KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Lukas Laurinaitis

OWL ONTOLOGIJŲ TRANSFORMACIJA Į XMI FORMATĄ

Baigiamasis magistro projektas

Vadovas
doc. dr. Rita Butkienė

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

OWL ONTOLOGIJŲ TRANSFORMACIJA Į XMI FORMATĄ

Baigiamasis magistro projektas
Informacinių sistemų inžinerijos studijų programa (kodas 621E15001)

Vadovas
doc. dr. Rita Butkienė
2017-05-22

Recenzentas
doc. dr. Lina Čeponienė
2017-05-22

Projektą atliko
Lukas Laurinaitis
2017-05-22

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

(Fakultetas)

Lukas Laurinaitis

(Studento vardas, pavardė)

Informacinių sistemų inžinerijos studijų programa, 621E15001

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „OWL ONTOLOGIJŲ TRANSFORMACIJA Į XMI FORMATĄ“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužės 22 d.

Kaunas

Patvirtinu, kad mano, **Luko Laurinaičio**, baigiamasis projektas tema „OWL ONTOLOGIJŲ TRANSFORMACIJA Į XMI FORMATĄ“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Laurinaitis, Lukas. OWL ontologijų transformacija į XMI formatą. Magistro baigiamasis projektas / vadovė doc. dr. Rita Butkienė; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Informatikos inžinerija, technologijos mokslai

Reikšminiai žodžiai: *ontologija, transformacija, OWL, XMI*

Kaunas, 2017. 88 p.

SANTRAUKA

Šiuo metu ontologijos yra vis dažniau taikomos informacinėse technologijose įvairiems uždaviniams spręsti. Norint perduoti ontologijas tarp taikomųjų programų yra susiduriama su problema: nėra sprendimo, kuris leistų transformuoti *OWL* ontologijas į *XMI* formatą, skitą apsikeisti metaduomenimis.

Šio darbo tikslas yra sudaryti metodiką, kurią naudojant būtų galima transformuoti OWL ontologijas į *XMI* formatą. Darbe išanalizuoti egzistuojantys panašūs problemos sprendimai siekiant nustatyti jų privalumus ir trūkumus. Atlikus *OWL* ontologijos specifikacijos ir jos meta modelio analizę buvo sudaryti esminiai principai transformacijai atlikti bei parengtas realizacijos projektas.

Siekiant pagrįsti sudarytos metodikos veiksmingumą buvo realizuotas prototipas atliekantis *OWL* ontologijų transformaciją į *XMI* formatą. Eksperimento metu transformuotoms *XMI* ontologijoms buvo atliekama atvirkštinė transformacija atgal į *OWL* panaudojant *s2o* programą. Palyginus ontologijas prieš transformaciją ir po atvirkštinės transformacijos buvo nustatyta, jog ontologijos yra vienodos. Šie rezultatai įrodo, kad sukurtas metodas yra veiksmingas.

Laurinaitis, Lukas. *OWL Ontology Transformation Into XMI Format*: Master‘s thesis in Information Systems Engineering / supervisor doc. dr. Rita Butkienė. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: Informatics Engineering, Technology Science

Key words: *ontology, transformation, OWL, XMI*

Kaunas, 2017. 88 p.

SUMMARY

At the present time, the ontology is frequently used in a variety of information technology challenges. In order to share ontologies between applications, developers face one of the problems: there is no current solution that will allow the transformation of OWL ontology into XMI format which is designed for metadata interchange.

The aim of research – to create methodology which will be used to transform OWL ontologies into XMI format. Existing solutions analysis were carried out to determine their advantages and disadvantages. The essential principles of transformation and project design were made using OWL ontology specification and metamodel.

In order to test the technique of developed prototype, the tool has been developed which transformed OWL ontologies information into XMI format. The experimental tests were made using s2o to transform XMI ontologies back into OWL. Moreover the test results showed that ontologies before and after inverse transformation were the same. It proves that created methodology for OWL ontology transformation into XMI format is error - free.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

Terminų ir santrumpų žodynas

- ***OWL*** – ontologijų aprašymo kalba (angl. *Web Ontology Language*);
- ***XMI*** – OMG grupės sukurtas standartas metaduomenų apsikeitimui (***X****ML* ***M****etadata* ***I****nterchange*);
- ***OMG*** – tarptautinė standartų kūrimo grupė (***O****bject* ***M****anagement* ***G****roup*);
- ***TB*** – terabaitas (1024 gigabaitų);

ĮVADAS

Šiandieniniame technologijų pasaulyje kasdien yra generuojamas milžiniškas informacijos kiekis (5 mlrd. TB per dieną, 2013m.), tačiau nedidelė dalis jos tėra panaudojama. Tam, jog kuo didesnis sukauptos informacijos kiekis įgautu praktinę naudą mokslo ir pramonės vystymuisi ją būtina pateikti standartizuotu, visiems suprantamu pavidalu. Yra sukurta daug priemonių, kurios padeda pateikti žinias standartizuotai, ir viena efektyviausių – ontologijos. Ontologijos – tam tikros srities sąvokų visumos specifikavimas išreikštu pavidalu (angl. *explicit specification of a conceptualization*) [1]. Pagrindinis ontologijų tikslas – galimybė dalytis tam tikra informacijos struktūra tarp žmonių ir programų. Šiam tikslui buvo sukurta nemažai ontologijų kalbų joms išreikšti, ir viena populiariausių yra *OWL* (angl. *Web Ontology Language*).

Atsiradus efektyviai žinių pateikimo platformai reikėjo dar vieno svarbaus komponento norint efektyviai panaudoti sukauptą informaciją – standarto, kuriuo butų galima keistis metaduomenimis tarp taikomųjų programų. Tam tikslui geriausiai tinka *XMI* standartas. Šis standartas yra paremtas *XML* standartu, kuris yra tapęs vienu iš pagrindinių apsikeitimo duomenimis tarp skirtingos programinės įrangos būdų.

Kaip ir minėjome, ontologijų tikslas – galimybė keistis informacija. Galime įsivaizduoti, jog mes vienoje pusėje turime ontologijas išreikštas *OWL* kalba, kitoje – *XMI* standartą apsikeisti metaduomenimis, tačiau šioje vietoje iškyla problema: norint pasiekti sėkmingą informacijos apsikeitimą tarp taikomųjų programų reikia metodo, kuris užpildytų atsiradusį „tarpą“, t. y. transformuotų *OWL* ontologijas į *XMI* standartą.

Taigi, šio darbo tikslas yra atlikti dalykinės srities analizę, sukurti metodą transformacijai atlikti, realizuoti metodo prototipą ir patikrinti jo veikimo teisingumą.

1. PROBLEMINĖS SRITIES ANALIZĖ

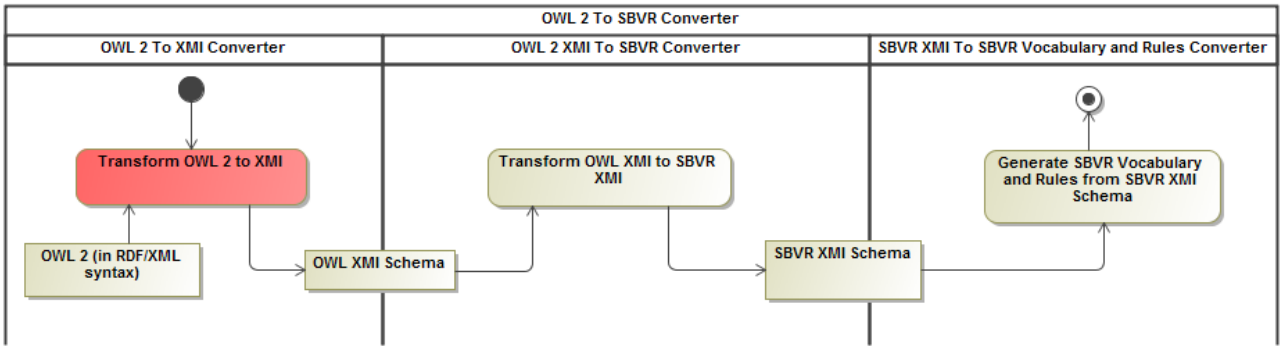
1.1. Analizės tikslas

Pagrindinis analizės tikslas – išanalizuoti probleminę sritį bei jos sprendimo būdus. Analizės metu siekiama atlikti ontologijų ir jos *OWL* kalbos analizę, jog būtų galima geriau suprasti ontologijų specifikavimo būdą, jos *OWL* kalbos principus, bei sąvokas. Taip pat atlikti panašių sprendimų analizę norint išsiaiškinti, kokie sprendimai egzistuoja, kokie yra jų trūkumai dėl kurių ši problema vis dar nėra išspręsta. Atlikus panašių sprendimų analizę bus galima sudaryti reikalavimus keliamus mūsų projektuojamai sistemai. Analizės metu taip pat bus analizuojamos Java bibliotekos skirtos darbui su *OWL* ontologijomis bei ontologijos kūrimo įrankiai siekiant išsiaiškinti, kuris įrankis geriausiai atitinka mūsų lūkesčius ir bus naudojamas ontologijoms sudaryti, peržiūrėti ar redaguoti.

1.2. Tyrimo objektas, sritis ir problema

OWL ontologijos gali būti išreikštos naudojant keletą sintaksės formų: *OWL Functional*, *RDF*, *OWL/XML* ir t.t. Šios sintaksės formos informacinėms sistemoms yra sunkiai suprantamos, o jų kūrėjams suprogramuoti palaikymą įvairioms sintaksės formoms yra per daug kompleksiškas ir didelių resursų reikalaujantis uždavinys. Norint perduoti ontologijas tarp taikomųjų programų ar jos komponentų reikalingas vieningas standartas, kuris būtų lengvai apdorojamas informacinėms sistemoms ir užtikrintų naudojamų sąvokų atitikimą ontologijos apibrėžtiems konceptams. Taigi, tyrimo objektas yra dalykinės srities ontologijos transformacijos į *XMI* standartu paremtą dokumentą metodika. Darbe nagrinėjama tyrimo sritis yra ontologijų inžinerija, taip pat ontologijų atvaizdavimas į informacinių sistemų inžinerijos modelius.

Be to, kuriama *OWL* ontologijų transformacijos į *XMI* formatą sistema bus naudojama kaip vienas iš komponentų (posistemė) „*OWL* to *SBVR*“ sistemoje (1 pav.). Tai sistema, kuri transformuoja *OWL* ontologijas į *SBVR* semantinius veiklos žodynus ir taisykles. Ši sistema susideda iš trijų komponentų: *OWL* ontologijų transformavimo į *XMI* formatą komponento, *OWL XMI* transformavimo į *SBVR XMI* ir *SBVR XMI* transformavimo į *SBVR* veiklos žodynus ir taisykles komponentų. Duomenų apsikeitimas tarp komponentų yra vykdomas *XMI* formatu. Šiame darbe projektuojamas komponentas diagramoje (1 pav.) yra pažymėtas raudona spalva.



1 pav. OWL ontologijų į SBVR veiklos žodynus ir taisykles transformavimo proceso diagrama

1.3. Tyrimo objekto analizė

1.3.1. Ontologijos apibrėžimas ir analizė

Ontologija – kompiuterijoje šio termino daugiskaitine forma ontologijos (skirtingai nuo filosofijoje – ontologija) vadinamas tam tikros srities sąvokų visumos specifikavimas išreikštu pavidalu [1].

Ontologijos apibrėžia nagrinėjamos srities:

- sąvokas, esybių (reiškinių, daiktų) tipus;
- sąvokų hierarchijas, esybių tipų tarpusavio sąryšius, priklausomybes;
- aksiomas, taisykles, dėsningumus apie esybių tipus ir sąryšius;

Remiantis ontologija galima aprašyti tam tikrą objektą, jo savybes, ryšius su kitais objektais, taip pat įvairius apribojimus. Ontologijos reikalingos tam, jog būtų galima dalintis tam tikros srities informacijos struktūra tarp žmonių ir programų, tam, jog būtų galima panaudoti vienas ontologijas kuriant kitas, srities sąvokoms ir sąryšiams detalizuoti ir tikslinti [2]. Ontologijų struktūra sudaryta iš šių dalių: klasių, atributų, ryšių ir individų (angl. *individuals*).

1.3.1.1. Ontologijų klasės

Klasės (angl. *classes*) reprezentuoja grupę skirtingų individų, kurie susieti bendromis charakteristikomis, kurios gali būti ir specifinės. Pvz., žmonės (kaip klasė) dalijasi tam tikromis bendromis charakteristikomis, tokiomis kaip *DNR*, tam tikros kūno dalys, galimybe kalbėti kompleksinėmis kalbomis. Taip pat žinduoliai irgi pasižymi tokiomis charakteristikomis, išskyrus gebėjimu kalbėti. Kai kurios ontologijų kalbos (viena iš jų *OWL*) leidžia turėti ir subklases, t. y. viena klasė gali paveldėti kitos klasės charakteristikas ir tuo pačiu turėti papildomų specifinių

charakteristikų, pvz.: klasė žmonės yra žinduolių klasės subklasė, kadangi turi savo DNR bei yra šilta kraujai.

1.3.1.2. Atributai

Ontologijų atributai nusako, apibrėžia objektų savybes, charakteristikas. Kiekvienas atributas gali būti tiek klasė, tiek individas. Objekto ir atributo tipas nusako ryšį tarp jų. Ryšys tarp objekto ir atributo parodo, jog tas ryšys yra specifinis šiam objektui.

Pavyzdžiui: objektas automobilis „Ford Explorer“ turi šiuos atributus:

- <turi pavadinimą> Ford Explorer,
- <turi detalę> duris,
- <turi detalę vieną iš> {4.0L variklis, 4.6L variklis},
- <turi detalę> 6 bėgių greičių dėžė,

Atributo reikšmė gali būti ir kompleksinio duomenų tipo, anksčiau minėto objekto atributas (variklis) gali turėti vieną iš dviejų subklasės elementų. Ontologijos yra tikros tada, kai klasės turi subklases (atributus).

1.3.1.3. Aksiomos

Aksioma – teiginys, kuris pasako kas yra tiesa. Ontologijose aksiomos yra naudojamos pasakyti tam tikrą informaciją apie klases, individus ar savybes, ir šita informacija yra užfiksuojama per ontologijos semantiką. Aksiomos užfiksuoja kurios klasės yra kitų klasių subklasės (angl. *subclass*), kurie individai priklauso (angl. *classAssertion*) kuriai klasei, priskiria savybes individams (angl. *propertyAssertion*), nurodo, kurios klasės yra ekvivalenčios (angl. *equivalentClasses*), o kurios yra nesusikertančios (angl. *disjointClasses*) ir pan.

1.3.1.4. Ryšiai

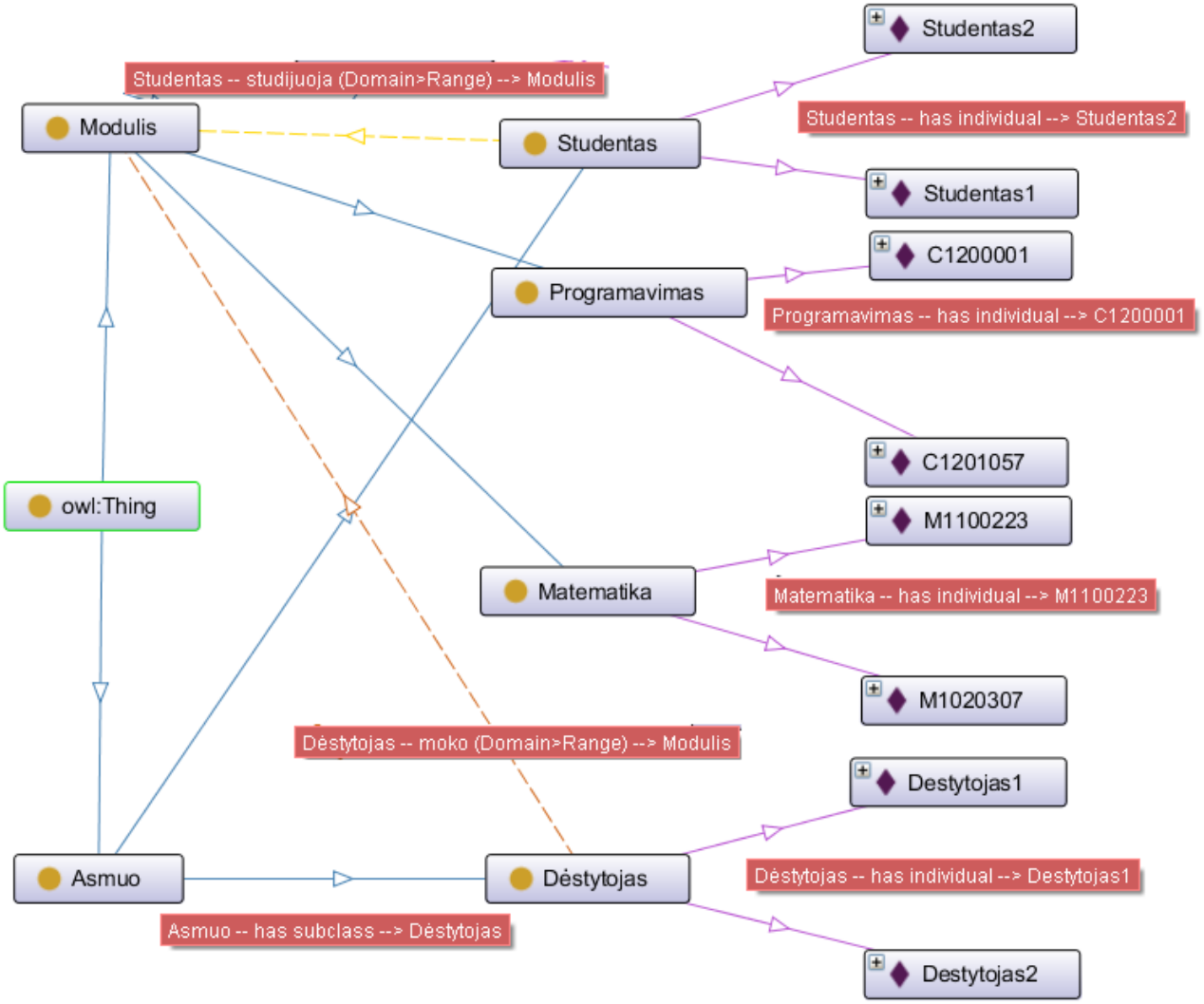
Ryšiai nusako kaip vieni ontologijos objektai yra susieti su kitais objektais. Įprastai ryšys yra atributas arba klasė, kuri nusako kaip vienas objektas yra susijęs su kitu objektu ontologijoje. Pavyzdžiui ontologijoje, kuri turi dvi klases: „Ford Explorer“ ir „Ford Bronco“ gali būti susietos ryšiu <yra apibrėžtas kaip įpėdinis>. Taigi pilna išraiška būtų: „Ford Explorer“ yra apibrėžtas kaip „Ford Bronco“ įpėdinis. Apibendrinant, ryšiai aprašo kaip vienas objektas susietas su kitu ontologijos objektu.

1.3.1.5. Individai

Individai (angl. *individuals*), dar kitaip žinomi kaip pavyzdžiai (angl. *instances*), yra bazinis ontologijos vienetas. Tai yra tai, ką ontologija aprašo arba gali aprašyti. Individai gali modeliuoti tokius konkrečius objektus kaip žmonės, mašinos ir pan. Taip pat individai gali modeliuoti ir abstrakčius dalykus, tokius kaip žmogaus darbas ar veikla, mašinos funkcija.

1.3.1.6. Ontologijos pavyzdys

Žemiau pateiktas sukurtas ontologijos pavyzdys naudojant „*Protégé*“ ontologijų kūrimo įrankį.



2 pav. Ontologijų modelis, aprašantis universitetą

Kaip matome, anksčiau pateiktas ontologijų modelis apie universitetą. Modelis prasideda klase „*Thing*“. Kas gi toji klasė? Tai yra klasių klasė, atskaitos taškas, apimantis viską, kas yra visatoje, kas galėtų būti aprašyta kaip klasė. „*Thing*“ klasė yra tėvinė klasė visuose ontologijų aprašuose. Iš šios klasės išeina jau šiam tyrimui aktualios dalykinės srities klasės ir jų ryšiai. Galima įžvelgti, jog klasės su *subklasėmis* sudaro hierarchinę struktūrą, kuri yra susieta ryšiais tarpusavyje. Pavyzdžiui, klasė „*Studentas*“ susieta su klase „*Modulis*“ per atributą „*studijuoja*“, taigi pilna iš būtų: „*Studentas*“ <*studijuoja*> „*Modulis*“.

Diagramoje taip pat galima įžvelgti ir individus, kurie priklauso konkrečioms klasėms, kurios juos aprašo.

1.3.2. OWL ontologijų kalba

Akronimas *OWL* yra kilęs iš vienos ontologijų kalbos pavadinimo „***W**eb **O**ntology **L**anguage*“. Ši ontologijų aprašymo kalba yra skirta publikuoti ir dalintis ontologijomis pasauliniame interneto tinkle. Ji buvo sukurta 2004 metais ir rekomenduojama *W3C* organizacijos kaip *RDF* (angl. *Resource Description Framework*) papildinys leidžiantis plačiau aprašyti ontologijas, bei nurodyti ryšius tarp hierarchijų. 2009 metais buvo išleista *OWL 2* versija. Nepaisant to, jog pirmoji versija buvo sėkminga, buvo ir pastebėta šiokiu tokių trūkumų. Vienas iš tokių trūkumų buvo tinkamų integruotų duomenų tipų nepakankamumas. Nors trūkumai ir nebuvo dideli, bet kartu sudėjus jie parodė, jog šią ontologijos kalbą reikia atnaujinti. Taip pat buvo siekiama sukurti tvirtą platformą ateities technologijoms.

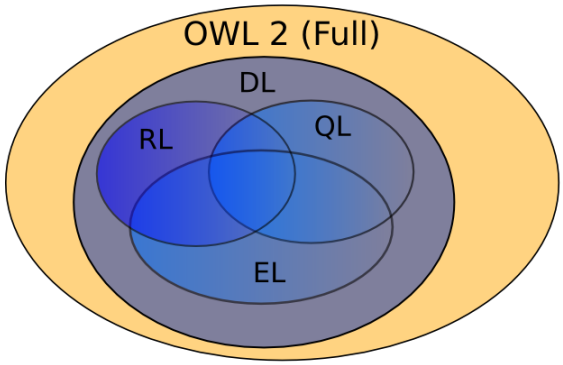
Ši ontologijų kalba turi tris skirtingos išraiškos lygius (dialektus): *OWL Lite*, *OWL DL*, *OWL FULL*.

OWL Lite versija skirta tiems vartotojams, kuriems pakanka tik hierarchines klasifikacijos ir paprastų apribojimų. Buvo tikėtasi, jog ši versija dėl savo paprastumo turės didesnį kiekį programinės įrangos *OWL* ontologijoms palaikyti, tačiau buvo pastebėta, jog programinės įrangos kūrimas *OWL Lite* versijai panašiai kompleksiškas kaip ir *OWL DL*, todėl pastaroji ir yra žymiai plačiau naudojama negu *Lite* versija.

