik keli įrankiai teikia grąžinimo atgal (ang. *undo*) funkciją, tačiau jų veikimas silpnas.

Empirinė kokybė arba kalbos suvokiamumo lengvumas. Ontolingua modeliai sunkiausiai suprantami ir yra tam tikrų problemų su grafiniu vaizdavimu. Protégé, OntoEdit ir OilEd, panašiai kaip Ontolingua, taksonomijų vaizdavimui naudoja tekstą, tačiau grafinis vaizdavimas naudojamas parodyti sprendimo tipus. WebOnto turi gerą grafinį vaizdavimo mechanizmą, bet jis nėra automatizuotas. Jo teikiamas objektų sąrašas turi minusą, kadangi jis nerodo skirtingų objektų tipų. WebODE teikia gerą tekstinį ir grafinį vaizdavimą.

Sintaksės kokybė skiriasi visuose įrankiuose. Ontolingua ir Protégé aptinka sintaksės klaidas. WebOnto, WebODE ir OntoEdit atlieka sintaksės klaidų prevenciją, ribojamos vaizdavimo galimybes. OilEd turi tendenciją dažnai sutrikti ir nepateikti naudotojui pranešimo, kad vaizdavimas neleistinas.

Semantinė kokybė priklauso nuo modelio fizinės ir sintaksės kokybės. Visi įrankiai turi tam tikrą modelio įvykdomumo ir užbaigtumo patikrinimo lygį. Ontolingua, WebOnto ir Protégé įdiegtas darnos tikrinimas, WebODE teikia geras tikrinimo paslaugas, tačiau OilEd turi galingiausią loginį mąstymą (FaCT). OntoEdit neturi integruoto loginio mąstymo, tačiau artimai susietas su OntoBroker išvadų darymo mašina. Protege turi modelio integravimo mechanizmus. WebOnto turi biblioteką, tačiau nepalaiko integracijos. OntoEdit ir OilEd nepalaikomas pakartotinis panaudojimas.

Pateikiamumo kokybę (suvokiamumą) sudėtinga nustatyti, kadangi tai priklauso ir nuo naudotojų žinių. Visi įrankiai turi tam tikras naudojimo instrukcijas, tačiau labiausiai išsamios ir išbaigtos yra Ontolingua ir Protégé įrankių instrukcijos. Pagalbinius pranešimus teikia WebODE, Protégé, OntoEdit ir OilEd.

Pragmatinė kokybė priklauso nuo fizinės, empirinės ir sintaksės kokybės. Ontolingua turi filtravimo ir vizualizacijos mechanizmus, kurie yra dokumentuoti, tačiau nepateikta išsamių generavimo paaiškinimų. WebOnto vienintelis įrankis, kuris siūlo modelių vykdymą. Tačiau nėra ryšių ir konceptų vienalaikio vaizdavimo. WebODe, Protégé, OntoEdit ir OilEd teikia panašius filtravimo ir vizualizacijos mechanizmus, tačiau OntoEdit blogiau atlieka generavimą.

Kitas 9 įrankių palyginimas (da Silva, Vankeisbelck, Lima 2004) buvo atliktas pagal šiuos kriterijus: modeliavimo galimybės, pagrindinė kalba, importavimo/eksportavimo formatai, grafininis vaizdavimas, darnos tikrinimas, leksinis palaikymas. Buvo nagrinėjami šie įrankiai: LexiCon Explorer, OilEd, Protégé, Onto Edit Free, LinkFactory, E-COSer, Terminae, Text-to-Onto, OntoLearn.

2 lentelė. Devynių įrankių palyginimas

Įrankis	**Modeliavimo galimybės**	**Pagrindinė kalba**	**Importavimo/eksportavimo formatai**	**Grafinis vaizdavimas**	**Darnos tikrinimas**
LexiCon Explorer	Sukurtas kaip objektų biblioteka pagal ISO DIS 12003-3 standartą. Konceptų aprašymas, apibrėžimas ir grafinis vaizdavimas, predikatai ir nuorodos.	RDB mySQL duomenų bazėms	XML eksportavimas ir importavimas iš IFC.	Tik per Lexicon Explorer naršyklę	Tam tikri darnos tikrinimo mechanizmai, nėra programiškai įdiegto tikrinimo
OilEd	DAML suvaržymų aksiomos, XML schemų duomenų tipai, metaduomenų kūrimas, tarpinės ekspresijos naudojamos užpildyti suvaržymų aksiomoms, detalus kiekų išreiškimas, vienas individų sąrašas, nėra hierarchinio savybių vaizdavimo.	DAMN+OIL (OWl kuriamas)	Importuoti iš OWL RDF/XML, GO, OIL Text ir SHIQ, eksportavimas į paprastą RDFS, SHIQ ir OWL.	Tik Graphviz failų klasių vaizdavimas	Serverio programa tikrina DAML+OIl sintaksę; naudojamas FaCT ir RACER
Protégé	Ryšių hierarchija ir daugialypio paveldėjimo koncepcijos. Meta klasės. Palaiko egzempliorių specifikaciją, F-Logic, OIL ir aksiomų kalba (PAL) ir su plėtiniais OWL.	OKBC modelis	RDF, XML schema, RDB su Data Genie praplėtimu, DAML+OIL ir OWL	EzOWL vaizdo redaktorius, Jambalaya vaizdavimo įrankis su SHriMP (Simple Hierarchical	DIG loginio mąstymo įrankis (RACER), Facet Constrain tab, kuris identifikuoja ir

				Multi-Perspective), OntoViz ir TGViz.	taiso egzempliorius, PAL išreikia žinių bazės suvaržymus ir galima daryti logines užklausas žinių bazei.
Onto Edit Free	F-Logic aksiomos klasės ir ryšiams, algebrinės ryšių savybės. Metaduomenų kūrimas. Riboti DAML savybių suvaržymai ir duomenų tipai, nėra klasių kombinacijų, vienodi egzemplioriai.	F-Logic	RDFS; F-Logic; Excel, DAML+OIL (ribota); RDB (SQL schema)	Vaizdavimui įrankis su plėtiniu	SiLRi (RDF interpretavim.)
LinkFactory	Klasių hierarchija, koncepcijų egzemplioriai, daugialypė ryšių hierarchija, sinonimų apibrėžimas, jungtinės sąlygos keliomis kalbomis.	Informacija neprieinama.	Informacija neprieinama.	Informacija neprieinama.	Savo išvadų kūrimo mechanizmas
E-COSer	Valdo ontologijų objektus: konceptus, ryšius, prašus ir egzempliorius.	DAML+OIL	Eksportuoja į DAML+OIL, importuoja bcXML.	Nėra	Nėra
TERMINAE	Valdo terminologijos failus (sąlygoms), bendri konceptai (klasėms), primityvūs konceptai (egzemplioriams) ir primityvūs vaidmenys (ryšiams).	XML	Importuoja OWL ir eksportuoja OIL formatus	Taip, bet ne automatinis	Galimas funkcijų klasifikavimas patikrinti bendrą koncepcijos teisingumą
Text-to-Onto	Tekstinės sąlygos kaip potencialūs konceptai ir ryšiai. Potencialūs taksonominiai ir ne taksonominiai ryšiai.	KAON formatas	RDF(S)	Taip, per KAON programa	Netaikomas
OntoLearn	Konceptai, taksonominiai ir ne taksonominiai sąryšiai ir	Informacija neprieina-	Informacija neprieinama.	Informacija neprieinama.	Informacija neprieinama.

	koncepcinis miškas (grafas)	ma.			

Šios analizės autoriai nusprendė, kad Protégé yra labiausiai rekomenduotinas įrankis (da Silva, Vankeisbelck, Lima 2004) , kadangi palaiko OWL yra nemokamas įrankis bei turi gausią kūrėjų ir naudotojų bendruomenę.

2.7.1. Pasirinkimo kriterijai

Tam, kad transformacija būtų sėkminga, pirmiausia reikia pasirinkti tinkamiausią įrankį, kurį naudojant sukurta ontologija bus panaudota aksiomų išgavimui. Skirtingi įrankiai palaiko įvairias ontologijų kalbas, pasižymi skirtinga struktūra bei skiriasi kitais kriterijais. Parametrai buvo suformuoti atsižvelgiant į tai, kad būtų užtikrintas būsimo įrankio tinkamumas darbui, aksiomų išgavimui, prieinamumas, patogumas bei įrankio naujumas. Pagrindiniai parametrai, kuriuos turi tenkinti ieškomas įrankis:

- Turi turėti aksiomų palaikymą ar aksiomų kalbą;
- Turi būti nemokamas (geriausia atviro kodo);
- Turi turėti draugišką vartotojo sąsają ir grafiškai atvaizduojamą taksonomiją;
- Turi būti atnaujinamas, aktyvus projektas;
- Turi būti galimybė įrankiui sukurti ir prijungti plėtinius (ang. plug-in).

Naudoajnt paminėtus ir kitus įrankių palyginimus buvo sudaryta 3 ir 4 lentelės, kuriose įrankiai lyginami pagal suformuotus pagrindinius kriterijus.

3 lentelė. 1-as palyginimas pagal suformuotus kriterijus

Reikalavimas	**WebODE**	**OilEd**	**Ontolingua**	**DUET**	**Protégé**
Ar turi aksiomų palaikymą, kalbą?	**Taip**	**Taip** (Casely-Hayford 2005)	**Taip** (Casely-Hayford 2005)	Ne	**Taip**
Ar nemokamas, atviro kodo	**Taip**	Mokama, nemokama vers. su mažiau galimybių	**Taip**, prieiga per internetą	Ne	**Taip**
Ar draugiška vartotojo sąsaja, grafinis taksonomijos vaizdavimas?	**Taip**	Ne	**Taip**	**Taip**	**Taip**
Ar aktyvus projektas?	Ne	Ne	**Taip**	**Taip**	**Taip**
Ar yra galimybė kurti	**Taip** (Casely-	**Taip** (Casely-	Ne	Ne	**Taip**

plėtinius?	Hayford 2005)	Hayford 2005)			

4 lentelė. 2-as palyginimas pagal suformuotus kriterijus

Reikalavimas	**Chimaera**	**OntoSaurus**	**DAD-Edit**	**OntoEdit**	**WebOnto**
Ar turi aksiomų palaikymą, kalbą?	**Taip**	**Taip** (Casely-Hayford 2005)	**Taip** (Casely-Hayford 2005)	**Taip**	**Taip** (Casely-Hayford 2005)
Ar nemokamas, atviro kodo	Atviro kodo, riboto laiko naudojimas (Casely-Hayford 2005)	Dalinai	Nemokamas	Dalina, tik prieiga per internetą	Dalinai
Ar draugiška vartotojo sąsaja, grafinis taksonomijos vaizdavimas?	**Taip**	Ne	Ne	Ne	**Taip** (Casely-Hayford 2005)
Ar aktyvus projektas?	**Taip**	**Taip**	Ne	**Taip**	Ne
Ar yra galimybė kurti plėtinius?	Ne	Ne (Corcho, Fernandez-Lopez, Gomez-Perez 2002)	Ne	**Taip**	Ne (Corcho, Fernandez-Lopez, Gomez-Perez 2002)

Kiti nustatyti įrankių trūkumai:

- OntoEdit – neturi integruoto loginio mąstymo;
- OntoEdit ir OilEd - nepalaikomas pakartotinis panaudojimas;
- OilEd – nėra hierarchinio savybių vaizdavimo, turi tendenciją dažnai sutrikti ir nepateikti naudotojui pranešimo, kad vaizdavimas neleistinas;
- OntoSaurus, WebODE – neturi ontologijų bibliotekos;
- WebODE – nepalaiko modelio integracijos.

Protégé tenkino iškeltus reikalavimus, todėl buvo atlikta detalesnė įrankio analizė:

- nemokamas, atvirjoo kodo įrankis;

- integruotas plėtinių rinkinys, turintis daug papildomų plėtinių;
- turi paprastą vartotojo sąsają, kuri palengvina ontologijų kūrimą ir valdymą;
- turi grafinį taksonomijos vaizdavimą;
- turi lanksčią Protégé struktūrą;
- turi didelę naudotojų ir kūrėjų bendruomenę;
- turi PAL (*Protege Axiom Language*) bei plėtinį EZPAL, kurio dėka galima lengvai kurti aksiomas;
- pateikiama išsami dokumentacija, instrukcijos;
- palaiko OWL;
- su plėtiniais leidžia eksportuoti į RDF, RDFS, DAML+OIL, XML, OWL, Clips, UML;
- suri grafinius įrankius, kurie naudingi kuriant sudėtingas semantinio tinklo kalbas, tokias kaip RDF ar OWL;
- nuolatos atnaujinamas, leidžiami klaidų ištaisymai.

Pagal šešių įrankių lyginamąją analizę Protégé buvo lygiavertis įrankis. Minėtoje palyginamojoje analizėje Protégé buvo gerai įvertintas lyginant empirinę, sintaksės, pateikiamumo ir pragmatinę kokybę. Kitame devynių įrankių palyginimo metu buvo išskirtas būtent Protégé. Darbui pasirinktas Protégé įrankis, kadangi pagal suformuotų reikalavimų tenkinimo lenteles matyti, kad tik vienintelis Protégé įrankis tenkina visus iškeltus reikalavimus (ir dar turi savo papildomų privalumų).

2.7.2. Protégé

Pasirinktas Protégé yra nemokamas, atviro kodo ontologijų žinių bazės kūrimo ir redagavimo įrankis. Protégé platforma palaiko du pagrindinius ontologijų modeliavimo būdus: Protégé-Frames ir Protégé-OWL redaktorius. Protégé ontologijos gali būti eksportuojamos į įvairius formatus tame tarpe RDF(S), OWL ir XML schema.

Protégé yra parašyta Java kalba, yra lengvai plečiamas, naudoja „prijunk ir paleisk“ (ang. *plug-and-play*) aplinką, kurios dėka galima greitai bei lengvai prijungti ir paleisti veikti plėtinius (Stanford Medical Informatics 2007).

Nėra abejonių, kad ateityje viena iš pamatinių semantinio tinklo dalių bus ontologijos. Tam yra labai tinkamas Protégé, kadangi jo kūrėjai ištobulino schemų panaudojimą, ontologijų kūrimą ir visos žinių bazės valdymą. Šis įrankis patenkina pagrindinius ontologijų kūrėjų poreikius (Alani, O'Hara, Shadbolt 2005):

- Protégé turi paprastą vartotojo sąsają, kuri palengvina ontologijų kūrimą ir valdymą;
- turi grafinius įrankius, kurie labai naudingi kuriant sudėtingas semantinio tinklo kalbas, tokias kaip RDF ar OWL;

- suteikia galimybę ontologijų kūrėjams vykdyti redagavimo schemos užklausas ir palyginti su kitomis ontologijomis, išplėsti esamą schemą ir sukurti egzempliorius.

Kitas svarbus Protégé privalumas yra jos atviras kodas ir plečiamumas. Daug įvairių papildomų įrankių galima lengvai integruoti į Protégé, tai stipriai padidina programos vertę. Protégé populiarumas ypatingai padidėjo jai tapus atviro kodo programa. Ji turi lengvai suprantamą, draugišką vartotojo sąsają. Pateikiama išsami dokumentacija ir užtikrinamas Protégé bendruomenės narių tarpusavio ryšys per elektroninį paštą, trumpus kursus ir konferencijas, ko trūksta daugumai panašaus pobūdžio įrankių (Alani, O'Hara, Shadbolt 2005). Nuolat kuriami atnaujinimai, klaidų pataisymai. Prie pateiktų Protégé privalumų galima pridurti, kad Protégé su plėtiniais leidžia eksportuoti į RDF, RDFS, DAML+OIL, XML, OWL, Clips, UML formatus bei plėsti ir tobulinti ontologijų, ribojimų aksiomų kūrimą (Denny 2004), palaiko išvadų darymą bei galimybę prijungti naujus plėtinius. Palaiko KADS metodologiją ir pilnai palaiko Unicode formatą.

Protégé-OWL palaiko tiek OWL tiek ir tradicinius Protégé ontologijų formatus. Su Protégé-OWL galima kurti ir redaguoti OWL ir Protégé ontologijas. Protégé-OWL lanksti architektūra leidžia lengvai konfigūruoti ir plėsti šį įrankį. Šie plėtiniai suteikia naujų funkcionalumo galimybių. Jų dėka darbas su aksiomomis - lengvas ir patogus. Protégé-OWL kūrėjas yra Stenfordo universitetas. Protégé glaudžiai integruota su Jena ir turi atviro kodo Java taikomosios programos programavimo sąsają tolesniam jos kūrimui (Stanford Medical Informatics 2007).

OWL yra pakankamai naujas standartas tarp ontologijų kalbų. Jį patvirtino *World Wide Web Consortium* (W3C) skatindama semantinio tinklo viziją. OWL yra aprašomos klasės, savybės ir egzemplioriai. OWL formali semantika specifikuoja kaip išvesti logines pasekmes, kadangi tiesiogiai faktai nėra vaizduojami ontologijoje, o tik aprašoma semantika. Šie aprašymai gali būti paremti vieninteliu dokumentu ar keliais išskirstytais dokumentais, kurie kartu naudoja OWL mechanizmus.

Protégé-OWL leidžia naudotojams:

1) Užkrauti ir saugoti OWL ir RDF ontologijas;
2) Redaguoti ir vaizduoti klases, savybes ir SWRL taisykles;
3) Apibrėžti logines klasių charakteristikas OWL išraiška;
4) Vykdyti loginius argumentus pagal loginių klasifikatorių aprašą;
5) Redaguoti OWL individus semantinio tinklo žymėjimui.

Be OWL palaikymo dar galima paminėti tai, kad pradinė Protégé-OWL turi šiuos plėtinius (galima ir daugiau prijunkti): OWL plug-in, RDF Backend, XML Tab WordNet Tab, UMLS Tab, String Search Tab, Script plug-in, PSM librarian, Prompt Tab, Pal Tabs, OWL Wizard, Jess Tab plug-in, Jambalaya, Basic Wizards, Algernon, EZPAL Tab.

Protégé-OWL tenkina visus ankščiau iškeltus reikalavimus įrankiui, todėl toliau bus kalbama apie darbą su juo ir jam kuriamą plėtinį (ang. *plugin*), kuris leis įgyvendinti siekiamą taisyklių transformaciją.

2.8. Analitinės dalies apibendrinimas

Modelio tipo transformacijas verta naudoti dėl ryšių ir atvaizdų savybių, kadangi

- modeliuotojas gali naudoti ryšius pradinėse sistemos kūrimo stadijose, kai įgyvendinimo sąlygos dar nėra galutinai aiškios (tai leidžia tiksliau apibrėžti transformacijas prieš nusprendžiant įgyvendinimo detales),
- šis tipas leidžia nuosekliai atlikti vienakryptę transformaciją iš šaltinio modelio į tikslo modelį.

Atlikus transformavimo kalbų analizę, padaryta išvada, kad tinkamam tikslo modelio kūrimui ir transformacijos įgyvendinimui būtinas visų galimų aibės elementų transformavimo aprašymas transformavimo taisyklėmis. Šios taisyklės transformacijos metu apdorotų šaltinio modelio elementus ir į tikslo modelio elementus.

Galima teigti, kad aprašius visas egzistuojančius šaltinio ir tikslo schemų egzempliorių dalių variacijas galima atlikti atvaizdavimo transformaciją. Tačiau tam būtina tinkamai ir aiškiai suprasti pradinį modelį ir galutinį transformacijos rezultatą.

Buvo aprašyti transformuojamų taisyklių aspektai susiję su verslo taisyklių sistemos formavimu ir modeliavimu. Išskirta pagrindinė problema sudėtingas verslo taisyklių rinkinio formavimas, kurią spręstų verslo taisyklių transformacija išgavus jas iš ontologijos, kur jos parašytos aksiomų pavidalu, į duomenų apdorojimo taisykles. Aprašyti įvairūs kandidatų verslo taisyklės apibrėžti pasirinkimo atvejai nuo tinkamo iki netinkamo. Aptarti keli laiko pasirinkimo variantai taisyklėms apibrėžti bei tai kaip gali įmonę paveikti įdiegta sistema. Taip pat buvo aptartas ne mažiau svarbus dalykas – verslo taisyklių metodo galima panaudojimo aplinka. Buvo apibūdintos įvairios įmonės, kurioms reikalinga ir nereikalinga verslo taisyklių mašina ir verslo taisyklių metodas, nurodytas jų tinkamumas verslo taisyklių metodui.

Verslo taisyklių transformacija padėtų spręsti šias problemas:

- sudėtingas verslo taisyklių rinkinio formavimas:
    - vieningos terminų sampratos trūkumas;
    - sudėtinga suformuoti darnų ir nuodugnų verslo taisyklių modelį;
    - sudėtingas verslo taisyklių išgavimas.

Kitos problemos, susijusios su verslo taisyklių sistemos formavimu ir modeliavimu:

- tinkamo taisyklių apibrėžimo laiko pasirinkimas;
- tinkamos verslo taisyklių metodo panaudojimo aplinkos parinkimas;

    - neatsižvelgimas į tai, kaip verslą paveiks įdiegta verslo taisyklių mašina.
- Taisyklių transformacijos metodui buvo pasirinktos verslo taisyklės, kadangi taikant taisyklių transformacijos metodą galima lanksčiau ir efektyviau atlikti pakeitimus, sutrumpinamas projekto kūrimo laikas, sumažėja kaina, kadangi nemaža dalis darbo atliekama automatizuotai.
- Taisyklių transformacijos objektu pasirinktos verslo taisyklės, kadangi jos gali būti išreikštos formaliomis kalbomis, kurios būtinos taisyklių transformacijai. Formaliai užrašytas verslo taisykles galima automatizuotai įdiegti ir panaudoti informacinėje sistemoje bei turi šiuos verslo taisyklių metodo privalumus sistemų kūrimo metu:
    - jos yra paprastos ir aiškios;
    - žinių inžinerijoje egzistuoja didelė teorinė ir praktinė patirtis;
    - taisyklės nėra susietos su technologija;
    - pakartotinis naudojimas, ir jos gali būti naudojamos dar neužpildžius duomenų bazės duomenimis;
    - supaprastintas sistemos projektavimas;
    - yra dinamiškos, jas galima greitai keisti;
    - sistemos kūrimą leidžia atlikti dalimis;
- Išnagrinėjus verslo taisyklių naudojimą informacinių sistemų gyvavimo cikle buvo nustatyta, kad verslo taisyklės yra svarbus ir neatsiejamas informacinės sistemos aspektas, leidžiantis įgyvendinti sistemos aktyvią elgseną. Verslo taisyklių modeliavimas, kaip atskiro komponento, įgalina operatyviai reaguoti į pokyčius versle.
- Buvo nustatyta, kad ontologija kūrėjui teikia lankstaus modifikavimo galimybę, kadangi galima pakartotinai panaudoti ir dalintis taikomosiomis srities žiniomis, naudojant tą patį žodyną įvairiose programinės įrangos platformose.
- Buvo nustatyta, kad metodui realizuoti, tinkamiausias ontologijų kūrimo įrankis – Protégé (Protégé-OWL), kuris tenkina visus suformuotus kriterijus.

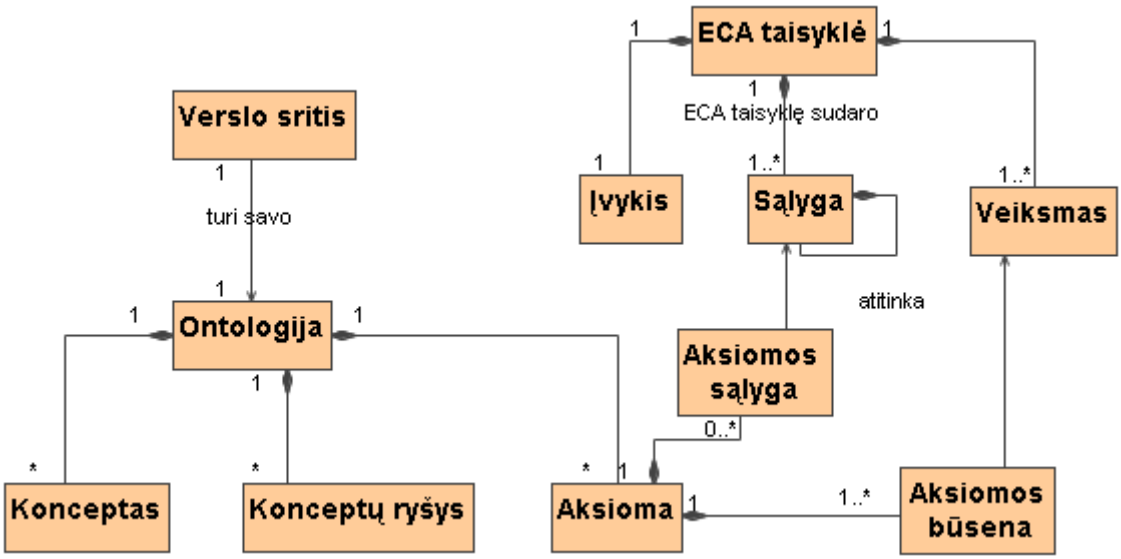
3. PROJEKTINĖ DALIS

3.1. Dalykinės srities ontologijos verslo taisyklių sistemos formavimo ir modeliavimo metodas

Kiekviena verslo sritis kartu su jos taisyklėmis turi savo ontologiją, kurią galima formalizuoti naudojant tam tikrą kalbą. Darbe vėliau kalbėsime apie Protege kalba sukurtą ontologiją. Ontologijoje aprašomos aksiomos, konceptai ir jų ryšiai. Pasinaudodami dalykinės srities ontologijoje apibrėžtomis aksiomomis galime automatizuotai formuoti ir modeliuoti verslo taisyklių sistemos taisykles į aktyviąsias taisykles, išreikštas SQL trigeriais arba UML naudojamą OCL ribojimus (sintaksę). Verslo taisyklių sistemos automatinis formavimas įgalina taupyti laiką ir finansinius išteklius, kadangi aksiomų nereikia aprašinėti ir interpretuoti darbuotojams, kuriantiems duomenų bazę rankiniu būdu. Be to, tai leidžia išvengti darbuotojų klaidų.

Aksiomomis modeliuojami sakiniai, nusakantys tiesą. Aksiomose nusakomos sąlygos, kurias turi tenkinti ontologijoje aprašyti konceptai. Iš aksiomos gali būti išskirtas dinaminis ribojimas. Nors egzistuoja įvairios taisyklių taksonomijos, tačiau įdiegimo prasme visos taisyklės gali būti klasifikuojamos į struktūrinius ir dinaminius ribojimus (Būgaitė, Vasilecas 2005). Struktūriniai ribojimai nurodo esybių apibrėžimus ir tarpusavio ryšius. Dinaminiai ribojimai skirstomi į išvestines taisykles ir reaguojančiąsias taisykles. Dinaminės reaguojančios taisyklės gali būti išreikštos ECA taisyklės modeliu, o ECA taisyklė transformuota į SQL trigerį (Būgaitė, Vasilecas 2005) ar OCL sakinius.

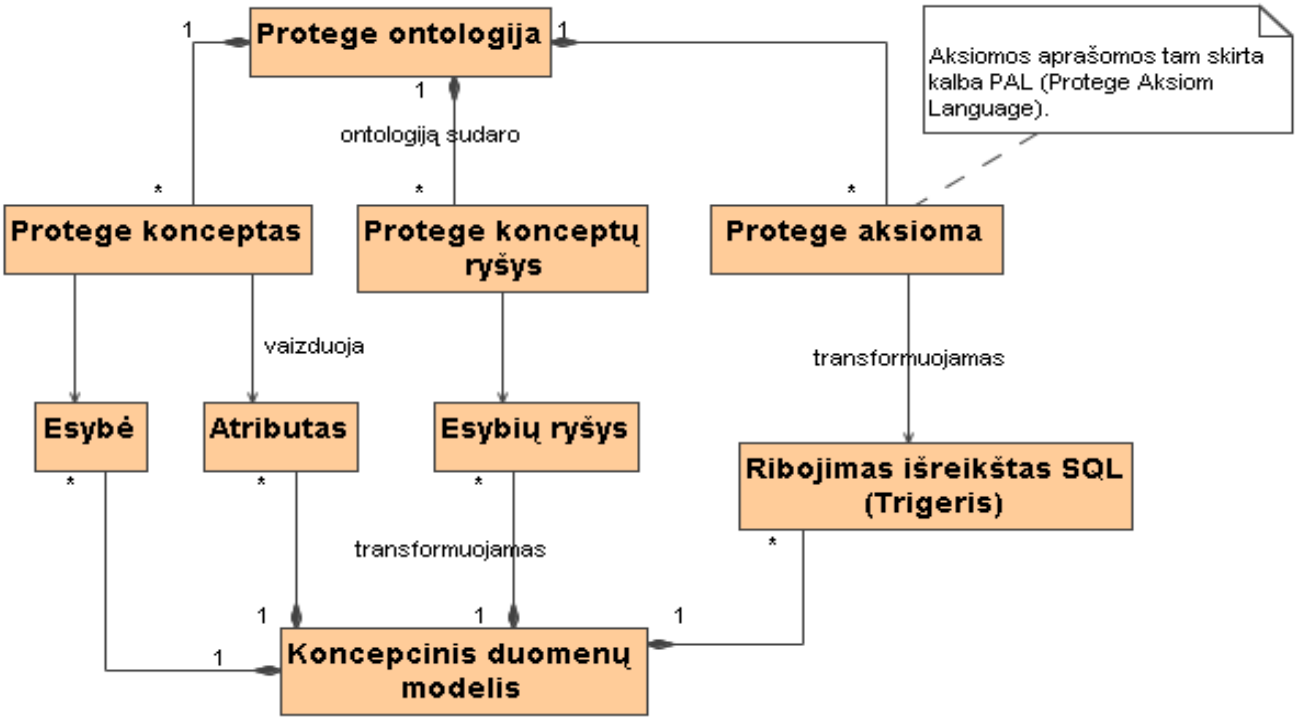
Aksiomos veiksmas ir sąlygos atitinka ECA veiksmą ir sąlygas (Būgaitė, Vasilecas 2005). Todėl aksiomos gali būti nagrinėjamos ECA taisyklių požiūriu bei kiekviena aksioma gali būti transformuota į ECA taisyklę, kuri sudaryta iš trijų elementų: įvykio, sąlygų ir veiksmo. T.y., iš aksiomų galima išgauti sąlygą(s) ir kartais panaudojamą būseną, kuri atitinką ECA taisyklės veiksmo dalį. Tai vaizduoja žemiau pateiktas paveikslas.



4 pav. Metodas, kuriuo aprašomas dalykinės srities aksiomų transformavimas (Būgaitė, Vasilecas 2005), pavaizduotas klasių diagrama

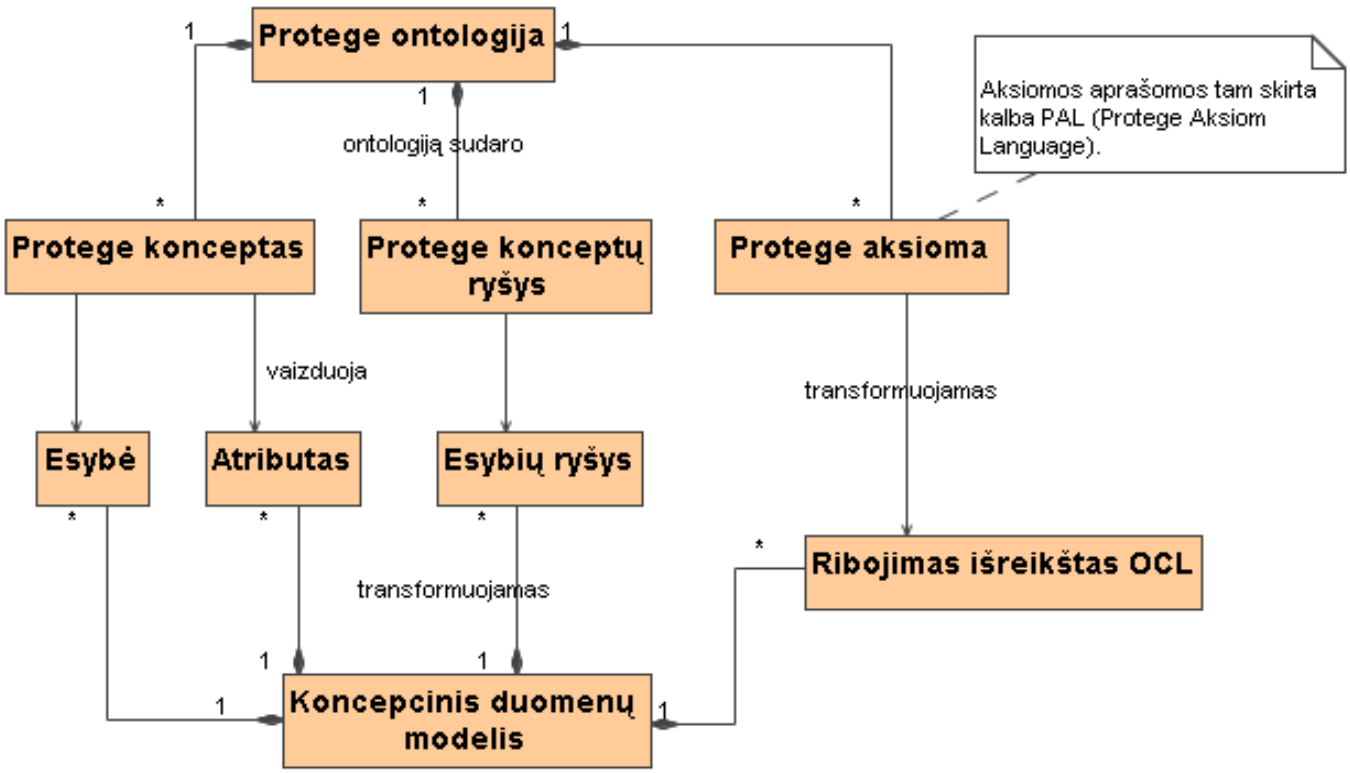
Aksiomos neegzistuoja atskirai nuo pačios dalykinės srities, todėl turi būti vykdomas lygiagretus koncepcinio modelio transformavimas. Turėtų būti gautos sistemos esybės ir atributai, kurias naudos SQL trigeris. Ontologijos konceptus galime paversti į duomenų bazės esybes su atitinkamais atributais. O konceptų ryšius – į ryšius tarp esybių.

Kadangi aksioma vietoje veiksmo dalies turi būseną, taisyklėje nurodomas dvejopas pranešimų išvedimas, kai tenkinamos ir/arba kai pažeidžiamos sąlygos. Jei būtų pažeista sąlygomis apibrėžta esybių darna, iš aksiomos suformuota taisyklė išvestų pranešimą apie klaidą, arba patvirtintų, kad tenkinamos sąlygos. Pirmoji metodo realizacija, transformavimas į SQL trigerį, pavaizduota grafiškai žemiau pateiktame paveiksle. Pavaizduota metodo transformacija atlikta panaudojus laikraščio ontologiją.



5 pav. UML klasų diagrama pavaizduota PAL aksiomos transformacijos schema į SQL išreikštą ribojimą (trigerį)

Kaip ir PAL aksiomų į SQL trigerio transformacijos realizacijos atveju, pasinaudodami dalykinės srities ontologijoje apibrėžtomis aksiomomis galime automatizuotai formuoti OCL sakinius ir juos vėliau naudoti UML aplinkoje. OCL ir PAL kalbų suderinamumo analizė pateikta OCL aprašymo dalyje. Ontologijos aksiomos kaip dinaminiai ribojimai gali būti išreikšti ECA taisyklės modeliu, kuris gali būti transformuotas į OCL ribojimus. Tai įrodyta vėliau pateiktoje PAL ir OCL kalbų suderinamumo analizėje.



6 pav. UML klasių diagrama pavaizduota PAL aksiomos transformacijos schema į OCL išreikštą ribojimą

Pateikta transformacijos metodo schema pritaikoma Protege ontologijos aksiomoms, aprašomoms PAL (angl. *Protege Axiom Language*) ribojimais, transformuoti į informacijos apdorojimo taisykles, išreikštas OCL ribojimais.

Transformacija į OCL naudinga tuo, kad:

- OCL ribojimai pateikiami kaip dalis UML diagramų;
- Siūloma transformacija gali būti tarpinė stadija transformuojant aksiomas į SQL trigerius, jei būtų pasirinkta tokia strategija

Atlikus vėliau darbe pateikiamą analizę buvo nustatyta, kad OCL ir PAL kalbom išreikšti ribojimai turi, tokias suderinamas dalis:

- Ribojimo pavadinimus
- Ribojimo aprašymus
- Klases, kurioms priskiriamas ribojimas
- Ribojimo sąlygas
- Prieš ir po sąlygų išraiška
- Reikšmių išvedimus

3.2. Metodo realizacija

Metodo realizacijai pasirinktas Protégé ontologijų kūrimo įrankis ir laikraščio ontologija. Ši ontologija yra Stenfordo universiteto sukurtas ontologijos pavyzdys. Ontologijos terminai ir

naudojamos aksiomos aprašytos anglų kalba. Metodą ir sukurtą plėtinį, aksiomų išgavimui ir transformavimui, galima pritaikyti bet kokiai dalykiniai sričiai. Tam, kad suprastume ar metodas yra įgyvendinamas, reikia suprasti įrankio duomenų saugojimo principus, kadangi pradžioje teks išgauti PAL aprašytas aksiomas.

3.2.1. Protégé projekto struktūra

Kiekvienas Protégé ontologijos projektas yra sudarytas iš trijų tekstinių failų:

- ***.pprj** (***P**rotégé **proj**ect)* faile yra saugoma visa informacija apie Protégé projektą. Čia taip pat saugoma ir informacija apie įtrauktus papildomus projektus. Pvz., gali būti saugoma EZPAL plėtinio (*ang. plug-in*) šablonų projektas. Kai yra atidaromas *.pprj failas, automatiškai yra paleidžiami ir kiti du projektui priklausantys failai.
- ***.pont** (***P**rotégé **ont**ology*) faile yra saugoma visa informacija apie ontologijoje esančias klases ir slotus (klasių savybės ir ne *is-a* (hierarchiniai) ryšiai su kitomis klasėmis).
- ***.pins** (***P**rotégé **ins**tances*) faile yra saugoma visa informacija apie egzempliorius, ribojimų aksiomas.

3.2.2. PAL (Protege Axiom Language)

Protégé-OWL kaip ir paprastas Protégé naudoja PAL (ang. *Protege Axiom Language* – Protégé aksiomų kalba). Jos paskirtis išreikšti žinių bazės ribojimus, ji taip pat gali būti panaudota kurti logines užklausas žinių bazės turiniui. Naudojami dvi įrankio dalys: PAL Constraints Tab (ribojimų dalis) ir PAL Queries Tab (užklausų dalis) (Grosso 2006).

PAL Constraints Tab yra visapusiška sistema, kuri leidžia naudotojui kurti, naršyti ir keisti ribojimus bei juos įvertinti. PAL Queries Tab analogiškai skirtas naudotojo darbui su užklausomis.

Prie jau minėtų PAL įgyvendinančių dalių, aksiomų kūrimo patobulinimų galima paminėti EZPAL Tab plėtinį, kuris sukurtas dar labiau palengvinti darbą su Protege aksiomomis bei jų struktūros įsisavinimą. Šis plėtinys leidžia lengvai kurti ribojimų aksiomas be pilno PAL supratimo (Grosso 2006). Naudotojas gali naudotis sukurta šablonų biblioteka tiesiog užpildydamas tuščias šablonų vietas, o objektų reikšmės aprašytos lengvai suprantamomis metaforomis.

3.2.3. EZPAL struktūra

EZPAL sąsajoje Protégé ontologijos informacija skirstoma į tris pagrindines kategorijas:

- Struktūra (ang. *pattern*) – apibrėžta loginė sentencija, išvesta iš aksiomų grupės, kurios struktūriškai identiškos, tačiau gali būti taikomos skirtingose sąlygose. Individualios struktūros bibliotekoje nėra saugomos detaliai, pateikiami tik bendri joms tinkantys

šablonai.

- Šablonas (ang. *template*) - kiekvienas šablonas aprašo dažniausiai naudojamų aksiomų rinkinį (pagal jų semantinius ir struktūrinius panašumus) ar tik vieną aksioma, jei ji unikali. Jis saugo tinkamiausią informacijos variaciją ir leidžia pasirinkti (*retrieval*) konkrečią struktūrą, į kurios vietas užtenka pasirinkti ontologijos klases ir apribojimus aksiomų kūrimui.
- Savybė (ang. *property*) – abstraktus aprašymas, naudojamas apibūdinti dažnai pasitaikančias šablonų grupių ypatybes. Savybės nėra skirtos konkrečiam atvejui: kiekvienas šablonas gali tenkinti daugiau nei vieną savybę.

Šiuo metu EZPAL bibliotekoje pateikiama 20 šablonų ir 4 savybės.

3.2.4. PAL struktūra

Nagrinėjant PAL sintaksę galima aiškiai atskirti aksiomos sudėtines dalis, kurios vėliau bus panaudotos formuojant aktyviąsias taisykles.

Ribojimai gali būti skirstomi pagal veikimo sferas: tipų, atributų ir duomenų bazių ribojimai. Palaikomi atitinkami perėjimų ribojimai. Ribojimai nusakomi ribojamaisiais sakiniais. Tuo tarpu PAL ribojimas yra egzempliorius, priklausantis *:PAL-Constraint* klasei. Klasė turi tokius slotus (ang. *slots*) (Stanford Medical Informatics. PAL Conceptual Framework 2006):

- *:PAL-name* – ribojimo pavadinimas;
- *:PAL-documentation* – ribojimo aprašymas;
- *:PAL-range* – lokalių ir globalių kintamųjų apibrėžimas, kuris naudojamas sakinyje;
- *:PAL-statement* – ribojimo sakinys, kuriame nurodomi sąlygos ir veiksmai.

Pagrindinė PAL ribojimo aksiomos dalis yra *:PAL-statement*, kuri gali būti apibūdinta kaip verslo taisyklė ir iš dalies išreikšta kaip ECA taisyklė, kurios struktūra įvykis-sąlygos-veiksmas. PAL-statement turi aiškiai apibrėžtą veiksmą ir sąlygos dalis, reikalingas transformacijai į SQL trigerius. Tačiau įvykio dalis nėra apibrėžiama, kadangi ji gali būti apibrėžta rankiniu būdu, kai yra būtina. Visi ribojimai parašyti PAL apibrėžia būseną, kurioje sritis turėtų būti. Tačiau, nėra teikiama informacija apie tai, kas turėtų būti įgyvendinta norint pasiekti reikiamą būseną.

3.3. Aksiomų transformacija į SQL

Pagal pateiktą 5 lentelės pavyzdį paaiškinsime metodo taikymą. Pirmoje dalyje pateiktas PAL ribojimas, saugomas projekto .pins faile. Antroje dalyje gaunamas rezultatas SQL sintakse. Transformacijai kaip pavyzdys yra naudojama laikraščio ontologijos aksiomos.

5 lentelė. PAL ribojimo gautas rezultatas

PAL ribojimas
(%3APAL-DESCRIPTION "A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.") (%3APAL-NAME "customer-not-supplier-in-contract") (%3APAL-RANGE "(defrange ?Contract :FRAME Contract)\n") (%3APAL-STATEMENT "(forall ?Contract\n (=> (and (own-slot-not-null 'customer' ?Contract) \n (own-slot-not-null 'supplier' ?Contract))\n (not (= ('customer' ?Contract) ('supplier' ?Contract)))))\n"))
Gautas rezultatas - formali taisyklė, SQL trigerio pavidalu

/* Documentation */
 	
/* A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value. */
 	
CREATE TRIGGER customer-not-supplier-in-contract
 	
ON Contract
 	
{FOR | AFTER | INSTEAD OF}
 	
{[DELETE] [,] [INSERT] [,] [UPDATE]}
 	
AS
 	
FOR EACH ROW
 	
IF (Contract.customer is not null AND Contract.supplier is not null) AND (Contract.customer<>Contract.supplier)
 	
BEGIN
 	
COMMIT TRANSACTION
 	
PRINT 'Transakcija įvykdyta'
 	
ELSE
 	
RAISERROR ('Taisyklė pažeista.')
 	
ROLLBACK TRANSACTION
 	
END

Transformacija atliekama 6 lentelėje pateiktu principu, dalys formuojamos ir pridedamos nuosekliai.

6 lentelė. Nuoseklus transformavimas į SQL

PAL ribojimo paimta dalis	**Išgauta dalis**	**Pridėta dalis**	**Rezultatas SQL sintakse**
(%3APAL-DESCRIPTION "A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.")	A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.	/* Documentation *//* *Išgauta dalis* */	/* Documentation */ /* A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value. */
(%3APAL-NAME "customer-not-supplier-in-contract")	customer-not-supplier-in-contract	CREATE TRIGGER *Išgauta dalis*	CREATE TRIGGER customer-not-supplier-in-contract
(%3APAL-RANGE "(defrange ?Contract :FRAME Contract)\n")	Contract	ON *Išgauta dalis*	ON Contract
Alternatyvus panaudojimas, jei nebūtų RANGE dalies: "(forall ?Contract	Contract	ON *Išgauta dalis*	ON Contract
		{FOR | AFTER | INSTEAD OF} {[DELETE] [,] [INSERT] [,] [UPDATE]} AS FOR EACH ROW	{FOR | AFTER | INSTEAD OF} {[DELETE] [,] [INSERT] [,] [UPDATE]} AS FOR EACH ROW

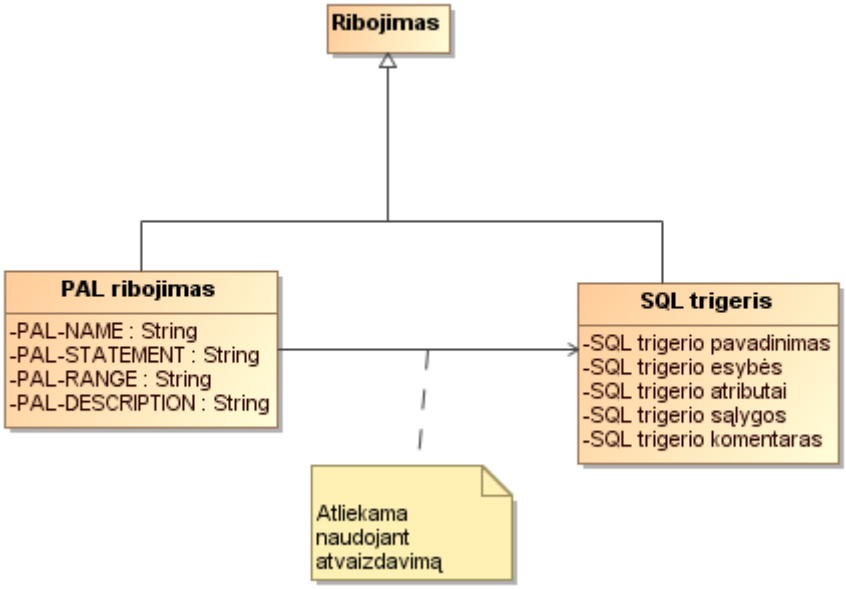
(%3APAL-STATEMENT "(forall ?Contract\n (=> (and (own-slot-not-null 'customer' ?Contract) \n (own-slot-not-null 'supplier' ?Contract))\n (not (= ('customer' ?Contract) ('supplier' ?Contract)))))\n"))	Contract.customer is not null; Contract.supplier is not null; Contract.customer <>Contract.supplier	IF (*Išgauta dalis* AND *Išgauta dalis*) AND (*Išgauta dalis* <> *Išgauta dalis*)	IF (Contract.customer is not null AND Contract.supplier is not null) AND (Contract.customer<>Contract.supplier)
Detalesnis STATEMENT panaudojimas			
"(forall ?Contract\n (=> (and (own-slot-not-null 'customer' ?Contract)	Contract.customer is not null	IF (*Išgauta dalis* AND	IF (Contract.customer is not null AND
(own-slot-not-null 'supplier' ?Contract))	Contract.supplier is not null	AND *Išgauta dalis*)	AND Contract.supplier is not null)
(not (= ('customer' ?Contract) ('supplier' ?Contract)))))\n"))	Contract.customer <>Contract.supplier	AND (*Išgauta dalis* <> *Išgauta dalis*)	AND (Contract.customer<>Contract.supplier)

		BEGIN
		
		COMMIT TRANSACTION
		
		PRINT 'Transakcija įvykdyta'
		
		ELSE
		
		RAISERROR ('Taisyklė pažeista.')
		
		ROLLBACK TRANSACTION
		
		END
			BEGIN
			
			COMMIT TRANSACTION
			
			PRINT 'Transakcija įvykdyta'
			
			ELSE
			
			RAISERROR ('Taisyklė pažeista.')
			
			ROLLBACK TRANSACTION
			
			END

3.4. Gautų SQL trigerių naudojimas informacinei sistemai formuoti

Gavus aksiomas ir transformavus į SQL sintaksę pagal aprašytą metodą, gaunami trigerių prototipai, kurie nėra baigtiniai. Trigerių prototipus reikia pakoreguoti, kad juos būtų galima taikyti

sistemai formuoti. Aksiomose neretai *RANGE* srityje nurodoma ne viena klasė, todėl galimas duomenų bazės lentelių pasirinkimas, pvz., *ON {article|editor}*, kur reikia pasirinkti *article* arba *editor*. Aksiomose taip pat nenusakoma *{FOR | AFTER | INSTEAD OF}* ir *{[DELETE] [,] [INSERT] [,] [UPDATE]}* dalis, kurią reikia rankiniu būdu įvesti, pvz., sąlygą, kad po atnaujinimo turi suveikti trigeris. Kita dalis, kurią galima pakeisti, tai *BEGIN* dalyje išvedamas tekstas, kuris nurodo tai, kas turi būti išvesta įvykus transakcijai ir/arba neįvykus. Šioje vietoje pradžioje gali būti nustatyti standartiniai pranešimai, tokie kaip „Transakcija įvykdyta“ ar „Transakcija neįvykdyta“ ar pan. šiuos pranešimus galima pakeisti taip, kad jie atitiktų konkretų trigerį. Transformacijos rezultatą turėtų peržiūrėti SQL ekspertas, kuris galutinai suderintų trigerį veikimui. Tačiau galutiniam taikymui reikalingus pakeitimus ganėtinai lengva atlikti, kai po transformacijos pateikiamas visas trigeris ir telieka pasirinkti iš nurodytų variantų.



7 pav. PAL-SQL transformavimas pateiktas UML veiksmų diagramoje, PAL metamodelio dalys transformajuojamos į SQL trigerio sudėtines dalis

Todėl, kad trigerių prototipai buvo sukurti remiantis SQL trigerių pavyzdžiais ir atitinka jų struktūrą, buvo atlikta jų funkcionalumo analizė, po kurios nustatyta, kad juos atinkamai pakeitus – juos galima naudoti informacinės sistemos kūrimui.

3.5. Axiom2SQL plėtinio ir programinio kodo aprašymas

Sukurtas Protégé plėtinys – programos prototipas, kuris išrenka iš Protégé .pins failo aksiomų kalba (PAL ir EZPAL) parašytas ribojimų aksiomas. Plėtinys, pavadintas „Axiom2SQL“, atlieka dažniausiai naudojamų ribojimų tipų transformaciją į informacijos apdorojimo taisykles, SQL trigerių prototipus. Programinis kodas yra parašytas Java kalba (visas programinis kodas yra pateiktas 2-ame

priede). Plėtinys pridedamas prie Protégé ontologijų kūrimo įrankio „Axiom2SQL.class“ failo pavidalu, saugomas „VETIS“ direktorijoje. „VETIS“ direktorija įdedama įrankio „plugins“ direktorijoje, kur saugomi visi Protégé plėtiniai. Tam, kad tinkamai veiktų Protégé plėtinys, jam būtina com.stevesoft.pat nemokama biblioteka, kuri yra saugoma faile patbinfree153.jar. Plėtinys taip pat naudoja standartines Java bibliotekas, o rezultatą įrašo į C disko „data“ direktoriją, kuri turi būti sukurta. Plėtinis sukurtas naudojant eksportavimo plėtinio pavyzdį, kuris buvo atitinkamai modifikuotas - jis iškviečia paleidimui transform() funckiją. Plėtinio veikimas pavaizduoti paveiksluose, kurie pateikti 1 priede. Plėtinys transformuoja ~20 PAL ir EZPAL skirtingų ribojimo *STATEMENT* dalies sintaksės šablonų ir įvairias jų variacijas.