OWL DL versija skirta suteikti didžiausią įmanomą išraiškingumą išlaikant skaičiavimo išbaigtumą (angl. *completeness*), išsprendžiamumą (angl. *decidability*) ir praktinį išvedimo (angl. *reasoning*) algoritmų prieinamumą. *OWL DL* turi visa *OWL* kalbos sandarą, bet ji gali būti naudojama tik su tam tikrais apribojimais. Akronimas *DL* kilęs iš angliško aprašymo logikos pavadinimo (angl. ***D**escription **L**ogic*).

OWL Full yra paremtas prieš tai esančia *OWL DL* versija, bei buvo pritaikytas suderinamumui su *RDF* schemomis. Pagrindinis skirtumas tarp *Full* ir *DL* versijų tai, jog *DL* versijoje daugiau apribotų funkcijų.

Naujai išleistoje *OWL 2* versijoje trys nauji dialektai (3 pav.), kurie buvo adaptuoti specialioms programinės įrangos užduotims spręsti: *OWL 2 EL*, *OWL 2 QL* ir *OWL 2 RL*.



3 pav. OWL2 kalbos dialektų aibės [3]

OWL 2 EL leidžia panaudoti polinominius laiko algoritmus standartinėms išvedimo (angl. *reasoning*) užduotims spręsti. Šis dialektas ypač tinkamas taikomosioms programoms, kurios naudoja ypač dideles ontologijas [4].

OWL2 QL dialektas skirtas programoms, kurios dirba su „lengvasvorėmis“ ontologijomis, dideliais individų kiekiais, bei tiesioginėmis užklausomis į duomenų bazes. Šis dialektas yra paremtas DL-Lite deskriptyvios logikos dialektais [4].

OWL2 RL yra *OWL 2* taisyklių poaibis, kuris aprėpia *EL* ir *QL* dialektus. Šis profilis (angl. *profile)* yra skirtas aplikacijoms, kurios reikalauja išplečiamumo per daug neprarandant išraiškingumo galios [4].

Šiuo metu naujausia yra OWL2 versija, todėl toliau darbe bus kalbama tik apie šia OWL ontologijų versiją.

1.3.2.1. OWL2 sintaksės pavyzdžiai

Siekiant aiškumo ne tik iš teorinės pusės, bet ir iš praktinės, žemiau pateikiami keturi *OWL2* kalbos sintaksės pavyzdžiai, kuriais gali būti aprašomos ontologijos.

1) *OWL2* funkcinė sintaksė:

```
Ontology(<http://example.com/tea.owl>
  Declaration( Class( :Tea ) )
)
```

4 pav. *OWL2* funkcinės sintaksės pavyzdys [4]

2) *OWL2 XML* sintaksė:

```
<Ontology ontologyIRI="http://example.com/tea.owl" ...>
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
  <Declaration>
    <Class IRI="Tea"/>
  </Declaration>
</Ontology>
```

5 pav. *OWL2 XML* sintaksės pavyzdys [4]

3) *RDF/XML* sintaksė:

```
<rdf:RDF ...>
    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:about="#Tea"/>
</rdf:RDF>
```

6 pav. *RDF/XML* sintaksės pavyzdys [4]

4) Turtle sintaksė:

```
<http://example.com/tea.owl> rdf:type owl:Ontology .
:Tea  rdf:type            owl:Class .
```

7 pav. Turtle sintaksės pavyzdys [4]

Kaip matyti, iš pirmojo pavyzdžio galime pasakyti, jog pateikta sintaksė nestruktūrizuota, toks pateiktos informacijos apsikeitimas komplikuotas, nes sudėtinga sukurti programinius įrankius gebančius korektiškai nuskaityti taip pateiktą informaciją ir ją interpretuoti. Antrame ir trečiame pavyzdyje matome, jog sintaksė turi struktūrą, yra aiškiau suprantama ne tik žmonėms, bet ir programinei įrangai. Kadangi *OWL* yra kilęs iš *RDF*, ontologijos *OWL* aprašus galima saugoti ir *RDF* formate, bei *RDF* formatu išsaugotus aprašus atgal transformuoti į *OWL*, tačiau, tai yra saugu tik *OWL DL* sluoksnyje. Kaip ir anksčiau jau minėjome, *OWL* yra labiau išplėtota ir turi daugiau funkcijų negu

RDF, todėl transformacija iš *OWL DL* ar *OWL Lite* į *RDF* nebus tiksli, nes RDF nėra tokia išraiški ir gali būti prarasta dalis informacijos.

1.3.3. XMI standartas

***X**ML **M**etadata **I**nterchange* arba sutrumpinus *XMI* yra *OMG* (*Object Management Group*) konsorciumo pasikeitimo metaduomenimis standartas. Šis standartas dažniausiai naudojamas metaduomenis tarp *UML* paremtų modeliavimo įrankių pasikeisti, taip pat gali būti naudojamas modelių transformacijai iš kitų ontologijų kalbų ir kaip priemonė perduoti modelius iš modeliavimo įrankių į programinės įrangos generavimo įrankius kaip modeliais grįstos inžinerijos dalis.

XMI standartas integruoja tris industrinius standartus:

1) ***XML*** (angl. *Extensible Markup Language*) – W3C konsorciumo rekomenduojama kalba duomenų apsikeitimui tarp skirtingų sistemų;
2) ***UML*** (angl. *Unified Modeling Language*) – vieninga modeliavimo ir specifikavimo kalba, skirta suteikti standartinį būdą vizualizuoti esamų arba būsimų sistemų architektūrai.
3) ***MOF*** (angl. *Meta Object Facility*) – *OMG* konsorciumo standartas skirtas modeliu paremtai inžinerijai.

Nuo *XMI* atsiradimo 2000 metais buvo sukurta dar keletas versijų: 1.0, 1.1, 1.2, 2.0, 2.1, 2.1.1, 2.4, 2.4.1 ir 2.4.2. Versija 2.x radikaliai skiriasi nuo 1.x versijos. Šiuo metu naudojama *XMI* versija 2.4.2 yra tarptautinis standartas, turintis *ISO* sertifikatą (*ISO/IEC 19509:2014*).

1.4. Tyrimo objekto naudotojų analizė

Tyrimo objekto naudotojų aibę sudaro:

1) Ontologijų kūrėjai;
2) *OWL* ontologijų transformacijų į *SBVR* standartą kūrėjai;
3) Kitų transformacijų susijusių su *OWL* ontologijomis kūrėjai;

Vartotojų tikslai ir problemos

Tyrimo objekto naudotojai siekia transformuoti ontologijas į kitus standartus, siekia paprasto apsikeitimo objektų modeliais ir kitais metaduomenimis internetu arba tarp taikomųjų programų.

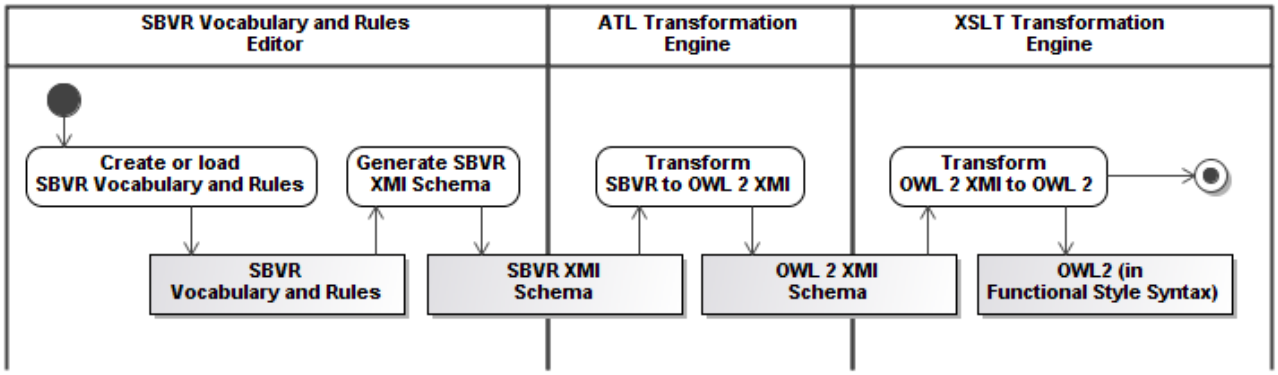
Kadangi šiuo metu transformuoti *OWL* ontologijas į *XMI* formatą nėra, tyrimo objekto naudotojai negali įgyvendinti savo siekių, todėl jiems reikia galimybes atlikti *OWL* transformaciją į *XMI.*

1.5. Panašių problemos sprendimo metodų analizė

Informacinių technologijų srityje ontologijos vis dar nėra labai populiari tema, tad ir panašių sistemų, kurios turėtų mus dominantį funkcionalumą, tik keletas.

1.5.1. SBVR žodynų ir taisyklių transformacija į OWL ontologijas

Vienas iš tokių sprendimų yra Daktaro Jaroslav Karpovič sukurtas „*SBVR to OWL (s2o)*“ transformacijos įrankis, kaip dalis tyrimo doktorantūros studijų metu, siekiant įrodyti, jog *SBVR* žodynai ir taisyklės yra tinkami *OWL2* ontologijų kūrimui [5].



8 pav. SBVR žodynų ir taisyklių į OWL ontologijas transformavimo įrankio transformacijos procesas

Šis įrankis (8 pav.) sudarytas iš trijų komponentų: *SBVR* žodynų ir taisyklių į *XMI* formatą transformavimo komponento, *SBVR XMI* į *OWL2 XMI* transformavimo komponento ir *OWL2 XMI* į *OWL* ontologijas transformavimo komponento. Bendravimas tarp komponentų *XMI* failų apsikeitimo būdu. Mus dominanti dalis yra transformavimo iš *OWL2 XMI* į *OWL* ontologijas komponentas, nors ir pati transformacija yra priešinga, negu šiame darbe projektuojama ir realizuojama. Transformacija yra atliekama naudojant *XSLT* transformacijos taisykles, tačiau šis sprendimas nėra atvirojo kodo, todėl nėra galimybės padaryti detalesnę sprendimo analizę.

1.5.2. OWL2XMI

Buvo rastas dar vienas panašus sprendimas - „*OWL2XMI*“ [6]. Šis pavadinimas tai *OWL* transformacijų į *XMI* trumpinys. Projektas sukurtas ir patalpintas „Semantic Web Architecture and Performance Group“ organizacijos, atvirojo kodo svetainėje. Tai Java programavimo kalba parašytas projektas palengvinti ontologijų klasių kūrimą *UML* diagramose. Įrankis iš ontologijų failo generuoja *XMI* failą, kurį galima importuoti į *UML* modeliavimo įrankius, tokius kaip: *StarUML*, *ArgoUML* ir kiti [6]. Projekto aprašymo svetainėje buvo pateikta nuoroda į programos versiją internetinėje

naršyklėje, tačiau nuoroda buvo neveikianti. Pabandžius pasileisti įrankį atsisiuntus iš projekto svetainės taip pat nepavyko. Kadangi programa atvirojo kodo, buvo analizuojamas programinis kodas. Pradėjus analizuoti išeities kodą buvo surastos klaidos, dėl kurių programa neveikdavo. Ištaisius jas buvo galima atlikti bandomąją transformaciją. Bandymo metu buvo pastebėta, jog programa priima tik *OWL* ontologijas išreikštas *RDF* sintakse, o *XMI* failas, gautas kaip bandymo rezultatas atskleidė, jog programa transformuoja *OWL* ontologijas į *XMI* formatą, kuris gali būti importuojamas į *UML* projektavimo įrankius gaunant ontologijos atvaizdą kaip *UML* diagramą. Gautą *XMI* failą buvo bandoma importuoti į *ArgoUML* ir *StarUML* taikomąsias programas, tačiau abiem atvejais nesėkmingai, todėl buvo nuspręsta detaliau paanalizuoti išeities kodą.

Jo analizė parodė, jog programa naudoja *Java* kalbos biblioteką *Jena RDF* ontologijų nuskaitymui iš failo, po to seka ontologijų klasių, subklasių, objektų atributų ir kardinalumų susiejimas pagal nuskaitytų ontologijų ryšius, *XMI* failo generavimas.

Taigi, šis įrankis yra geras tais atvejais, kai yra norima sugeneruoti *UML* klasių diagramą iš *OWL* ontologijų, kitais atvejais yra patiriamas informacijos nuostolis, nes *OWL* ontologijos be aukščiau išvardintų elementu dar turi ir duomenų atributus, individus, aksiomas ir kitus.

1.5.3. Panašių problemos sprendimų analizės išvados

Atlikę dviejų sprendimų analizę paaiškėjo, jog pirmasis sprendimas, „*SBVR to OWL 2 Converter*“ mums nėra tinkamas dėl to, nes jame naudojamas OWL transformacijos komponentas atlieka atvirkštinę transformaciją (*OWL XMI* į *OWL*), nei projektuojama šiame darbe. Antrasis sprendimas „*OWL2XMI*“ nėra priimtinas dėl kelių priežasčių: programos kode yra klaidų, dėl kurių programos nepavyksta paleisti iš karto atsisiuntus iš kūrėjų svetainės. Norint tai padaryti, teko pataisyti programinį kodą. Antra priežastis yra tai, jog programa transformuoja *OWL* ontologijų failą į *XMI* formatą skirta *UML* programose atvaizduoti ontologiją *UML* klasių diagramoje. Ši programa sukuria didelį duomenų praradimą transformacijos metu, kadangi netransformuoja tokias ontologijos dalis kaip: aksiomos, duomenų atributai, individai ir kitos.

1.6. Darbo tikslas, uždaviniai, planas ir siekiami privalumai

Darbo tikslas yra sukurti priemones dalykinės srities aprašymui tokia forma, kuri supaprastintų ir formalizuotų dalykinės srities žinių inžineriją ir leistų efektyviai panaudoti šias žinias taikomosiose programose. Turi būti sukurta taikomoji programa ir algoritmas leidžiantis transformuoti ontologijų *OWL* aprašus į *XMI* formatą.

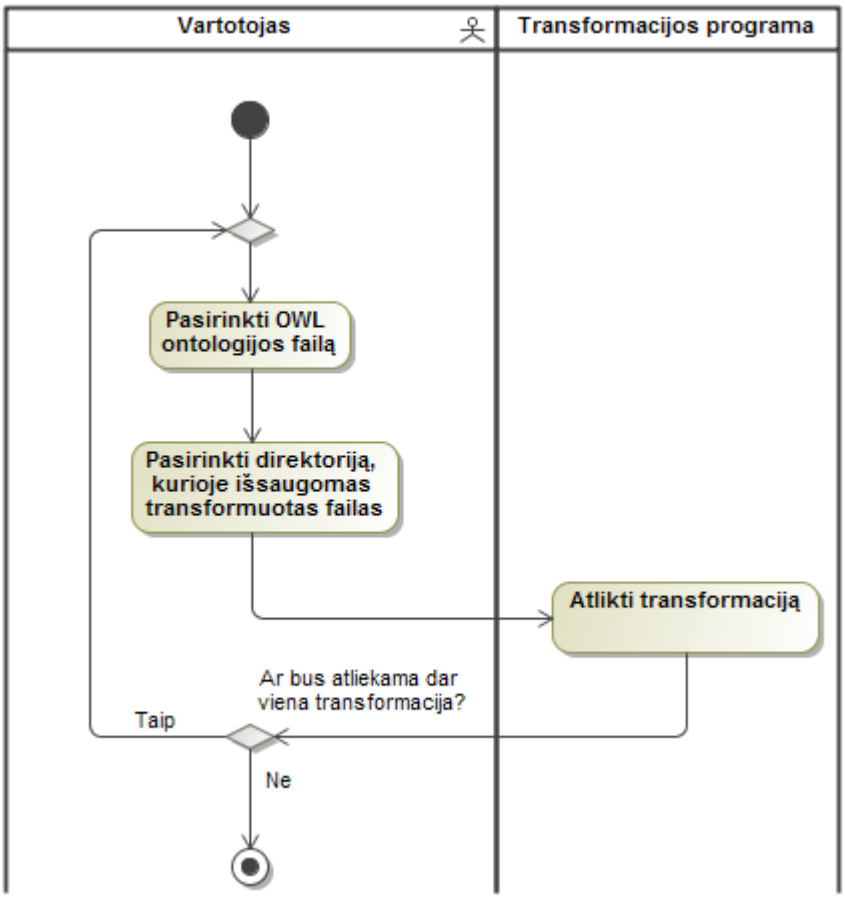
Darbo uždaviniai:

1. Atlikti *OWL* ir *XMI* analizę;
2. Specifikuoti reikalavimus ir suprojektuoti OWL ontologijų transformacijos į XMI standartą programą;
3. Realizuoti transformacijos programos prototipą;
4. Atlikti eksperimentą algoritmo tinkamumui įvertinti;

1.7. Siekiamo sprendimo apibrėžimas

Pagrindinis projektuojamo sprendimo tikslas – *OWL2* ontologijų transformacija į *XMI* failus. Norint atlikti transformaciją be didelių duomenų nuostolių reikia transformacijos algoritmą suprojektuoti taip, jog apimtų visą dalykinės srities meta modelį, kuris detalizuotas ir aprašytas *2.2* skyriuje ir jo poskyriuose. Nepaisant to, jog sistema turi būti realizuota kaip įrankis ontologijų kūrėjams, ji tuo pačiu turi būti realizuota taip, jog ją būtų galima integruoti į kitą sistemą, pavyzdžiui vykdyti transformaciją naudojant tik komandinę eilutę su parametrais. Šis sprendimas bus naujoviškas vien dėl priežasties, jog tokios *OWL2* ontologijų į *XMI* formatą transformacijos sistemos, kuri transformuotų neprarandant didelio kiekio informacijos, nėra. Sprendimui realizuoti bus naudojama OWL API biblioteka Java programavimo kalbai.

Norint pradėti projektuoti kuriamą programą, buvo sudaryta principinė programos naudojimosi schema (9 pav.), kuri padeda apibrėžti programoje atliekamus veiksmus ir ką programa turi daryti. Vartotojas pasirenka *OWL* ontologijų failą, kurį yra norima transformuoti, pasirenkama direktorija, kurioje bus išsaugomas *XMI* failas, atliekama transformacija.



9 pav. Projektuojamos programos principinė naudojimosi schema

1.8. Bibliotekų, skirtų darbui su ontologijomis, palyginimas

Nors ontologijos vis dar nėra labai populiari sritis informacinėse technologijose, tačiau galima rasti net keletą Java bibliotekų skirtų darbui su ontologijomis. Šios Java bibliotekos labai palengvina programų, kurios dirba su ontologijomis, realizaciją, turi realizuotus metodus ontologijų nuskaitymui, redagavimui ir manipuliavimui.

Detalesnei analizei buvo pasirinktos trys bibliotekos*: Jena RDF*, *OWLAPI* ir *Protege OWL API*.

1) Bene labiausiai paplitusi biblioteka yra *Jena RDF*. Tai yra atvirojo kodo semantinio tinklo platforma skirta Java programavimo kalbai. Biblioteką sukūrė ir iki 2009 m. tobulino *HP* įmonės tyrimų grupė Ši biblioteka yra orientuota į *RDF* ontologijas, kurias traktuoja kaip *RDF* grafus, padeda juos nuskaityti bei įrašyti. Grafai yra reprezentuojami kaip abstraktūs modeliai. Duomenys modeliui gali būti imami iš failų, duomenų bazių ar internetinių šaltinių, arba jų kombinacijų. Ši biblioteka palaiko keletą loginio išvedimo (angl. *reasoner*) ir keletą ontologijos kalbų: *RDF/XML, Turtle, Notation 3* [7].

2) Vis labiau populiarėjanti atvirojo kodo biblioteka *OWLAPI* buvo sukurta 2012 m., Mančesterio universitete, Jungtinėje Karalystėje. Ši biblioteka orientuota į OWL ontologijas, skirta jas kurti, nuskaityti, manipuliuoti ir publikuoti naudojant Java programavimo kalbą. Naujausia šios bibliotekos versija turi našų nuorodų į objektus saugojimo kompiuterio sparčiojoje atmintyje algoritmą, palaiko *OWL2* ontologijas, devynis loginio išvedimo (angl. *reasoner*) bei šešias *OWL* ontologijų aprašymo kalbas: *OWL Functional, OWL/XML, RDF/XML, Turtle, KRSS, OBO format* [8].
3) *Protege OWL API* – atvirojo kodo Java programavimo kalbai skirta biblioteka skirta darbui su *OWL* ir *RDF(S)* ontologijomis. Biblioteka yra paremta OWLAPI biblioteka, todėl turi implementuotas klases ir metodus skirtus nuskaityti ir išsaugoti *OWL* failus, manipuliuoti *OWL* duomenų modelius, yra optimizuota grafinės vartotojo sąsajos realizacijai. Ši biblioteka yra suprojektuota ir labiau orientuota į *Protege* įrankio įskiepių kūrimą. Palaikomos *OWL* aprašymo kalbos: *OWL Functional, OWL/XML, RDF/XML, Turtle, KRSS, OBO format, Manchester syntax* [9].

1 lentelė. Bibliotekų, skirtų darbui su OWL ontologijomis palyginimas

Įrankis / Savybė	OWL API	JENA RDF	Protege OWL API
Programavimo kalba	Java	Java	Java
Palaikomos ontologijų kalbos	OWL/XML, RDF/XML, Turtle, KRSS, OBO format	RDF/XML, Turtle, Notation 3	OWL Functional, OWL/XML, RDF/XML, Turtle, KRSS, OBO format, Manchester syntax
Orientuota į	OWL	RDF	OWL
Dokumentacija ir literatūra	Išsami, plati vartotojų bendruomenė	Išsami, plati vartotojų bendruomenė	Išsami, tačiau siaura vartotojų bendruomenė

Atlikę OWL ontologijas palaikančių bibliotekų analizę nustatyta, kad geriausiai lūkesčius atitinka OWL API biblioteka dėl to, kad turi gausų ontologijas išreiškiančių kalbų sąrašą, yra orientuota į OWL ontologijas (JENA RDF OWL ontologijas traktuoja kaip RDF grafus), bei turi plačia dokumentaciją ir vartotojų, kūrėjų bendruomenę. *Protege OWL API* yra labiau skirta *Protege* įrankio įskiepiams kurti.

1.9. Ontologijų kūrimo įrankių analizė

Atliekant probleminės srities analizę, projektuojant sprendimą bei atliekant testavimus bus būtinas įrankis, kuriame būtų galima peržiūrėti, redaguoti ar kurti ontologijas, todėl reikia atlikti ontologijų kūrimo įrankių analizę norint išsiaiškinti, kuris įrankis atitinka poreikius.

Ontologijų kūrimo įrankių analizei buvo pasirinkti trys iš rinkoje egzistuojančių įrankių: „*Protégé*“, „*TopBraid Composer*“ ir „*OntoStudio*“.

1) Šiuo metu populiariausias rinkoje esantis sprendimas darbui su ontologijomis yra „*Protégé*“. Viena pagrindinių priežasčių kodėl šis įrankis smarkiai išpopuliarėjo kitų įrankių tarpe yra tai, jog jis yra visiškai nemokamas ir yra atvirojo kodo. Pastaroji priežastis lėmė didelį kieki dokumentacijos ir atvirojo kodo bendruomenės aktyvumo dalinantis patirtimi, bei sukurtais įrankio plėtiniais išplečiančiais įrankio galimybes. Šios priežastys turėjo didelę įtaką įrankio išpopuliarėjimui. Įrankio vartotojo sąsaja yra aiški ir lengvai suprantama, bei patogi naudojimuisi.