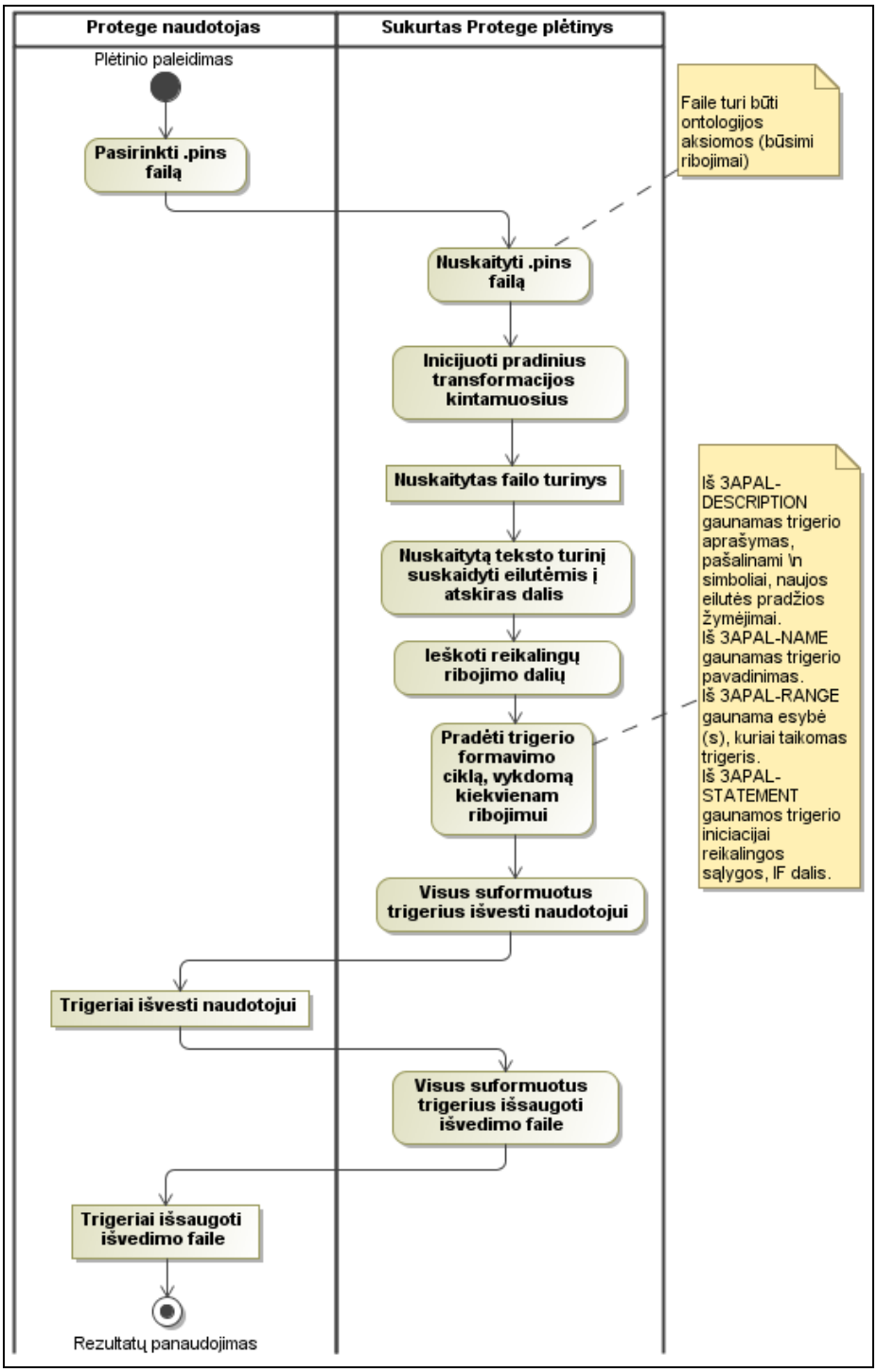
3.5.1. Algoritmas

Aprašymo pradžioje pateiktas išsamus Protégé plėtinio algoritmo aprašymas. Grafiškai algoritmas pateiktas vėliau pateiktame paveiksle.

- Pasirinkti .pins failą, kuriame yra PAL aksiomos.
- Nuskaityti .pins failą.
- Inicijuoti pradinius transformavimo kintamuosius.
- Nuskaityti turinio tekstą ir suskaidyti eilutėmis į atskiras dalis.
- Ieškoti reikalingų ribojimo dalių.
- Pradėti trigerio formavimo ciklą, vykdomą kiekvienam ribojimui.
    - Tkrinti eilutės po vieną ar konkreti eilutė aprašo ribojimų aksiomą. Eilutėje ieškoti *3APAL-NAME, 3APAL-DESCRIPTION, 3APAL-RANGE, 3APAL-STATEMENT* simbolių sekų, kurios apibūdina keturias ribojimo aksiomos dalis.
    - Pagal rastą dalį atlieka transformacija. Paimama reikalinga dalis, ji pakeičiama ar pertvarkoma, kad kuo labiau atitiktų SQL sintaksę.
    - Iš *3APAL-DESCRIPTION* gaunamas trigerio aprašymas, pašalinami *\n* simboliai, naujos eilutės pradžios žymėjimai.
    - Iš *3APAL-NAME* gaunamas trigerio pavadinimas.
    - Iš *3APAL-RANGE* gaunama esybė(s), kuriai taikomas trigeris.
    - Iš *3APAL-STATEMENT* gaunamos trigerio iniciacijai reikalingos sąlygos, IF dalis. Eilutės duomenys suskaidomi į atskiras dalis ties *\n* simboliais. Vėliau vykdomas vidinis IF dalies formavimo ciklas, kurio metu gautos *3APAL-STATEMENT* eilutės dalys lyginamos su aprašytomis programoje konkrečiais transformavimo šablonais. Nustačius eilutės dalies atitinkamą šabloną, atliekama transformacija į SQL sintaksę.
    - Atsižvelgiant į kiekvienos dalies skliaustelių pozicijas, skaičių bei kitus parametrus nustatyti IF dalies jungiamuosius žodžius (AND, OR) ir jų padėtis.

- Pilnai suformuoti IF dalį.
- Prie sąlygų pridėti BEGIN dalį su standartiniu vykdymo pranešimu.
- Suformuoti trigerį (-ius).

- Visus suformuotus trigerius išvesti naudotojui.
- Visus suformuotus trigerius išsaugoti išvedimo faile.



8 pav. PAL-SQL transformacijos algoritmas pateiktas UML veiksmų diagramoje

3.5.2. Transformacijos kodo dalis

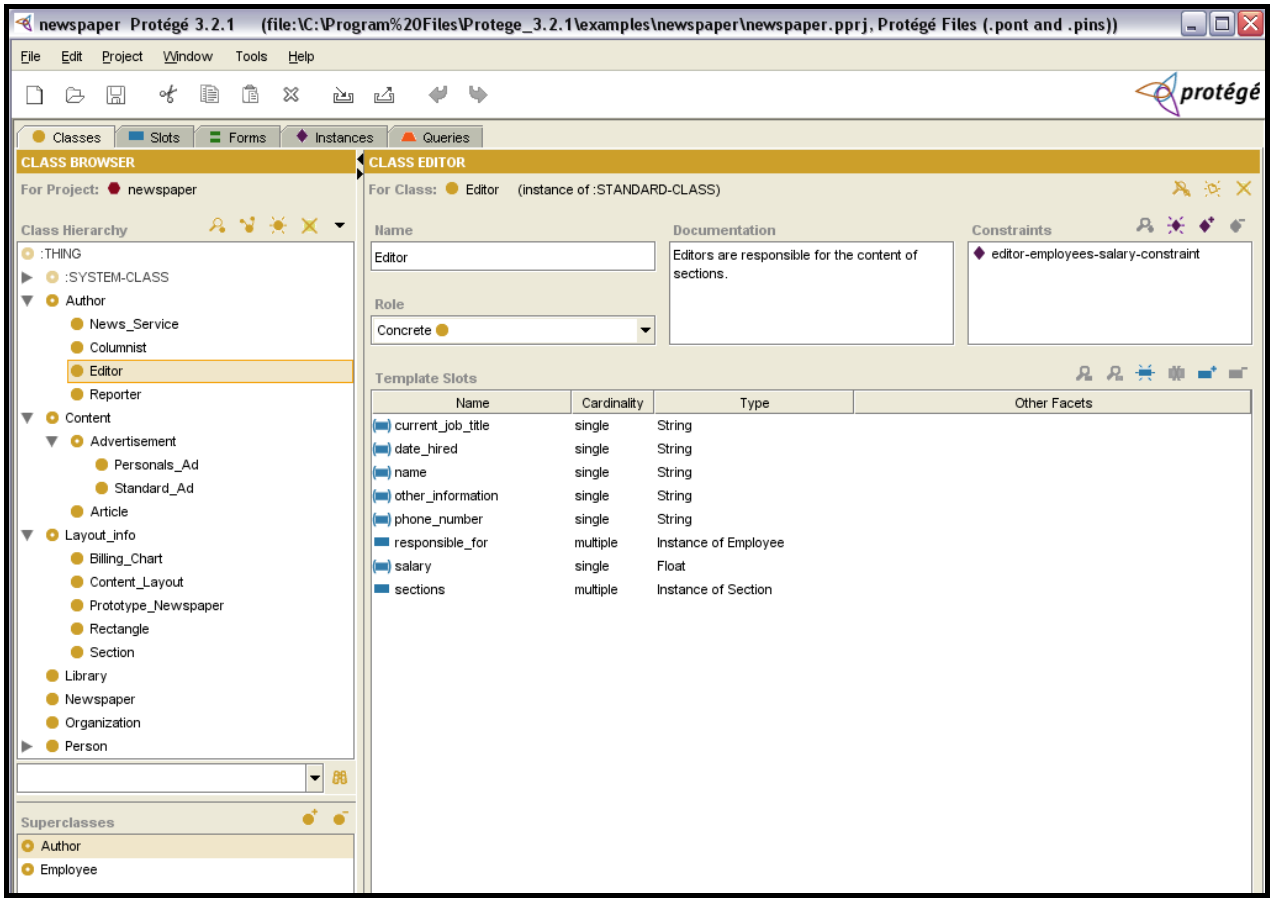
Pateikta kodo dalis su detaliais paaiškinimais, kuri naudojama transformuoti *Statement* dalį.

```
// Transformavimui naudojama Java regex biblioteka
// Transformuojamo teksto pavyzdys -  (> ('quantity' ?Contract_Product) 19)
// Naudojamas masyvas paimti lyginimo  ženklams (>, <, =).
   n=0;
   while (n<3) {
   Input=Mas[n];
   n++; // Nustatoma ar dalis atitinka aprašytą šabloną.
   Regex range461 =  new Regex("\\(" + Input + " \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)
([A-Z_a-z0-9.]+)\\)");
   range461.search(dalis);
// Jei atitiko, vykdoma.
   if (range461.didMatch()) {
// Prieš sąlygos transformavimą, tikrinama, ar reikia pridėti AND ir OR. Jei tenkinamos sąlygos,
tai padaroma. Atsižvelgiama į Exists naudojimą. Booland ir boolor kintamaisiais tikrinama ar
AND arba OR grandinės buvo inicijuotos. Nrand ir nror skaičiuoja kiek kartų buvo pridėti AND ir
OR. Jei nrand arba nror skaičius mažesnis už 2, sąlyga nevykdoma, nes buvo fiksuota tik pirma
AND arba OR grandinės dalis, kuri neprijungiama. Ir atvirkščiai, užfiksavus, kad AND arba OR
buvo panaudotas daugiau nei 1 kartą, pridedamas AND arba OR, kadangi grandinė jau yra
prasidėjusi ir būtina sujungti jos dalis.
if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
// Formuojama ir pridedama IF sąlyga.
         naudstatement = naudstatement + range461.stringMatched(2) + "." +
range461.stringMatched(1) + Input + range461.stringMatched(3);
        }
//EZPAL sintaksei
else {
// Nustatoma ar dalis atitinka aprašyta šabloną.
 Regex range461b =  new Regex("\\(" + Input + " \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-
9.]+)\\) ([A-Z_a-z0-9.]+)\\)");
  range461b.search(dalis);
  if (range461b.didMatch()) {
  if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
  if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
  if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
  naudstatement = naudstatement + range461b.stringMatched(2) + "." +
```

```
range461b.stringMatched(1) + Input + range461b.stringMatched(3);
 }}} // Baigiasi masyvo while dalis.
```

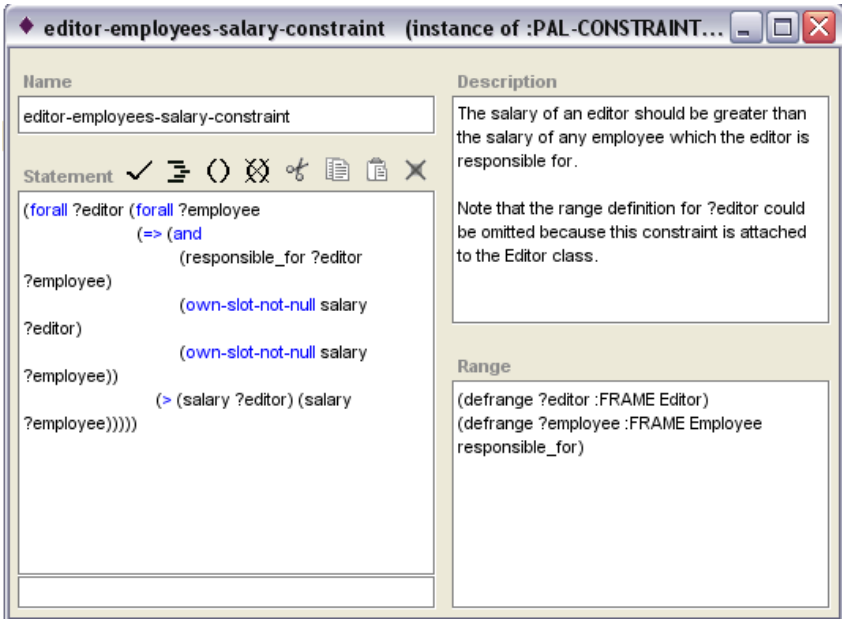
3.5.3. Testavimas

Realizacijai pademonstruoti buvo pasirinkta laikraščio ontologija, tačiau metodas gali būti taikomas bet kokiai dalykiniai sričiai. Bus pateikta tik metodui reikalinga ontologijos dalis, ribojimo aksiomos.



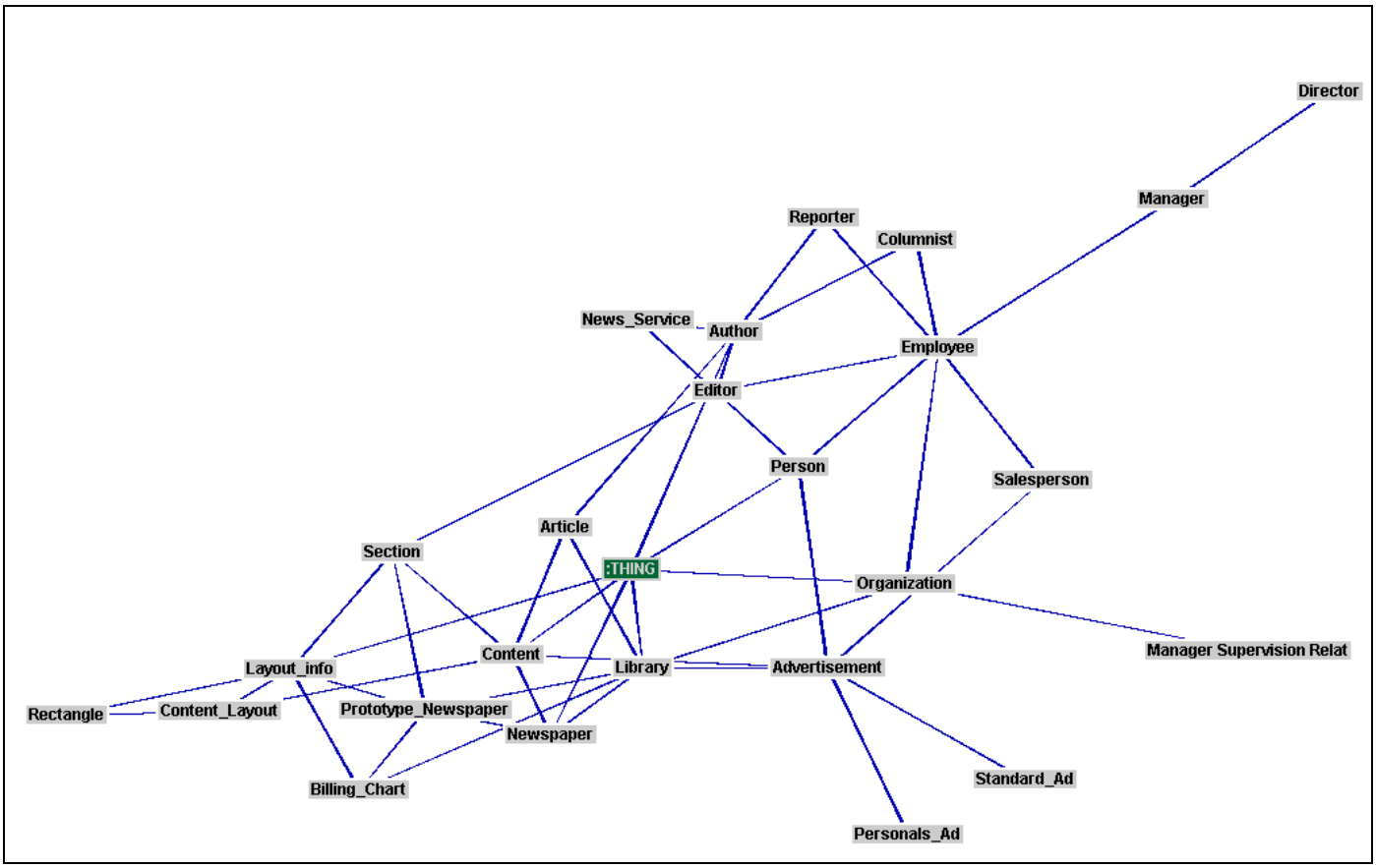
9 pav. Laikraščio ontologija Protégé lange

Šiame paveiksle pavaizduota laikraščio ontologija. Matomos konceptų klasės pavaizduotos hierarchiškai vaizduojamos Protégé įrankyje. Matoma redaktoriaus klasė, jos atributai ir taikomas ribojimas *editor-employees-salary-constraint*.

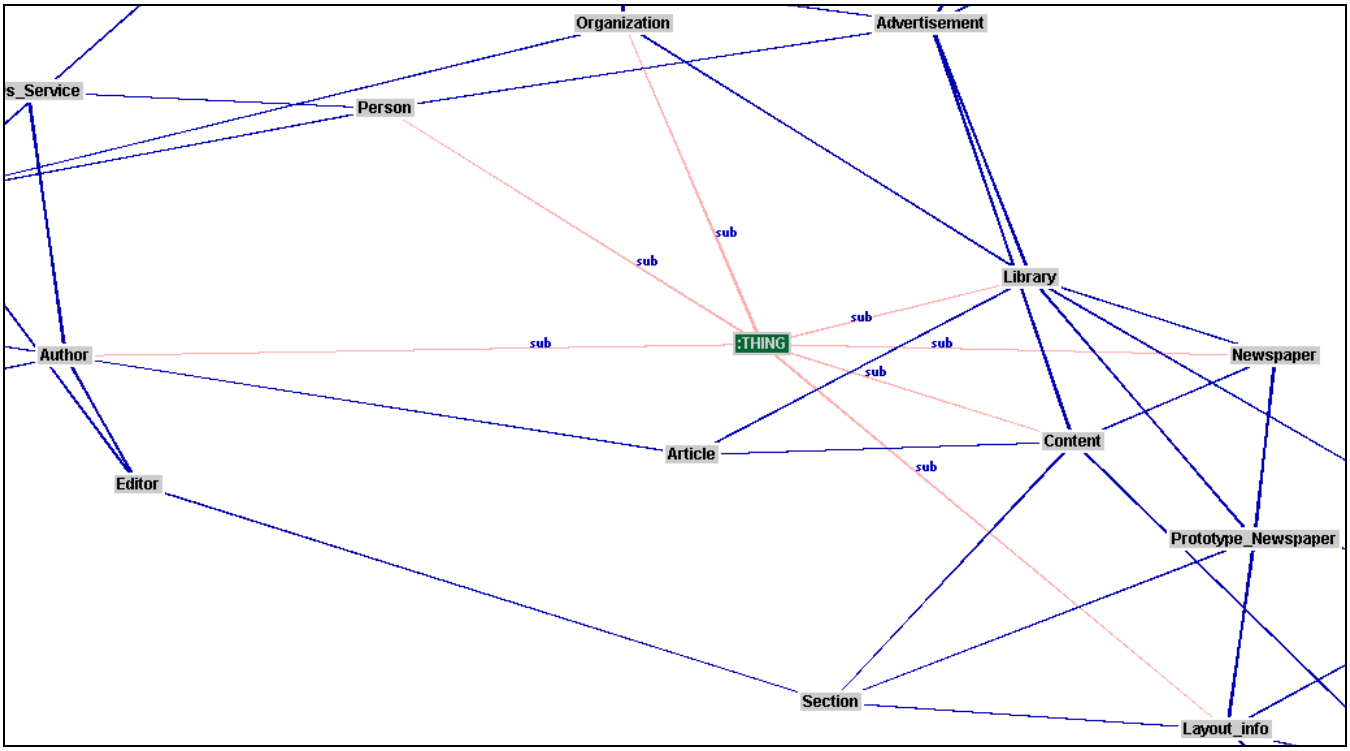


10 pav. *Edit-employees-salary-constraint* ribojimas

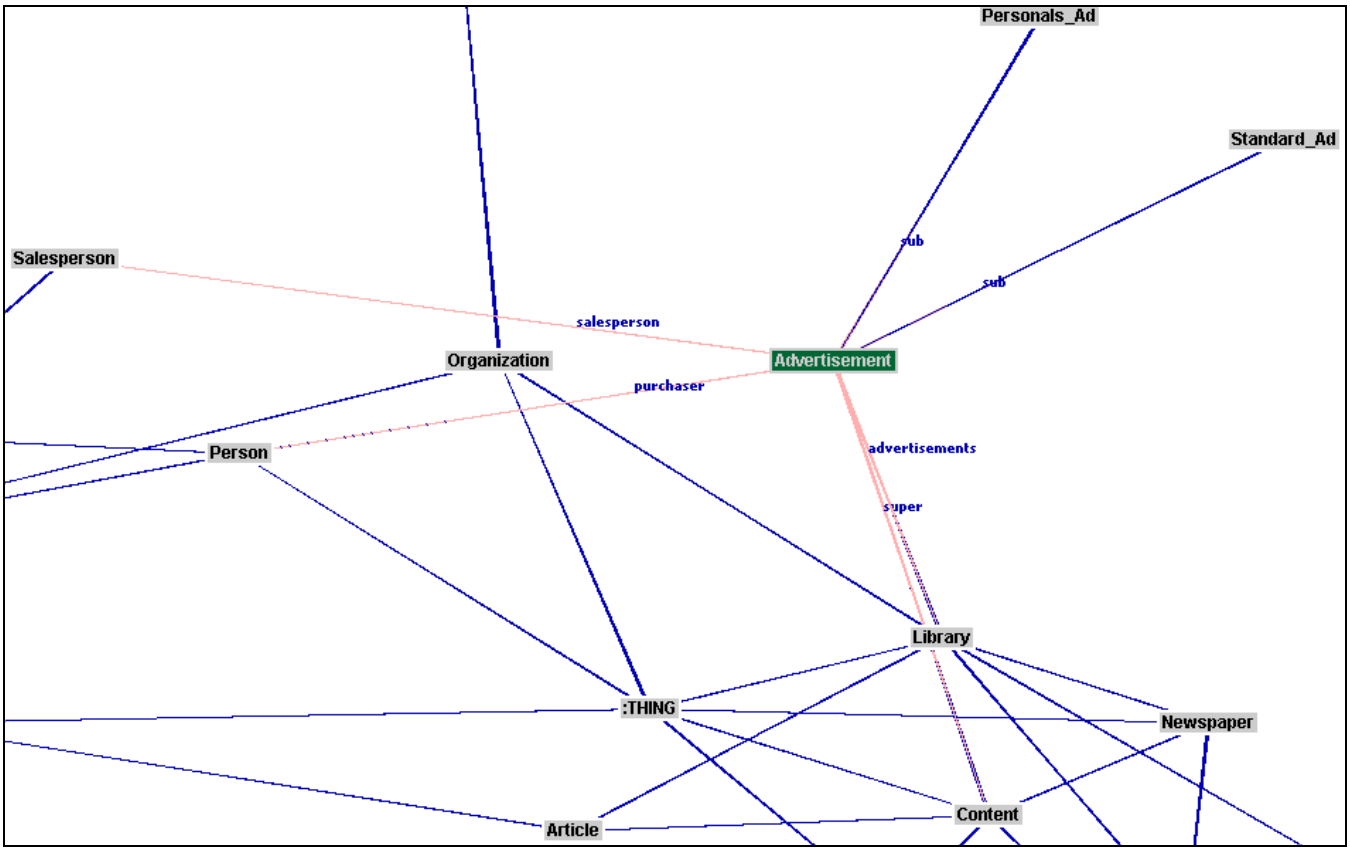
10 paveiksle pateiktas Protégé ribojimo aprašymo langas, matoma PAL sintaksė. Pasinaudojus TGVizTab plėtiniu, sekančiame paveiksle matomi visi laikraščio ontologijos konceptai ir ryšiai tarp konceptų.