2) Kitas analizei pasirinktas įrankis yra „*TopBraid Composter*“. Šis įrankis sukurtas „*TopQuadrant*“ kompanijos 2006 metais. Nuo sukūrimo pradžios išleistos trys produkto versijos: v1.0, v2.0 ir šiuo metu naudojama v3.0. Šis įrankis yra mokamas, tačiau yra galimybė išbandyti šį įrankį nemokamai trisdešimt dienų. Bandomoji programos versija taip pat yra apriboto funkcionalumo, mokamos programos versijos („*Standard*“ arba *Maestro*) turi tokias funkcijas kaip *XML* schemų generavimą ir HTML dokumentų generavimą iš ontologijų, bei kitas funkcijas.

3) Trečiasis įrankis yra „*OntoStudio*“, kurį sukūrė kompanija „*Semafora systems*“. Kaip ir „*TopBraid Composter*“, taip ir šis įrankis yra mokamas, bet turi 30 dienų bandomąją versiją. „*OntoStudio*“ palaiko OWL failų importą ir eksportą, turi grafinį ontologijų atvaizdavimą, bei funkcionalumo išplėtimo galimybes panaudojant papildinius (angl. *plug-ins*). Šis įrankis turi galimybę kolektyvinio ontologijų kūrimo galimybę, t. y. kuomet ontologijos saugomos serveryje ir su jomis gali dirbti keletas vartotojų.

2 lentelė. Ontologijų kūrimo įrankių palyginimas

Įrankis / Savybė	Protégé	TopBraid Composter	OntoStudio
Licencija	Nemokama	30 dienų bandomoji versija	30 dienų bandomoji versija
Palaikomos ontologijų kalbos	OWL,OWL/XML, RDF(S), OBO, Turtle	OWL, RDF(S), SPARQL	OWL, RDF(S), RIF, SPARQL
Papildiniai	Taip	Taip	Taip
Dokumentacija ir literatūra	Išsami, plati vartotojų bendruomenė	Siaura, nes dokumentacija skirta mokamoms versijoms	Siaura, nes dokumentacija skirta mokamoms versijoms

Atlikę ontologijų kūrimo įrankių analizę buvo padaryta išvada, jog mūsų poreikius geriausiai atitinka „*Protégé*“ įrankius dėl to, jog jis yra visiškai nemokamas, yra atvirojo kodo, bei turi išsamią dokumentaciją, daug internetinės literatūros ir plačia vartotojų bendruomenę.

1.10. Analizės išvados

1) Nėra vieningo standarto keistis ontologijomis tarp taikomųjų programų;
2) *OWL* ontologijų ir su jomis susiję kūrėjai galėtų keistis ontologijomis tarp taikomųjų programų, jei turėtų *OWL* ontologijų transformavimo į *XMI* formatą įrankį;
3) Atlikę panašių sprendimų analizę buvo nustatyta, jog poreikius geriausiai atitinka mūsų projektuojamas ir kuriamas sprendimas, dėl 1.5.3 skyrelyje išvardintų priežasčių;
4) Darbui su *OWL* ontologijomis Java bibliotekų analizė parodė, jog geriausiai lūkesčius atitinka *OWLAPI* biblioteka dėl plačios vartotojų ir kūrėjų bendruomenės.
5) Atlikę ontologijų kūrimo įrankių analizę išsiaiškinome, jog geriausiai mūsų poreikius išpildo „*Protégé*“ įrankis, dėl savo plačios dokumentacijos, visiškai nemokamos programos versijos, bei atviro kodo.

2. PROJEKTUOJAMOS OWL ONTOLOGIJŲ TRANSFORMAVIMO Į XMI FORMATĄ PROGRAMOS SPRENDIMO REIKALAVIMŲ SPECIFIKACIJA

2.1. Funkciniai projektuojamos programos reikalavimai

Žemiau pateikti projektuojamos sistemos funkciniai reikalavimai.

3 lentelė. Funkcinis reikalavimas R1

Aprašymas:	Sistema turi transformuoti OWL failus į XMI failus
Pagrindimas:	Norint perduoti ontologijas kitoms taikomosioms programoms būtina jas konvertuoti į XMI formatą
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Parodomas pranešimas, jog transformacija baigta ir yra sugeneruotas XMI failas su ontologija.

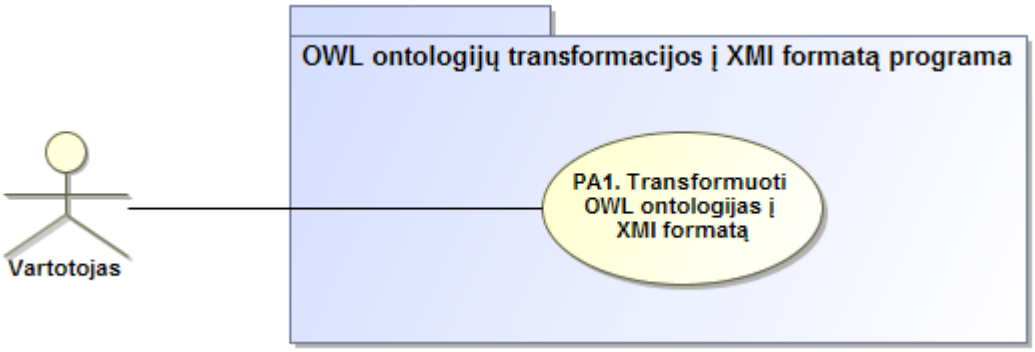
4 lentelė. Funkcinis reikalavimas R2

Aprašymas:	Sistema turi leisti nurodyti kokį OWL failą transformuosime
Pagrindimas:	Transformacijai reikalingas pradinis ontologijos failas, kurį vartotojas nori transformuoti
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Atsiranda transformacijos rezultatų išsaugojimo direktorijos nurodymo mygtukas

5 lentelė. Funkcinis reikalavimas R3

Aprašymas:	Sistema turi leisti nurodyti kur išsaugoti XMI failą
Pagrindimas:	Transformacijai reikalinga nurodyti, kur išsaugoti rezultatus
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Atsiranda transformacijos paleidimo mygtukas

Projektuojamos sistemos panaudos atvejų diagrama pateikta (10 pav.).



10 pav. Projektuojamos sistemos panaudos atvejų diagrama

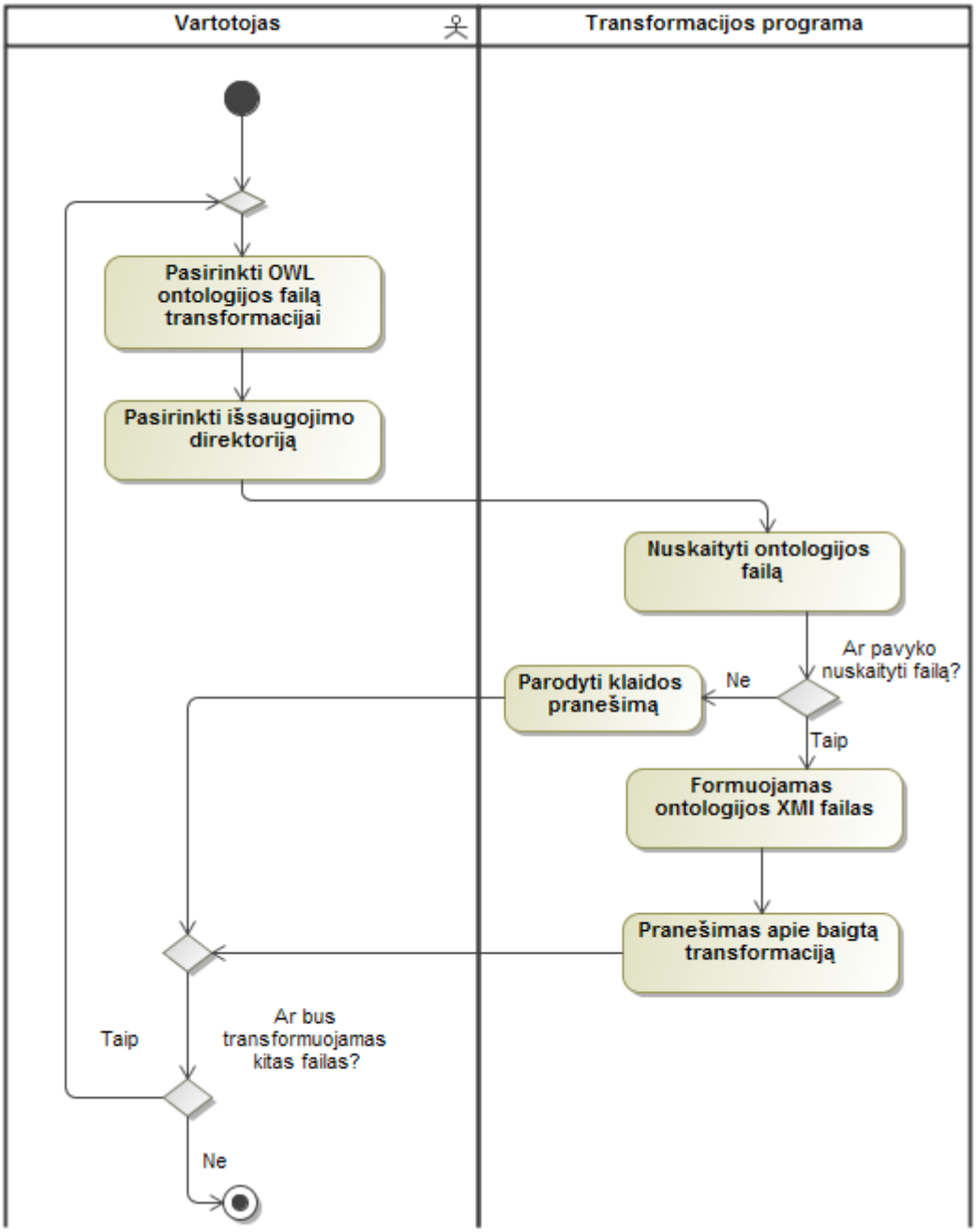
Panaudos atvejų diagramoje (10 pav.) matome, jog yra vienas vartotojas, kuris turi turėti galimybę transformuoti ontologijų failus. Norint atliktį transformaciją vartotojas privalo nurodyti kokį failą transformuosime bei direktoriją, kurioje transformuotas failas bus išsaugotas.

Žemiau pateiktos panaudos atvejo specifikacijos lentelė.

6 lentelė. PA1 „Transformuoti OWL ontologijas į XMI formatą“

PA1 „Transformuoti OWL ontologijas į XMI formatą“		
Tikslas: atlikti transformaciją.		
Prieš sąlyga		Turi būti nurodyta kokį OWL ontologijų failą transformuosime bei kur jį išsaugosime
Aktorius		Vartotojas
Sužadinimo sąlyga		Vartotojas paspaudžia transformavimo mygtuką
Susiję panaudojimo atvejai	**Išplečia PA**	–
	Apima PA	–
	Specializuoja PA	–
Pagrindinis įvykių srautas		**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia transformavimo mygtuką		Sistema pradeda transformaciją. Kuomet transformacija baigta išmetamas pranešimas.
Po sąlyga:		Nurodytoje vietoje yra išsaugomas transformuotos ontologijos failas.

Panaudojimo atvejui buvo sudaryta jį specializuojanti veiklos diagrama. Diagramoje detalizuotas panaudojimo atvejis smulkesne veiksmų seka koncentruojantis į pačias veiklas.



11 pav. PA1 „Transformuoti OWL ontologijas į XMI failus“ veiklos diagrama

Veiklos diagramoje (11 pav.) matome, jog vartotojas atsidaręs programą turi pasirinkti failą kurį transformuos programa, pasirinkti direktoriją, kurioje išsaugos transformuotą failą. Atlikus šiuos veiksmus ir pradėjus transformaciją programa nuskaito ontologijos failą, transformuoja jį į *XMI* formatą, bei parodo pranešimą apie transformaciją. Jei nuskaityti failo nepavyko išmetamas klaidos pranešimas.

2.1.1. Nefunkciniai reikalavimai

Žemiau pateikti išskirti projektuojamos sistemos nefunkciniai reikalavimai.

2.1.1.1. Reikalavimai sistemos išvaizdai

7 lentelė. Nefunkcinis reikalavimas R4

Aprašymas:	Sistema turi turėti vartotojo sąsają.
Pagrindimas:	Sistema naudosis kiti vartotojai, todėl reikalinga vartotojo sąsaja, jog būtų galima patogiai naudotis programa.
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Devyni vartotojai iš dešimties suprato ką reikia daryti norint transformuoti failą.

2.1.1.2. Reikalavimai vykdymo savybėms

8 lentelė. Nefunkcinis reikalavimas R5

Aprašymas:	Sistemoje turi būti indikacija apie vykstantį transformacijos procesą
Pagrindimas:	Vartotojas gali nesuprasti ar sistema „pastrigo“ ar vis dar vyksta transformacija, turi būti indikacija rodanti vykstantį transformavimo procesą.
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Transformacija negali trukti ilgiau nei dvi minutes

9 lentelė. Nefunkcinis reikalavimas R6

Aprašymas:	Sistema turi sklandžiai veikti su ontologijų failais, kurių dydis neviršija 15 megabaitų.
Pagrindimas:	Didelės apimties ontologijų failai gali būti sunkiai transformuojami dėl didelio duomenų kiekio, o transformacija gali ilgai užtrukti, todėl buvo apsibrėžtas toks failo dydis su kuriuo sistema sklandžiai dirba.
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Sistemoje nėra apribotas maksimalus ontologijos failo dydis, tačiau sklandus veikimas užtikrintas su failais, kurių dydis neviršija 15 megabaitų.

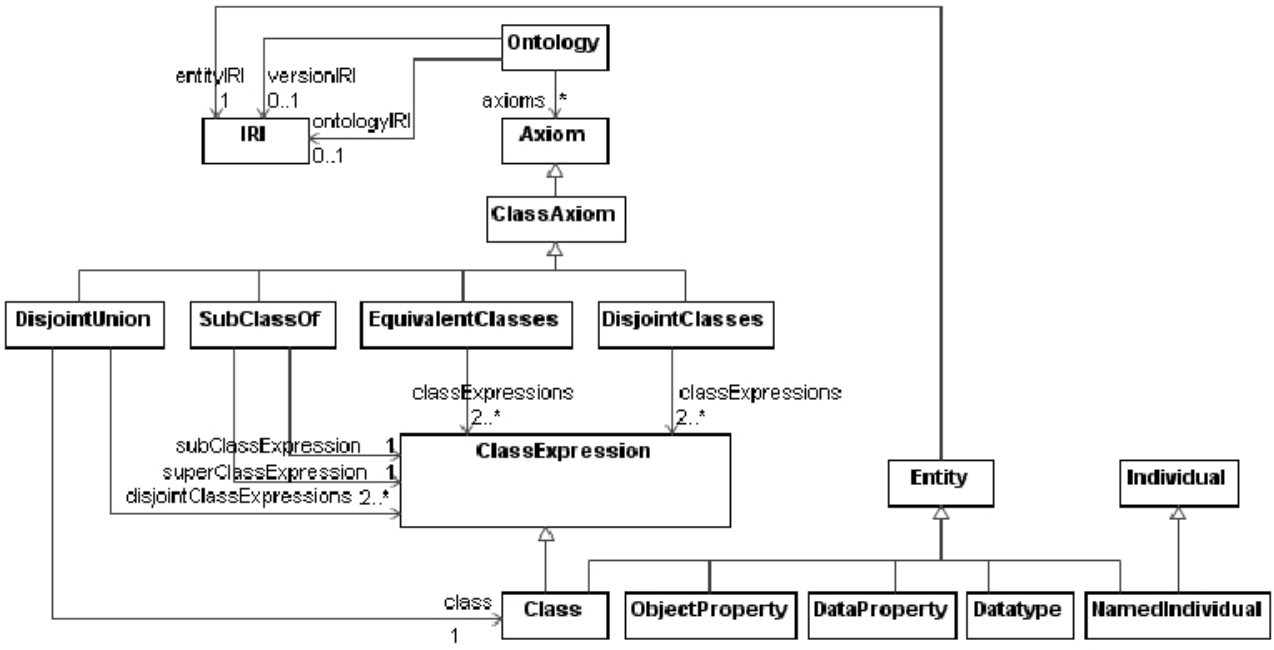
10 lentelė. Nefunkcinis reikalavimas R7

Aprašymas:	Sistema turi būti realizuota taip, jog atlikti transformaciją būtų galima ir be vartotojo sąsajos, t. y. naudojant komandinę eilutę su parametrais.
Pagrindimas:	Sistema ne tik bus naudojama kaip įrankis, bet bus naudojama ir kaip vienas iš komponentų ontologijų į SBVR veiklos žodynus ir taisykles transformavimo sistemoje.
Šaltinis:	Užsakovas
Tinkamumo kriterijus:	Transformacija galima iškvietus programą per komandinę eilutę su parametrais.

2.2. Dalykinės srities modelis

2.2.1. OWL2 meta modelis

OWL2 ontologijų specifikacija turi tris naujas išraiškas: EL, RL, QL, tačiau šiame projekte toliau bus kalbama tik apie *OWL2 DL* profilį (angl. *profile*) dėl jos išraiškingumo ir išbaigtumo. *OWL2* meta modelis (12 pav.) sudarytas iš trijų pagrindinių elementų: esybių, ontologijos ir aksiomų. Visi kiti komponentai yra šių trijų elementų subklasės. Norint atlikti *OWL* ontologijų transformaciją į *XMI* formatą reikia suprojektuoti ir realizuoti sistemą, kuri aprėptų pateiktą meta modelį (pateiktame modelyje 12 pav. yra atvaizduota tik nedidelė dalis aksiomų, plačiau jos aprašytos skyriuje 2.2.4). Norint geriau suprasti dalykinės srities modelį, jis bus detalizuojamas sekančiuose (2.2.2, 2.2.3, 2.2.4) skyriuose.

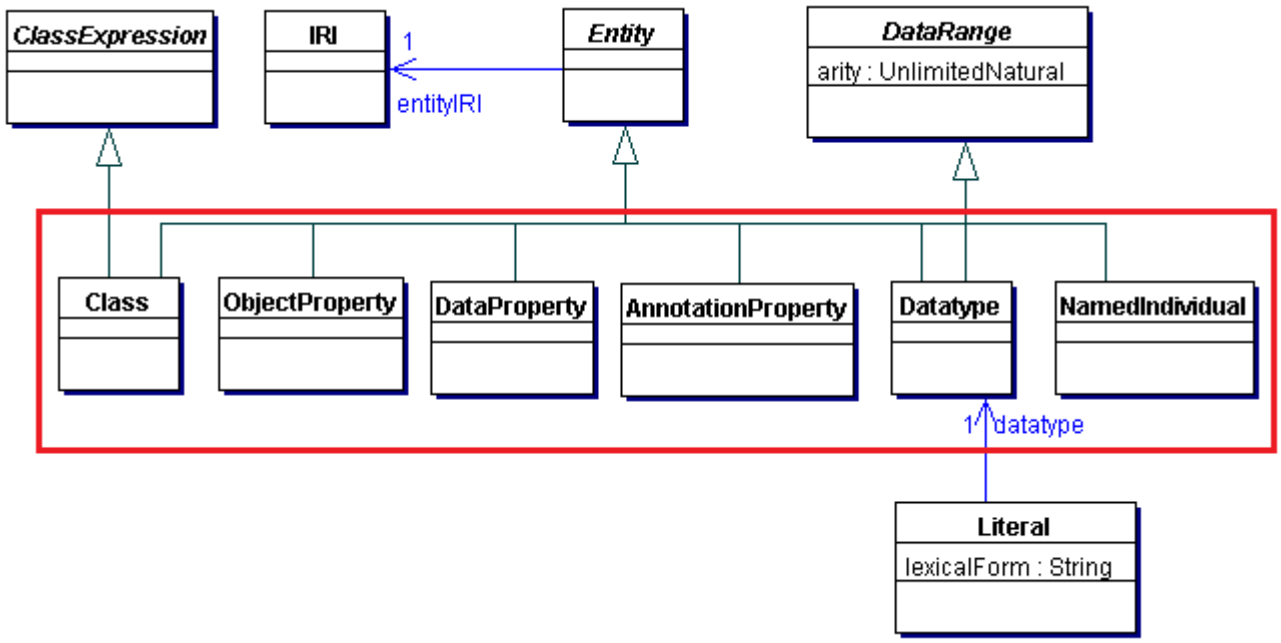


12 pav. OWL2 aukščiausio lygio meta modelis

2.2.2. Esybės

Dalykinės srities esybių modelį (13 pav.) sudaro 6 esybės: klasės (*OWLClass),* individai (*NamedIndividual),* anotacijos savybė (*AnnotationProperty),* duomenų savybė (*DataProperty),* objektų savybė (*ObjectProperty),* duomenų tipas (*Datatype).* Klasės reprezentuoja tam tikras individų grupes. Individai naudojami atvaizduoti tam tikrus realius objektus. Duomenų tipai yra literalų (angl. *literals*) rinkiniai. Kiekvienas literalas susideda iš atributo *lexicalForm*, kuris saugo tam tikrą reikšmę, o duomenų tipas, kuriam priklauso šis literalas nurodo kaip literalo reikšmę interpretuoti. Objektų ir duomenų savybės naudojamos atvaizduoti ryšiams, kaip objektai ar duomenys yra susieti. Anotacijos savybė yra naudojama susieti ontologijas arba esybes su informacija, kuri yra tik informacinio pobūdžio, tokia kaip aprašymai, pastabos ir pan. Kiekviena esybė turi savo, ir tik jai priklausantį

unikalų (*IRI*) identifikatorių, kuris naudojamas ontologijoje ryšiams sudaryti. Visos šios esybės sudaro ontologijos žodyną.



13 pav. OWL2 Dalykinės srities esybių modelis

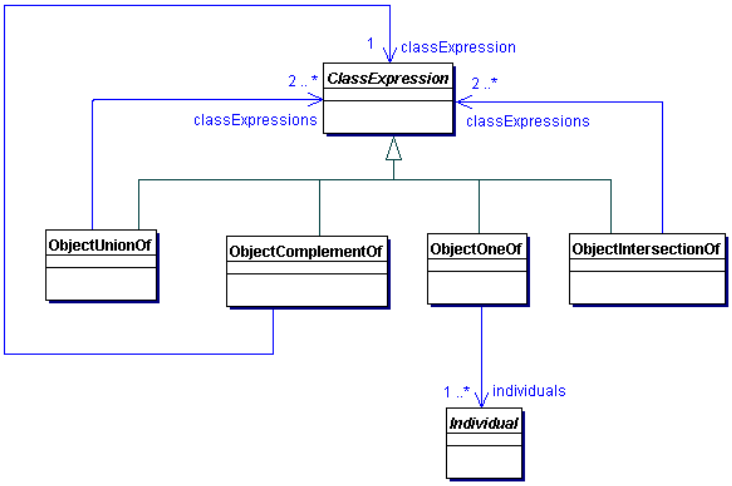
11 lentelė. Esybių išreiškimas OWL funkcine sintakse

Class:	*Class(a:Person)*
ObjectProperty:	*ObjectProperty(owl:topObjectProperty)*
DataProperty:	*DataProperty(owl:topDataProperty)*
AnnotationProperty:	*AnnotationProperty(<ns:Photo_equipment#label_sbvr>)*
NamedIndividual:	*NamedIndividual(a:Peter)*
DataType:	*Datatype(rdfs:Literal)*

2.2.3. Klasių išraiškos

Pagrindinis *OWL2* ontologijos meta modelio konceptas yra *OWL2* klasė, kuri yra klasių išraiškos subklasė. Tokių subklasių iš viso yra aštuoniolika. Galima išskirti dvi pagrindines klasių išraiškų grupes: klasių išraiškos, kurios siejasi su objektų savybėmis (14 pav., 15 pav., 16 pav.) ir klasių išraiškos, kurios siejasi su duomenų savybėmis (17 pav., 18 pav.).

OWL2 ontologijos turi klasių išraiškas, kurios skirtos teiginių jungimui ir individų sąrašų sudarymui (14 pav.).

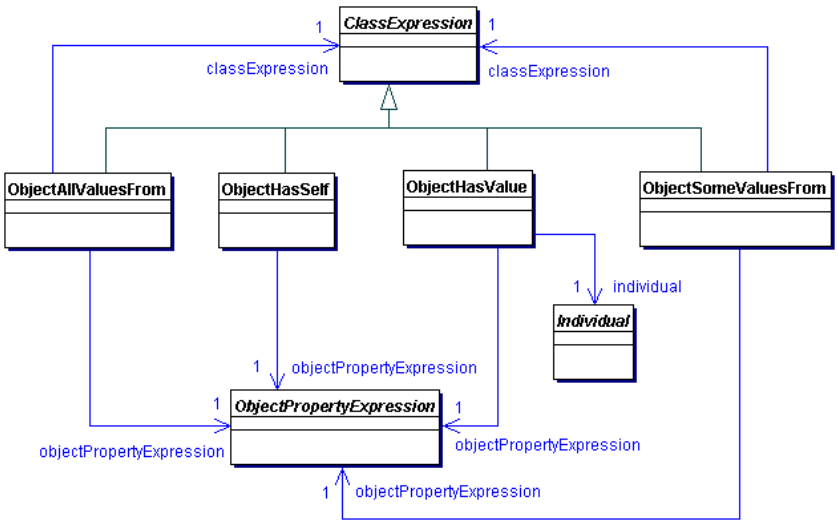


14 pav. Klasių išraiškos teiginių jungimui ir individų sąrašų sudarymui [10]

12 lentelė. Klasių išraiškų išreiškimas OWL funkcine sintakse (1 dalis)

ObjectUnionOf	*ObjectUnionOf(a:Man a:Woman)*
ObjectOneOf	*ObjectOneOf(a:Peter a:Lois a:Stewie a:Meg a:Chris a:Brian)*
ObjectComplementOf	*ObjectComplementOf(a:GriffinFamilyMember)*
ObjectIntersectionOf	*ObjectIntersectionOf(a:Child a:Man)*

Klasių išraiškos *OWL2* ontologijose gali būti suformuojamos uždedant apribojimus objektų savybėms. Tokios išraiškos pavaizduotos žemiau esančiame paveikslėlyje (15 pav.).

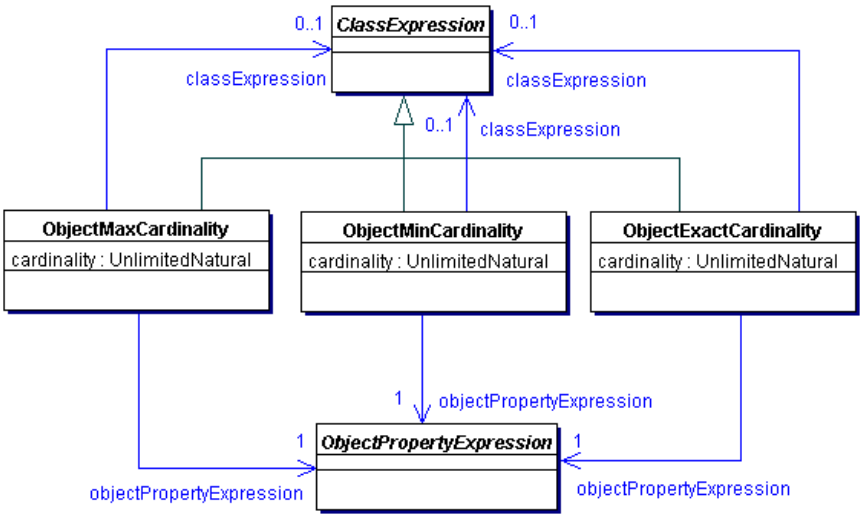


15 pav. Klasių išraiškos apribojančios objektų savybes [10]

13 lentelė. Klasių išraiškų išreiškimas OWL funkcine sintakse (2 dalis)

ObjectAllValuesFrom	*ObjectAllValuesFrom(a:hasPet a:Dog)*
ObjectHasSelf	*ObjectHasSelf(a:likes)*
ObjectHasValue	*ObjectHasValue(a:hasChild a:Stewie)*
ObjectSomeValuesFrom	*ObjectSomeValuesFrom(a:fatherOf a:Man)*

Klasių išraiškos *OWL2* ontologijose gali būti suformuojamos uždedant apribojimus objektų savybių kardinalumui. Kardinalumo apribojimai galioja tik tiems individams, kurie yra susieti su objekto savybe. Tokios išraiškos pavaizduotos žemiau esančiame paveikslėlyje (16 pav.).

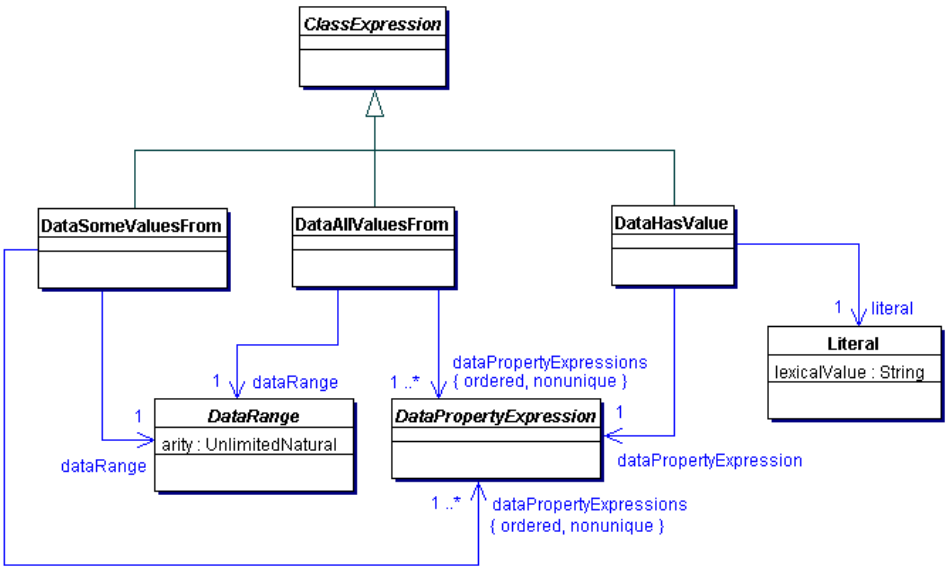


16 pav. Klasių išraiškos ribojančios objektų savybių kardinalumą [10]

14 lentelė. Klasių išraiškų išreiškimas OWL funkcine sintakse (3 dalis)

ObjectMaxCardinality	*ObjectMaxCardinality(1 a:hasPet)*
ObjectMinCardinality	*ObjectMinCardinality(2 a:fatherOf a:Man)*
ObjectExactCardinality	*ObjectExactCardinality(1 a:hasPet a:Dog)*

Šios klasių išraiškos (17 pav.) yra suformuojamos dedant apribojimus duomenų savybėms. Šie apribojimai yra panašūs objektų savybių apribojimams, tačiau esminis skirtumas tarp jų yra tai, jog apribojimams yra naudojami masyvai su duomenų intervalais (angl. *data ranges*).

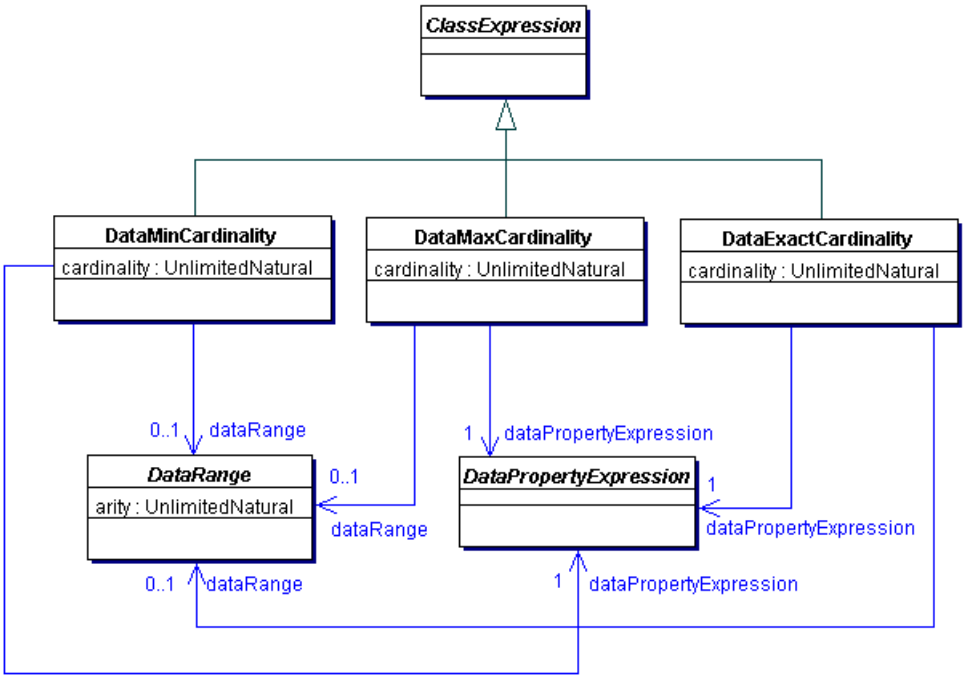


17 pav. Klasių išraiškos ribojančios duomenų savybes [10]

15 lentelė. Klasių išraiškų išreiškimas OWL funkcine sintakse (4 dalis)

DataSomeValuesFrom	*DataSomeValuesFrom(a:hasAge DatatypeRestriction(xsd:integer xsd:maxExclusive "20"^^xsd:integer))*
DataAllValuesFrom	*DataAllValuesFrom(a:hasZIP xsd:integer)*
DataHasValue	*DataHasValue(a:hasAge "17"^^xsd:integer)*

Šios klasių išraiškos (18 pav.) yra suformuojamos uždedant apribojimus duomenų savybių kardinalumams. Šie apribojimai panašūs kaip ir objektų savybių kardinalumų apribojimai, tačiau kardinalumų apribojimai galioja literalams, kurie priskirti duomenų savybei.



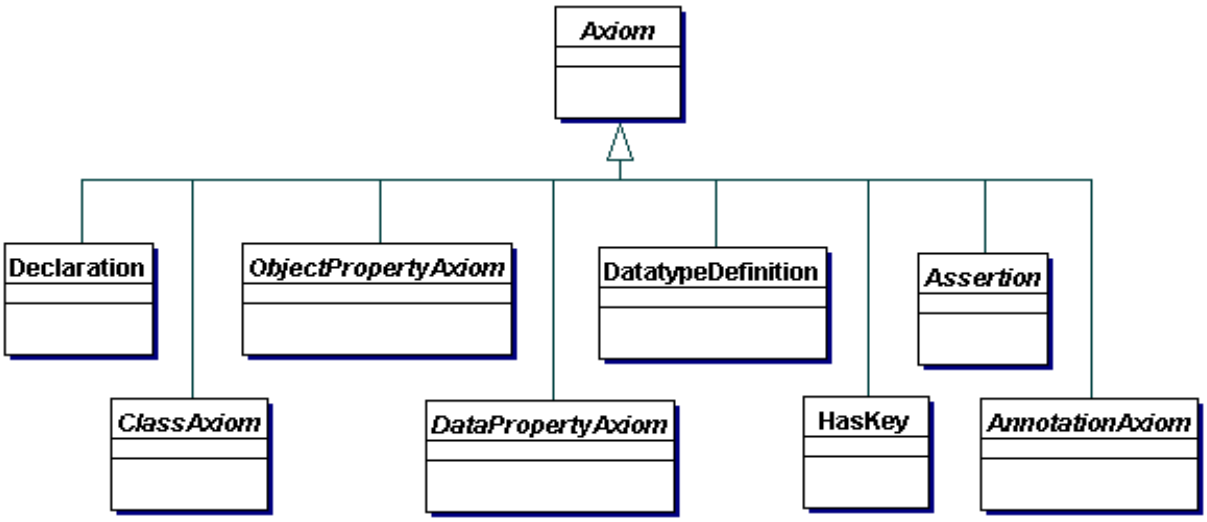
18 pav. Klasių išraiškos ribojančios duomenų savybių kardinalumą [10]

16 lentelė. Klasių išraiškų išreiškimas OWL funkcine sintakse (5 dalis)

DataMinCardinality	*DataMinCardinality(2 a:hasName)*
DataMaxCardinality	*DataMaxCardinality(2 a:hasName)*
DataExactCardinality	*DataExactCardinality(1 a:hasName)*

2.2.4. Aksiomų tipai

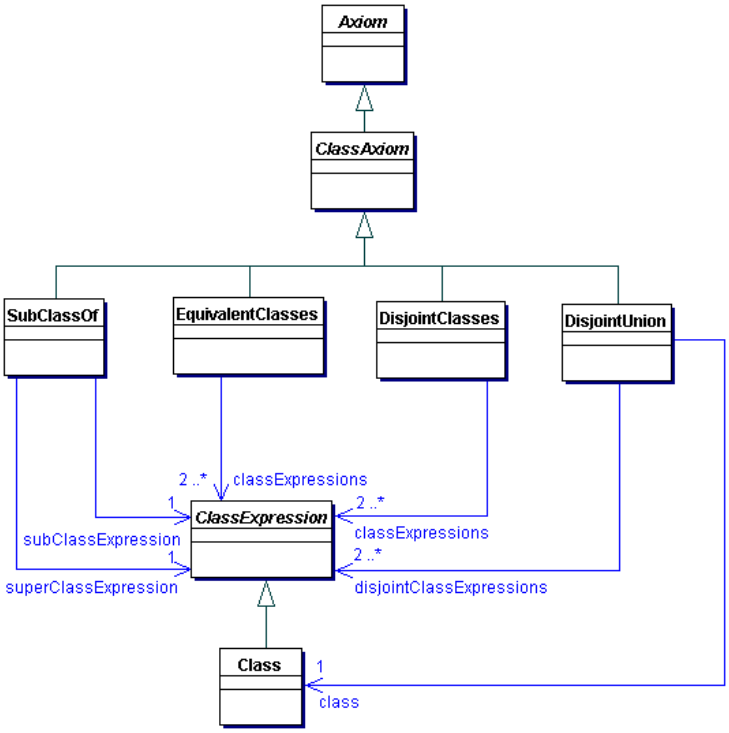
OWL2 ontologijose žinios yra vaizduojamos aksiomų pagalba. Aksiomos – pagrindinis *OWL2* ontologijų komponentas, o jų tipų yra trisdešimt septynios. Šiuos aksiomų tipus galima suskirstyti į aštuonias grupes: klasių, objektų savybių, duomenų savybių, duomenų tipų apibrėžimų, teiginių, deklaracijos, raktų ir anotacijų.



19 pav. OWL2 aksiomos [10]

2.2.4.1. Klasių aksiomų tipas

Klasių aksiomos leidžia sudaryti ryšius tarp klasių išraiškų (angl. *ClassExpression*).



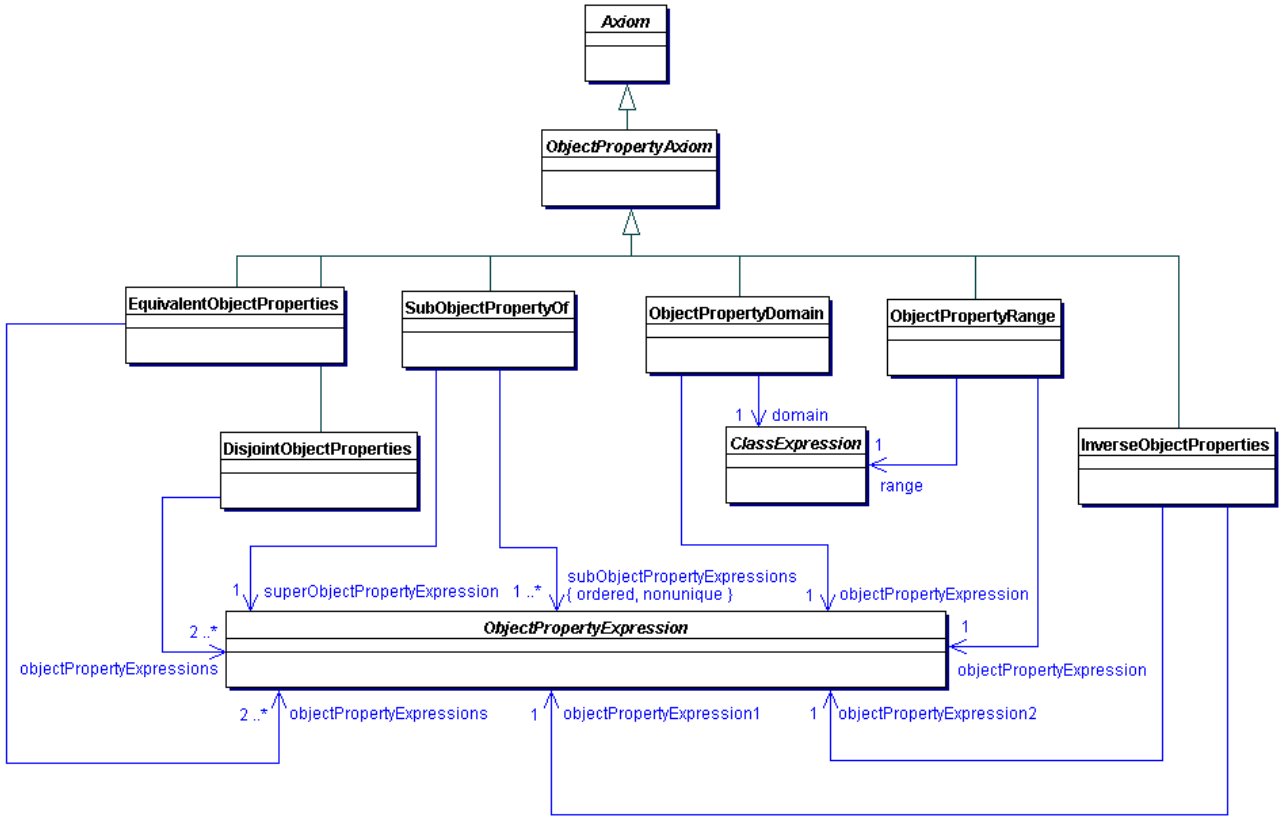
20 pav. OWL2 klasių aksiomų tipai [10]

17 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse (1 dalis)

DataMinCardinality	*DataMinCardinality(2 a:hasName)*
DataMaxCardinality	*DataMaxCardinality(2 a:hasName)*
DataExactCardinality	*DataExactCardinality(1 a:hasName)*

2.2.4.2. Objektų savybių aksiomų tipas

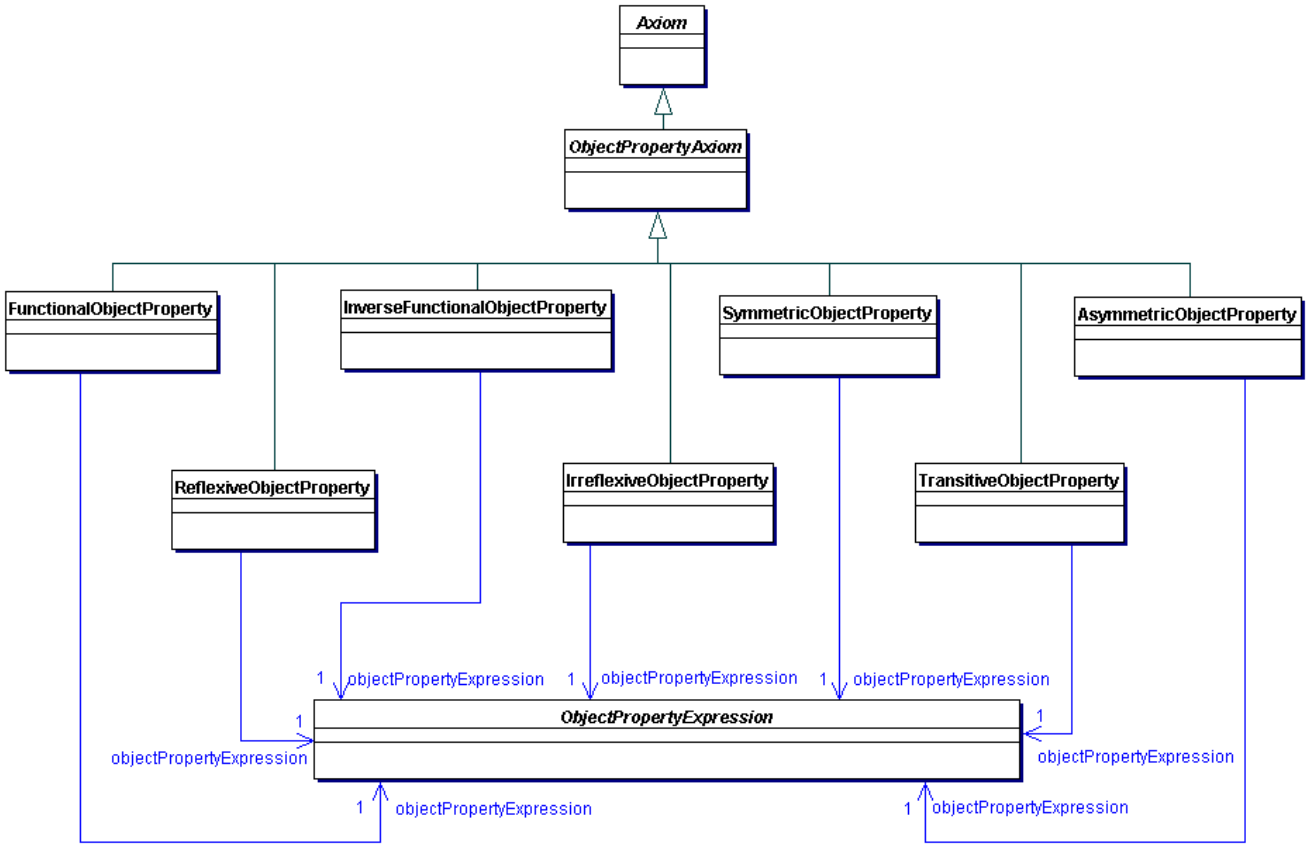
Objektų savybių aksiomos gali būti naudojamos charakterizuoti ir sudaryti ryšius tarp objektų savybių. Dėl aiškumo atvaizduojant aksiomų tipus, jie buvo išskaidyti į dvi diagramas: 21 pav. ir 22 pav.