11 pav. Laikraščio ontologijos struktūra, pavaizduoti visi konceptai



12 pav. Kiti konceptai – aukščiausio koncepto klasės *Thing* (dalyko) poklasiai

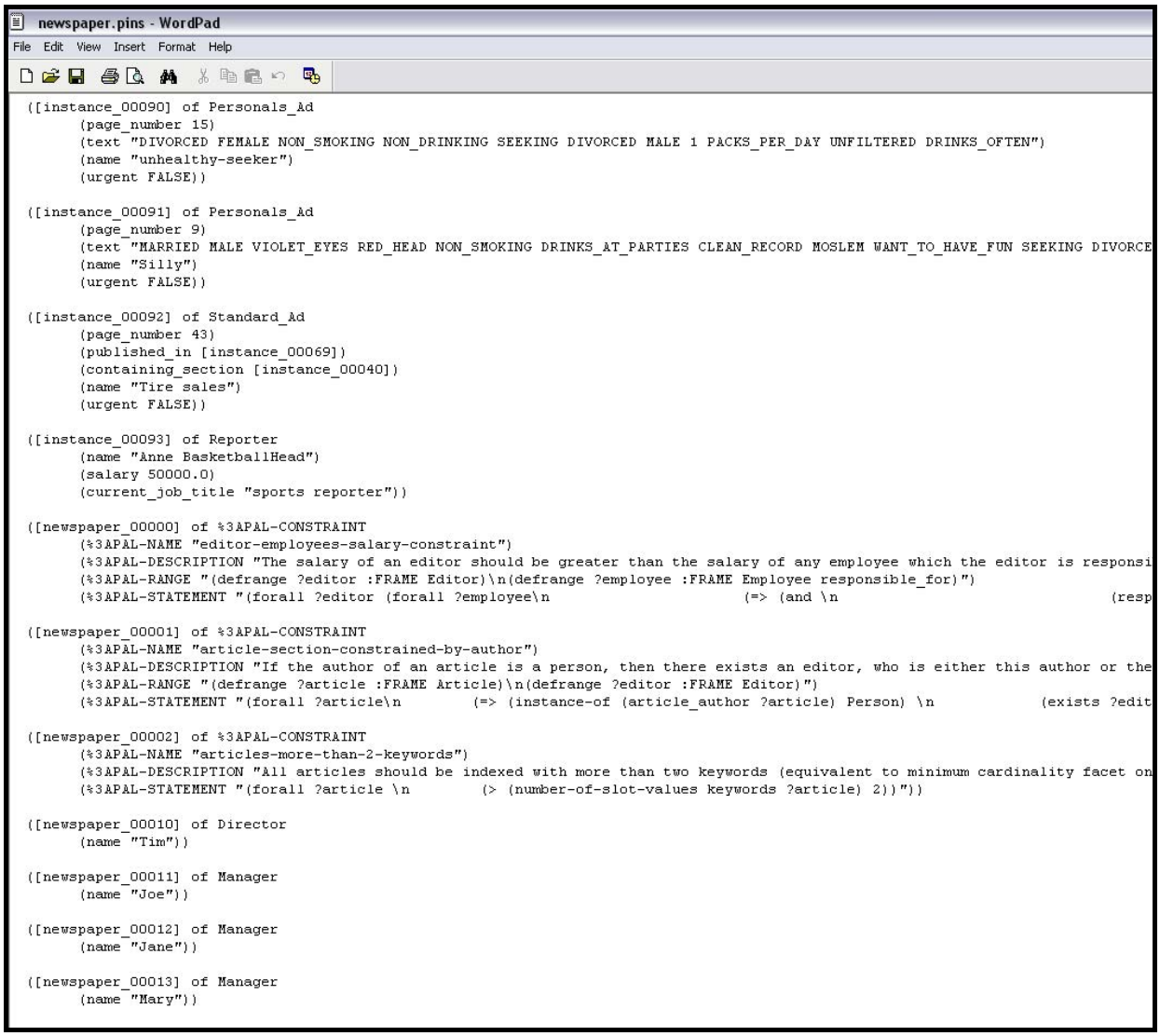


13 pav. Vaizduojami *Advertisment* (reklamos) koncepto ryšiai su kitais konceptais

3.5.4. Testavimui naudotos laikraščio ontologijos aksiomos

7 lentelė. Testavimui naudotos ontologijos aksiomos

Nr.	Aksiomos pavadinimas	Naudojami konceptai	Naudojamos savybės	Aksiomos aprašymas
1	***editor-employees-salary-constraint***	*Editor* (redaktorius)	*Salary* (atlyginimas), *responsible_for* (atsakingas)	Redaktoriaus atlyginimas turi būti didesnis nei jam pavaldžių darbuotojų.
		Employee (darbuotojas)		
2	***article-section-constrained-by-author***	*Article* (straipsnis)	*Article_author* (straipsnio autorius), *Containing_section* (saugoma skiltyje)	Jei autorius yra asmuo ir egzistuoja redaktorius, kuris yra arba autorius, arba už autorių atsakingas redaktorius. Straipsnio skiltį prižiūri redaktorius.
		Editor	*Article_author, Responsible_for, sections* (skiltys)	
		Person (asmuo)	*Article_author*	
3	***articles-more-than-2-keywords***	*Article*	*Keywords* (reikšmingi žodžiai)	Visi straipsniai turi būti indeksuojami daugiau nei 2 reikšmingais žodžiais.



14 pav. Ribojimo aksiomos laikraščio ontologijos .pins faile

Paveiksle pavaizduotas .pins failo vidus, matomos ribojimo aksiomos, kurios bus išgautos ir transformuotos.

3.5.5. Testavimo tvarka

Plėtinio testavimo ir derinimo metu buvo atlikti žemiau pateikti veiksmai.

1) Pasirenkama testavimui naudojamos laikraščio ontologijos .pins formato failas, kuriame saugomi ribojimai.
2) Nagrinėjami gauti rezultatai. Rezultatai lyginami su panašiais SQL trigerių pavyzdžiais. Jei buvo rasta klaidų, plėtinys tobulinamas, kol tinkamai vykdo transformaciją.
3) Rezultatai pateikiami įvertinti ekspertui (darbo konsultantei D. Kalibatienei).
4) Tobulinamas plėtinys, šalinamos nustatytos klaidos. Kartojamas testavimas iki tol, kol bus tinkamai vykdoma visų aksiomų transformacija.

3.5.6. Testavimo rezultatai

Žemiau pateiktas laikraščio ontologijos 3 ribojimų aksiomų išgavimas ir transformavimas į SQL trigerių prototipus. Pagal galutinius rezultatus galima spręsti, kad plėtinys tinkamai atlieka savo funkciją, išgavimą ir transformavimą, kadangi tiksliai atlieka Aksiomų transformacija į SQL poskyryje aprašytą transformavimą.

STATEMENT dalyje sąlygos ties \n buvo suskaidytos siekiant parodyti, kokiomis dalimis programoje nagrinėjama eilutė.

3.5.6.1. Aksioma editor-employees-salary-constraint

PAL

```
([newspaper_00000] of %3APAL-CONSTRAINT
        (%3APAL-NAME "editor-employees-salary-constraint")
        (%3APAL-DESCRIPTION "The salary of an editor should be greater than the salary of any employee which the editor is responsible for.\n\nNote that the range definition for ?editor could be omitted because this constraint is attached to the Editor class.")
        (%3APAL-RANGE "(defrange ?editor :FRAME Editor)\n
(defrange ?employee :FRAME Employee responsible_for)")
        (%3APAL-STATEMENT "(forall ?editor (forall ?employee\n
          (=> (and \n
           (responsible_for ?editor ?employee)\n
            (own-slot-not-null salary ?editor) \n
            (own-slot-not-null salary ?employee)) \n
                    (> (salary ?editor) (salary ?employee)))))"))
```

SQL

```
/* Documentation */
/* The salary of an editor should be greater than the salary of any employee which the editor is responsible for.  Note that the range definition for ?editor could be omitted because this constraint is attached to the Editor class. */
CREATE TRIGGER editor-employees-salary-constraint
ON  {editor|employee}
{FOR | AFTER | INSTEAD OF}
{[DELETE] [,] [INSERT] [,] [UPDATE]}
AS
FOR EACH ROW
IF (editor.responsible_for=employee.responsible_for AND editor.salary is not null AND employee.salary is not null) AND (editor.salary>employee.salary)
BEGIN
```

```
        COMMIT TRANSACTION
        PRINT 'Viskas gerai.'
ELSE
        RAISERROR ('Įvyko klaida.')
        ROLLBACK TRANSACTION
END
```

3.5.6.2. Aksioma article-section-constrained-by-author

PAL

```
([newspaper_00001] of %3APAL-CONSTRAINT
        (%3APAL-NAME "article-section-constrained-by-author")
        (%3APAL-DESCRIPTION "If the author of an article is a person, then there exists an editor, who is either this author or the editor responsible for this author. The section that contains this article is one of the sections managed by this editor.\n\nNote that the range definition for ?article could be omitted because this constraint is attached to the Article class.")
        (%3APAL-RANGE "(defrange ?article :FRAME Article)\n(defrange ?editor :FRAME Editor)")
        (%3APAL-STATEMENT "(forall ?article\n
    (=> (instance-of (article_author ?article) Person) \n
    (exists ?editor \n
    (and (or (article_author ?article ?editor) \n
    (responsible_for ?editor (article_author ?article))) \n
    (sections ?editor (containing_section ?article))))))"))
```

SQL

```
/* Documentation */
/* If the author of an article is a person, then there exists an editor, who is either this author or the editor responsible for this author. The section that contains this article is one of the sections managed by this editor.  Note that the range definition for ?article could be omitted because this constraint is attached to the Article class. */
CREATE TRIGGER article-section-constrained-by-author
ON  {article|editor}
{FOR | AFTER | INSTEAD OF}
{[DELETE] [,] [INSERT] [,] [UPDATE]}
AS
FOR EACH ROW
IF (article.article_author=Person.article_author AND EXISTS editor) AND (article.article_author=editor.article_author OR editor.responsible_for=article.article_author) AND (editor.sections=article.containing_section)
BEGIN
```

```
        COMMIT TRANSACTION
        PRINT 'Viskas gerai'
ELSE
        RAISERROR ('Ivyko klaida')
        ROLLBACK TRANSACTION
END
```

Article.article_author=Person.article_author aprašomas sąryšis, vėliau tikrinama ar egzistuoja editor, kurio galimi du variantai: *article.article_author=editor.article_author* ir w*ditor.responsible_for=article.article_author*. Turi būti *editor.sections=article.containing_section*.

3.5.6.3. Aksioma articles-more-than-2-keywords

PAL

```
([newspaper_00002] of %3APAL-CONSTRAINT
        (%3APAL-NAME "articles-more-than-2-keywords")
        (%3APAL-DESCRIPTION "All articles should be indexed with more than two keywords
(equivalent to minimum cardinality facet on slot attachment)\n\nBecause this constraint is
attached to the Article class, the variable ?article implicitly ranges over all instances of the Article
class.\n")
        (%3APAL-STATEMENT "(forall ?article \n
    (> (number-of-slot-values keywords ?article) 2))"))
```

SQL

```
/* Documentation */
/* All articles should be indexed with more than two keywords (equivalent to minimum cardinality
facet on slot attachment)  Because this constraint is attached to the Article class, the variable
?article implicitly ranges over all instances of the Article class.  */
CREATE TRIGGER articles-more-than-2-keywords
ON article
{FOR | AFTER | INSTEAD OF}
{[DELETE] [,] [INSERT] [,] [UPDATE]}
AS
FOR EACH ROW
IF (COUNT (article.keywords)>2)
BEGIN
        COMMIT TRANSACTION
        PRINT 'Viskas gerai.'
ELSE
        RAISERROR ('Įvyko klaida!')
        ROLLBACK TRANSACTION
END;
```

Pastaba: Šioje aksiomoje nėra 3APAL-RANGE, klasė nurodoma 3APAL-STATEMENT dalyje po forall.

Turi būti tenkinama sąlyga, pagal kurią *article.keywords* turi būti daugiau nei 2.

Atlikus rezultatų (virš 20 įvairių tipų aksiomų transformacijų) analizę, nustatyta, kad transformacija atlikta tinkamai, todėl plėtinys, realizuojantis pasiūlytą metodą, yra tinkamas taisyklių modeliui kurti ir palengvina VT sistemos formavimą ir modeliavimą.

3.6. OCL (Object Constraint Language)

Nagrinėjant verslo taisyklių struktūrą aprašytų Protege ontologijoje nustatyta, kad įmanoma ir kitoks aksiomų transformavimas į OCL kalbą.

Transformacija į OCL naudinga tuo, kad:

- OCL ribojimai pateikiami kaip dalis UML diagramų, naudojami klasėms ir tipams specifikuoti invariantines išraiškas (OMG 2006);
- Su UML Backend plėtiniu (ang. *plugin*) galima gauti klasių diagramą, kuriai bus naudojami ribojimai (Protege UML Backend 2007).
- Tai gali būti tarpinė stadija transformuojant į SQL trigerius, jei būtų pasirinkta tokia strategija.

Pradžioje prieš lyginant ir susiejant OCL ir Protege bendrus elementus, būtina detaliau susipažinti su OCL struktūra ir jos veikimo ypatumais.

OCL tradiciškai naudojama specifikuoti invariantines sąlygas, kurios nusako sistemos būsenos modeliavimą ar užklausas. Kai OCL yra vertinama, tai atliekama be pasekmių, įvertinimas negali pakeisti vykdančios sistemos būsenos. OCL išraiška gali būti naudojama specifikuoti operacijas / veiksmus, kurie vėliau vykdomi ir keičia sistemos būseną. OCL gali būti naudojama specifikuoti įvairius ribojimus.

OCL gali naudoti naudotojai neturintys plačių matematinių žinių bei tie, kurie modeliuoja verslą ar sistemas. OCL yra formali kalba, kurią yra nesudėtinga skaityti ir rašyti. OCL buvo kurta kaip verslo modeliavimo kalba kartu su IBM draudimo padaliniu ir yra kildinama iš Syntropy metodo. OCL yra gryna specifikacijų kalba, todėl OCL išraiška veikia be pašalinių efektų \ pasekmių sistemai/modeliui, nes kai OCL išraiška yra įvertinama, ji tiesiog grąžina reikšmę. Ji negali nieko pakeisti modelyje. Tai reiškia, kad sistemos būsena niekada nepasikeis dėl OCL išraiškos įvertinimo. Tačiau OCL išraiška gali būti naudojama specifikuoti pokyčio būseną (pvz., vykstančią po sąlygos (ang. *post-condition*)). OCL nėra programavimo kalba, todėl ja neįmanoma užrašyti programos logikos ar kontroliuoti duomenų srauto.

Mes negalime iškviesti procesus arba aktyvuoti ne užklausų operacijas. OCL yra modeliavimo kalba, kurios išraiškos nėra apibrėžtos vykdymui tiesiogiai.

OCL yra tipų kalba, tai reiškia, kad kiekviena OCL išraiška turi tipą. Tai reiškia, kad lyginant atributus (kintamuosius) jie turi būti to paties tipo.

3.6.1. OCL struktūra

Context žodis apibūdina kontekstą. Raktažodžiai inv, pre ir post žymi stereotipus, kuriuos naudoja ribojimai: invariantinis «invariant», prieš sąlygą «precondition» ir po sąlygos «postcondition».

OCL naudojama specifikuoti ribojimus, kuriais gali būti išreikštos veiklos taisyklės (OMG 2003). Iš OCL išraiškos gali būti sugeneruota dokumentacija natūralia kalba (anglų). OCL yra glaustesnė nei SQL, daugiausiai dėl savo taškinio žvalgos stiliaus. OCL taip pat gali būti naudojama ankstyvoje sistemos analizės fazėje.

OCL išraiška turi būti susieta su klasių objektu, vadinamu kontekstine klase ir gali sietis su pačiu objektu (per savo atributą) arba su kitu bet kokiu atributu ar užklausos metodu. Pvz., vaizduojant darbuotojo amžiaus skaičiavimą.

```
Context Employee::age(): Integer
Post: result = year( systemDate() – birthDate )
```

3.6.1.1. Ribojimų išraiška

Ribojimas gali būti išreikštas OCL vidiniu kintamuoju (ang. *invariable*). Vidinis kintamasis – loginė išraiška, kuri turi būti teisinga visiems klasės egzemplioriams.

Standartinė struktūra:

```
context TipoPavadinimas inv:
'tai yra tradicinė OCL <<invariant>> išraiška kuri naudoja TypeName kontekstą' =
    'kita eilutė'
```

Šiame pavyzdyje ir vėliau paryškintai žymimi OCL kintamieji, o kursyvu žymima OCL išraiškos dalis.

Pvz., darbuotojo atlyginimas negali būti mažesnis nei 3000 litų.

```
Context Darbuotojas inv:
MenesioAtlyginimas >= 3000
```

Ribojimai yra susieti su įvykiais – tam tikras įvykis (ir sąlygų atitikimas) sužadina ribojimą. Ankščiau pavaizduota OCL išraiška gali būti tikrinamas dviem atskirais atvejais: kai priimamas naujas darbuotojas arba kai atnaujinama esamo darbuotojo alga.

Kiekviena OCL išraiška užrašoma naudojant specialaus tipo konteksto egzempliorių. OCL išraiškoje gali būti panaudotas žodis self, kuris užrašomas konteksto egzemplioriuje. Pvz., jei kontekstas yra Bendrove, tai self nurodo egzemplioriaus priklausomybę klasei Bendrove.

Kaip ir ankstesniu atveju šiuo būdu užrašytos ribojimo «invariant» išraiška turi būti teisinga visiems nurodytos klasės egzemplioriams. Pvz., OCL išraiškos sakinys teigia, kad darbuotojų kiekis turi būti visada didesnis už 25, tai būtų užrašoma taip:

```
self.DarbuotojuKiekis > 25
```

Šitoje išraiškoje *self* yra klasės *Bendrove* egzempliorius. Vertinamas objektas yra invariantinis *self* ir apima kiekvieną *Bendrove* klasės egzempliorių. Kontekstinės klasės tipas turi būti užrašomas prieš tai einančiame sakinyje:

```
context Bendrove inv:
self.DarbuotojuKiekis > 25
```

Dažniausiai raktinis žodis *self* gali būti nenaudojamas, kadangi kontekstas yra aiškus, kaip prieš tai pateiktame pavyzdyje. Vietoje to alternatyviai galima naudoti klasės identifikatorių kaip pvz.:

```
context c : Bendrove inv:
c.DarbuotojuKiekis > 25
```

Ribojimo pavadinimą (meta atributą) galima užrašyti po raktiniu žodžiu *inv*, kad vėliau būtų galima kreiptis į ribojimą. Pvz.,:

```
context c : Bendrove inv DarbuotojuGana:
c.DarbuotojuKiekis > 25
```

OCL išraiškoje galima nurodyti prieš sąlygą «precondition» ir/arba po sąlygos «postcondition» ribojimą, kurios susijusios su operacijos arba kitomis elgsenos ypatybėmis. Kontekstinis egzempliorius self yra tipo egzempliorius, kuriam priklauso operacijos ar metodo ypatybė. OCL konteksto deklaravimas naudoja context raktinį žodį, po kurio seka tipas ir operacijos deklaravimas. Prieš sąlyginą ribojimas žymimas pre:, o po sąlygos post:, kurie rašomi prieš tikrąsias prieš sąlygą ir po sąlygos. Pvz.,:

```
context TipoPavadinimas :: OperacijosPavadinimas(param1 : Tipas1, ... ):
GrazinamasTipas
pre: param1 > ...
post: result =
```

Raktinis žodis result žymi rezultatą, jei toks yra. Param1 žymi pirmą parametrą, o Tipas1 tipą. Pvz.:

```
Context Asmuo::Pajamos(d : Date) : Integer
post: result = 5000
```

Analogiškai inv operacijai ribojimo pavadinimas gali būti parašytas prieš dvitaškį po post ir pre.

3.6.1.2. OCL ir Protege suderinamumo analizė

Pradinei analizei buvo pasirinktos panašios OCL ir Protege aksiomos.

OCL (Object Constraint Language):

```
context c : Bendrove inv DarbuotojuGana:
c.DarbuotojuKiekis < 26
```

PAL (Protege Axiom Language)

```
([Bendrove_00001] of %3APAL-CONSTRAINT
        (%3APAL-NAME "DarbuotojuGana")
        (%3APAL-DESCRIPTION "Bendrovėje turi dirbti ne daugiau kaip 26
darbuotojai\n")
        (%3APAL-STATEMENT "(forall ?Bendrove \n
(< (DarbuotojuKiekis ?Bendrove) 26))"))
```

OCL ribojimas gali būti sudarytas iš keturių dalių:

- *konteksto klasės*, kuri apibrėžia ribojimo sritį;
- *OCL sąlygos išraiška:*
    1. *savybės*, kuri nusako konteksto charakteristikas;
    2. *operacijos*, kurios rezultatas savybė;
- *raktiniai žodžiai*, kuriais išreiškiama sąlyga.

Priminsime, kad PAL ribojimas yra egzempliorius, priklausantis *:PAL-Constraint* klasei. PAL ribojimas kaip ir OCL sudaromas iš 4 dalių, klasės *PAL-Constraint* slotų (ang. *slots*) :

- :*PAL-name* – ribojimo pavadinimas;
- *:PAL-documentation* – ribojimo aprašymas;
- *:PAL-range* – lokalių ir globalių kintamųjų apibrėžimas, kuris naudojamas sakinyje;
- *:PAL-statement* – ribojimo sakinys, kuriame nurodomi sąlygos ir veiksmai.

Analizuojant pateiktų OCL ir PAL paprastų ribojimų struktūrą, yra išskiriami tokie OCL ir PAL panašumai ir skirtumai pateikti žemiau esančioje lentelėje.

8 lentelė. OCL ir PAL suderinamumo analizė

Elemento pavadinimas	OCL pavyzdyje	PAL pavyzdyje	Ar tarpusavyje suderinami?
Ribojimo pavadinimas (pvz., *DarbuotojuGana*)	**Yra.** Rašomas po inv ribojimo tipo.	**Yra.** Rašomas po `%3APAL-NAME` tarp kabučių.	**Pilnai suderinami**
Ribojimo aprašymas	**Yra.** Rašoma tarp viengubų kabučių.	**Yra, tačiau nebūtinas.** Dažniausiai pateikiamas naudotojo patogumui.	**Suderinami**, aprašymas nėra esminė ribojimo dalis.
Klasė, kuriai priskiriamas ribojimas (pvz., *Bendrove*)	**Yra.** Pateikiamas po `context` dalies.	**Yra**. *:PAL-range* dalis, pavyzdyje pateikiama pirmoje `%3APAL-STATEMENT` dalies eilutėje.	**Pilnai suderinami**
Ribojimo sąlyga	**Yra.** `c.DarbuotojuKiekis < 26`	**Yra.** `(< DarbuotojuKiekis ?Bendrove) 26))"))`	**Pilnai suderinami.** Abiejuose ribojimuose nurodomi naudojami klasės atributai, klasė, sąlygos operatoriai.
Prieš sąlyga	**Yra.** Pateikiama po raktinio žodžio `pre-condition`. (žr. 1 pavyzdį)	**Yra.** Pateikiama po simbolio `=>`. (žr. 2 pavyzdį)	**Suderinami.**
Po sąlyga	**Yra.** Pateikiama po raktinio žodžio `post-condition`. (žr. 2 pavyzdį)	**Yra.** Gali būti bet kokia išraiška, einanti po prieš sąlygą.	**Suderinami.**
Esamos ir išvedamos reikšmės	**Yra.** Pateikiama po raktinio žodžio `init` ir `derive`. (žr. 3 pavyzdį)	**Yra.** Gali būti užrašytos paprasta PAL išraiška. (žr. 4 pavyzdį)	**Suderinami.**

1 pavyzdys.

OCL (*Object Constraint Language*):

context Person::getCurrentSpouse() : Person
pre: self.isMarried = true
body: self.mariages->select(m | m.ended = false).spouse

PAL (Protege Axiom Language):

```
(%3APAL-DESCRIPTION "Redaktoriaus alga turi būti didesnė už jam vadovaujamų darbuotojų algą\n")
(%3APAL-STATEMENT "( forall ?editor (forall ?employee
        (=> (and
                (responsible_for ?editor ?employee)
                (own-slot-not-null salary ?editor)
                (own-slot-not-null salary ?employee))
(> (salary ?editor) (salary ?employee))))))
```

2 pavyzdys:

OCL (*Object Constraint Language*):

context Typename::operationName(param1 : Type1, ...): ReturnType
pre parameterOk: param1 > ...
post resultOk : result = ...

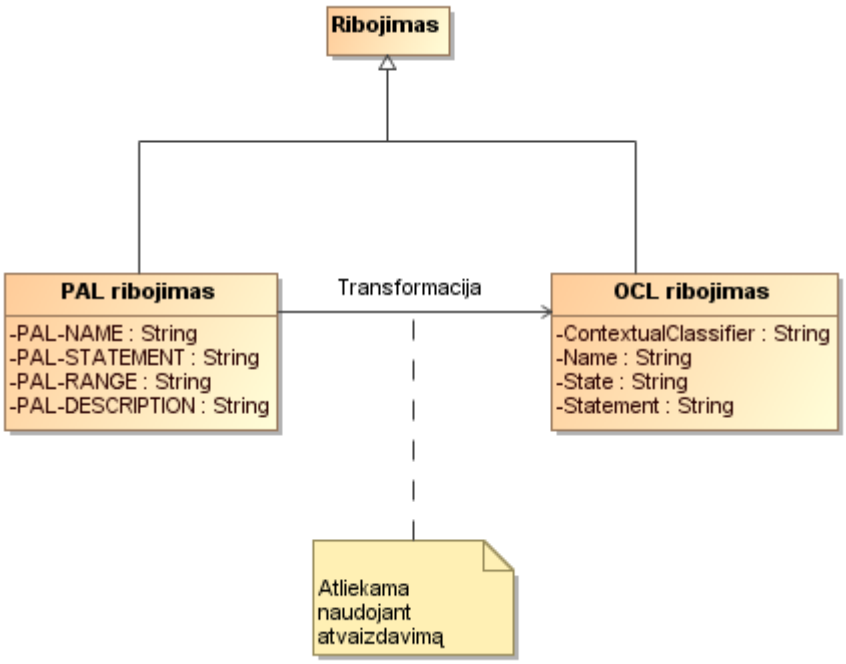
3 pavyzdys:

OCL (*Object Constraint Language*):

context Person::income : Integer
init: parents.income->sum() * 1% -- pocket allowance
derive: if underAge
        then parents.income->sum() * 1% -- pocket allowance
        else job.salary -- income from regular job
        endif

PAL (Protege Axiom Language)

```
(%3APAL-NAME "discount-10-percent")
(%3APAL-DESCRIPTION "Nuolaidos dydis priklauso nuo perkamų prekių kiekio. Jeigu kiekis yra nuo 50 iki 100, tai nuolaida yra 10%.\n")
(%3APAL-STATEMENT "(forall ?Contract_Product
        (=> (and
                (> ('quantity' ?Contract_Product) 50)
                        (< ('quantity' ?Contract_Product) 100))
                (= ('discount' ?Contract_Product) 0.1)))"
```



15 pav. PAL ir OCL struktūriniai metamodeliai ir jų transformacija klasių diagramoje

OCL ribojimo išraiškos schema:

context TypeName inv constraint-name:

-- comment

[if] OCL statement [[then] OCL statement endif]

OCL statement dalis:

self.[TypeName].attribute [operation | function] [self.[TypeName].attribute | function]

Pagrindinė OCL savybė lyginant su PAL ar net SQL kalbomis yra ta, kad ribojimas susiejamas su kontekstine klase per kurią ir jos ryšius išreiškiama ribojimo sąlyga.

3.7. Aksiomų transformacija į OCL

Pagal pateiktą žemiau pateiktos lentelės pavyzdį aprašysime metodo taikymą, aksiomos transformuojamos į OCL. Stengsimės išlaikyti panašią į praėjusios realizacijos struktūrą. Pirmoje dalyje pateiktas PAL ribojimas, saugomas projekto .pins faile. Antroje dalyje gaunamas rezultatas OCL ribojimas. Panaudotos tos pačios aksiomos.

9 lentelė. PAL ribojimo gautas rezultatas

PAL ribojimas
(%3APAL-DESCRIPTION "A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.") (%3APAL-NAME "customer-not-supplier-in-contract")

(%3APAL-RANGE "(defrange ?Contract :FRAME Contract)\n") (%3APAL-STATEMENT "(forall ?Contract\n (=> (and (own-slot-not-null 'customer' ?Contract) \n (own-slot-not-null 'supplier' ?Contract))\n (not (= ('customer' ?Contract) ('supplier' ?Contract)))))\n"))
Gautas rezultatas – modeliavimo ribojimas, OCL ribojimo pavidalu
' A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.' context Contract inv customer-not-supplier-in-contract: if (self.customer ->notEmpty() and self.supplier ->notEmpty()) then (self.customer<>self.supplier)

Transformacija atliekama principu pateiktu žemiau lentelėje, dalys formuojamos ir pridedamos nuosekliai.