21 pav. OWL2 objektų savybių aksiomų tipai (pirma dalis) [10]

18 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse (2 dalis)

EquivalentObjectProperties	*EquivalentObjectProperties(a:hasBrother a:hasMaleSibling)*
DisjointObjectProperties	*DisjointObjectProperties(a:hasFather a:hasMother)*
SubObjectPropertyOf	*SubObjectPropertyOf(a:hasUncle a:hasRelative)*
ObjectPropertyDomain	*ObjectPropertyDomain(a:hasDog a:Person)*
ObjectPropertyRange	*ObjectPropertyRange(a:hasDog a:Dog)*
InverseObjectProperties	*InverseObjectProperties(a:hasFather a:fatherOf)*



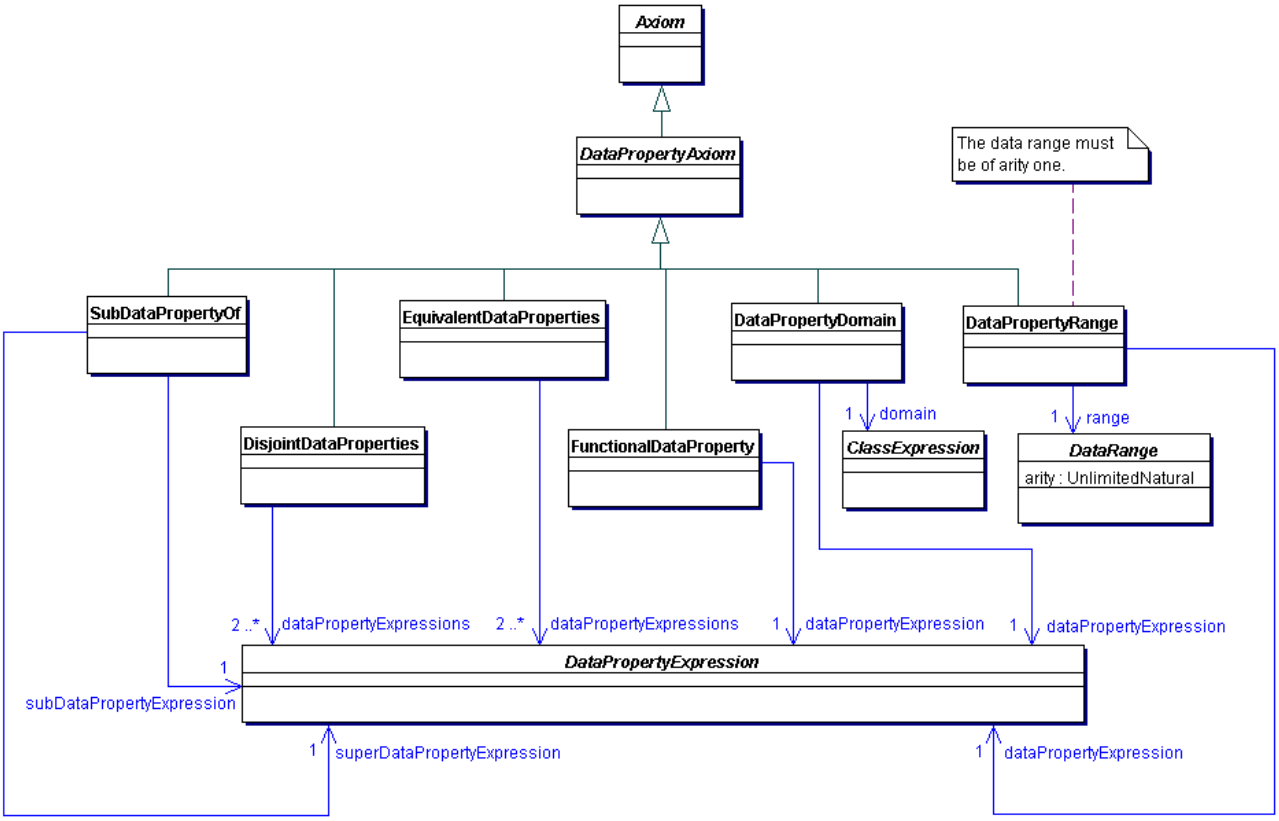
22 pav. OWL2 objektų savybių aksiomų tipai (antra dalis) [10]

19 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse (3 dalis)

FunctionalObjectProperty	*FunctionalObjectProperty(a:hasFather)*
ReflexiveObjectProperty	*ReflexiveObjectProperty(a:knows)*
InverseFunctionalObjectProperty	*InverseFunctionalObjectProperty(a:fatherOf)*
IrreflexiveObjectProperty	*IrreflexiveObjectProperty(a:marriedTo)*
SymmetricObjectProperty	*SymmetricObjectProperty(a:friend)*
TransitiveObjectProperty	*TransitiveObjectProperty(a:ancestorOf)*
AsymmetricObjectProperty	*AsymmetricObjectProperty(a:parentOf)*

2.2.4.3. Duomenų savybių aksiomų tipas

OWL2 taipogi turi aksiomas duomenų savybėms. Struktūriškai duomenų savybių aksiomų tipai (23 pav.) yra panašūs objektų duomenų savybių aksiomų tipams (21 pav., 22 pav.).



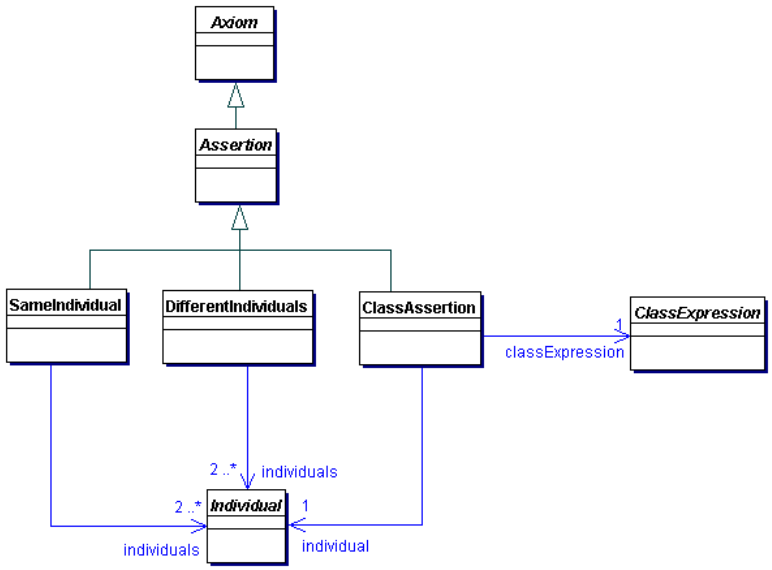
23 pav. OWL2 duomenų savybių aksiomų tipai [10]

20 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse (4 dalis)

SubDataPropertyOf	*SubDataPropertyOf(a:hasLastName a:hasName)*
DisjointDataProperties	*DisjointDataProperties(a:hasName a:hasAddress)*
EquivalentDataProperties	*EquivalentDataProperties(a:hasName a:seLlama)*
FunctionalDataProperty	*FunctionalDataProperty(a:numberOfChildren)*
DataPropertyDomain	*DataPropertyDomain(a:hasName a:Person)*
DataPropertyRange	*DataPropertyRange(a:hasAge xsd:integer)*

2.2.4.4. Teiginių aksiomų tipai

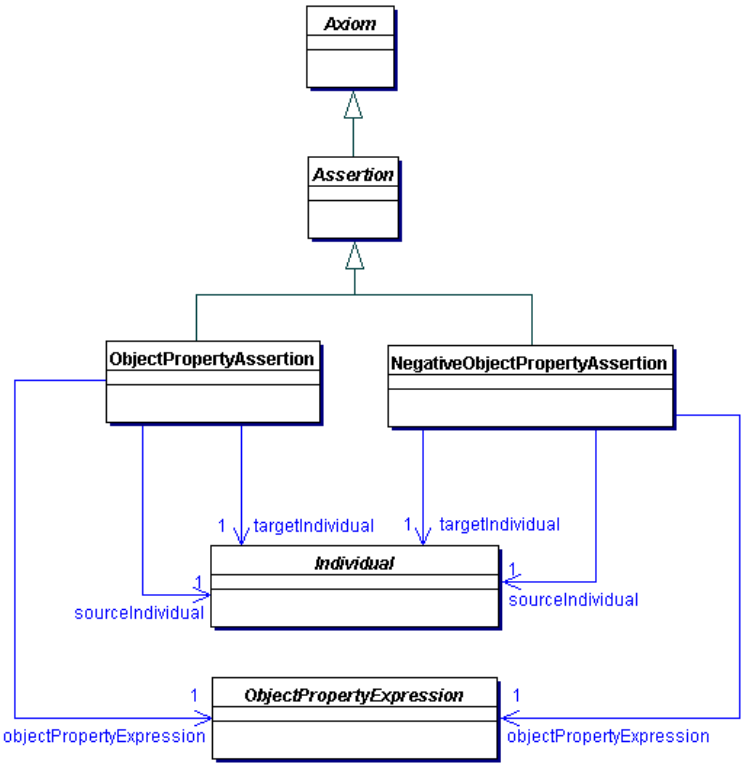
Teiginių (angl. *assertion*) aksiomos, dar žinomos kaip faktai (anlg. *facts*) aprašo individus. Siekiant aiškumo teiginių aksiomų tipai pateiktos trijose diagramose: 24 pav., 25 pav., 26 pav.



24 pav. Teiginių aksiomų tipai (pirma dalis) [10]

21 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse (5 dalis)

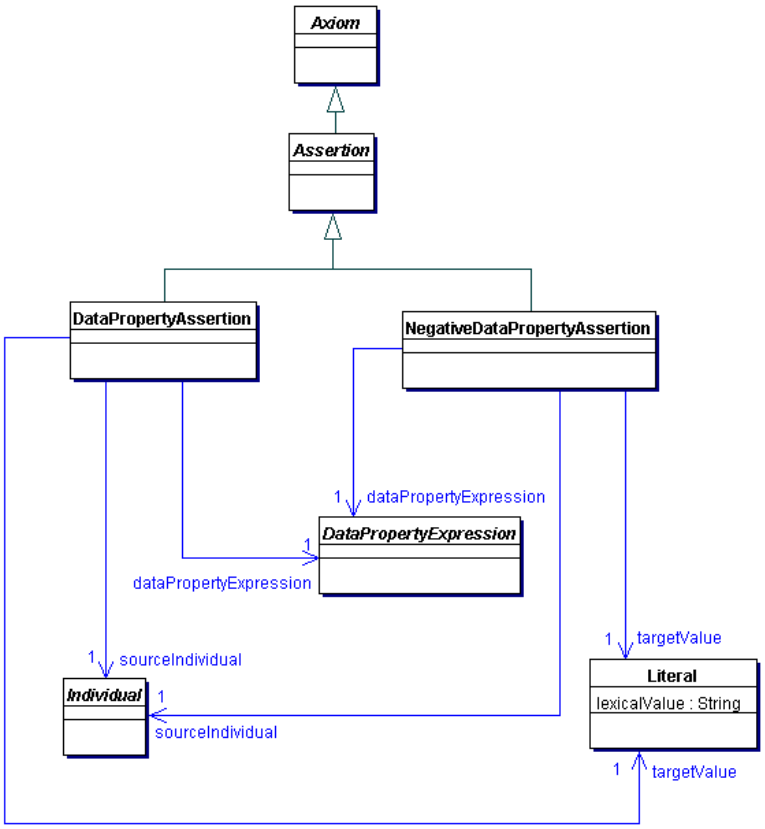
SameIndividual	*SameIndividual(a:Meg a:Megan)*
DifferentIndividuals	*DifferentIndividuals(a:Peter a:Meg a:Megan)*
ClassAssertion	*ClassAssertion(a:Dog a:Brian)*



25 pav. Teiginių aksiomų tipai (antra dalis) [10]

22 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse

ObjectPropertyAssertion	*ObjectPropertyAssertion(a:hasPet a:Peter a:Brian)*
NegativeObjectPropertyAssertion	*NegativeObjectPropertyAssertion(a:hasSon a:Peter a:Meg)*



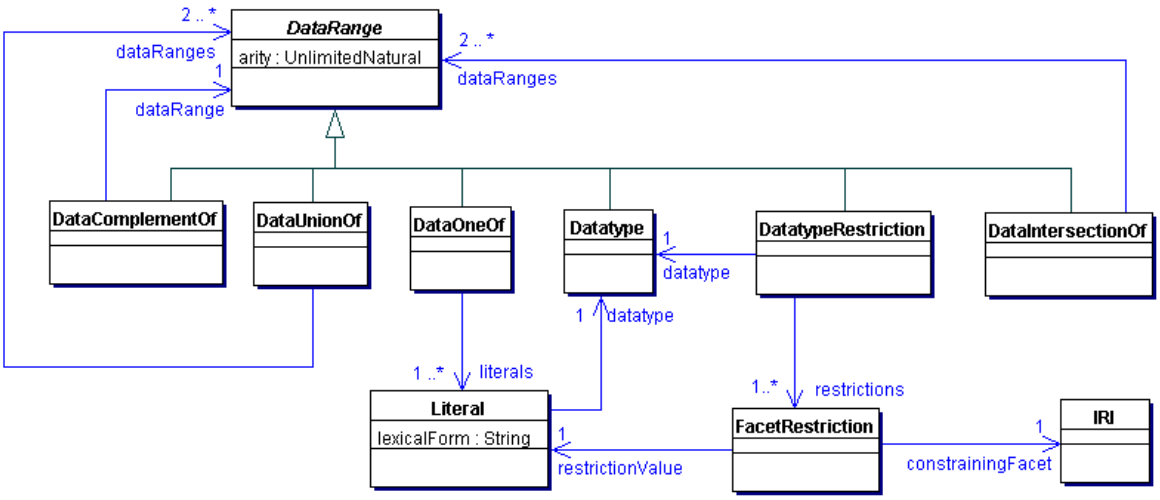
26 pav. Teiginių aksiomų tipai (trečia dalis) [10]

23 lentelė. Aksiomų tipų išreiškimas OWL funkcine sintakse (6 dalis)

DataPropertyAssertion	*DataPropertyAssertion(a:hasAge a:Meg "17"^^xsd:integer)*
NegativeDataPropertyAssertion	*NegativeDataPropertyAssertion(a:hasAge a:Meg "5"^^xsd:integer)*

2.2.5. Duomenų intervalas

Duomenų intervalai (angl. *data ranges*) (27 pav.) gali būti naudojami duomenų savybių apribojimams sudaryti. Paprasčiausi duomenų intervalai yra *Datatype* ir *DataOneOf*. Jie sudaryti iš atitinkamai vieno ir kelių literalų. Kiti duomenų intervalai yra kompleksiniai bei sudaryti iš kitų duomenų intervalų.



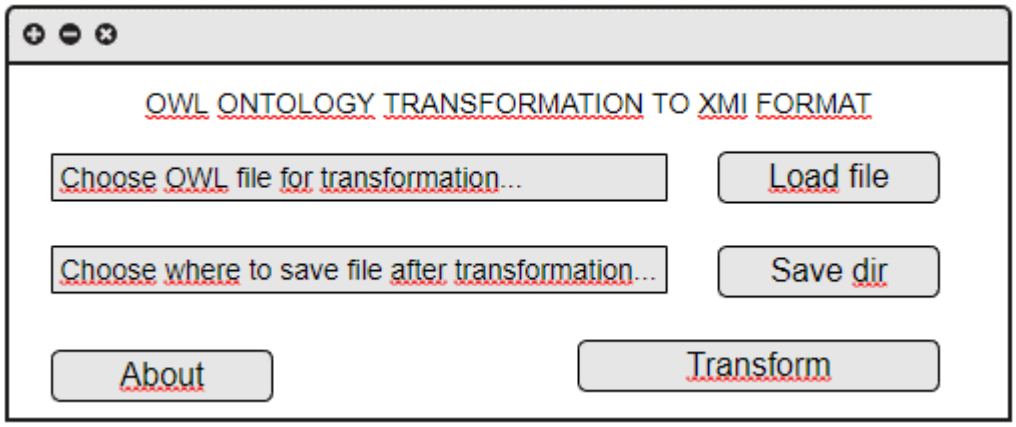
27 pav. Duomenų intervalo tipai (subklasės) *OWL2* ontologijoje [10]

24 lentelė. Duomenų intervalo tipų išreiškimas OWL funkcine sintakse

DataComplementOf	*DataComplementOf(xsd:positiveInteger)*
DataUnionOf	*DataUnionOf(xsd:string xsd:integer)*
DataOneOf	*DataOneOf("Peter" "1"^^xsd:integer)*
DatatypeRestriction	*DatatypeRestriction(xsd:integer xsd:minInclusive "5"^^xsd:integer xsd:maxExclusive "10"^^xsd:integer)*
DataIntersectionOf	*DataIntersectionOf(xsd:nonNegativeInteger xsd:nonPositiveInteger)*

2.3. Naudotojų sąsajos modelis

Paleidęs programa vartotojas išvys programos langą (28 pav.). Šiame lange vartotojas galės pasirinkti ontologijos failą, kurį yra norima transformuoti, direktoriją, kurioje bus išsaugotas transformuotas failas. Atsidarius programą išsaugojimo direktorijos ir transformavimo mygtukai yra neaktyvūs. Kuomet yra pasirenkamas ontologijos failas, aktyvuojamas išsaugojimo direktorijos pasirinkimo mygtukas. Atlikus ir šį pasirinkimą aktyvuojamas transformacijos paleidimo mygtukas. Tokių būdu yra apsisaugoma nuo klaidingos veiksmų sekos. Programos lango dizainas labai primityvus ir minimalistinis siekiant programos naudojimą vartotojui padaryti kuo paprastesnį.

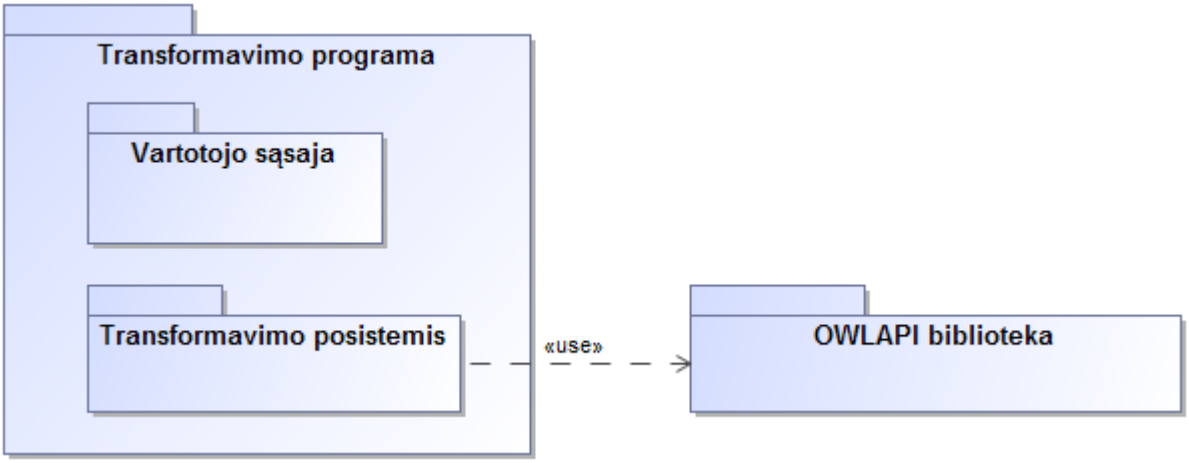


28 pav. Programos lango vartotojo sąsajos modelis

3. OWL ONTOLOGIJŲ TRANSFORMACIJOS Į XMI FORMATĄ REALIZACIJOS PROJEKTAS

3.1. Projektuojamo sprendimo loginė architektūra

Sistemos loginė architektūra (29 pav.) susideda iš vartotojo sąsajos, transformavimo posistemio ir *OWLAPI* bibliotekos. Projektas realizuojamas kaip atskira programa, todėl turės savo vartotojo sąsają. Transformavimo posistemis atlieka *OWL* ontologijų transformaciją į *XMI* formatą. Ontologijų nuskaitymui ir manipuliavimui naudojama *OWLAPI* biblioteka. Sprendimui įgyvendinti duomenų bazė nėra būtina, todėl ji naudojama nebus.