10 lentelė. Transformavimas į OCL

PAL ribojimo paimta dalis	**Išgauta dalis**	**Pridėta dalis**	**Rezultatas OCL sintakse**
(%3APAL-DESCRIPTION "A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.")	A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.	' *Išgauta dalis* '	' A Customer can not be a Supplier at the same time in the same Contract<=>For every instance of Class Contract, Slot customer: Class Contract and Slot supplier: Class Contract cannot have the same value.'
(%3APAL-NAME "customer-not-supplier-in-contract")	customer-not-supplier-in-contract	*Išgauta dalis* :	customer-not-supplier-in-contract:
(%3APAL-RANGE "(defrange ?Contract :FRAME Contract)\n")	Contract	context *Išgauta dalis*	context Contract

Alternatyvus panaudojimas, jei nebūtų RANGE dalies: "(forall ?Contract	Contract	context *Išgauta dalis*	context Contract
forall	inv	*Išgauta dalis*	inv
(=>	If	*Išgauta dalis*	if
(%3APAL-STATEMENT "(forall ?Contract\n (=> (and (own-slot-not-null 'customer' ?Contract) \n (own-slot-not-null 'supplier' ?Contract))\n (not (= ('customer' ?Contract) ('supplier' ?Contract)))))\n"))	(self.customer ->notEmpty() and self.supplier ->notEmpty() and self.customer<>self.supplier)	If *Išgauta dalis*	if (self.customer ->notEmpty() and self.supplier ->notEmpty()) then (self.customer<>self.supplier)
Detalesnis STATEMENT panaudojimas			
(=> (and (own-slot-not-null 'customer' ?Contract) \n	(self.customer ->notEmpty()	If *Išgauta dalis* and	If (self.customer ->notEmpty() and
(own-slot-not-null 'supplier' ?Contract))\n	Contract.supplier ->notEmpty())	*Išgauta dalis* then	Contract.supplier ->notEmpty()) then
(not (= ('customer' ?Contract) ('supplier' ?Contract)))))\n"))	(self.customer<>self.supplier)	*Išgauta dalis*	(self.customer<>self.supplier)

3.8. Išgautų OCL ribojimų naudojimas sistemai formuoti

Išgavus aksiomas ir transformavus į OCL sintaksę pagal aprašytą metodą, gaunami OCL ribojimai, kuriems reikalinga UML klasių diagrama. UML diagrama gaunama iš Protege panaudojus plėtinį Backend. Gautus ribojimus galima taikyti sistemos modeliavimui ir kūrimui. Paminėtina, kad aksiomose neretai nėa *RANGE* dalies, todėl kontekstine klase imama prie *forrall* nurodyta klasė.

Jeigu lygintume pirmosios transformacijos gautą rezultatą su šiuo, matome, kad papildoma adaptacija po transformacijos nebūtina, tačiau kadangi rezultatai įkeliami į UML rankiniu būdu, ją lengvai galima atlikti.

3.9. Axiom2OCL plėtinio ir programinio kodo aprašymas

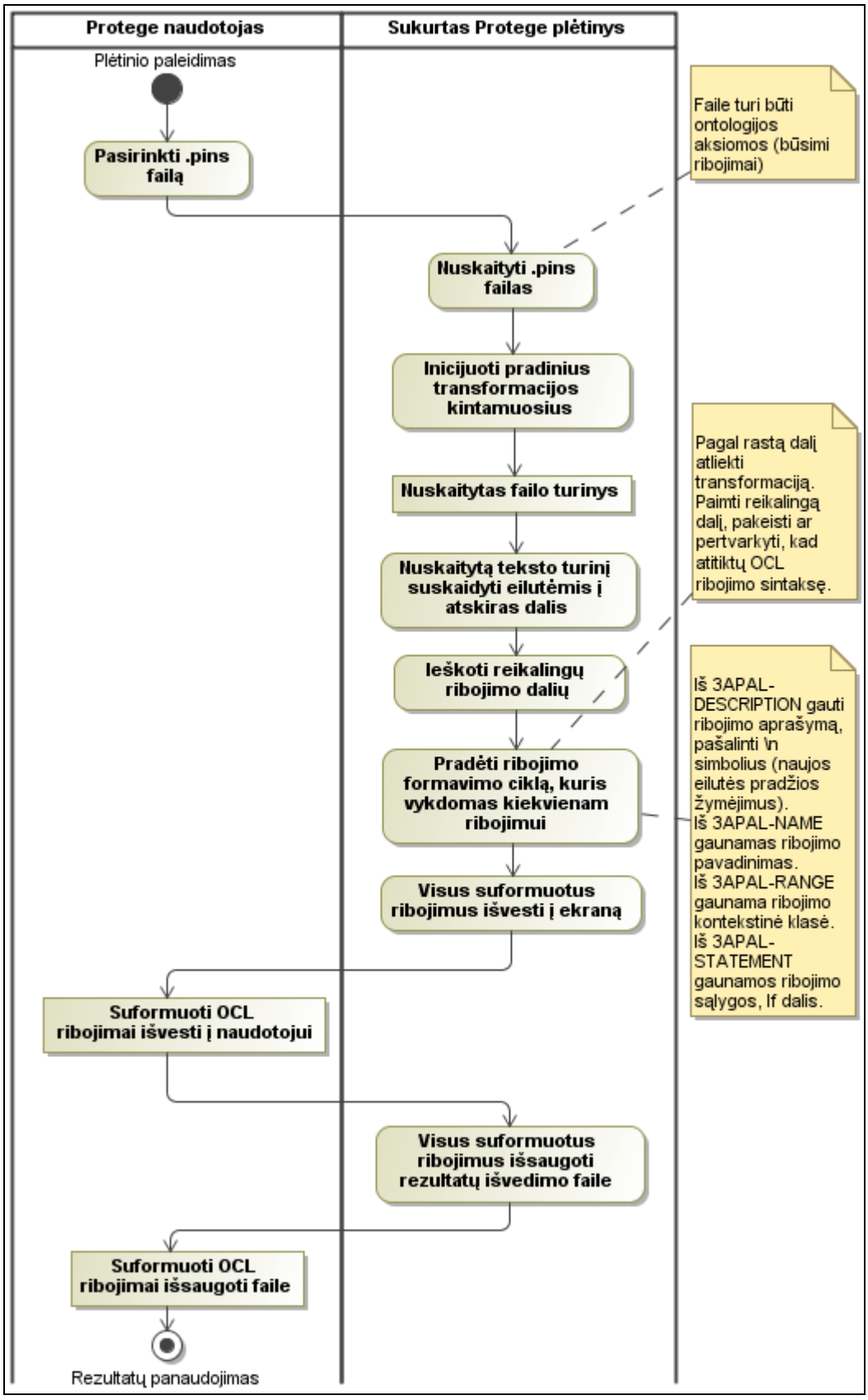
Sukurtas Protégé plėtinys – programos prototipas, kuris išrenka iš Protégé .pins failo aksiomų kalba (PAL ir EZPAL) parašytas ribojimų aksiomas. Antrasis plėtinys, pavadintas „Axiom2OCL“, atlieka tokių pat dažniausiai naudojamų ribojimų tipų transformaciją į OCL ribojimus, naudojamus UMLinformacinių sistemų modeliavimui. Programinis kodas yra parašytas Java kalba (visas programinis kodas yra pateiktas 3-iame priede). Plėtinys pridedamas prie Protégé ontologijų kūrimo įrankio „Axiom2OCL.class“ failo pavidalu, saugomas „VETIS“ direktorijoje. „VETIS“ direktorija įdedama įrankio „plugins“ direktorijoje, kur saugomi visi Protégé plėtiniai. Tam, kad tinkamai veiktų Protégé plėtinys, jam būtina com.stevesoft.pat nemokama biblioteka, kuri yra saugoma faile patbinfree153.jar. Plėtinys taip pat naudoja standartines Java bibliotekas, o rezultatą įrašo į C disko „data“ direktoriją, kuri turi būti sukurta. Plėtinis sukurtas naudojant eksportavimo plėtinio pavyzdį, kuris buvo atitinkamai modifikuotas - jis iškviečia paleidimui atitinkamą transform() funckiją. Aboiejų plėtinių veikimas pavaizduoti paveiksluose, kurie pateikti 1 priede. Plėtinys transformuoja ~20 PAL ir EZPAL skirtingų ribojimo *STATEMENT* dalies sintaksės šablonų ir įvairias jų variacijas.

3.9.1. Algoritmas

Aprašymo pradžioje pateiktas išsamus Protégé plėtinio algoritmo aprašymas. Grafiškai algoritmas pateiktas vėliau einančioje paveiksle.

- Pasirinkti .pins failą, kuriame yra ribojimų aksiomų PAL kalba.
- Nuskaityti .pins failas.
- Inicijuoti pradinius transformacijos kintamuosius.
- Nuskaitytą teksto turinį suskaidyti eilutėmis į atskiras dalis.
- Ieškoti reikalingų ribojimo dalių.
- Pradėti ribojimo formavimo ciklą, kuris vykdomas kiekvienam ribojimui.
    - Eilutes tikrinti po vieną, ar konkreti eilutė aprašo ribojimo aksiomą. Eilutėje ieškoti

3APAL-NAME, 3APAL-DESCRIPTION, 3APAL-RANGE, 3APAL-STATEMENT simbolių sekų, kurios apibūdina keturias ribojimo aksiomos dalis.
    - Pagal rastą dalį atlikti transformaciją. Paimti reikalingą dalį, pakeisti ar pertvarkyti, kad atitiktų OCL ribojimo sintaksę.
    - Iš *3APAL-DESCRIPTION* gauti ribojimo aprašymą, pašalinti *\n* simbolius (naujos eilutės pradžios žymėjimus).
    - Iš *3APAL-NAME* gaunamas ribojimo pavadinimas.
    - Iš *3APAL-RANGE* gaunama ribojimo kontekstinė klasė.
    - Iš *3APAL-STATEMENT* gaunamos ribojimo sąlygos, If dalis. Eilutės duomenys suskaidomi į atskiras dalis ties *\n* simboliais. Vėliau vykdomas vidinis If dalies formavimo ciklas, kurio metu gautos *3APAL-STATEMENT* eilutės dalys lyginamos su aprašytomis programoje konkrečiais transformavimo šablonais. Nustačius eilutės dalies atitinkamą šabloną, atliekama transformacija į OCL.
    - Atsižvelgiant į kiekvienos dalies skliaustelių pozicijas, skaičių bei kitus parametrus nustatyti If dalies jungiamuosius žodžius (AND, OR) ir jų padėtis.
    - Pilnai suformuoti if dalį iš sąlygų.
    - Suformuoti ribojimą (-us).
- Visus suformuotus OCL ribojimus išvesti naudotojui.
- Visus suformuotus OCL ribojimus išsaugoti rezultatų išvedimo faile.



16 pav. PAL-OCL transformacijos algoritmas UML veiksmų digramoje

3.9.2. Testavimui naudotos laikraščio ontologijos aksiomos

Testavimui naudotos tokios pat aksiomos kaip ir PAL-SQL plėtinio atveju (lentelė pateikta prie

pirmojo plėtinio testavimo).

3.9.3. Testavimo tvarka

Plėtinio testavimo ir derinimo metu buvo atlikti žemiau pateikti veiksmai.

1) Pasirenkama testavimui naudojamos laikraščio ontologijos .pins formato failas, kuriame saugomi ribojimai.
2) Nagrinėjami gauti rezultatai. Rezultatai lyginami su panašiais OCL ribojimų pavyzdžiais. Jei buvo rasta klaidų, plėtinys tobulinamas, kol tinkamai vykdo transformaciją. Taip pat atliekamas validavimas naudojant UML Backend sugeneruotojo klasių diagramoje.
3) Rezultatai pateikiami įvertinti ekspertui (darbo konsultantei D. Kalibatienei).
4) Tobulinamas plėtinys, šalinamos nustatytos klaidos. Kartojamas testavimas iki tol, kol bus tinkamai vykdoma visų aprašytų aksiomų transformacija.

3.9.4. Testavimo rezultatai

Žemiau pateiktas laikraščio ontologijos 3 ribojimų aksiomų išgavimas ir transformavimas į OCL robojimus. Pagal galutinius rezultatus galima spręsti, kad plėtinys tinkamai atlieka savo funkciją, išgavimą ir transformavimą, kadangi tiksliai atlieka Aksiomų transformacija į OCL poskyryje aprašytą transformavimą.

STATEMENT dalyje sąlygos ties \n buvo suskaidytos siekiant parodyti, kokiomis dalimis programoje nagrinėjama eilutė.

3.9.4.1. Aksioma editor-employees-salary-constraint

PAL

```
([newspaper_00000] of %3APAL-CONSTRAINT
        (%3APAL-NAME "editor-employees-salary-constraint")
        (%3APAL-DESCRIPTION "The salary of an editor should be greater than the salary of any employee which the editor is responsible for.\n\nNote that the range definition for ?editor could be omitted because this constraint is attached to the Editor class.")
        (%3APAL-RANGE "(defrange ?editor :FRAME Editor)\n
(defrange ?employee :FRAME Employee responsible_for)")
        (%3APAL-STATEMENT "(forall ?editor (forall ?employee\n
 (=> (and \n
(responsible_for ?editor ?employee)\n
 (own-slot-not-null salary ?editor) \n
 (own-slot-not-null salary ?employee)) \n
                (> (salary ?editor) (salary ?employee)))))"))
```

OCL

' The salary of an editor should be greater than the salary of any employee which the editor is responsible for.'

context editor inv editor-employees-salary-constraint:
if (self.responsible_for->notEmpty() and self.salary ->notEmpty() and employee.salary ->notEmpty())
then (self.salary>self.responsible_for.salary)

3.9.4.2. Aksioma article-section-constrained-by-author

PAL

([newspaper_00001] of %3APAL-CONSTRAINT
(%3APAL-NAME "article-section-constrained-by-author")
(%3APAL-DESCRIPTION "If the author of an article is a person, then there exists an editor, who is either this author or the editor responsible for this author. The section that contains this article is one of the sections managed by this editor.\n\nNote that the range definition for ?article could be omitted because this constraint is attached to the Article class.")
(%3APAL-RANGE "(defrange ?article :FRAME Article)\n(defrange ?editor :FRAME Editor)")
(%3APAL-STATEMENT "(forall ?article\n
(=> (instance-of (article_author ?article) Person) \n
(exists ?editor \n
(and (or (article_author ?article ?editor) \n
(responsible_for ?editor (article_author ?article))) \n
(sections ?editor (containing_section ?article))))))"))

OCL

' If the author of an article is a person, then there exists an editor, who is either this author or the editor responsible for this author. The section that contains this article is one of the sections managed by this editor. Note that the range definition for ?article could be omitted because this constraint is attached to the Article class.'
context article inv article-section-constrained-by-author:

if (self.article_author=Person.article_author) then (self.article_author->notEmpty() or editor.responsible_for=self.article_author) and (editor.sections=self.containing_section)

3.9.4.3. Aksioma articles-more-than-2-keywords

PAL

([newspaper_00002] of %3APAL-CONSTRAINT
(%3APAL-NAME "articles-more-than-2-keywords")

(%3APAL-DESCRIPTION "All articles should be indexed with more than two keywords (equivalent to minimum cardinality facet on slot attachment)\n\nBecause this constraint is attached to the Article class, the variable ?article implicitly ranges over all instances of the Article class.\n")

(%3APAL-STATEMENT "(forall ?article \n
(> (number-of-slot-values keywords ?article) 2))"))

OCL

' All articles should be indexed with more than two keywords (equivalent to minimum cardinality facet on slot attachment) Because this constraint is attached to the Article class, the variable ?article implicitly ranges over all instances of the Article class. '

context Article inv articles-more-than-2-keywords:

self.keywords->size() > 2

Ribojimas aprašo sąlygą, pagal kurią *keywords* kiekis turi būti daugiau nei 2.

Atlikus rezultatų (virš 20 įvairių tipų aksiomų transformacijų) analizę, nustatyta, kad transformacija atlikta tinkamai, todėl plėtinys, realizuojantis pasiūlytą metodą, yra tinkamas taisyklių modeliui kurti ir palengvina VT sistemos formavimą ir modeliavimą.

3.10. Projektinės dalies apibendrinimas

- Buvo atliktos dvi metodo realizacijos aksiomų transformacijos į SQL prototipų trigerius ir OCL ribojimus. Pirmuoju atveju gaunami trigerių prototipai sukurti informacinei sistemai, o antruoju gaunami OC ribojimai sistemos modeliavimui naudojant UML.
- Atlikus OCL ir PAL analizę nustatyta, kad OCL ir PAL ribojimai, kurių apibūdinimui naudojama sąlyga, yra tarpusavyje suderinami su tam tikromis išlygomis, kadangi turi bendrus struktūrinius elementus.
- OCL prieš sąlygą ir po sąlygos taip pat gali būti tapatinamos su PAL ribojimais. OCL prieš sąlyga gali būti tapatinama su PAL ribojimo sąlyga. OCL po sąlyga yra tapatinama su PAL ribojimo išraiška einančia po prieš sąlygos.
- OCL išvedamos reikšmės yra tapatinamos su tam tikromis PAL ribojimo išraiškomis, kurios leidžia iš esamų reikšmių išvesti naujas reikšmes.
- OCL tam tikro tipo ribojimams, vykdomiems prieš sąlygą ir po sąlygos, išvedamoms reikšmėms ir kt., užrašyti yra naudojami tam skirti raktiniai žodžiai. PAL ribojimu galima apibrėžti daug konstrukcijų, kaip vykdomos prieš sąlygą ir po sąlygos, išvedamoms reikšmėms ir kt. Tačiau PAL ribojimuose nėra tam skirtų raktinių žodžių. OCL ribojimus atitinkančios išraiškos yra užrašomos tiesiog tam tikrais PAL sakiniais, neišsiskiriančiais ypatingais raktiniais žodžiais.
- Trigerių prototipai buvo sukurti remiantis SQL trigerių pavyzdžiais ir atitinka jų struktūrą, buvo atlikta jų funkcionalumo analizė, po kurios nustatyta, kad juos atinkamai pakeistus galima naudoti informacinės sistemos kūrimui.
- Išgavus aksiomas ir transformavus į OCL sintaksę pagal aprašytą metodą, gaunami OCL ribojimai, kurie panaudojami UML klasių diagramoje.
- Lyginant pirmosios transformacijos gautą rezultatą su antrosios, matyti, kad papildoma adaptacija po transformacijos nebūtina, tačiau kadangi rezultatai įkeliami į UML rankiniu būdu, ją lengvai galima atlikti.

3.11. Ateities darbai

- Patobulinti sukurtus transformacijų plėtinius.
- Sukurti plėtinį, atliekantį tiesioginę transformaciją iš ontologijų kūrimo įrankio į OCL palaikančią programų sistemą (UML Magic Draw).

4. IŠVADOS

- Atlikus transformacijos sąvokos ir jų tipų analizę, nustatyta, kad sistemos kūrimo metu verslo taisykles, aprašytas ontologijos aksiomomis, tikslinga transformuoti į informacijos apdorojimo taisykles, kurios tiktų pradinei sistemos kūrimo stadijai, kai įgyvendinimo sąlygos dar nėra galutinai aiškios.
- Išanalizavus transformacijų metodus, nustatyta, kad šaltinio modelio struktūra turi būti tapati su gaunamo modelio struktūra.
- Buvo nustatyta, kad Protege ontologijos aksiomų struktūra didžiąja dalimi atitinka SQL vykdomųjų taisyklių struktūrą. Taip pat buvo nustatyta, kad Protege ontologijos aksiomų struktūra gali būti atvaizduota OCL ribojimais.
- Nustatytos pagrindinės problemos, kurias spręstų ontologijos aksiomų transformacijos: sudėtingas verslo taisyklių išgavimas, apimantis, vieningos terminų sampratos trūkumą, sistemų nesuderinamumą, netinkamą kandidatų, kurie turi apibrėžti verslo taisykles, parinkimą.
- Lyginant ontologijos aksiomas ir verslo taisykles, išreikštas ECA taisyklių forma, yra nustatyta, kad aksiomose yra apibrėžiamos sąlygos ir/arba veiksmai, kurie gali būti transformuoti į ECA taisyklės sąlygas ir veiksmus naudojant įvairias realizacijas.
- Sukūrus transformacijų metodą realizuojančius programų sistemų prototipus, nustatyta, kad galima atlikti ontologijoje aprašytų verslo taisyklių transformaciją į SQL ir OCL išreikštus taisyklių rinkinius, kurie palengvintų informacinės sistemos kūrimą. Tačiau prieš galutinį SQL trigerių įgyvendinimą ir taikymą juos gali tekti pritaikyti prie konkrečios situacijos. PAL transformavus į OCL ribojimus gaunami struktūriškai tikslūs ribojimai, kurie gali būti tiesiogiai naudojami UML klasių diagramos modelyje.

5. LITERATŪRA

Alani, H.; O'Hara, K.; Shadbolt, N. Stanford Medical Informatics. Common Features of Killer Apps: A Comparison with Protégé. 2005. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://protege.stanford.edu/conference/2005/submissions/abstracts/accepted-abstract-alani.pdf>

Appukuttan, B.; Clark, T.; Reddy, S.; Tratt, L.; Venkatesh, T. A model driven approach to building implementable model transformations. 2003. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://www.tratt.net/laurie/research/publications/papers/tratt03model.pdf>

BizTalk Server Developer Center. Transformation vs. Translation. 2004. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://msdn2.microsoft.com/en-us/library/ms916577.aspx>

Business Rules Group: Defining Business Rules ~ What Are They Really? Business Rules Group, 3rd Ed. 2000. [Žiūrėta 2009-01-21]. Prieiga per Internetą: <http://rewerse.net/downloads/BRG-whatisBR_3ed.pdf>

Būgaitė, D.; Vasilecas, O. Framework on application domain ontology transformation into set of business rules. 2005. [Žiūrėta. 2009-01-07]. Prieiga per Internetą: <http://ecet.ecs.ru.acad.bg/cst05/Docs/cp/SIII/IIIB.8.pdf>

Chisholm, M. Business Rules Evangelist: Do Business Rules Fit the Systems Development Life Cycle? DMReview.com. 2004. [Žiūrėta. 2009-01-20]. Prieiga per Internetą: <http://www.dmreview.com/article_sub.cfm?articleId=1005616>

Casely-Hayford L. CCLRC Daresbury Laboratories. A comparative analysis of methodologies, tools and languages used for building Ontologies. 2005. [Žiūrėta 2009-01-21]. Prieiga per Internetą: <http://epubs.cclrc.ac.uk/bitstream/894/OntologyReport.pdf>

Corcho, O.; Fernandez-Lopez, M.; Gomez-Perez, A. Methodologies, tools and languages for building ontologies. Where is their meeting point? Elsevier Science Publishers. 2002. [Žiūrėta 2009-01-21]. Prieiga per Internetą: <http://www.cs.man.ac.uk/~ocorcho/documents/DKE2003_CorchoEtAl.pdf>

Chisholm, M. How to Build Business Rules Engine. Elsevier, Morgan Kaufman Publishers. 2004. p. 29-40

Galley, M.; Hopkins, M.; Knight, K.; Marcu, D. Galley, M.; Hopkins, M.; Knight, K.; Marcu, D. What's in a translation rule? 2003. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://www.stanford.edu/~mgalley/papers/syntax-mt.pdf>

Guarino, N. Formal Ontology and Information Systems. In Proc. of FOIS'98, Trento, Italy, IOS Press. 1998. p. 3-15. [Žiūrėta 2009-01-21]. Prieiga per Internetą: <http://www.loa-cnr.it/Papers/FOIS98.pdf>

Gomez-Perez, A.; Gernandez-Lopez M., Corcho, O. Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. Spinger-Verlag. 2004. p. 6-14

D'Ambrogio, A. A Model Transformation Framework for the Automated Building of Performance Models from UML Models. 2005. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://www.sel.uniroma2.it/pubs/2005-WOSP.pdf>

Date, C. J. Twelve Rules for Business Rules. 2000. [Žiūrėta 2009-01-21]. Prieiga per Internetą: <http://www.informit.com/content/images/0201708507/samplechapter%5Crules.pdf>

Denny, M. Ontology Building: A Survey of Editing Tools. 2004. p. 2. [Žiūrėta 2009-01-21]. Prieiga per Internetą: <http://www.xml.com/pub/a/2002/11/06/ontologies.html?page=last&x-showcontent=off>

Jouault, F.; Kurtev, I. ATLAS Group. On the Architectural Alignment of ATL and QVT. p. 3. 2005. [Žiūrėta 2009-01-23]. Prieiga per Internetą: <http://www.sciences.univ-nantes.fr/lina/atl/www/papers/ATLandQVT-PRELIMINARY%20VERSION.pdf>

Grosso, W. Stanford Medical Informatics. PAL TABs. 2006. [Žiūrėta 2009-01-15]. Prieiga per internetą: <http://protege.cim3.net/cgi-bin/wiki.pl?PALTabs>

Lublinsky, B.; Le Tien, D. Implementation of business rules and business processes in SOA. 2007. [Žiūrėta 2009-01-24]. Prieiga per Internetą: <http://www.infoq.com/articles/business-rules-processes>

Lohse, N.; Valtchanov, G.; Ratchev, S.; Onori, M.; Barata, J. Towards a Unified Assembly System Design Ontology using Protégé. 2005. [Žiūrėta 2009-01-20]. Prieiga per Internetą : <http://protege.stanford.edu/conference/2005/submissions/abstracts/accepted-abstract-lohse.pdf>

Milo, T.; Zohar, S. Using Schema Matching to Simplify Heterogeneous Data Translation. 1998. [Žiūrėta 2009-01-20]. Prieiga per Internetą.: <http://www.vldb.org/conf/1998/p122.pdf>

Murshed, Md. A.; Singh, R. Evaluation and Ranking of Ontology Construction Tools. Technical Report DIT-05-013. Informatica e Telecomunicazioni. University of Trento. 2005. [Žiūrėta 2009-01-22]. Prieiga per Internetą: <http://eprints.biblio.unitn.it/archive/00000747/01/013.pdf>

Neil, C. G.; Pons, C. Formalizing the Model Transformation Using Metamodeling Techniques. 2004. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://www.lifia.info.unlp.edu.ar/papers/2004/Claudia2004.pdf>

Navigli, R.; Velardi, P. Gangemi, A. Ontology Learning and Its Application to Automated Terminology Translation. Intelligent Systems, IEEE. 2003. p. 24. [Žiūrėta 2009-01-21]. Priega per Internetą : <http://www.loa-cnr.it/Papers/IEEE2003.pdf>

Ressler, J.; Dean, M.; Benson, E.; Dorner, E.; Morris, C. Application of Ontology Translation. 2007 [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://iswc2007.semanticweb.org/papers/823.pdf>

Ross, R. G. Principles of the Business Rule Approach. Addison-Wesley Professional (2003). p. 185.

Stanford Medical Informatics. PAL Conceptual Framework. 2006. [Žiūrėta 2009-01-24]. Prieiga per Internetą: <http://protege.stanford.edu/plugins/paltabs/pal-documentation/lang_framework.htm>

OMG. UML 2.0 OCL Specification. p. 21-25. 2003. [Žiūrėta 2009-09-20]. Prieiga per Internetą: <http://www.omg.org/docs/ptc/03-10-14.pdf>

OMG. Object Constraint Language OMG Available Specification Version 2.0. p. 5. 2006. [Žiūrėta 2009-04-07]. Prieiga per Internetą: <http://www.omg.org/docs/formal/06-05-01.pdf>

Oracle. Rule Enabling Applications with Oracle Business Rules. 2005. p. 5, 8. [Žiūrėta 2009-01-24]. Prieiga per Internetą: <http://www.oracle.com/technology/products/ias/business_rules/pdf/businessWhitepaper.pdf>

da Silva, C. F.; Vankeisbelck R.; Lima, C. Final Draft CWA5 Proposal European eConstruction Software Implementation Toolset (EeS). 2004. p. 24-26.

Stanford Medical Informatics.What is Protégé-OWL. 2007. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://protege.stanford.edu/overview/protege-owl.html>

Su, X.; Ilebrekke, L.; Banks, A. A Comparative Study of Ontology Languages and Tools. Springer Berlin / Heidelberg. 2002. [Žiūrėta 2009-01-20]. Prieiga per Internetą: <http://www.idi.ntnu.no/~xiaomeng/paper/caise02WorkshopCRC.pdf>

Šoblinskas, T.; Būgaitė, D. „Verslo taisyklių transformacijos kuriant informacines sistemas.“ 11-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“. 2008. p. 79-86

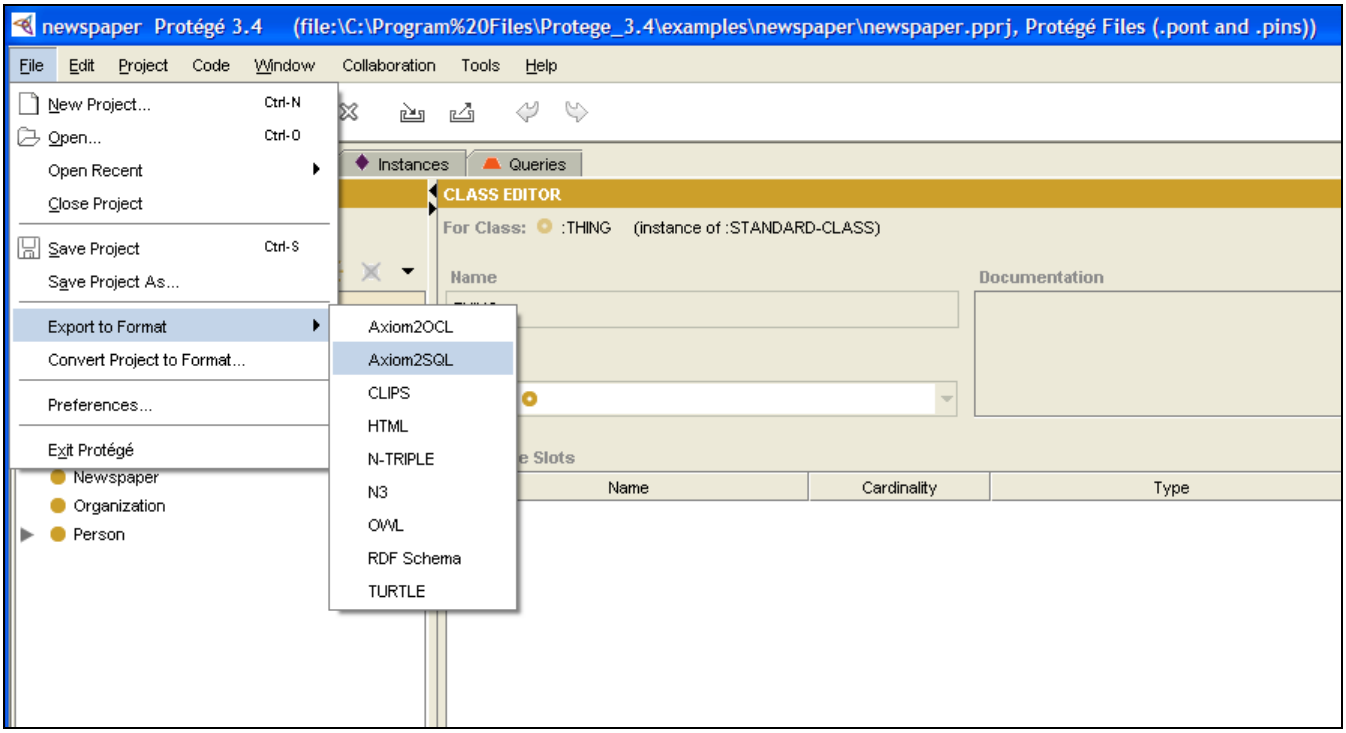
UML Backend. 2007. [Žiūrėta 2009-03-19]. Prieiga per Internetą: <http://protege.cim3.net/cgi-bin/wiki.pl?UMLBackend>

Von Halle, B. What is a Business Rules Approach? TDAN.com. 2002. [Žiūrėta 2009-01-24]. Prieiga per Internetą: <http://www.tdan.com/view-articles/4983>

Zimbrão, G.; Miranda, R.; de Souza, J. M.; Estolano, M. H.; Neto, F. P. Enforcement of Business Rules in Relational Databases Using Constraints. 2002. p. 4-5. [Žiūrėta 2009-01-24]. Prieiga per Internetą: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbbd/2003/paper010.pdf>

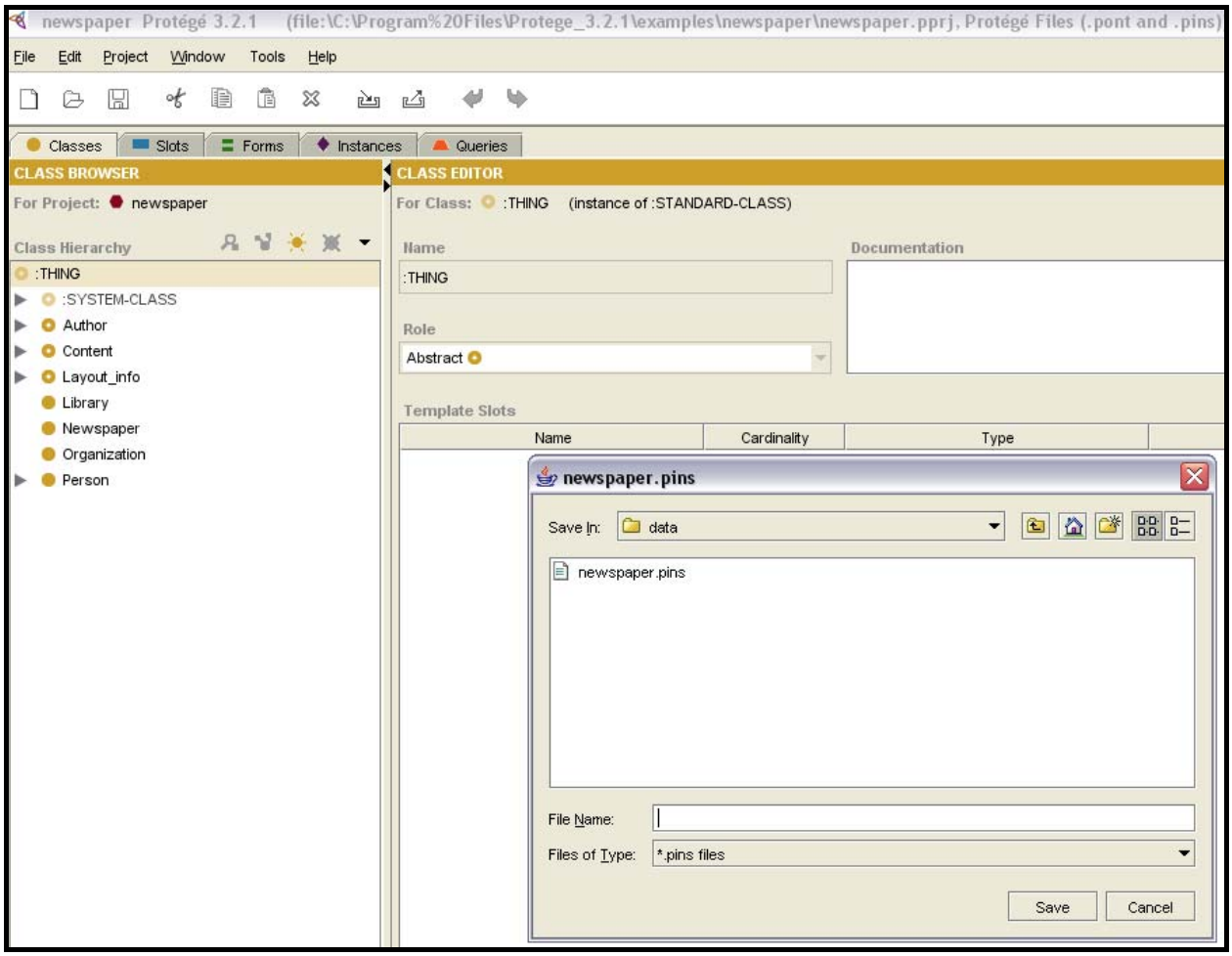
6. PRIEDAI

6.1. Priedas. Papildomi paveikslai



17 pav. Transformavimo plėtinio paleidimas

Pateiktame paveiksle vaizduojamas plėtinio(ių) paleidimas naudojant Protégé įrankį. Buvo sukurti du plėtiniai: Axiom2OCL ir Axiom2SQL.

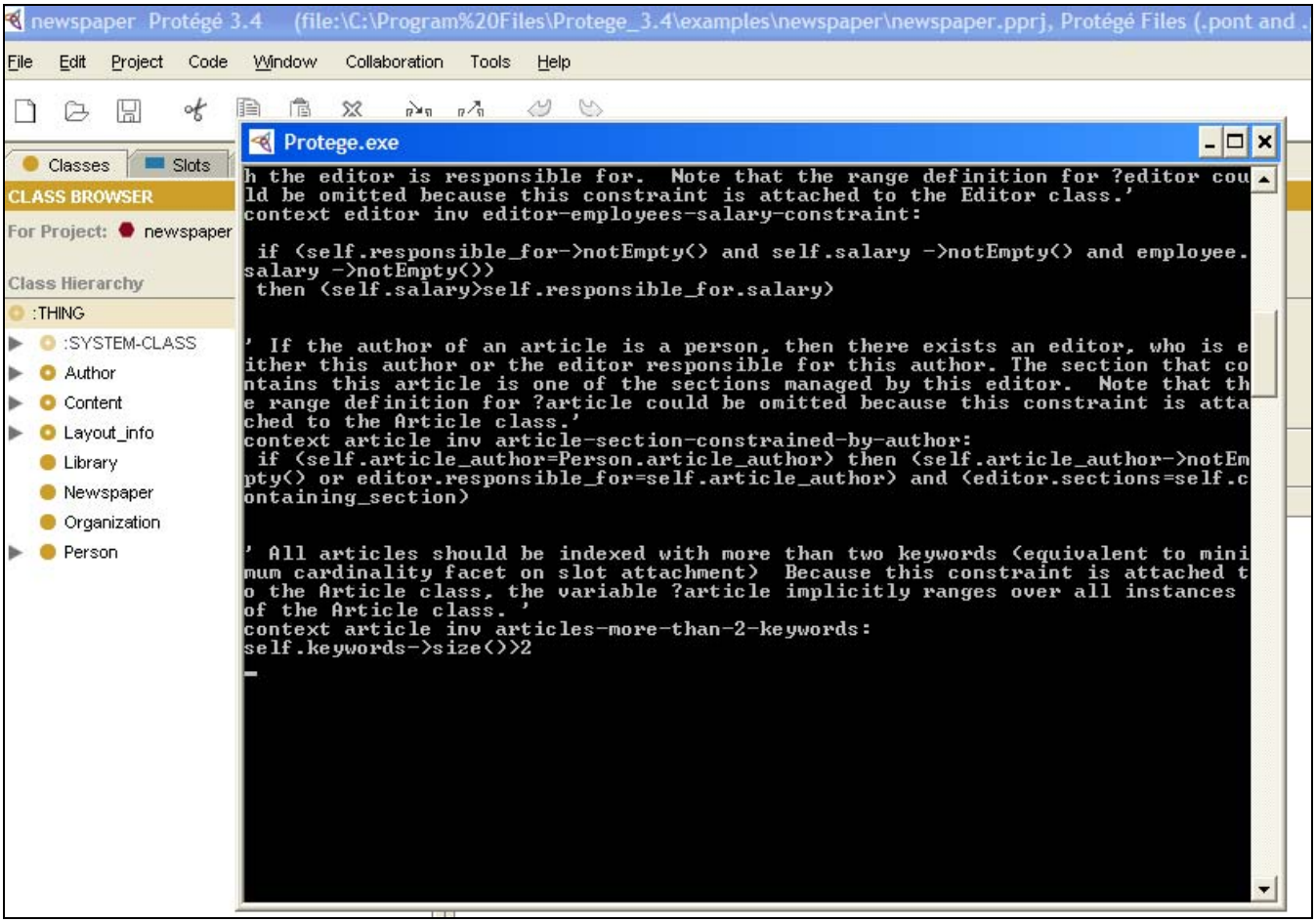


18 pav. Paleidus plėtinį atsidaro pasirinkimo langas

Pateiktame paveiksle pavaizduotas atsidarantis langas, kuriame galima pasirinkti bet kurį .pins formato failą aksiomų išgavimui.

19 pav. PAL-SQL rezultatų išvedimas ekrane

Pateiktame paveiksle vaizduojamas išvedamas pradinis keičiamas aksiomos tekstas ir gaunamas rezultatas, toks pat tekstas išvedamas ir į tekstinį failą output.sql, esantį C:\data direktorijoje. Analogiškas išvedimas padarytas ir PAL-OCL transformacijai.



20 pav. PAL-OCL rezultatų išvedimas ekrane.

Pateiktame paveiksle vaizduojamas išvedamas pradinis keičiamas aksiomos tekstas ir gaunamas rezultatas, toks pat tekstas išvedamas ir į tekstinį failą output.txt, esantį C:\data direktorijoje.

6.2. Priedas. Axiom2SQL programinis kodas

Abiejų plėtinių klasės pajungiamos prie Protege, jas padėjus į <Protege įrašymo katalogas>\plugin į Axiom2BR katalogą bei įdėjus manifest failą (į katalogą META-INF) su turiniu:

```
Manifest-Version: 1.0
Main-Class: edu/stanford/smi/protege/Application

Name: Axiom2BR/Axiom2SQL.class
Export-Plugin: True

Name: Axiom2BR/Axiom2OCL.class
Export-Plugin: True
```