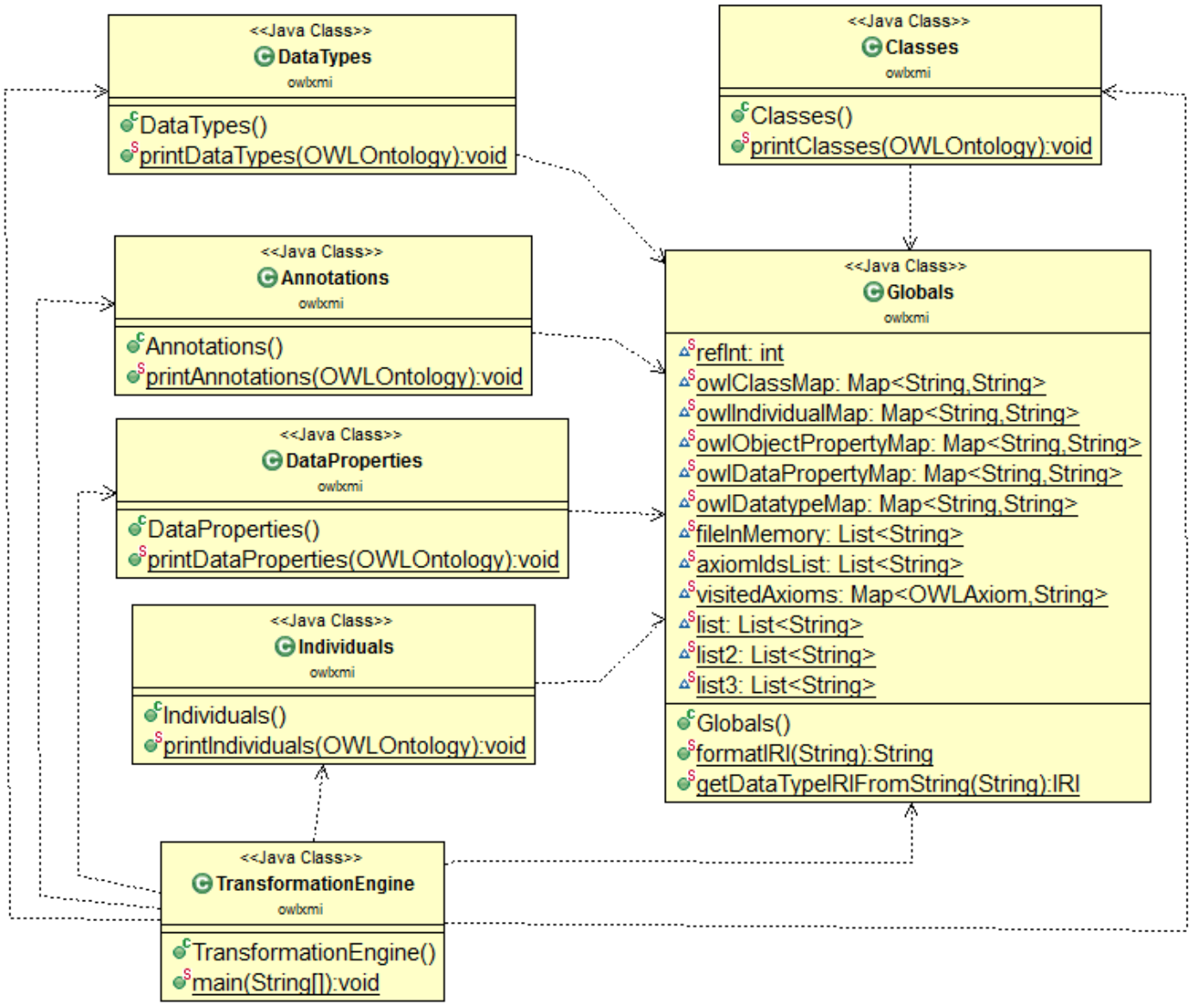


29 pav. Sistemos loginė architektūra

3.2. Projekto klasių modelis

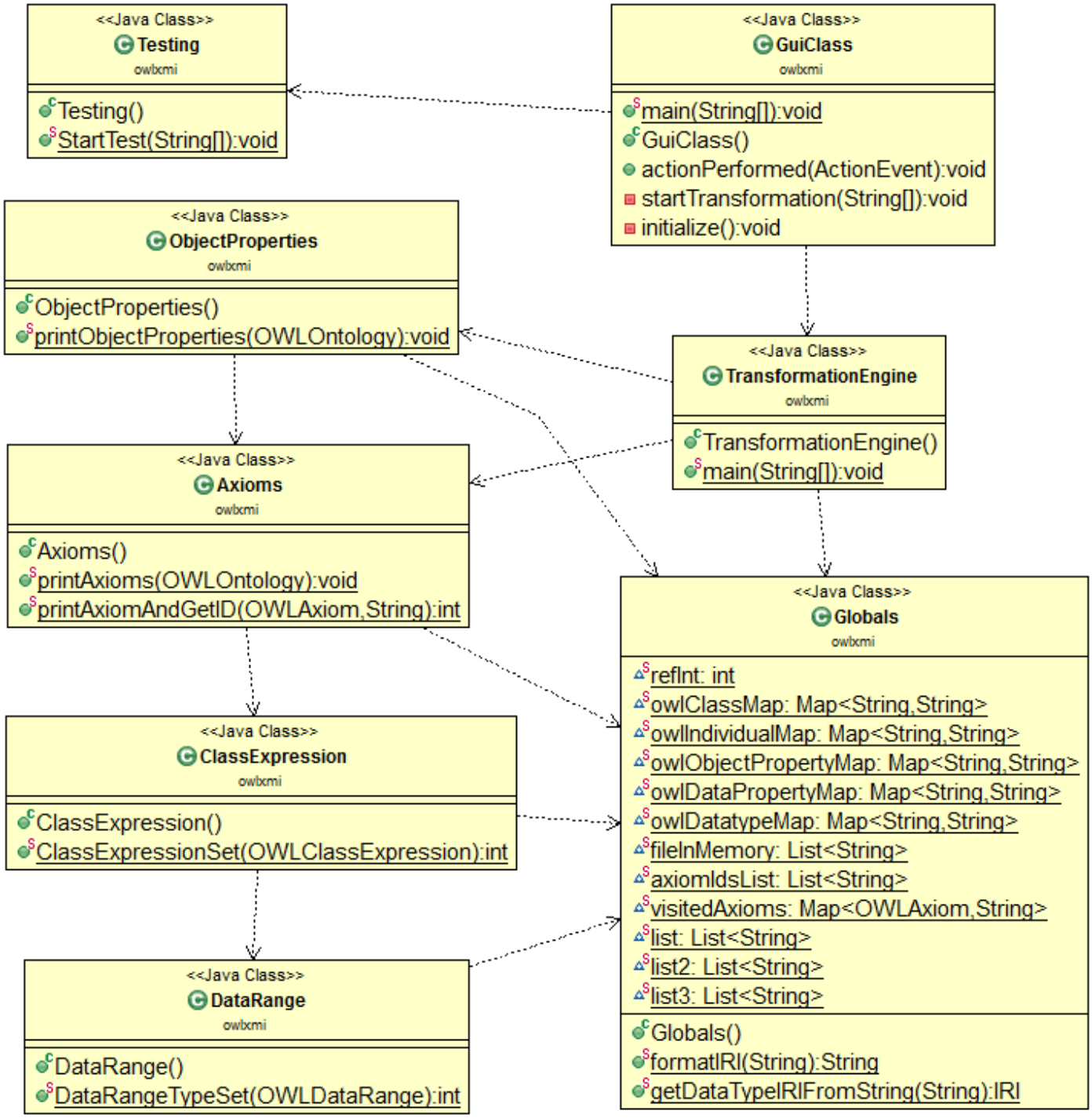
Projekto klasių modelį sudaro trylika klasių. Siekiant aiškumo ir dėl nemažo klasių skaičiaus projekto klasių modelis buvo išskaidytas į dvi dalis. Pirmoje dalyje (30 pav.) yra atvaizduotos septynios klasės: *DataTypes, Classes, Annotations, DataProperties, Individuals, TransformationEngine* ir *Globals*. Pastarosios dvi klasės abiejose klasių modelio dalyse yra bendros. Transformavimo programa *OWL* ontologijas į *XMI* formatą transformuoja etapais, taigi klasė *TransformationEngine* yra atsakinga už transformacijos vykdymo eigą. Šioje klasėje yra aprašyta transformavimo etapų vykdymo tvarka, todėl ši klasė kreipiasi į kitas klases, kurios turi metodus tam tikros dalies transformacijai atlikti. Klasėje *Globals* yra aprašyti sistemos veikimui reikalingi ir globaliai naudojami kintamieji. Šie kintamieji yra matomi ir gali būti manipuliuojami visose kitose projekto klasėse, todėl diagramoje galime matyti, kurios kitos klasės naudoja šioje klasėje esančius kintamuosius. *Classes* klasė turi metodą *printClasses(OWLOntology)*, kuris išveda visas klases

esančias ontologijoje *XMI* formatu į failą. Ontologija yra perduodama į metodą kaip metodo parametras. *DataTypes* klasė turi metodą *printDataTypes(OWLOntology)*, kuris išveda papildomai ontologijoje aprašytus duomenų tipus *XMI* formatu. Klasė *Annotations* turi metodą *printAnnotations(OWLOntology)*, kuris išveda visas ontologijoje esančias anotacijas *XMI* formatu. *DataProperties* klasė išveda ontologijoje esančias duomenų savybes *XMI* formatu. Klasės *Individuals* metodas *printIndividuals(OWLOntology)* išveda ontologijoje esančius individus *XMI* formatu.



30 pav. Klasių diagrama (pirma dalis)

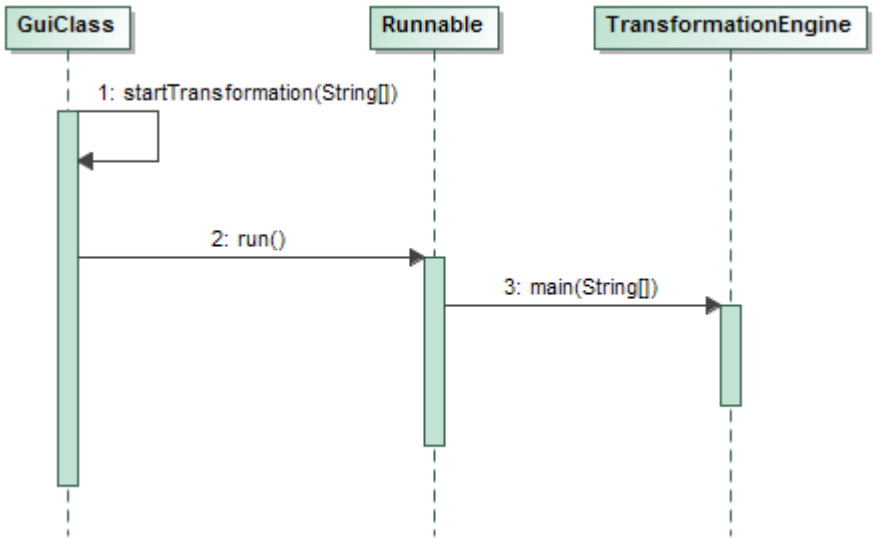
Antroje klasių diagramos dalyje (31 pav.) atvaizduotos dar šešios klasės, be jau anksčiau paminėtų *Globals* ir *TransformationEngine* klasių: *GuiClass, Testing, ObjectProperties, Axioms, ClassExpression, DataRange.* Kuomet atidaroma programa ji kreipiasi į *GuiClass* klasę, kadangi ji nustatyta kaip pradinė klasė ir turi *main(String[])* metodą. Šioje klasėje yra aprašyta vartotojo sąsaja ir turi kreipinį į transformavimo posistemę transformacijos proceso sužadinimui. *Testing* klasė yra skirta testavimo metodo realizacijai aprašyti ir naudojama tik testavimo metu. Klasė *ObjectProperties* turi metodą *printObjectProperties(OWLOntology)*, kuris išveda ontologijoje esančias objektų savybes *XMI* formatu. Klasė *Axioms* turi du metodus: *printAxioms(OWLOntology)* ir *printAxiomsAndGetID(OWLAxiom, String).* Pirmasis metodas skirtas transformuoti visas ontologijos aksiomas, antrasis – tik konkrečią aksiomą. *Axioms* klasė turi aprašytus visų aksiomų tipų transformacijos algoritmus. *ClassExpression* klasė atsakinga už konkrečių klasių išraiškų išvedimą *XMI* formatu. Klasės išraiška yra perduodama šios klasės metodui *ClassExpressionSet(OWLClassExpression)* kaip parametras, o metodas pagal perduotos klasės išraiškos tipą transformuoja ją į *XMI* formatą. *ClassExpression* klasėje yra aprašytą aštuoniolika klasės išraiškų tipų. *DataRange* klasė yra atsakinga už duomenų intervalų (angl. *dataranges*) išvedimą *XMI* formatu. Ši klasė turi metodą *DataRangeTypeSet(OWLDataRange)*, kuris pagal perduotą duomenų intervalo tipą atitinkamai jį ir transformuoja.



31 pav. Klasių diagrama (antra dalis)

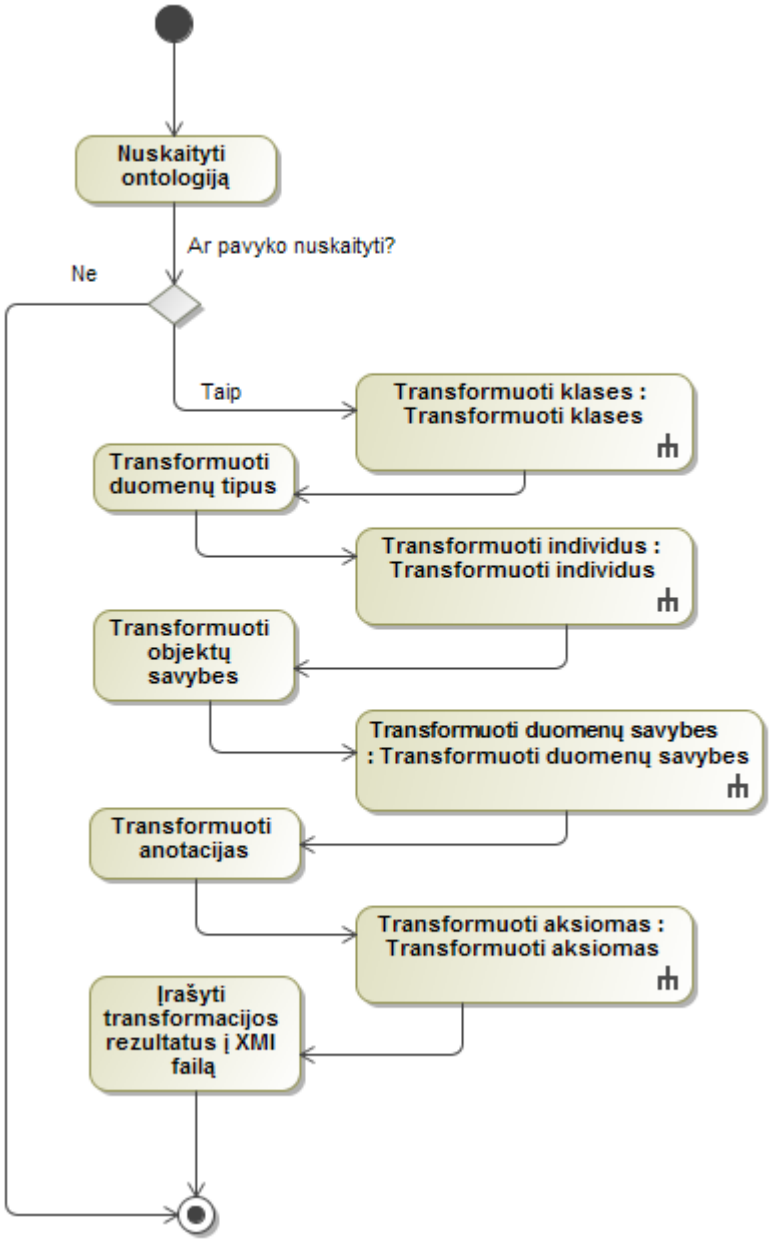
3.3. Projektuojamos programos elgsena

Projektuojant programos elgseną reikia sudaryti ją atvaizduojančias sekų diagramas. Skyriuje 1.7 buvo apibrėžta sistemos principinė naudojimosi schema (9 pav.): vartotojas turi pirmiausia pasirinkti ontologijos failą, direktoriją, kurioje išsaugos transformuotą failą, ir tuomet tik leidžiama atlikti transformaciją. Siekiant aiškumo diagramose transformacijos paleidimo procesui atvaizduoti buvo sudaryta atskira sekų diagrama (32 pav.). Šioje diagramoje esanti *GuiClass* klasė yra skirta vartotojo sąsajai ir paleidus programą yra vykdoma pati pirmoji. Kuomet vartotojas pasirenka ontologijos failą, rezultatų išsaugojimo vietą ir paspaudžia transformacijos mygtuką programa kreipiasi į *startTransformation(String[])* metoda. Metodui perduodamame parametre *String[]* saugomi sisteminiai „keliai“ iki ontologijos failo ir rezultatų išsaugojimo direktorijos. Kadangi transformacijos procesas gali užtrukti keletą minučių, programa transformacijos procesą vykdo asinchroniškai nuo pagrindinio programos vykdymo proceso. Tai atspindi sekų diagramoje kreipinys *run()* į sisteminę *Java* kalbos klasę *Runnable*, kuri kreipiasi į *TransformationEngine* klasės metodą *main(String[])* ir taip pradeda transformaciją. Asinchroninis transformacijos vykdymas „nesustingtina“ programos ir leidžia realizuoti indikaciją transformacijos proceso būsenai atvaizduoti.



32 pav. Transformacijos sužadinimo proceso sekų diagrama

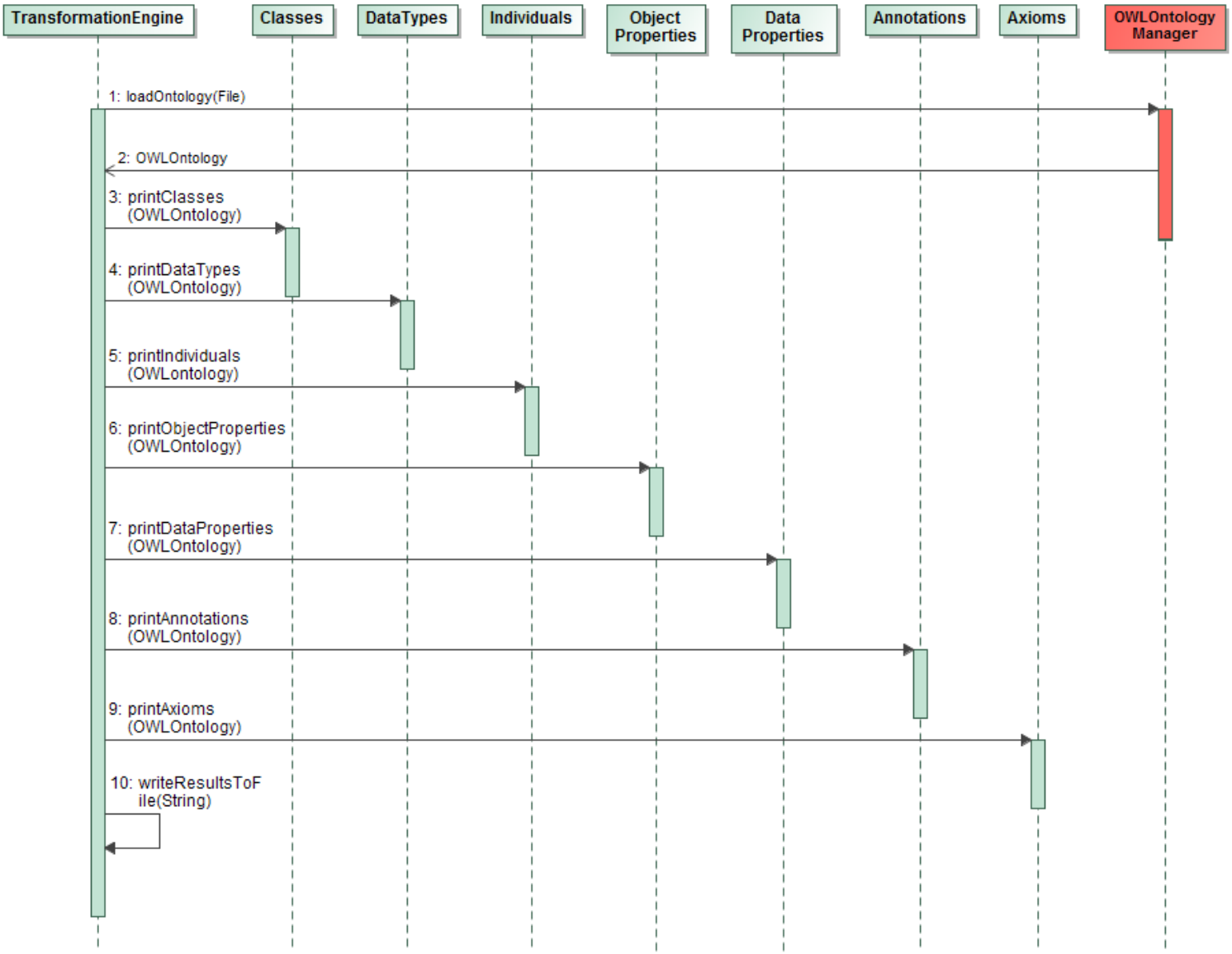
Transformacijos algoritmą atvaizduoja veiklos diagrama (33 pav.). Transformacija prasideda ontologijos nuskaitymu: jei nuskaitymas pavyksta vykdoma tolimesnė transformacija, jei ne – grąžinama klaida. Kadangi ontologijos esybės sudaro ontologijos pagrindą, jos turi būti transformuotos pačios pirmos. Transformavus esybes toliau seka aksiomų transformacija ir transformacijos rezultatų įrašymas į failą.



33 pav. Transformavimo operacijos veiklos diagrama

Atsižvelgiant į veiklos diagramą (33 pav.) buvo sudaryta transformavimo operacijos sekų diagrama (34 pav.). Pirmiausia yra užkraunamas ontologijos failas. Tai atliekama panaudojant *OWLAPI* bibliotekos klasės *OWLOntologyManager* metodą *loadOntology(File).* Nuskaičius ontologiją grąžinamas *OWLOntology* klasės objektas. Šis objektas kaip parametras yra perduodamas į visus, toliau sekančius transformavimo proceso metodus. Baigus transformaciją jos rezultatai, kurie yra saugomi kompiuterio atmintyje, yra surašomi į rezultatų failą nurodytoje vietoje. Kiekvienas transformuojamas elementas programoje yra pažymimas unikaliu ir tik jam priskirtu kodu (ID), kuris yra kaip nuoroda į tą elementą. Šie kodai padeda išlaikyti ryšį tarp elementų, kuris elementas kuriam priklauso.

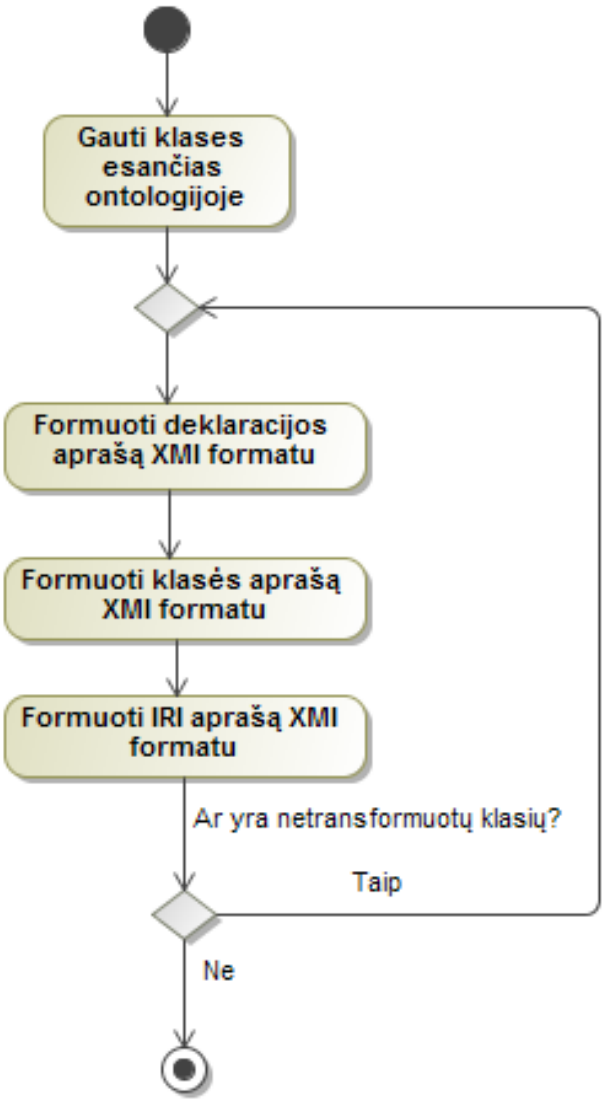
Toliau darbe bus detalizuojamos transformacijos operacijos dalys.



34 pav. Transformavimo operacijos sekų diagrama

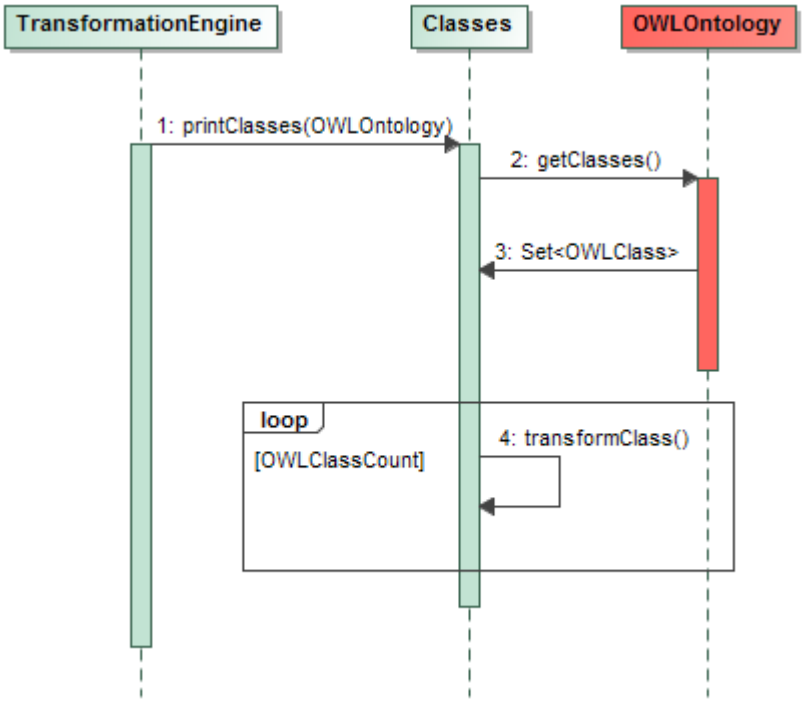
Diagramoje ***35 pav.*** yra atvaizduota detalizuota klasių transformavimo veiklos diagrama. Pirmiausia iš ontologijos yra išgaunamos joje esančios klasės, po to kiekviena klasė transformuojama individualiai. Transformacija vykdoma tol, kol nelieka netransformuotų klasių.

Klasių transformacijos vykdymo algoritmas yra identiškas ir duomenų tipų bei anotacijų transformacijai, todėl šios transformacijos detalizuojamos nebus.



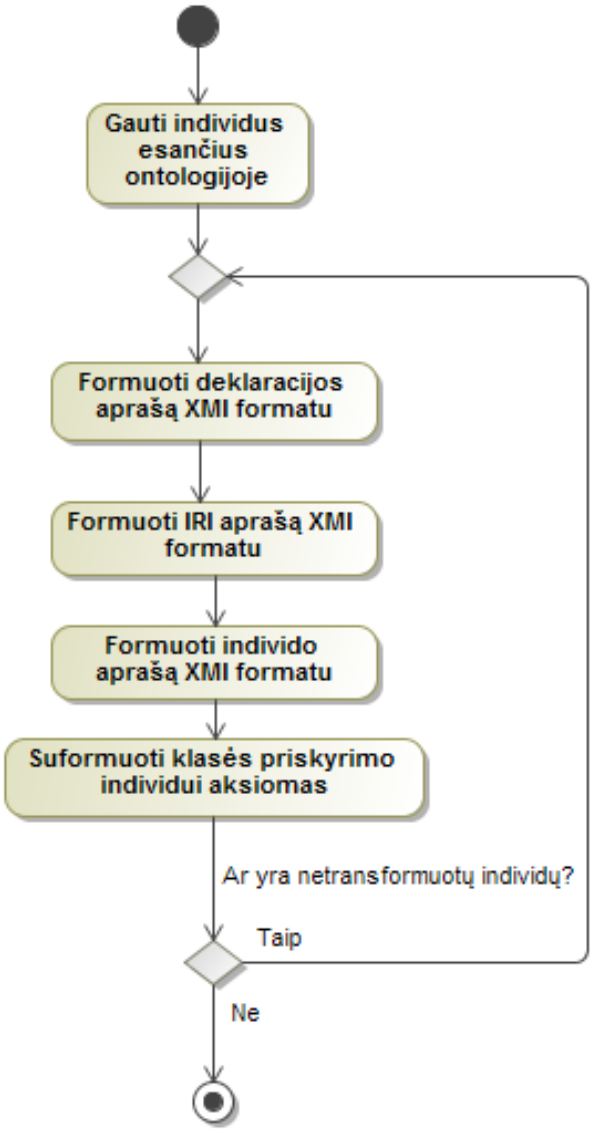
35 pav. Detalizuota klasių transformavimo veiklos diagrama

Klasių transformavimo operaciją detalizuoja jos sekų diagrama (36 pav.). Klasių gavimui iš ontologijos naudojame *OWLOntology* klasės metodą *getClasses()*. Transformavimui naudojamas ciklas, kurio pakartojimų skaičius priklauso nuo klasių skaičiaus, o ciklo viduje transformuojama kiekviena klasė individualiai.



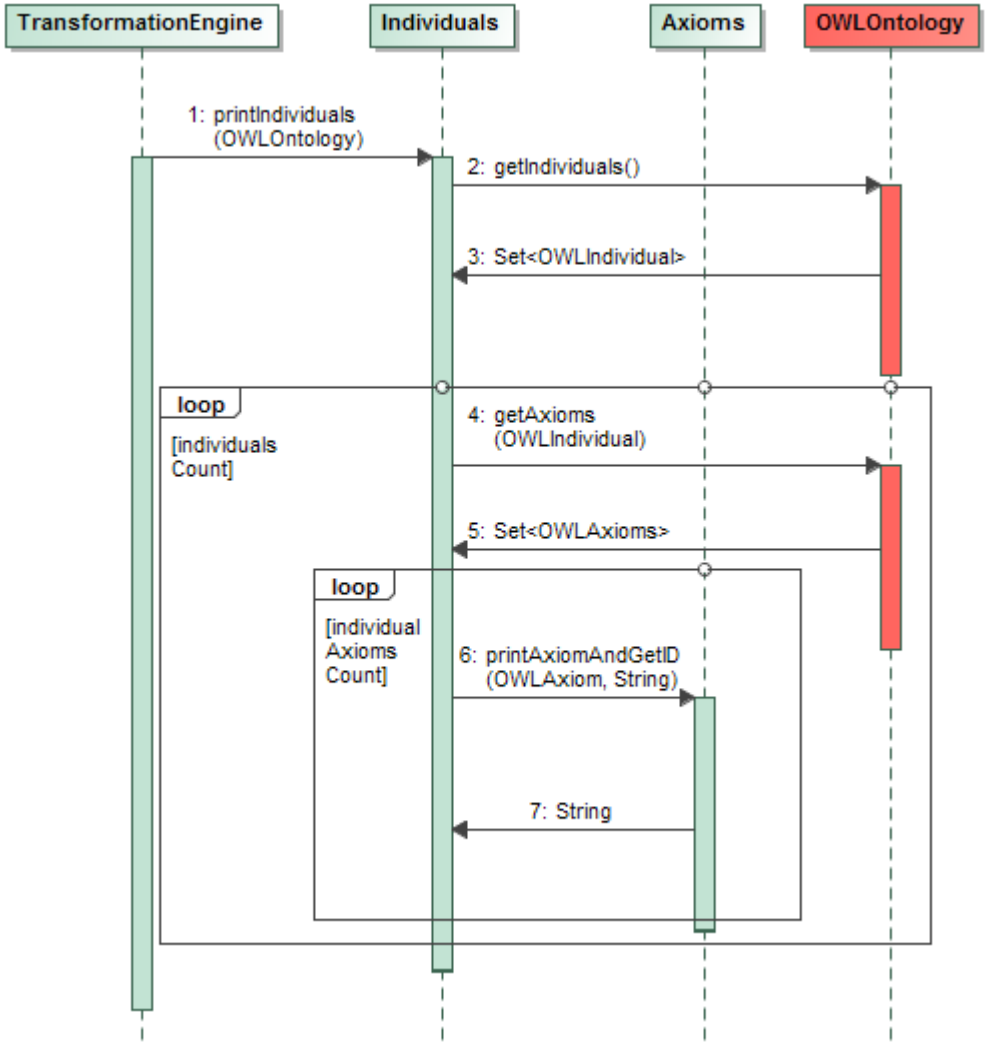
36 pav. Klasių transformavimo operacijos sekų diagrama

Individų transformavimo algoritmas (37 pav.) labai panašus į klasių transformavimo algoritmą, tačiau formuojant individų aprašus juose reikia nurodyti aksiomas, kurios priskiria individą klasėms. Norint tai padaryti reikia suformuoti ir pačias klasių priskyrimo aksiomas.



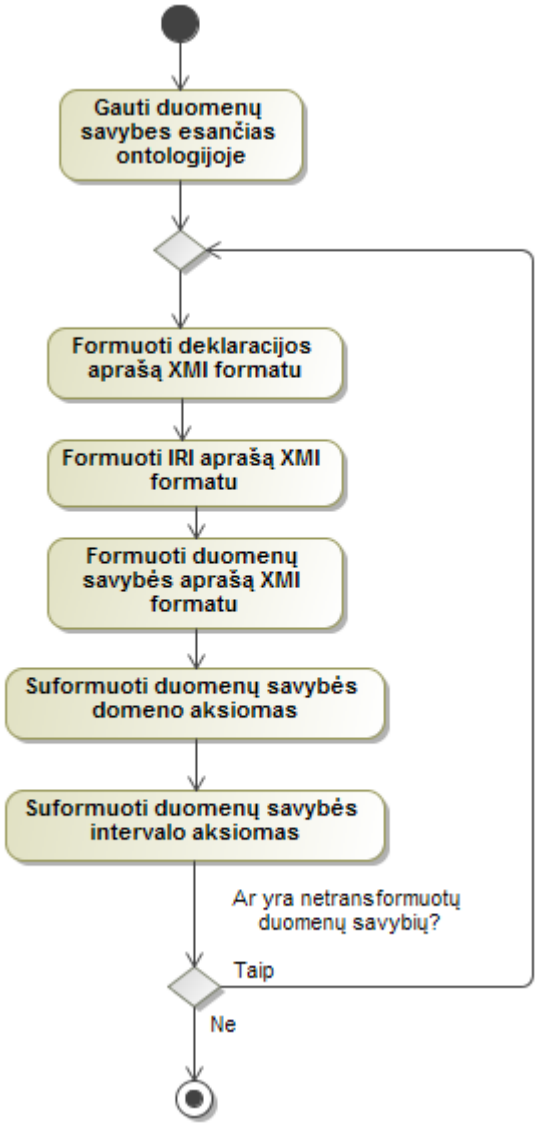
37 pav. Detalizuota individų transformavimo veiklos diagrama

Pagal pateiktą individų transformavimo algoritmą (37 pav.) sudaryta sekų diagrama (38 pav.). Individai gaunami per klasės *OWLOntology* metodą *getIndividuals()*, o jų transformacija atliekama panaudojant ciklą, kurio pakartojimų skaičius priklauso nuo individų kiekio. Kiekvieno ciklo metu yra gaunami individui priklausantys klasių priskyrimo aksiomų rinkiniai. Ciklo pagalba kiekviena aksioma yra transformuojama panaudojant klasės *Axioms* metodą *printAxiomAndGetID(OWLAxiom)*.



38 pav. Individų transformavimo operacijos sekų diagrama

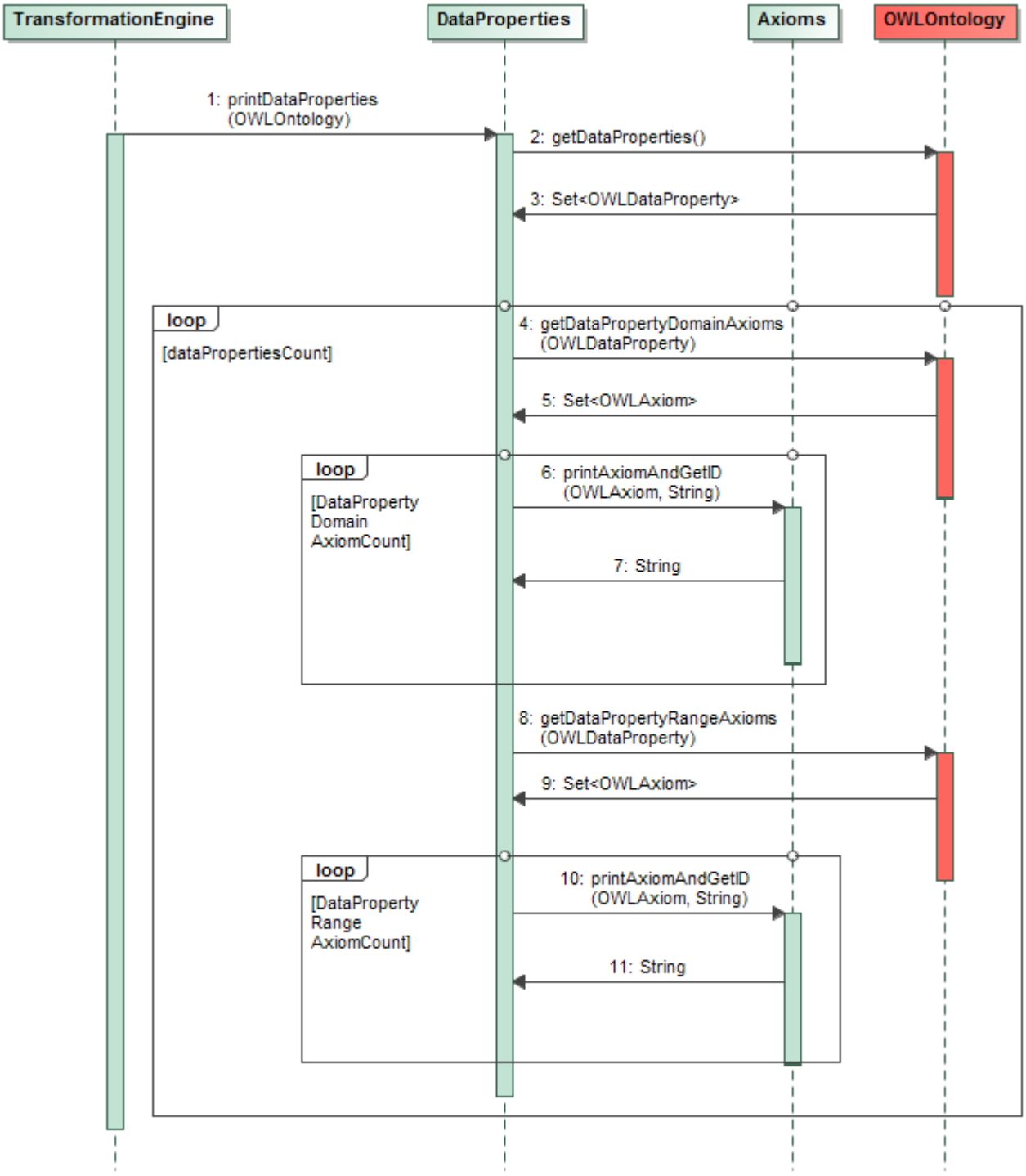
Diagramoje ***39 pav.*** pavaizduota detalizuota duomenų savybių transformacijos į *XMI* formatą veiklos diagrama. Pirmiausia yra gaunamas duomenų savybių rinkinys iš ontologijos. Kiekviena duomenų savybė apdorojama individualiai. Pirmiausia suformuojamas deklaracijos ir duomenų savybės *IRI* identifikatoriaus aprašai *XMI* formatu. Formuojant duomenų savybės aprašą reikia nurodyti aksiomas, kurios apibrėžia duomenų savybės domeną ir intervalą (angl. *range*), todėl šios transformavimo operacijos metu mes taip pat transformuojame ir konkrečiai duomenų savybei priklausančias domeno ir intervalo priskyrimo aksiomas. Analogiškas transformavimo operacijos algoritmas yra vykdomas ir transformuojant objektų savybes, todėl jų transformavimo algoritmas detalizuojamas nebus.



39 pav. Detalizuota duomenų savybių transformavimo veiklos diagrama

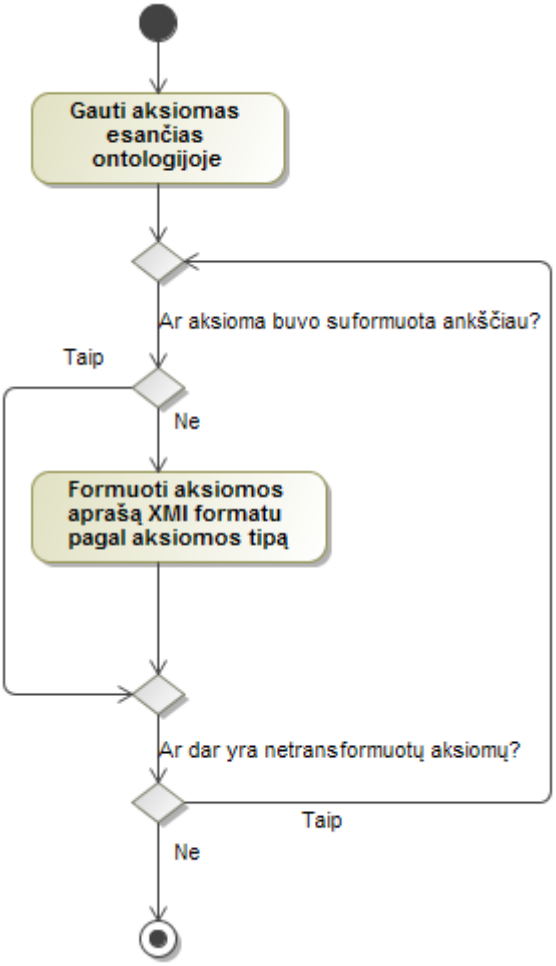
Sekų diagramoje (40 pav.) atvaizduotas duomenų savybių transformavimo operacija. Duomenų savybių gavimui kreipiamės į *OWLOntology* klasės metodą *getDataProperties()*. Metodas grąžina

duomenų savybių rinkinį *Set<OWLDataProperty>*. Panaudojant ciklą yra apdorojama kiekviena duomenų savybė. Transformacijos metu reikia suformuoti domeno ir intervalo priskyrimo duomenų savybei aksiomas. Pirmiausia gaunami aksiomų rinkiniai, tada kiekviena aksioma yra perduodama kaip parametras *Axioms* klasės metodui *printAxiomAndGetID(OWLAxiom, String),* kuris ir atlieka aksiomos formavimą. *String* tipo parametras yra nuoroda (ID) į duomenų savybę, kuri įrašoma į aksiomos aprašą jos formavimo metu.



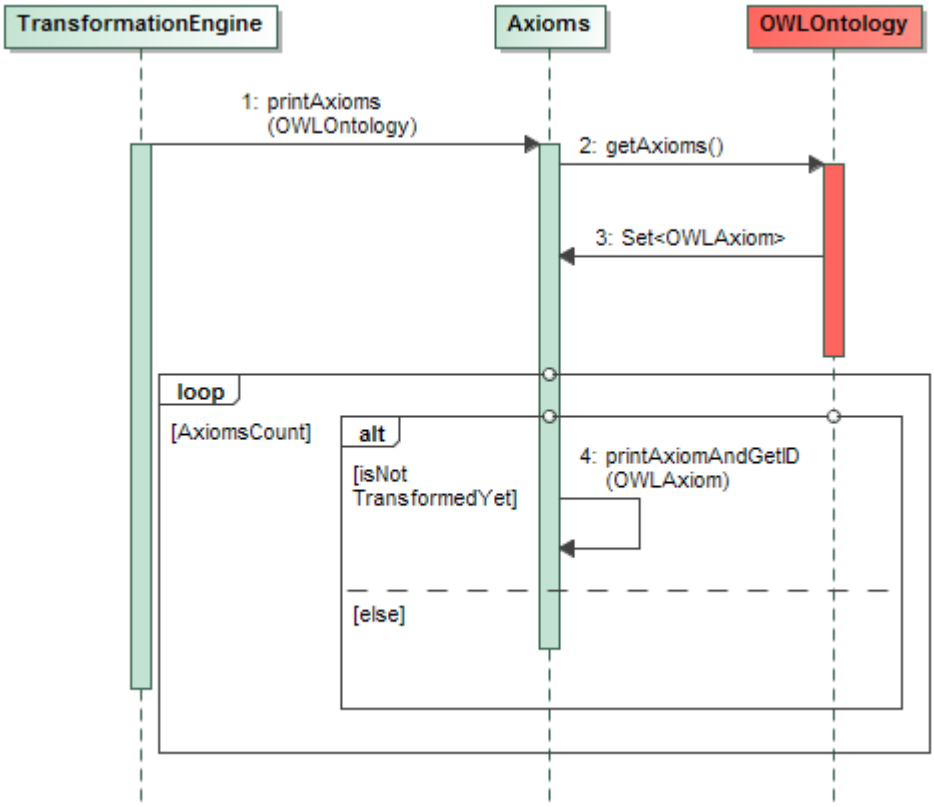
40 pav. Duomenų savybių transformavimo operacijos sekų diagrama

Detalizuota aksiomų transformavimo operacija atvaizduota veiklos diagramoje (41 pav.). Pirmiausia yra gaunamos ontologijos aksiomos. Kiekviena aksioma transformuojama individualiai, tačiau prieš pradedant aksiomos transformavimą ji yra patikrinama jau transformuotų aksiomų sąraše, kadangi kai kurios aksiomos yra transformuojamos esybių transformacijos metu. Jei aksioma nebuvo transformuota, ją transformuojame. Diagramoje detalizavimas nėra žemo lygio dėl to, jog dauguma aksiomų yra transformuojamos skirtingai ir vienoje veiklos diagramoje aksiomų transformavimo algoritmo atvaizduoti negalima.



41 pav. Detalizuota aksiomų transformavimo veiklos diagrama

Sekų diagramoje (42 pav.) atvaizduota aksiomų transformavimo operacija. Aksiomų gavimui yra kreipiamasi į *OWLOntology klasę.* Kiekviena aksioma ciklo pagalba transformuojama individualiai. Kiekvienos iteracijos metu aksioma yra tikrinama ar nėra transformuotų aksiomų sąraše, jei ne – transformuojama.



42 pav. Aksiomų transformavimo operacijos sekų diagrama

Kaip ir buvo minėta ankščiau šiame darbe (skyrelis 2.2) aksiomų yra trisdešimt septynios. Transformuojant aksiomas yra naudojamos klasių išraiškos, kurių yra aštuoniolika bei duomenų intervalų tipai, kurių yra šeši. Aksiomos yra sudarytos iš esybių, klasių išraiškų, duomenų intervalų, tad galimų aksiomos elementų kombinacijų kiekis labai didelis. Dėl šios priežasties visų aksiomų transformavimo operacijų detalizuoti nepavyks, todėl bus aprašyta keletas konkrečių aksiomų transformacijos pavyzdžių siekiant parodyti principinį aksiomų transformavimo algoritmą.

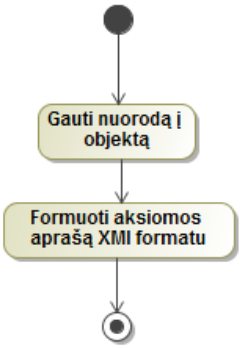
Aksiomų transformavimo operacijos pavyzdžiai

Aksiomų pavyzdžiai bus pateikti *OWL* kalbos funkcinės sintaksės išraiška.