Pateikiamas Axiom2SQL.java turinys. Detalus aprašymas buvo pateiktas prie plėtinio ir programinio kodo aprašymo.

```
package Axiom2BR;
/* Sukure Tautvydas Soblinskas*/
import java.io.*;
import java.util.*;
import javax.swing.*;
import edu.stanford.smi.protege.*;
import edu.stanford.smi.protege.model.*;
import edu.stanford.smi.protege.plugin.*;
import edu.stanford.smi.protege.util.*;
import java.util.regex.*;
import com.stevesoft.pat.*;
import java.util.regex.Matcher;

public class Axiom2SQL implements ExportPlugin {
    private static final String EXTENSION = ".pins";

    public static void main(String[] args) {
        Application.main(args);
    }

  public String getName() {
        return "Axiom2SQL";}
//nurodomas pasirenkamos f-jos vardas


    public void dispose() {
    }

   public void handleExportRequest(Project project)  {
        File file = promptForFooFile(project);
       // Gaunamas failas
        if (file != null)
        {
   //       saveToFile(project, file);
           }
     }

    //parenkamas failas
    public File promptForFooFile(Project project) {
        String name = project.getName();
        String proposedName = new File(name + EXTENSION).getPath();
        JFileChooser chooser = ComponentFactory.createFileChooser(proposedName, EXTENSION);
        File file = null;
        if (chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
            file = chooser.getSelectedFile();
        }
       ///tam to, kad pamgintu pagauti
        try {transform(file);}
        catch(java.io.IOException exp){ exp.printStackTrace();}
        return file;
     }
```

```
   private void saveToFile(Project project, File file) {
       PrintWriter writer = FileUtilities.createPrintWriter(file, false);
      //saveProject(project, writer);
      writer.close();
    }

    // Isveda kiekviena klas? atskiroje eilut?je
    private void saveProject(Project project, PrintWriter writer) {
    }

////////////// Funkcijos ////////////////////////////

  /* Paima visa failo turini ir grazinu stringu (eilute).
  * @param aFile yra failas, kuris jau egzistuoja ir gali butt perskaitytas.
  */
static public String getContents(File aFile) {
    StringBuffer contents = new StringBuffer();

    BufferedReader input = null;
    try {
      input = new BufferedReader( new FileReader(aFile) );
      String line = null; //ne deklaruotas cikle

      /* Grazina null tik srauto pabaigoje.
      * Grazina eilute, jei dvi eilutes pasitaiko iseiles.*/
      while (( line = input.readLine()) != null){
        contents.append(line);
        contents.append(System.getProperty("line.separator"));
      }
    }
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
    catch (IOException ex){
      ex.printStackTrace();
    }
        finally {
      try {
        if (input!= null) {
          input.close();
        }
      }
      catch (IOException ex) {
        ex.printStackTrace();
      }
    }
    return contents.toString();
  }
/////////////////////////////////////////////////////////////////////////
  /* Keicia failo turini nauju tekstu.
  * @param aFile - egzistuojantis faila i kuri galima irasyti.
  * @throws IllegalArgumentException, jei param neivykdomas.
  * @throws FileNotFoundException, jei failas neegzistuoja.
  * @throws IOException, ivyksta problema irasymo metu.*/

public void setContents(File aFile, String aContents)
                          throws FileNotFoundException, IOException {
    if (aFile == null) {
      throw new IllegalArgumentException("Failas neturi buti tuscias.");
    }
    if (!aFile.exists()) {
      throw new FileNotFoundException ("Neegzistuoja failas: " + aFile);
    }
    if (!aFile.isFile()) {
      throw new IllegalArgumentException("Klaida susijusi su pateikiamu kategorijos: " + aFile);
    }
    if (!aFile.canWrite()) {
      throw new IllegalArgumentException("Negalima irasyti i faila: " + aFile);
    }

    Writer output = null;
    try {
      //naudojamas buferis
```

```
      //FileWriter pagal pradin? nustatyma visada teigia, kad uzkodavimas tinkamas
      output = new BufferedWriter( new FileWriter(aFile) );
      output.write( aContents );
    }
    finally {
      //uzdaro output.
      if (output != null) output.close();
    }}

//////////////////////Pabaiga
///////////////// Didziausia transformavimo f-ja

public void transform (File input) throws IOException {

    //faila pavadinimai
    String tempfpav = "C:/data/datatmp.txt"; //laikinas failas is kurio bus nuskaitoma
    String outputfpav = "C:/data/output.sql";

    File f = new File(tempfpav);

   // Kopijavimo pradzia
    InputStream in = new FileInputStream(input);
    OutputStream out = new FileOutputStream(f);

   // Failo turinio perkopijavimas i laikina faila
    byte[] buf = new byte[1024];
    int len;
    while ((len = in.read(buf)) > 0) out.write(buf, 0, len);
      in.close();
      out.close();
   /// Kopijavimo pabaiga

     File f2 = new File(outputfpav);
        try {
           // sukuria f2 faila, jei tokio n?ra
            boolean success2 = f2.createNewFile();
         }
        catch (IOException e) { }

// f failo turinys paimamas su getContents(f), kurio formatas vientisa eilute (String).
Scanner s = new Scanner(getContents(f)).useDelimiter("\n"); // Eilute nuskaitoma dalimis po eilute.

      setContents(f2, ""); //Isvalo isvedimo fail?.

     //**//Ciklo kintamieji//**//
     int i=0; //visus pereina per cikla parodydamas dalis
     //trigerio prototipui isvedimui kintamieji
      String nauddesc = "";  //DESCRIPTION dalies pa?mimui
      String naudname = ""; //NAME dalies pazmimui
      String naudrange = ""; //RANGE dalies pazmimui
      String naudstatement = ""; //STATEMENT dalies pa?mimui
      boolean trigbaigtas = false;

      //Pasitikrinimui kintamieji
      String pirmalent = null;
      String antralent = null;
      String trecialent = null;
      String raktas = null; //sarysio raktui

      //Salyg? dalies kintamieji
      String Input = null;
      String beginstate = null;
      boolean boolor=false; //nurodo ar tesiasi OR grandine
      boolean booland=false; //nurodo ar tesiasi AND grandin?, zymima jos pradzia ir pabaiga
      boolean ifon=false; // zymi pirmine dali nuo IF/EXIST iki )) (ar daugiau skliaustu)
      int nrand=0; // skaiciuoja AND dalis (neskaiciuojant dideliu tarpusavio salygos sakiniu daliu)
      int nror=0; //skaiciuoja OR dalis
      boolean existson=false; //nurodo ar pries tai buvo EXIST tipo salyga
      boolean galpab=false; //ziuri ar galima salygos sakiniu pabaiga, zymeklis
      boolean praleist=false; //naudojamas praleidimui pazymeti
      boolean buvoskl=false;

      //Vidinio masyvo kintamieji, naudojami perrinkti per ivairius variantus
      String[] Mas = {">","<" ,"="};
```

```
    int n=0;
  // ********** ///Pagrindinis ciklas/// **************
  // Ciklo metu atrenkamos reikalingos eilutes
  while(s.hasNext()) // ziuri ar egzistuoja kita eilute
  {

    String naudojamas = s.next(); //eilut?s stringas permetamas kintamajam

    // su 3APAL-NAME eiluciu isrinkimas
    //nurodomas pilnas funkcijos ir bibliotekos pavadinimas, kad atrinktu kuria naudoti
    java.util.regex.Pattern p1 = java.util.regex.Pattern.compile("3APAL-NAME"); //ko ieskoma
    Matcher m1 = p1.matcher(naudojamas); // kur bus keiciama/ieskoma

    if(m1.find()) // ziuri ar true (ar yra ieskomas gabalas) | suradimas
    {

    // Vieta transformacijai
    // paima PAL constraint pavadinima ir ideda ji i SQL trigerio pavadinima
       Regex name = new Regex("(?is)%3APAL-NAME (?=\"(.+)\")");
       name.search(naudojamas);
       naudojamas = "CREATE TRIGGER " + name.stringMatched(1); //i "naudojamas" kintamaji idedama isrinktas NAME
       naudname = naudojamas;
    }

 //// su 3APAL-DESCRIPTION eiluciu isrinkimas
   java.util.regex.Pattern p2 = java.util.regex.Pattern.compile("3APAL-DESCRIPTION"); //ko ieskoma
   Matcher m2 = p2.matcher(naudojamas); // kur bus keiciama/ieskoma

    if(m2.find())
    {
       /// Vieta transformacijai
       Regex r = new Regex("(?im)%3APAL-DESCRIPTION (?=\"(.+)\")");
       r.search(naudojamas);
        naudojamas = "\n/* Documentation */ " + "\n" + "/* " + r.stringMatched(1) + " */";
        nauddesc = naudojamas;

       //Pasalina /n ir patalpina tarpe
       java.util.regex.Pattern p = java.util.regex.Pattern.compile("\\\\n"); //ko ieskoma
       Matcher m = p.matcher(nauddesc); // (get a matcher object) kur bus keiciama/ieskoma
       nauddesc = m.replaceAll(" ");
    }

    ///////////////// su 3APAL-RANGE  eilu?i? i□rinkimas
   java.util.regex.Pattern p3 = java.util.regex.Pattern.compile("3APAL-RANGE "); //ko ieskoma
    Matcher m3 = p3.matcher(naudojamas); // kur bus keiciama/ieskoma

    if(m3.find())
    {
     /// Vieta transformacijai
      Regex range = new Regex("([[A-Z_a-z0-9.]+) :FRAME ([[A-Z_a-z0-9.]+)\\)"); //randa 1 lentele
      Regex range2 = new Regex("([[A-Z_a-z0-9.]+) :FRAME ([[A-Z_a-z0-9.]+)[]");//randa 2 lentele
      Regex range21 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)"); //randa antra lentele
2 var.
      Regex range3 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+) ([A-Z_a-z0-9.]+)");
//randa sarysio atributa

       range.search(naudojamas);
       range2.search(naudojamas); range21.search(naudojamas);
       range3.search(naudojamas);

       if (range3.didMatch()) {raktas = range3.stringMatched(3);}
       naudrange = "ON " + range.stringMatched(1);
       naudrange = naudrange+ "\n{FOR | AFTER | INSTEAD OF}" + "\n{[DELETE] [,] [INSERT] [,] [UPDATE]} \nAS";
       pirmalent = range.stringMatched(1); //priskirimas 1 lentelei
       antralent = range2.stringMatched(1); //priskirimas 2 lentelei

       if (antralent==null&&range21.didMatch()) {antralent = range21.stringMatched(1);}
       if (pirmalent!=null&&antralent!=null) {
        naudrange = "ON " + " {" + pirmalent + "|" + antralent + "}";

     //Jei trys FRAME dalys
        Regex range213 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)\\)\\\\n\\(defrange
\\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)"); //
       range213.search(naudojamas);
```

```
     // PVZ., (%3APAL-RANGE "(defrange ?Order1 :FRAME Order)\n(defrange ?Order2 :FRAME Order)\n(defrange ?Order3
:FRAME Order)\n")
         if (range213.didMatch()) {trecialent=range213.stringMatched(3); naudrange = "ON " + " {" + pirmalent + "|" +
antralent + "|"  + trecialent + "}";}


       naudrange = naudrange+ "\n{FOR | AFTER | INSTEAD OF}" + "\n{[DELETE] [,] [INSERT] [,] [UPDATE]} \nAS"; }
}

    ///////////////// su 3APAL-STATEMENT eiluciu isrinkimas
    java.util.regex.Pattern p4 = java.util.regex.Pattern.compile("3APAL-STATEMENT"); //ko ieskoma
    Matcher m4 = p4.matcher(naudojamas); // kur bus keiciama/ieskoma

  if(m4.find())
    {  /// Vieta transformacijai

      // Vidinis skaneris atrinkti s?lygoms ////
    Scanner s2 = new Scanner(naudojamas).useDelimiter("\\\\n"); // String nuskaitoma dalimis suskaidant ties \n.
     //Vidinis ciklas atrinkti dalims

    int eilnr = 0;
    String laikinas=null;
    while(s2.hasNext()) // ziuri ar egzistuoja kita eilute
   {

    eilnr=eilnr+1;
    String dalis = s2.next(); //eilutes stringas priskiriamas kintamajam
    System.out.println(dalis);

    java.util.regex.Pattern p41 = java.util.regex.Pattern.compile("forall"); //ko ieskoma
    Matcher m41 = p41.matcher(dalis); // kur bus keiciama/ieskoma
    if (m41.find()){
      naudstatement="FOR EACH ROW\nIF (";
      //jei rastas forall pridedamas FOR EACH ROW
      if (naudrange==null) { //suveikia jei nera apibrestos lenteles
          Regex range = new Regex("forall \\?([A-Z_a-z0-9.]+)");
          range.search(dalis);
          if (range.stringMatched(1)!=null) {
              naudrange="ON " + range.stringMatched(1);
           naudrange = naudrange + "\n{FOR | AFTER | INSTEAD OF}" + "\n{[DELETE] [,] [INSERT] [,] [UPDATE]} \nAS";
          }
      }
    }
    // pvz., (not (and (
    java.util.regex.Pattern p411 = java.util.regex.Pattern.compile("\\(not \\(and \\("); //ko ie□koma
    Matcher m411 = p411.matcher(dalis); // kur bus keiciama/iesskoma
    if (m411.find()&&boolor==false&&booland==false&&naudrange!=null){
        naudstatement="FOR EACH ROW\nIF NOT (";}

    //Atrenkami IF, AND ir OR, kur prasideda
    java.util.regex.Pattern p42 = java.util.regex.Pattern.compile("\\(=>"); //ko ie□koma
    Matcher m42 = p42.matcher(dalis);
    if (m42.find()) {ifon=true;}

    java.util.regex.Pattern p43 = java.util.regex.Pattern.compile("\\(and"); //ko ie□koma
    Matcher m43 = p43.matcher(dalis);
    if (m43.find()) {booland = true; nrand=0;} // prasideda And grandin?

    Regex range431 = new Regex("\\(or ");
    range431.search(dalis);
    if (range431.didMatch()) { boolor = true; nror=0;}

   //Jei AND ir OR salia
   java.util.regex.Pattern p432 = java.util.regex.Pattern.compile("\\(and \\(or "); //ko ie□koma
   Matcher m432 = p432.matcher(dalis); // kur bus keiciama/ieskoma
   if (m432.find()) {
      boolor = true;
      booland = false;
      nror=0;
   } // prasideda And grandin?

    /// iesko )))
   java.util.regex.Pattern p489 = java.util.regex.Pattern.compile("([A-z0-9.']+)\\)\\)\\)");
   Matcher m489 = p489.matcher(dalis);
```

```
   if (m489.find()&&ifon==true&&booland==false&&boolor==false&&existson==false)  {naudstatement = naudstatement
+ ") AND (";  galpab=true; ifon=false;}

   /// S?lygos sakiniu pradzia
   //su EXISTS daliai //PVZ. (exists ?Contract_Product2
    Regex range441 = new Regex("\\(exists \\?([A-Z_a-z0-9.]+)");
    range441.search(dalis);
       if (range441.didMatch()) {
        if (naudstatement==null) {naudstatement="\nIF (";}
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
        if (ifon) {naudstatement = naudstatement + " AND EXISTS " + range441.stringMatched(1) + ")";
    }
        else {naudstatement = naudstatement + "EXISTS " + range441.stringMatched(1)+ ")";} //Jei nera prasidejes
        ifon=true;
        existson=true;
     }

   Regex range440 = new Regex("\\(instance-of \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)");
   range440.search(dalis);
   if (range440.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
       naudstatement = naudstatement + range440.stringMatched(2) + "." + range440.stringMatched(1) + "=" +
range440.stringMatched(3) + "." + range440.stringMatched(1);
   }
   else {
      Regex range440b = new Regex("\\(instance-of \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)");
   range440b.search(dalis);
   if (range440b.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
       naudstatement = naudstatement + range440b.stringMatched(2) + "." + range440b.stringMatched(1) + "=" +
range440b.stringMatched(3) + "." + range440b.stringMatched(1);
   }}

    /// PVZ. (contract ?Contract_Product2 ?Contract1)))
    Regex range44 = new Regex("\\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)");
    range44.search(dalis);
     if (range44.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
      naudstatement = naudstatement + range44.stringMatched(2) + "." + range44.stringMatched(1) + "=" +
range44.stringMatched(3) + "." + range44.stringMatched(1);
     }
     else {//EZPAL statement sintaksei variantas ('contract' ?Contract_Product2 ?Contract1))
         Regex range44b = new Regex("\\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)");
         range44b.search(dalis);
       if (range44b.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
             naudstatement = naudstatement + range44b.stringMatched(2) + "." + range44b.stringMatched(1) + "=" +
range44b.stringMatched(3) + "." + range44b.stringMatched(1);
      }
     }

    Regex range442 = new Regex("\\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+) \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)");
    range442.search(dalis);
     if (range442.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
          naudstatement = naudstatement + range442.stringMatched(2) + "." + range442.stringMatched(1) + "=" +
range442.stringMatched(4) + "." + range442.stringMatched(3);
     }

   // Galima paversti ?vairiafunkciniu IF-u su masyvo pagalba
    Input ="own-slot-not-null";
    Regex range443 = new Regex("\\(" + Input + " ([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)");
```

```
     range443.search(dalis);
     if (range443.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
         naudstatement = naudstatement + range443.stringMatched(2) + "." + range443.stringMatched(1) + " is not null";
    }
    else { //EZPAL sintaksei
    Regex range443b = new Regex("\\(" + Input + " '([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)");
     range443b.search(dalis);
      if (range443b.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
         naudstatement = naudstatement + range443b.stringMatched(2) + "." + range443b.stringMatched(1) + " is not null";
        }
    }

    n=0;
    while (n<3) {
    Input=Mas[n];
    n++;
    // PVZ. (> ('quantity' ?Contract_Product) 19)
    Regex range461 =  new Regex("\\(" + Input + " \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)\\)");
    range461.search(dalis);
    if (range461.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
           naudstatement = naudstatement + range461.stringMatched(2) + "." + range461.stringMatched(1) + Input +
range461.stringMatched(3);
         }
    else { //EZPAL sintaksei
    Regex range461b =  new Regex("\\(" + Input + " \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)\\)");
    range461b.search(dalis);
    if (range461b.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
         naudstatement = naudstatement + range461b.stringMatched(2) + "." + range461b.stringMatched(1) + Input +
range461b.stringMatched(3);
     }}}    //baigiasi masyvo while

    //PVZ. (not (= ?Contract1 ?Contract2))
    Regex range463 = new Regex("\\(not \\(\\= \\?([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)\\)");
    range463.search(dalis);
    if (range463.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
        naudstatement = naudstatement + range463.stringMatched(2) + "<>" + range463.stringMatched(1);
       }

   // pvz. EZPAL - (= (get-class ('delivering_address' ?Order)) (get-class ('delivering_place_location' ?Order))))\n"))
    n=0;
  while (n<3) {
    Input=Mas[n];
    n++;
   Regex range464 = new Regex("\\("+Input+" \\(get-class \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)\\) \\(get-class
\\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)\\)\\)");
   range464.search(dalis);
   if (range464.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
         naudstatement = naudstatement + range464.stringMatched(2) + "." + range464.stringMatched(1) + Input +
range464.stringMatched(4) + "." + range464.stringMatched(3);
        }
  } //baigiasi while

  //PVZ. (> (number-of-slot-values keywords ?article) 2
    n=0;
  while (n<3) {
    Input=Mas[n];
```

```
   n++;
   Regex range452 = new Regex("\\(" + Input + " \\(number-of-slot-values ([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([0-
9.]+)");
   range452.search(dalis);
    if (range452.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
      naudstatement = naudstatement + "COUNT ("  +  range452.stringMatched(2) + "." + range452.stringMatched(1) +
")" + Input + range452.stringMatched(3);
    }
    else {
     Regex range452b = new Regex("\\(" + Input + " \\(number-of-slot-values '([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)
([0-9.]+)");
     range452b.search(dalis);
     if (range452b.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
        naudstatement = naudstatement + "COUNT ("  +  range452b.stringMatched(2) + "." + range452b.stringMatched(1)
+ ")" + Input + range452b.stringMatched(3);
     }
   }
  } //Baigiasi masyvo ciklas

  ////////

   n=0;
   while (n<3) {
   Input=Mas[n];
   n++;
  Regex range21b = new Regex("\\(" + Input + " \\( '([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) \\('([A-Z_a-z0-9.]+)'
\\?([A-Z_a-z0-9.]+)\\)\\)");
   range21b.search(dalis);
   if (range21b.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
        naudstatement = naudstatement + range21b.stringMatched(2) + "." + range21b.stringMatched(1) + Input +
range21b.stringMatched(4) + "." + range21b.stringMatched(3);
     }
   } //masyvo tikrinimo pabaiga

  ////////

   //PVZ. //(not (= (contract_code ?Contract1) (contract_code ?Contract2)) EZPAL
   Regex range454 = new Regex("\\(not \\(\\= \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) \\(([A-Z_a-z0-9.]+) \\?([A-
Z_a-z0-9.]+)\\)\\)\\)");
   range454.search(dalis);
   if (range454.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
       naudstatement = naudstatement + range454.stringMatched(2) + "." + range454.stringMatched(1) + "<>" +
range454.stringMatched(4) + "." + range454.stringMatched(3);
   }

  //PVZ. //(not (= ('contract_code' ?Contract1) ('contract_code' ?Contract2)) EZPAL,
    Regex range454b = new Regex("\\(not \\(\\= \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) \\('([A-Z_a-z0-9.]+)'
\\?([A-Z_a-z0-9.]+)\\)\\)\\)");
    range454b.search(dalis);
    if (range454b.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
       naudstatement = naudstatement + range454b.stringMatched(2) + "." + range454b.stringMatched(1) + "<>" +
range454b.stringMatched(4) + "." + range454b.stringMatched(3);
    }
   else { //Pastarosios s?lygos poaibis
      n=0;
  while (n<3) {
   Input=Mas[n];
   n++;                //PVZ.  (< ('contract_data' ?Contract) ('expiry_date' ?Contract))
```

```
   Regex range444 = new Regex("\\(" + Input + " \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) \\(([A-Z_a-z0-9.]+) \\?([A-
Z_a-z0-9.]+)\\)\\)");
   range444.search(dalis);
   if (range444.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
       naudstatement = naudstatement + range444.stringMatched(2) + "." + range444.stringMatched(1) + Input +
range444.stringMatched(4) + "." + range444.stringMatched(3);
     }
   else { //EZAPL sintakesei

      //(= ('contract' ?Order1) ('contract' ?Order1))) kai vienodi turi buti neisvedamas

    Regex range444b = new Regex("\\(" + Input + " \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) \\('([A-Z_a-z0-9.]+)'
\\?([A-Z_a-z0-9.]+)\\)\\)");
    range444b.search(dalis);
    if (range444b.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " AND ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " AND ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " OR ";}}
        if (range444b.stringMatched(4).equals(range444b.stringMatched(2)) &&
range444b.stringMatched(1).equals(range444b.stringMatched(3))) {ifon=false; praleist=true;}
        else {naudstatement = naudstatement + range444b.stringMatched(2) + "." + range444b.stringMatched(1) + Input
+ range444b.stringMatched(4) + "." + range444b.stringMatched(3);}
     }
    }
   } //while baigiasi
   } //baigiasi else

  java.util.regex.Pattern p800 = java.util.regex.Pattern.compile("\\(\\= 0 0\\)"); //ko ieskoma
  Matcher m800 = p800.matcher(dalis); // (= 0 0)
  if (m800.find()&&booland&&nrand==1) {booland=false; nrand=0;}

 ///////// Baigiasi s?lygos
/// Uzbaigimo pozymiu ieskojimas

    /// iesko )))
   java.util.regex.Pattern p4493 = java.util.regex.Pattern.compile("([A-Z_a-z0-9.']+)\\)\\)\\)"); //ko ieskoma
   Matcher m4493 = p4493.matcher(dalis); // kur bus keiciama/ieskoma
   if (m4493.find()&&ifon==true&&galpab==false)  {
     galpab=true;
     ifon=false;
     if (nror>1) {naudstatement = naudstatement + ") AND (";}  // del 1 - 2-o
     if (booland==true&&boolor==false&&nrand>1) {booland=false;  nrand=0;}
     if (booland==false&&boolor==true&&nror>1) {boolor=false; nror=0;}
     if (booland==true&&boolor==true&&nrand>1&&nror>1) {booland=false;  nrand=0; boolor=false; nror=0;}
   }

   //Gale pridedamas ") AND (" arba ")" uzbaigiama.
   java.util.regex.Pattern p449 = java.util.regex.Pattern.compile("([A-Z_a-z0-9.']+)\\)\\)"); //ko ieskoma
    Matcher m449 = p449.matcher(dalis); // kur bus ieskoma/kei?iama
    if (m449.find()&&galpab==false) { //Baigia AND arba OR dali

    if (booland==true&&boolor==false&&nrand>1&&trecialent==null) {booland=false;  nrand=0;}
    if (booland==true&&boolor==false&&nrand>2&&trecialent!=null)  {booland=false;  nrand=0;} //su trim lentel?m

    if (booland==false&&boolor==true&&nror>1&&trecialent==null) {boolor=false; nror=0;}
    if (booland==false&&boolor==true&&nror>2&&trecialent!=null) {boolor=false; nror=0;} //su trim lentel?m

     if (booland==true&&boolor==true&&nrand>1&&nror>1&&trecialent==null) {booland=false;  nrand=0; boolor=false;
nror=0;}
     if (booland==false&&boolor==true&&nror>2&&trecialent!=null) {boolor=false; nror=0;} //su trim lentelem
     if (praleist==false&&ifon&&boolor==false&&booland==false) {naudstatement=naudstatement +  ") AND (";}
      else {if (praleist==false&&boolor==false&&booland==false&&buvoskl==false) {naudstatement=naudstatement +
")"; buvoskl=true;}}
   if (boolor==false&&booland==false) {ifon=false;}
    }

  java.util.regex.Pattern p4494 = java.util.regex.Pattern.compile("\\)\\)\\)\\)"); //ko ieskoma
   Matcher m4494 = p4494.matcher(dalis);  // kur bus ieskoma/keiciama
   if (m4494.find()&&buvoskl==false)  {
     galpab=true;
     ifon=false;
```

```
   }
    /// Visiska pabaiga
    java.util.regex.Pattern p4491 = java.util.regex.Pattern.compile("\\\"\\)\\)"); //ko ieskoma
    Matcher m4491 = p4491.matcher(dalis);
   if (m4491.find()&&galpab&&buvoskl==false) { naudstatement=naudstatement + ")"; buvoskl=true;}
   if (m4491.find()) {galpab=false;}

////////// Baigesi skliausteliu ir AND pridejimo zona
    praleist=false;
    buvoskl=false;
    Input= null;

}   //Vidinio ciklo pabaiga


naudstatement=naudstatement + "\n" + "BEGIN\n                COMMIT TRANSACTION\n         PRINT 'Viskas
gerai.'\nEND\nELSE\n             RAISERROR ('Ivyko klaida.')\n  ROLLBACK TRANSACTION\nEND";
  trigbaigtas=true;
  s2.close();
  }
  //ivairus isvedimai i faila ir patikrinimui i ekrana ir kintamaji valymai
     if (trigbaigtas==true) { //trigerio irasymas i faila, jei buvo rasta Statement eilute
       if (nauddesc!=null) setContents(f2, getContents(f2) + nauddesc); //ideda i faila
      setContents(f2, getContents(f2) + naudname);
      if (naudrange!=null) setContents(f2, getContents(f2) + naudrange);
      setContents(f2, getContents(f2) + naudstatement);
  //Pasitikrinimai isvedimas
     if ( nauddesc!=null) System.out.println(nauddesc);
     System.out.println(naudname);
     if (naudrange!=null) System.out.println(naudrange);
     System.out.println(naudstatement);
     //Isvalomi kintamieji
      trigbaigtas=false;
      nauddesc = null;
      naudname = null;
      naudrange = null;
      naudstatement = null;
      pirmalent = null;
      antralent = null;
      trecialent = null;
      raktas = null;
      booland=false;
      boolor=false;
      beginstate = null;
      ifon =false;
      nrand=0;
      nror=0;
      galpab=false;
      praleist=false;
     }
  } //while pagrindinio ciklo pabaiga

   s.close();
boolean success14 = (new File(tempfpav)).delete();
    if (!success14) {
       System.out.println("Failo " + tempfpav + " istrinti nepavyko");
    }}}.out.println("Failo " + tempfpav + " istrinti nepavyko");
    }}}
```

6.3. Priedas. Axiom2OCL programinis kodas

```
package Axiom2BR;
/* Sukure Tautvydas Soblinskas*/
import java.io.*;
import java.util.*;
import javax.swing.*;
import edu.stanford.smi.protege.*;
import edu.stanford.smi.protege.model.*;
import edu.stanford.smi.protege.plugin.*;
import edu.stanford.smi.protege.util.*;
import java.util.regex.*;
import com.stevesoft.pat.*;
import java.util.regex.Matcher;

public class Axiom2OCL implements ExportPlugin {
    private static final String EXTENSION = ".pins";

    public static void main(String[] args) {
        Application.main(args);
    }

  public String getName() {
        return "Axiom2OCL";}
//nurodomas pasirenkamos f-jos vardas


    public void dispose() {
    }

   public void handleExportRequest(Project project)  {
        File file = promptForFooFile(project);
       // Gaunamas failas
        if (file != null)
        {
    //      saveToFile(project, file);
           }
     }

    //parenkamas failas
    public File promptForFooFile(Project project) {
        String name = project.getName();
        String proposedName = new File(name + EXTENSION).getPath();
        JFileChooser chooser = ComponentFactory.createFileChooser(proposedName, EXTENSION);
        File file = null;
        if (chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
            file = chooser.getSelectedFile();
        }

        try {transform(file);}
        catch(java.io.IOException exp){ exp.printStackTrace();}
        return file;
     }

   private void saveToFile(Project project, File file) {
        PrintWriter writer = FileUtilities.createPrintWriter(file, false);
       //saveProject(project, writer);
       writer.close();
    }

    // Isveda kiekviena klase atskiroje eiluteje
    private void saveProject(Project project, PrintWriter writer) {
    }

/////////////// Funkcijos ////////////////////////////

  /* Paima visa failo turini ir grazinu stringu (eilute).
  * @param aFile yra failas, kuris jau egzistuoja ir gali buti perskaitytas.
  */
static public String getContents(File aFile) {
```

```
    StringBuffer contents = new StringBuffer();

    BufferedReader input = null;
    try {
      input = new BufferedReader( new FileReader(aFile) );
      String line = null; //ne deklaruotas cikle

      /* Grziina null tik srauto pabaigoje.
      * Grazina tuscia eilute, jei dvi eilutes pasitaiko iseiles.*/
      while (( line = input.readLine()) != null){
        contents.append(line);
        contents.append(System.getProperty("line.separator"));
      }
    }
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
    catch (IOException ex){
      ex.printStackTrace();
    }
        finally {
      try {
        if (input!= null) {
          input.close();
        }
      }
      catch (IOException ex) {
        ex.printStackTrace();
      }
    }
    return contents.toString();
  }
////////////////////////////////////////////////////////////////////////
  /* Keicia failo turini nauju tekstu.
  * @param aFile - egzistuojantis faila i kuri galima irasyti.
  * @throws IllegalArgumentException, jei param neivykdomas.
  * @throws FileNotFoundException, jei failas neegzistuoja.
  * @throws IOException, ivyksta problema irasymo metu.*/

  public void setContents(File aFile, String aContents)
                              throws FileNotFoundException, IOException {
    if (aFile == null) {
      throw new IllegalArgumentException("Failas neturi b?ti null.");
    }
    if (!aFile.exists()) {
      throw new FileNotFoundException ("Neegzistuoja failas: " + aFile);
    }
    if (!aFile.isFile()) {
      throw new IllegalArgumentException("Klaida susijusi su pateikiamu kategorijos: " + aFile);
    }
    if (!aFile.canWrite()) {
      throw new IllegalArgumentException("Negalima irasyti i faila: " + aFile);
    }

    Writer output = null;
    try {
      //naudojamas buferis
      //FileWriter pagal pradine nustatyma visada teigia, kad uzkodavimas tinkamas
      output = new BufferedWriter( new FileWriter(aFile) );
      output.write( aContents );
    }
    finally {
      //uzdaro output.
      if (output != null) output.close();
    }}

  ////////////////////////Pabaiga
  ////////////////// Didziausia transformavimo f-ja

  public void transform (File input) throws IOException {
    String tempfpav = "C:/data/temp.txt";
    String outputfpav = "C:/data/output.txt";

    File temp_failas = new File(tempfpav); // priskkiriam f kintamajam nurodyta faila
```

```
  File saltinio_failas = input;

    // Kopijavimo pradzia
   InputStream in = new FileInputStream(input);
   OutputStream out = new FileOutputStream(temp_failas);

  // Failo turinio perkopijavimas i laikina faila
   byte[] buf = new byte[1024];
   int len;
   while ((len = in.read(buf)) > 0) out.write(buf, 0, len);
     in.close();
     out.close();
  /// Kopijavimo pabaiga

   File rez_failas = new File(outputfpav); //sukuriamas duomenu isvedimo failas
      try {
         // sukuria rez_failas faila, jei tokio nera
          boolean success2 = rez_failas.createNewFile();
       }
      catch (IOException e) { }


  // f failo turinys paimamas su getContents(f), kurio formatas vientisa eilute (String).
Scanner sc = new Scanner(System.in);

  Scanner s = new Scanner(getContents(temp_failas)).useDelimiter("\n"); // Eilute nuskaitoma dalimis po eilute.


     //**//Ciklo kintamieji//**//
    int i=0; //visus pereina per cikla parodydamas dalis
    //PAL daliu paemimo kintamieji
    String nauddesc = null;  //DESCRIPTION dalies paemimui
    String naudname = null; //NAME dalies paemimui
    String naudrange = null; //RANGE dalies paemimui
    String naudstatement = null; //STATEMENT dalies paemimui
    String ribojimas = null;
    String visiribojimai = "PAL-OCL transformacijos rezultatai:";
    String tempname = "";


    boolean ribbaigtas = false;

    //Pasitikrinimui kintamieji
    String klase1 = "";
    String klase2 = null;
    String klase3 = null;
    String raktas = null; //saryšio raktui
    String lygklase1 = null;
    String lygklase2 = null;
    String lygklase3 = null;
    String unik_konklase= null;
    boolean inv_yra = false;
    boolean not_on = false;
    String Input = null;

    boolean boolor=false; //nurodo ar tesiasi OR grandine
    boolean booland=false; //nurodo ar tesiasi AND grandine, žymima jos pradžia ir pabaiga
    boolean ifon=false; // Žymi pirmine dali nuo IF iki )) (ar daugiau skliaustu)
    int nrand=0; // skaiciuoja AND dalis (neskaiciuojant dideliu tarpusavio salygos sakiniu daliu)
    int nror=0; //skaiciuoja OR dalis
    boolean galpab=false; //žiuri ar galima salygos sakiniu pabaiga, žymeklis
    boolean existson=false; //nurodo ar prieš tai buvo EXIST tipo salyga
    boolean praleist=false; //naudojamas praleidimui pažymeti, po then
    boolean buvoskl=false;
    boolean endif=false;
    boolean abu_ANDOR=false;


    //Salygu dalies kintamieji

    //Vidinio masyvo kintamieji, naudojami perrinkti per ivairius variantus
    String[] Mas = {">","<" ,"="};
    int n=0;
  // ********** ///Pagrindinis ciklas/// **********
```

```
  // Ciklo metu atrenkamos reikalingos eilutes
  while(s.hasNext()) // žiuri ar egzistuoja kita eilute
{

    String naudojamas = s.next(); //eilutes stringas permetamas kintamajam

    // su 3APAL-NAME eiluciu išrinkimas
    //nurodomas pilnas funkcijos ir bibliotekos pavadinimas, kad atrinktu kuria naudoti
    java.util.regex.Pattern p1 = java.util.regex.Pattern.compile("3APAL-NAME"); //ko ieškoma
    Matcher m1 = p1.matcher(naudojamas); // kur bus keiciama/ieškoma

    if(m1.find()) // žiuri ar true (ar yra ieškomas gabalas) | suradimas
    {

    // Vieta transformacijai
    // Ištraukia PAL constraint PAVADINIMA ir ideda ji i OCL ribojima
       Regex name = new Regex("(?is)%3APAL-NAME (?=\"(.+)\")");
       name.search(naudojamas);
       naudname = " " + name.stringMatched(1); //i "naudojamas" kintamaji idedama išrinktas NAME
    }

    //// su 3APAL-DESCRIPTION eiluciu išrinkimas
   java.util.regex.Pattern p2 = java.util.regex.Pattern.compile("3APAL-DESCRIPTION"); //ko ieškoma
   Matcher m2 = p2.matcher(naudojamas); // kur bus keiciama/ieškoma

    if(m2.find())
    {
       /// Vieta transformacijai
       Regex r = new Regex("(?im)%3APAL-DESCRIPTION (?=\"(.+)\")");
       r.search(naudojamas);
        naudojamas = "' " + r.stringMatched(1) + "'";
        nauddesc = naudojamas;

       //Pašalina /n ir patalpina tarpa
       java.util.regex.Pattern p = java.util.regex.Pattern.compile("\\\\n"); //ko ieškoma
       Matcher m = p.matcher(nauddesc); // (get a matcher object) kur bus keiciama/ieškoma
       nauddesc = m.replaceAll(" ");
    }

   /// Tik tada kai nurodomas range
   ///////////////// su 3APAL-RANGE  eiluciu išrinkimas
   java.util.regex.Pattern p3 = java.util.regex.Pattern.compile("3APAL-RANGE "); //ko ieškoma
    Matcher m3 = p3.matcher(naudojamas); // kur bus keiciama/ieškoma

    if(m3.find())
    {
     /// Vieta transformacijai
      Regex range = new Regex("([[A-Z_a-z0-9.]+) :FRAME ([[A-Z_a-z0-9.]+)\\)"); //randa pirma lentele
      Regex range2 = new Regex("([[A-Z_a-z0-9.]+) :FRAME ([[A-Z_a-z0-9.]+)[]"); //randa antra lenetele
      Regex range21 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)"); //randa antra lentele
2 var.
      Regex range3 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+) ([A-Z_a-z0-9.]+)");
//randa saryšio atributa

      //(%3APAL-RANGE "(defrange ?Article1 :FRAME Article)\n(defrange ?Article2 :FRAME Article) \n")
      Regex range31 = new Regex("\\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)\\)\\\\n\\(defrange \\?([A-Z_a-z0-
9.]+)");
      //Regex range31 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)\\)\\\\n(defrange
\\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)\\)");

      range.search(naudojamas);
      range2.search(naudojamas);
      range21.search(naudojamas);
      range3.search(naudojamas);
      range31.search(naudojamas);

      if (range3.didMatch())
      {
         raktas = range3.stringMatched(3); //raktas
      }

      if (range31.didMatch())
      {
        System.out.println("TESTASs==" + range31.stringMatched(2));
```

```
        unik_konklase=range31.stringMatched(2);
      }



       klase1 = range.stringMatched(1); //priskirimas 1 lentelei
       klase2 = range2.stringMatched(1); //priskirimas 2 lentelei

       if (klase2==null&&range21.didMatch())
       {
          klase2 = range21.stringMatched(1);
          naudrange =range21.stringMatched(2);
            //  System.out.println(range21.stringMatched(2));
       }
       if (klase1!=null&&klase2!=null)
       {

     //Jei trys FRAME dalys
        Regex range213 = new Regex("\\\\n\\(defrange \\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)\\)\\\\n\\(defrange
\\?([A-Z_a-z0-9.]+) :FRAME ([A-Z_a-z0-9.]+)"); //
        range213.search(naudojamas);
     // PVZ., (%3APAL-RANGE "(defrange ?Order1 :FRAME Order)\n(defrange ?Order2 :FRAME Order)\n(defrange ?Order3
:FRAME Order)\n")
         if (range213.didMatch())
         {
            klase3=range213.stringMatched(3);
            naudrange=range213.stringMatched(2); //priskirta pirmoji klase

         }

       }

 } //baigiama klasiu paieska RANGE dalyje


    ////////////////// su 3APAL-STATEMENT eiluciu išrinkimas
    java.util.regex.Pattern p4 = java.util.regex.Pattern.compile("3APAL-STATEMENT"); //ko ieškoma
    Matcher m4 = p4.matcher(naudojamas); // kur bus keiciama/ieškoma

  if(m4.find())
  {  /// Vieta transformacijai

    // Vidinis skaneris atrinkti salygoms ////
    Scanner s2 = new Scanner(naudojamas).useDelimiter("\\\\n"); // String nuskaitoma dalimis suskaidant ties \n.
     //Vidinis ciklas atrinkti dalims

  // System.out.println("\n------------- constraint--PAL to OCL ------------------");
    while(s2.hasNext()) // žiuri ar egzistuoja kita eilute
   {

     String dalis = s2.next(); //eilutes stringas priskiriamas kintamajam
    // System.out.println(dalis); // paziureti kokios dalys paimamos

    java.util.regex.Pattern p41 = java.util.regex.Pattern.compile("forall"); //ko ieškoma
    Matcher m41 = p41.matcher(dalis); // kur bus keiciama/ieškoma
    if (m41.find())
    {
      inv_yra =true; //jei rastas forall pridedamas inv atitikuomui
      if (klase1==null && klase2==null && klase3==null) // kai nera RANGE dalies ir neisgaunama ne viena klase
      { //suveikia jei nera apibreztos klases
         Regex range = new Regex("forall \\?([A-Z_a-z0-9.]+)");
         range.search(dalis);
         if (range.stringMatched(1)!=null)
         {
            naudrange="" + range.stringMatched(1); // Isgaunama taikymo sritis (jei jos reikia)


         }
      }
    }
    /////// SALYGU SAKINIAM FORMUOTI


    ////////////////////
```

```
    //// SALYGOS //////
    ////////////////////

  /**************/

//      (=> (instance-of (article_author ?article) Person) \n
     Regex range46 = new Regex("\\(=> \\(instance-of \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)\\)");
     range46.search(dalis);
     if (range46.didMatch()) {
     //    System.out.println("\nRADOM\n");
     if (existson==true) {naudstatement = naudstatement + "("; existson=false;}
     if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
     if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
     raktas=range46.stringMatched(1); //jungiamasis atributas
     naudstatement="\n if (" + "self." + range46.stringMatched(1) + "="+
range46.stringMatched(3)+"."+range46.stringMatched(1) + ")";
     ifon=true;
     }
     else {
    //(=> //// prideda If ocl sakinyje
     //Atrenkami IF, kur prasideda
     Regex range_if = new Regex("\\(=> ");
    range_if.search(dalis);
    if (range_if.didMatch()) {
    naudstatement = "\n if ";
    //System.out.println("\nRADOM\n");
      ifon=true;
    }
     }
    //Jei AND ir OR šalia
    java.util.regex.Pattern p43 = java.util.regex.Pattern.compile("\\(and \\(or "); //ko ieškoma
    Matcher m43 = p43.matcher(dalis);
    if (m43.find()) {
        boolor = true;
        booland = false;
        nror=0;
        buvoskl=true;
        abu_ANDOR=true; // atrinkimui OR ir AND
    } // prasideda And grandine
    else {

     java.util.regex.Pattern p431 = java.util.regex.Pattern.compile("\\(and"); //ko ieškoma
     Matcher m431 = p431.matcher(dalis);
     if (m431.find()) {booland = true; nrand=0; naudstatement = naudstatement + "("; buvoskl=true;} // prasideda And
grandin?
     }

     Regex range432 = new Regex("\\(or ");
     range432.search(dalis);
     if (range432.didMatch()) { boolor = true; nror=0; buvoskl=true;}


    //Iesko Exist on =then
    //(exists ?editor \n
    java.util.regex.Pattern p4exist = java.util.regex.Pattern.compile("\\(exists ?"); //ko ieškoma
    Matcher m4exist = p4exist.matcher(dalis); // kur bus kei?iama/ieškoma
    if (m4exist.find()) {
       existson=true;
       endif=true;
       naudstatement = naudstatement + " then";
    }

        /// ieško )))
   Regex range601 = new Regex("\\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+) \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-
9.]+)\\)\\)\\)\\)\\)");
    range601.search(dalis);
     if (range601.didMatch()&& abu_ANDOR==true&&boolor==true&&booland==false)
    {naudstatement = naudstatement + ") and (";  abu_ANDOR=false; boolor=false; booland=true; buvoskl=true;}

    if (range601.didMatch()&& abu_ANDOR==true&&boolor==false&&booland==true)
    {naudstatement = naudstatement + ") and (";  abu_ANDOR=false; boolor=true; booland=false; buvoskl=true;}
```

```
 //(responsible_for ?editor (article_author ?article))) \n
 //(sections ?editor (containing_section ?article))))))"))
 // editor.responsible_for= article.article_author
    Regex range600 = new Regex("\\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+) \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-
9.]+)\\)\\)");
    range600.search(dalis);
    if (range600.didMatch()) {
      if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
         if (booland&&buvoskl==false) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
       lygklase1=naudrange;
       lygklase2=range600.stringMatched(2);
       lygklase3=range600.stringMatched(4);
       if (lygklase1.equals(lygklase2)) {naudstatement = naudstatement + "self." + range600.stringMatched(1) + "=" +
range600.stringMatched(4) + "."+ range600.stringMatched(3);}
       if (lygklase1.equals(lygklase3)) {naudstatement = naudstatement + range600.stringMatched(2)+ "." +
range600.stringMatched(1) + "=self."+ range600.stringMatched(3);}
  //naudstatement = naudstatement + range600.stringMatched(2)+ "." + range600.stringMatched(1) + "=" +
range600.stringMatched(4) + "."+ range600.stringMatched(3);
  lygklase1=null;
  lygklase2=null;
  lygklase3=null;

    }

    // (responsible_for ?editor ?employee) traktuojama kaip OCL (self.responsible_for->notEmpty())
    Regex range41a = new Regex("\\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)");
    range41a.search(dalis);
    if (range41a.didMatch()) {
      if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
      raktas = range41a.stringMatched(1); //jungiamasis atributas
      lygklase1=naudrange;
      lygklase2=range41a.stringMatched(2);
      lygklase3=range41a.stringMatched(3);
      if (lygklase1.equals(lygklase2)) {naudstatement = naudstatement + "self." + raktas +"->notEmpty()";}
      if (lygklase1.equals(lygklase3)) {naudstatement = naudstatement + "self." + raktas +"->notEmpty()";}
       //  naudstatement = naudstatement + "self." +
range41a.stringMatched(1)+"="+"self."+range41a.stringMatched(3)+"."+range41a.stringMatched(1);
    }


     // Prasideda salygu sakinai
    //PVZ. (not (= (DarbuotojuKiekis ?Bendrove) 100)))"))
    Regex range463a = new Regex("\\(not \\(= \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)");
    range463a.search(dalis);
    if (range463a.didMatch()) {
        not_on=true;
       if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
        if (naudstatement==null) {naudstatement = "\n";}
       lygklase1=naudrange;
       lygklase2=range463a.stringMatched(2);
       if (lygklase1.equals(lygklase2)) {naudstatement = naudstatement +  "self." + range463a.stringMatched(1) + "<>" +
range463a.stringMatched(2) + "." + range463a.stringMatched(3);}
       else {naudstatement = naudstatement +  range463a.stringMatched(2) + "." + range463a.stringMatched(1) + "<>" +
range463a.stringMatched(2) + "." + range463a.stringMatched(3);}
       }
    else {
     if (not_on==false) {

        Regex range466 = new Regex("\\(not \\(= \\('([A-Z_a-z0-9.]+)\\' \\?([A-Z_a-z0-9.]+)\\) \\('([A-Z_a-z0-9.]+)\\'
\\?([A-Z_a-z0-9.]+)\\)");
    range466.search(dalis);
    if (range466.didMatch()) {
        not_on=true;
       if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
       if (naudstatement==null) {naudstatement = "\n";}

          lygklase1=naudrange;
```

```
          lygklase2=range466.stringMatched(2);
          lygklase3=range466.stringMatched(4);
        if (lygklase1.equals(lygklase2)&&lygklase1.equals(lygklase3)) {naudstatement = naudstatement +  "self." +
range466.stringMatched(1) + "<>" + "self" + "." + range466.stringMatched(3);}
        if (lygklase1.equals(lygklase2)&&lygklase1.equals(lygklase3)==false) {naudstatement = naudstatement +  "self." +
range466.stringMatched(1) + "<>" + range466.stringMatched(4) + "." + range466.stringMatched(3);}
        if
(lygklase1.equals(lygklase2)==false&&lygklase1.equals(lygklase3)&&!range466.stringMatched(1).equals(range466.stringM
atched(3))) {naudstatement = naudstatement +  range466.stringMatched(2) + "." + range466.stringMatched(1) + "<>" +
"self"  + "." + range466.stringMatched(3);}

        //(not (= ('headline' ?Article1) ('headline' ?Article2)))))))
        if
(lygklase1.equals(lygklase2)==false&&lygklase1.equals(lygklase3)&&range466.stringMatched(1).equals(range466.stringMa
tched(3)))
        {naudstatement = "\n" + unik_konklase + "::allInstances()->isUnique(" + range466.stringMatched(3);
        //not_on=false;
        }
          /*
                context Article inv unique-headline:
Article::allInstances()->isUnique(headline)

       ------------- constraint--PAL to OCL ------------------
        (%3APAL-STATEMENT " (forall ?Article1
        (forall ?Article2
              (=> (and (own-slot-not-null 'headline' ?Article1)
                       (own-slot-not-null 'headline' ?Article2))
                 (=> (not (= ?Article1 ?Article2))
        (not (= ('headline' ?Article1) ('headline' ?Article2)))))))
        */

          lygklase1=null;
          lygklase2=null;
          lygklase3=null;
        }}


    // Jei nebuvo rasta pries tai dalis
     if (not_on==false) {
    //(not (< (DarbuotojuKiekis ?Bendrove) 26)))"))
    // not self.DarbuotojuKiekis<26
     Regex range463c = new Regex("\\(not");
    range463c.search(dalis);
    if (range463c.didMatch()) {
         if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
         if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
         if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
         if (naudstatement==null) {naudstatement = "not (";}
        else {naudstatement = naudstatement + " (";}
      //   else {naudstatement = naudstatement + "not (";}
     }
    }
           } //baigiasi else dalis

    if (not_on==false) { // kai nera not sakinio tada imam tokia salyga
    n=0;
    Input=null;
    while (n<3) {
    Input=Mas[n];
    n++;
    // PVZ. (> ('quantity' ?Contract_Product) 19)
    // self.quantity>19 //////(kai kontekstine klase 1)
    Regex range501 =  new Regex("\\(" + Input + " \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-9.]+)\\)");
    range501.search(dalis);
   // System.out.println(dalis);
    if (range501.didMatch()) {
     if (naudstatement==null) naudstatement="";
     if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
     naudstatement = "\n" + naudstatement + "self" + "." + range501.stringMatched(1) + Input +
range501.stringMatched(3);
        }
    else { //EZPAL sintaksei
    Regex range501ezpal =  new Regex("\\(" + Input + " \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) ([A-Z_a-z0-
9.]+)\\)");
```

```
    range501ezpal.search(dalis);
    if (range501ezpal.didMatch()) {
    if (naudstatement==null) naudstatement="";
    if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
        naudstatement ="\n"+ naudstatement + "self" + "." + range501ezpal.stringMatched(1) + Input +
range501ezpal.stringMatched(3); //neima klases nes nereikia
     }}}    //Baigiasi masyvo while
    }
    /*****************/

    // (= isFinished yes)
  n=0;
  while (n<3) {
    Input=Mas[n];
    n++;
   Regex range470 = new Regex("\\("+Input+" ([A-Z_a-z0-9.]+) ([A-Z_a-z0-9.]+)\\)");
   range470.search(dalis);
   if (range470.didMatch()) {
   //    if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
   //  if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
   //    if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}

     lygklase1=naudrange;
     lygklase2=range470.stringMatched(1);
     if (!lygklase1.equals(lygklase2)&&lygklase1.equals(lygklase3)) {naudstatement = naudstatement + " then (self" + "." +
range470.stringMatched(1) + Input + range470.stringMatched(2);}
      }
   } //baigiasi while

    /*************/

   // pvz. EZPAL - (= (get-class ('delivering_address' ?Order)) (get-class ('delivering_place_location' ?Order))))\n"))
  n=0;
  while (n<3) {
    Input=Mas[n];
    n++;
   Regex range464 = new Regex("\\("+Input+" \\(get-class \\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)\\) \\(get-class
\\('([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)\\)\\)");
   range464.search(dalis);
   if (range464.didMatch()) {
      if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
     if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
       if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}

     lygklase1=naudrange;
     lygklase2=range464.stringMatched(2);
     lygklase3=range464.stringMatched(3);
     if (lygklase1.equals(lygklase2)&&lygklase1.equals(lygklase3)) {naudstatement = naudstatement + "self" + "." +
range464.stringMatched(1) + Input + "self" + "." + range464.stringMatched(3);}
     if (lygklase1.equals(lygklase2)&&lygklase1.equals(lygklase3)==false) {naudstatement = naudstatement + "self" + "."
+ range464.stringMatched(1) + Input + range464.stringMatched(4) + "." + range464.stringMatched(3);}
    if (lygklase1.equals(lygklase2)==false&&lygklase1.equals(lygklase3)) {naudstatement = naudstatement +
range464.stringMatched(2) + "." + range464.stringMatched(1) + Input + "self" + "." + range464.stringMatched(3);}
  }
   } //baigiasi while

     /**************/
  //PVZ. (> (number-of-slot-values keywords ?article) 2
  // self.keyword->size() > 2
    n=0;
   while (n<3) {
    Input=Mas[n];
    n++;
    Regex range452 = new Regex("\\(" + Input + " \\(number-of-slot-values ([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) ([0-
9.]+)");
    range452.search(dalis);
     if (range452.didMatch()) {
     if (existson==true) {naudstatement = naudstatement + "("; existson=false;}
     if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
      if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
     if (naudstatement==null) naudstatement="";
     lygklase1=naudrange;
     lygklase2=range452.stringMatched(2);
     if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
```

```
    if (lygklase1.equals(lygklase2)) {naudstatement = "\n"+ naudstatement + "self." + range452.stringMatched(1) + "-
>size()" + Input + range452.stringMatched(3);}
    else {naudstatement = "\n"+ naudstatement + range452.stringMatched(2) + "." + range452.stringMatched(1) + "-
>size()" + Input + range452.stringMatched(3);}
    }
    else {
     Regex range452bezpal = new Regex("\\(" + Input + " \\(number-of-slot-values '([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-
9.]+)\\) ([0-9.]+)");
      range452bezpal.search(dalis);
      if (range452bezpal.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
       if (naudstatement==null) naudstatement="";
    lygklase1=naudrange;
    lygklase2=range452bezpal.stringMatched(2);  // lyginimas ar klases sutampa tada galime naudoti self (jei range ir lyg.
atributo klases sutampa)
    if (lygklase1.equals(lygklase2)) {naudstatement = "\n"+ naudstatement + "self." + range452bezpal.stringMatched(1) +
"->size()" + Input + range452bezpal.stringMatched(3);}
    else {naudstatement = "\n"+ naudstatement +  range452bezpal.stringMatched(2) + "." +
range452bezpal.stringMatched(1) + "->size()" + Input + range452bezpal.stringMatched(3);}
     }
   }
  } //Baigiasi masyvo ciklas

    // Galima paversti ivairiafunkciniu IF-u su masyvo pagalba
    Input ="own-slot-not-null";
    Regex range443 = new Regex("\\(" + Input + " ([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)");
    range443.search(dalis);
    if (range443.didMatch()) {
        if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}

       lygklase1=range443.stringMatched(2);
       lygklase2=naudrange;
       if (lygklase1.equals(lygklase2)) {naudstatement = naudstatement + "self." + range443.stringMatched(1) + " -
>notEmpty()";}
       else {naudstatement = naudstatement + range443.stringMatched(2) + "." + range443.stringMatched(1) + " -
>notEmpty()"; }
   }
   else { //EZPAL sintaksei
        //(own-slot-not-null 'editor' ?Article))
       //own-slot-not-null 'headline' ?Article1
  Regex range443b = new Regex("\\(" + Input + " '([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\)");
    range443b.search(dalis);
     if (range443b.didMatch()) {



        if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
        if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
        if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}

      lygklase1=naudrange;
      lygklase2=range443b.stringMatched(2);

       if (lygklase1.equals(lygklase2)) {naudstatement = naudstatement + "self." + range443b.stringMatched(1) + " -
>notEmpty()";}
      else {naudstatement = naudstatement + range443b.stringMatched(2) + "." + range443b.stringMatched(1) + " -
>notEmpty()";}


    lygklase1=null;
    lygklase2=null;
      }
   }

     /**************/
    //////(> (salary ?editor) (salary ?employee)))))"))
 //self.salary>self.responsible_for.salary
   n=0;
   while (n<3) {
   Input=Mas[n];
```

```
   n++;
   Regex range211bezpal = new Regex("\\(" + Input + " \\(([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\) \\(([A-Z_a-z0-9.]+)
\\?([A-Z_a-z0-9.]+)\\)\\)");
   range211bezpal.search(dalis);
   if (range211bezpal.didMatch()) {

    if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
    if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
    if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
       if (naudstatement==null) naudstatement="";

      lygklase1=naudrange;
      lygklase2=range211bezpal.stringMatched(2);
      lygklase3=range211bezpal.stringMatched(4);
       if (lygklase1.equals(lygklase2)) {
          if (raktas==null)
          {naudstatement = "\n"+ naudstatement + "self." + range211bezpal.stringMatched(1) + Input +
range211bezpal.stringMatched(4) +  "." + range211bezpal.stringMatched(3);}
          else {{naudstatement = "\n"+ naudstatement + "self." + range211bezpal.stringMatched(1) + Input + "self." +
raktas +  "." + range211bezpal.stringMatched(3);}}
        }
         if (lygklase1.equals(lygklase3)) {
          if (raktas==null) {naudstatement = "\n"+ naudstatement +  range211bezpal.stringMatched(1) + Input + "self." +
range211bezpal.stringMatched(4) +  "." + range211bezpal.stringMatched(3);}
          else {naudstatement = "\n"+ naudstatement + range211bezpal.stringMatched(2) + raktas +
range211bezpal.stringMatched(1) + "." + Input + "self" +"." + range211bezpal.stringMatched(3);}
          }
         lygklase1=null;
         lygklase2=null;
         lygklase3=null;

   }}

  // pvz.   (< ( 'start_date' ?Employee) ('end_date' ?Employee)))\n"))
  //  self.start_date <self.end_date
    n=0;
   while (n<3) {
   Input=Mas[n];
   n++;
  Regex range212bezpal = new Regex("\\(" + Input + " \\( '([A-Z_a-z0-9.]+)' \\?([A-Z_a-z0-9.]+)\\) \\('([A-Z_a-z0-
9.]+)' \\?([A-Z_a-z0-9.]+)\\)\\)");
   range212bezpal.search(dalis);
   if (range212bezpal.didMatch()) {

    if (existson==true) {naudstatement = naudstatement + " ("; existson=false;}
    if (booland) {nrand++; if (nrand>1) {naudstatement = naudstatement + " and ";}}
    if (boolor) {nror++; if (nror>1) {naudstatement = naudstatement + " or ";}}
       if (naudstatement==null) naudstatement="";
      lygklase1=naudrange;
      lygklase2=range212bezpal.stringMatched(2);
      lygklase3=range212bezpal.stringMatched(4);
     if (lygklase1.equals(lygklase2)&&lygklase1.equals(lygklase3)) {naudstatement = "\n"+ naudstatement + "self." +
range212bezpal.stringMatched(1) + Input + "self." +  range212bezpal.stringMatched(3);}
     else {
         if (lygklase1.equals(lygklase2)) {naudstatement = "\n"+ naudstatement + "self." +
range212bezpal.stringMatched(1) + Input + range212bezpal.stringMatched(4) + "." +  range212bezpal.stringMatched(3);}
         if (lygklase1.equals(lygklase3)) {naudstatement = "\n"+ naudstatement + range212bezpal.stringMatched(2) + "." +
range212bezpal.stringMatched(1) + Input + "self." +  range212bezpal.stringMatched(3);}
         } //else pabaiga
  lygklase1=null;
  lygklase2=null;
  lygklase3=null;

   }//salygos suradimo pabaiga

   } //masyvo tikrinimo pabaiga

     ///////// Baigiasi salygos
 /// Užbaigimo požymiu ieškojimas
   //paprastam pal
    //salary ?employee)) \n

   java.util.regex.Pattern p4then = java.util.regex.Pattern.compile("([A-Z_a-z0-9.]+) \\?([A-Z_a-z0-9.]+)\\)\\) "); //ko
ieškoma
```

```
    Matcher mp4then = p4then.matcher(dalis); // kur bus keiciama/ieškoma
    if (mp4then.find()&&ifon&&praleist==false) {
       endif=true;
       booland=false;
       boolor=false;
       praleist=true;
       naudstatement = naudstatement + ")\n then (";
       //jei yra abu vienas baigiasi (OR)
       if (abu_ANDOR==true) {booland=true; abu_ANDOR=false;}
     }

   //EzPal
  //  (own-slot-not-null 'editor' ?Article))\n
      java.util.regex.Pattern p41then = java.util.regex.Pattern.compile("\\'([A-Z_a-z0-9.]+)\\' \\?([A-Z_a-z0-9.]+)\\)\\)");
//ko ieškoma
    Matcher mp41then = p41then.matcher(dalis); // kur bus keiciama/ieškoma
    if (mp41then.find()&&ifon&&praleist==false) {
       endif=true;
       booland=false;
       boolor=false;
       praleist=true;
       naudstatement = naudstatement + ")\n then (";
       //jei yra abu vienas baigiasi (OR)
       if (abu_ANDOR==true) {booland=true; abu_ANDOR=false;}

     }

     java.util.regex.Pattern p4494 = java.util.regex.Pattern.compile("\\)\\)\\)\\)"); //ko ieškoma
     Matcher m4494 = p4494.matcher(dalis);  // kur bus ieškoma/keiciama
     if (m4494.find()&&buvoskl==false)  {
      galpab=true;
       naudstatement = naudstatement + ")";
      ifon=false;
   }
     /// Visiška pabaiga


   java.util.regex.Pattern p4491 = java.util.regex.Pattern.compile("\\\"\\)\\)"); //ko ieškoma
   Matcher m4491 = p4491.matcher(dalis);  // kur bus ieškoma/kei?iama
   //uždedamas paskutinis skliaustas
   if (m4491.find()&&buvoskl==true&&ifon) {naudstatement= naudstatement + ")"; buvoskl=true; ifon=false;
galpab=true; booland=false; boolor=false;}
   if (m4491.find()) {galpab=true; ifon=false; booland=false; boolor=false;}


////////// Baigiasi skliausteliai ir AND pridejimo zona
    // Pagrindinio išrinkimo ciklo pabaiga
  }
      s2.close();
}
    // OCL Ribojimo formavimas


    if (naudname!=null&&naudrange!=null&&naudstatement!=null&&ifon==false) {
      if (nauddesc==null)      {ribojimas="";}
      if (nauddesc!=null)      {ribojimas=  "\n"+nauddesc;}
      if (naudrange!=null)
        {
          //imama kontekstine klase is Range dalies
          if (unik_konklase!=null) {ribojimas= ribojimas + "\n"  + "context " + unik_konklase;}
          else {ribojimas= ribojimas + "\n"  + "context " + naudrange;}
        }
      if (inv_yra)            {ribojimas= ribojimas+" inv" + naudname + ":";}
      if (naudstatement!=null) {ribojimas= ribojimas + naudstatement;}
      if (endif) {endif=false;}

   System.out.println("\n" +  ribojimas);


      visiribojimai=visiribojimai+"\n"+ribojimas;

      setContents(rez_failas, visiribojimai);

      nauddesc = null;
```

```
      naudname = null;
      naudrange = null;
      naudstatement = null;
      unik_konklase = null;
      inv_yra = false;
      // Sintaksiniai kintamieji
      praleist=false;
      buvoskl=false;
      Input= null;
      abu_ANDOR=false;
      raktas = null;
    }

    //Išvalomi kintamieji
     // IŠVALOMI PO KIEKVIENOS EILUTES!
      klase1 = null;
      klase2 = null;
      klase3 = null;
      lygklase1=null;
      lygklase2=null;
      lygklase3=null;

      Input= null;
      not_on=false;


 /// Ieskoma Dalis

     // IŠVEDIMO PABAIGA
    }
  //  System.out.println(ribojimas); // Išvedamas tik paskutinis

     s.close();
     boolean success = (new File(tempfpav)).delete();
    if (!success) {System.out.println("Failo " + tempfpav + " ištrinti nepavyko"); }

    }//void main pabaiga
} // programos pabaiga
```