1) Aksiomų transformavimo operacijos pavyzdys Nr. 1

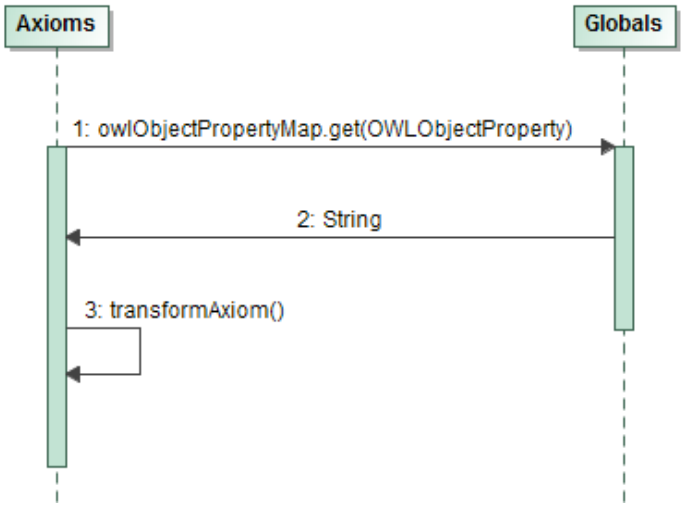
Aksioma: *FunctionalObjectProperty(<ns:Photo_equipment#has__camera_model>)*

Ši aksioma pažymi objekto savybę *has__camera_model* kaip funkcinę savybę. Pagal *OWL2* ontologijos meta modelį (22 pav.) žinome, jog aksioma turi ryši su viena objekto savybe. Taigi, aksiomos transformavimo operacija buvo atvaizduota veiklos diagramoje 43 pav. Pirmiausia yra gaunama nuoroda (objekto savybės ID) į objekto savybę, kurią aprašo aksioma, po to suformuojamas aksiomos aprašas.



43 pav. Aksiomos transformavimo pavyzdžio Nr. 1 operacijos veiklos diagrama

Sekų diagramoje (44 pav.) matome, jog nuorodos (ID) į objekto savybę gavimui kreipiamasi į klasės *Globals* objektų savybių sąrašą *owlObjectPropertyMap* panaudojant *get* metodą. Rezultatas gražinamas kaip *String* eilutė, kuri yra panaudojama formuojant aksiomos aprašą.

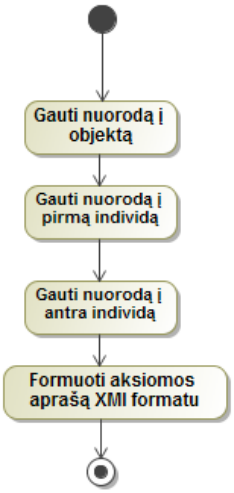


44 pav. Aksiomos transformavimo pavyzdžio Nr. 1 operacijos sekų diagrama

2) Aksiomų transformavimo operacijos pavyzdys Nr. 2

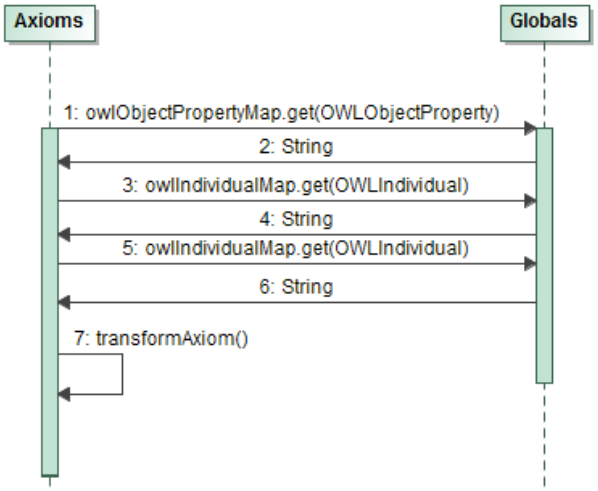
Aksioma: *ObjectPropertyAssertion(<ns:Photo_equipment#has_produced__product> <ns:Photo_equipment#Jim> <ns:Photo_equipment#Photo_1>)*

Aksioma sudaryta iš dviejų individų ir vienos objekto savybės bei nurodo, kad individas *Jim* yra *has_produced__product* individą *Photo_1*. Aksiomos transformavimo algoritmas labai panašus į algoritmą, aprašytą pirmame pavyzdyje. Pagrindinis skirtumas yra tai, jog aksiomai sudaryti vienos objekto savybės ir papildomai dviejų individų. Transformavimo operacija atvaizduota veiklos diagramoje ***45 pav***.



45 pav. Aksiomos transformavimo pavyzdžio Nr. 2 operacijos veiklos diagrama

Sekų diagramoje (46 pav.) matome, jog nuorodos (ID) į objekto savybę gavimui kreipiamasi į klasės *Globals* objektų savybių sąrašą *owlObjectPropertyMap* panaudojant *get* metodą. Nuorodų į individus gavimui naudojamas individų sąrašo *owlIndividualMap* metodas *get*. Atlikus šiuos veiksmus suformuojamas aksiomos aprašas su nuorodomis į objekto savybę ir individus.



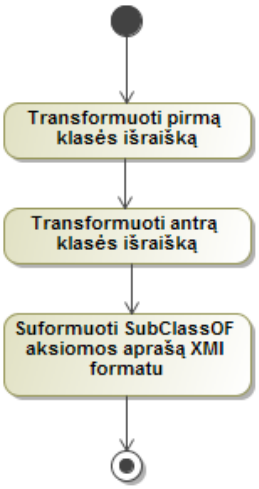
46 pav. Aksiomos transformavimo pavyzdžio Nr. 2 operacijos sekų diagrama

3) Aksiomų transformavimo operacijos pavyzdys Nr. 3

Aksioma: *SubClassOf(<ns:Photo_equipment#photo_camera> ObjectExactCardinality(2 <ns:Photo_equipment#contains__battery> <ns:Photo_equipment#battery>))*

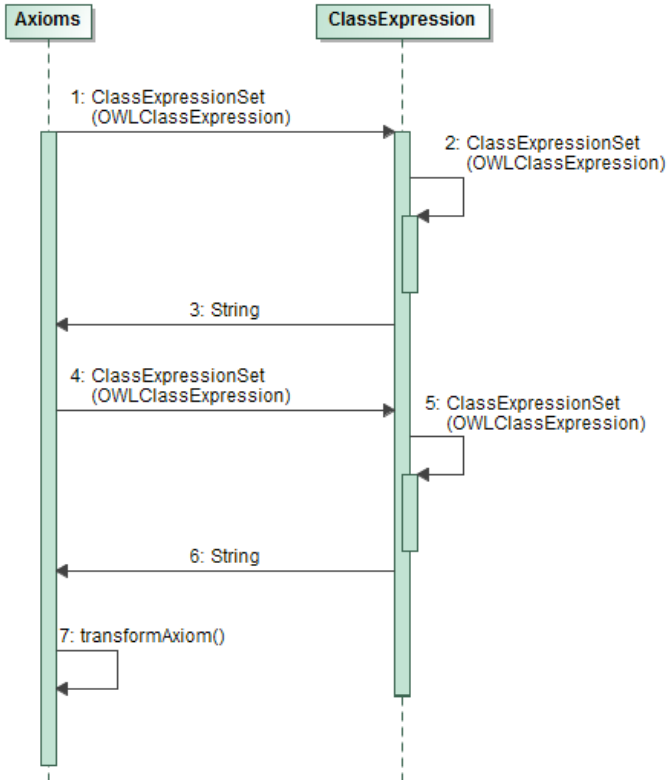
Ši aksioma nurodo, kad klasės *photo_camera* individo objektinė savybė *contains__battery* turi ir tik turi *2* klasės *battery* individus, kitais žodžiais tariant: foto kamera yra sudaryta tik ir tik iš dviejų baterijų. Pagal *OWL2* meta modelį (20 pav.) matome, jog *SubClassOf* aksioma turi ryšį su dvejomis klasių išraiškomis. Kaip jau ir minėta šiame darbe, klasių išraiškų tipų yra aštuoniolika. Yra tokių klasės išraiškos tipų, kurios turi ryšį su kitu klasės išraiškos tipu, todėl norint transformuoti tokią klasės išraišką yra būtina daryti rekursinę funkciją, kuri eitų per klasės išraiškas gilyn tol, kol būtų pasiektas „šakos“ galas, klasės išraiška, kuri neturi kreipinio į rekursinę funkciją. Kuomet yra pasiekiama paskutinė klasės išraiška, ji yra suformuojama, o jos unikalus kodas (nuoroda į ją) yra gražinama kaip rekursinės funkcijos rezultatas. Tuomet prieš paskutinė klasės išraiška, kuri paskutinį kartą iškvietė rekursinę funkciją, įsirašo rekursinės funkcijos grąžintą rezultatą (nuorodą į paskutinę klasės išraišką) ir yra suformuojama. Tokiu principu prieš paskutinėje klasės išraiškoje yra nuoroda į paskutiniąją, o prieš prieš paskutinė turi nuorodą į prieš paskutinę ir t.t. Tai tarsi grandinė, sudaryta iš klasės išraiškų.

Tokią aksiomos transformavimo operaciją vaizduoja veiklos (47 pav.) diagrama.



47 pav. Aksiomos transformavimo pavyzdžio Nr. 3 operacijos veiklos diagrama

Sekų diagramoje (48 pav.) atvaizduota šio pavyzdžio aksiomos transformavimo operacija. Kadangi *SubClassOf* aksioma sudaryta iš dviejų klasės išraiškų, todėl turime transformuoti abi klasės išraiškas. Iškvietus *ClassExpressionSet* metodą pagal klasės išraiškos tipą yra vykdoma išraiškos transformacija. Jei transformuojama klasės išraiška turi ryšį į kita klasės išraišką, tuomet iškviečiamas *ClassExpressionSet* metodas, taip gaunama rekursija. Transformavus klasių išraiškas, suformuojamas aksiomos aprašas *XMI* formatu.



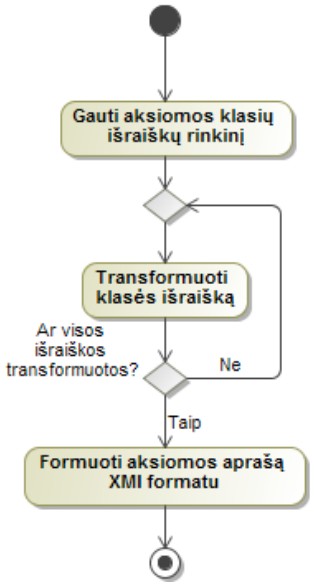
48 pav. Aksiomos transformavimo pavyzdžio Nr. 3 operacijos sekų diagrama

4) Aksiomų transformavimo operacijos pavyzdys Nr. 4

Aksioma: *EquivalentClasses(<ns:Photo_equipment#moderate_weight_camera> ObjectIntersectionOf(DataSomeValuesFrom(<ns:Photo_equipment#camera_weight> DatatypeRestriction(<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> xsd:minExclusive "400"^^xsd:integer xsd:maxInclusive "800"^^xsd:integer)) <ns:Photo_equipment#camera_by_weight>))*

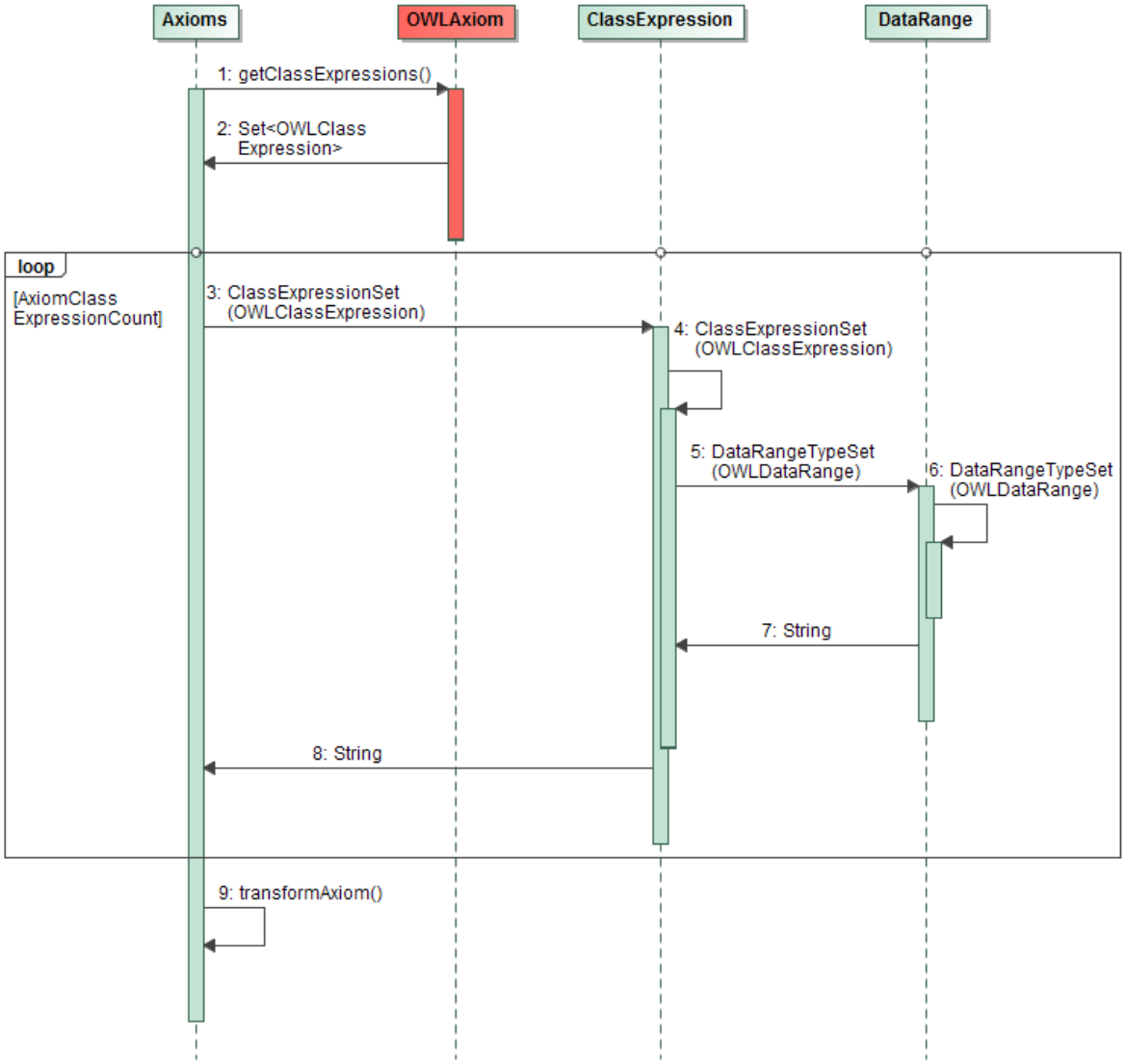
Pagal *OWL2* meta modelį (20 pav.) *EquivalentClasses* aksioma sudaryta iš bent dviejų klasių išraiškų, kurių kiekis gali būti *n*. Šios aksiomos transformavimo algoritmas labai panašus į prieš tai pavyzdyje Nr. 3 nagrinėta aksiomos transformavimo algoritmą. Pagrindinis skirtumas yra tai, jog klasės išraiškų skaičius nėra apibrėžtas.

Prieš transformuojant šiame pavyzdyje pateiktą aksiomą reikia transformuoti jos klasių išraiškas. Pirmiausia yra gaunamas klasių išraiškų rinkinys, o po to kiekviena išraiška transformuojama individualiai. Baigus išraiškų transformavimą, suformuojamas aksiomos aprašas *XMI* formatu. Transformavimo operacijos algoritmas atvaizduotas veiklos diagramoje (49 pav.).



49 pav. Aksiomos transformavimo pavyzdžio Nr. 4 operacijos veiklos diagrama

Sekų diagramoje (50 pav.) pirmiausia vykdomas klasės išraiškų rinkinio išgavimas iš aksiomos panaudojant *OWLAxiom* klasės metodą *getClassExpressions()*. Kiekviena klasės išraiška transformuojama individualiai naudojant ciklą. Ciklo pakartojimų skaičius lygus klasės išraiškų rinkinio dydžiui. Transformuojant klasės išraišką yra kviečiama rekursinė funkcija *ClassExpressionSet(OWLClassExpression)*. Pavyzdyje nagrinėjama aksioma turi klasės išraiškos tipą *DataSomeValuesFrom*, kuris skirtas duomenų apribojimams sudaryti. Pagal *OWL2* meta modelį (17 pav.) klasės išraiška *DataSomeValuesFrom* turi ryšį su vienu duomenų intervalo tipu. Taigi, transformuojant šią klasės išraiška atliekamas kreipinys į *DataRange* klasės metodą *DataRangeTypeSet(OWLDataRange)* duomenų intervalui transformuoti. Kadangi duomenų intervalo tipų yra šeši, o kai kurie iš jų turi ryšius į kitus duomenų intervalo tipus, todėl ir šiuo atveju yra būtina naudoti rekursinę funkciją duomenų intervalams transformuoti. Transformavus klasių išraiškas yra transformuojama aksioma į *XMI* formatą.



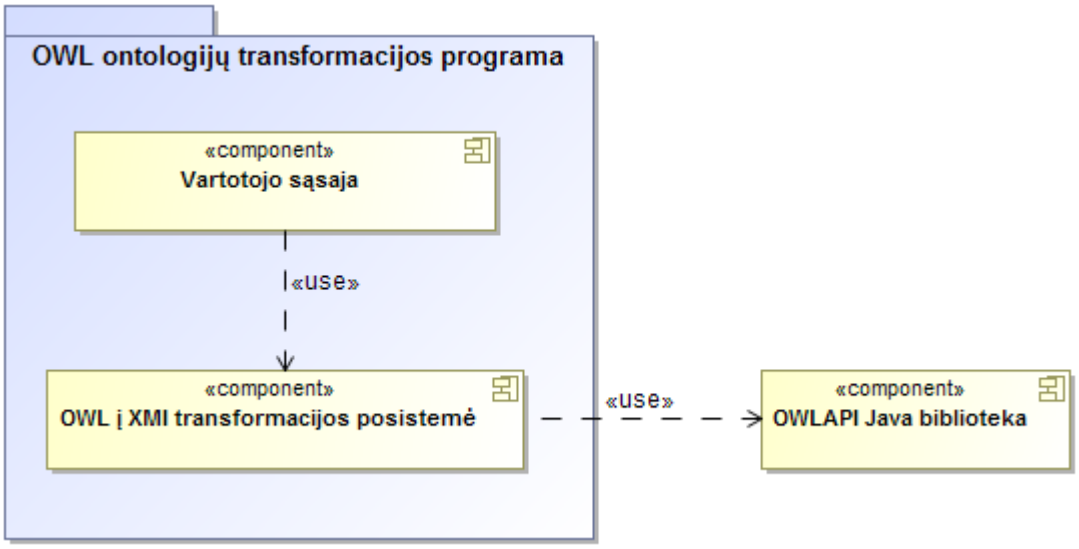
50 pav. Aksiomos transformavimo pavyzdžio Nr. 4 operacijos sekų diagrama

Taigi, detalizavus kelių aksiomų transformavimo į *XMI* formatą operacijos algoritmus buvo pademonstruoti pagrindiniai aksiomų transformavimo algoritmų principai, kurie yra panašūs kituose algoritmuose, transformuojančiuose kitų tipų aksiomas.

3.4. Realizacijos modelis

Realizuota transformacijos programa susideda iš trijų komponentų: vartotojo sąsajos, *OWL* į *XMI* transformacijos posistemės ir *OWLAPI Java* bibliotekos. Vartotojo sąsajos komponentas reikalingas pradiniams duomenims įvesti ir transformacijos vykdymo būsenos atvaizdavimui. Vartotojo sąsaja leidžia atlikti tokius veiksmus kaip pasirinkti pradinį ontologijos failą transformacijai, nurodyti rezultatų išvedimo direktoriją, bei pradėti transformaciją. Transformacijos komponentas reikalingas transformacijai atlikti. Šis komponentas naudodamas išorinę *OWLAPI* biblioteką nuskaito OWL ontologijas bei transformuoja jas į XMI formatą.

Komponentų diagramą (51 pav.) parodo kaip sistemos komponentai sąveikauja tarpusavyje.

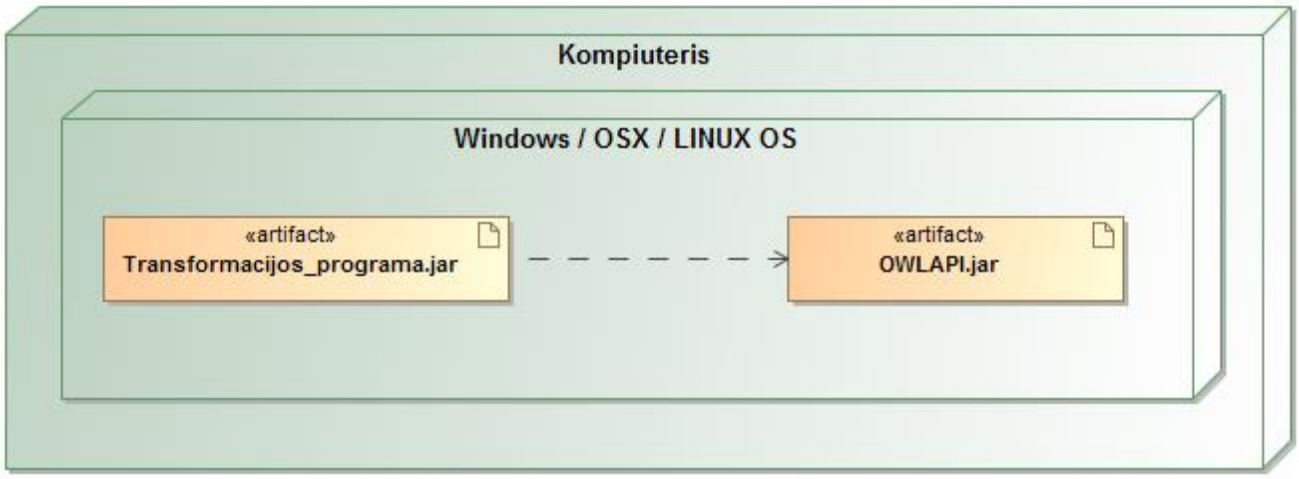


51 pav. Projekto komponentų diagrama

Progra,a realizuojama panaudojant Java programavimo kalbą, todėl norint programą paleisti kompiuteryje yra būtina įsidiegti *Java Runtime Environment (JRE)* programinę įrangą. JRE tai programinės įrangos paketas, kuriame yra visi reikalingi komponentai norint paleisti programą parašytą Java programavimo kalba. Ši PĮ yra nemokama ir ją galima atsisiųsti iš www.*java.com* interneto svetainės. Pagrindinis Java kalbos privalumas, jog šia kalba parašytos programos be didesnių keblumų gali veikti keliose operacinėse sistemose: *Windows OS, OSX, Linux OS.*

Sistemos diegimas yra labai paprastas, įsikeliate programą į savo kompiuterį ir iš karto galite ja naudotis. Programa susideda iš dviejų dalių: pačios programos failo *Transformacijos_programa.jar* ir bibliotekų direktorijos *Transformacijos_programa_lib*, kurioje yra patalpintos programos veikimui reikalingos bibliotekos. Tiek failas, tiek bibliotekų direktorija privalo būti toje pačioje vietoje, kitaip programa negalės pasiekti bibliotekų. *Java* kalbos programavimo įrankiai suteikia galimybę visą programą ir jos veikimui reikalingas bibliotekas sutalpinti į vieną *.jar* failą, tačiau tai smarkiai įtakoja programos veikimo spartą (sparta sulėtėja nuo penkių iki dešimties kartų), todėl buvo nuspręsta bibliotekas iškelti į atskirą direktoriją.

Sistemos diegimo modelis (52 pav.) parodo kaip sistema fiziškai turi būti patalpinta į jos vykdymo aplinką (šiuo atveju - kompiuterį).



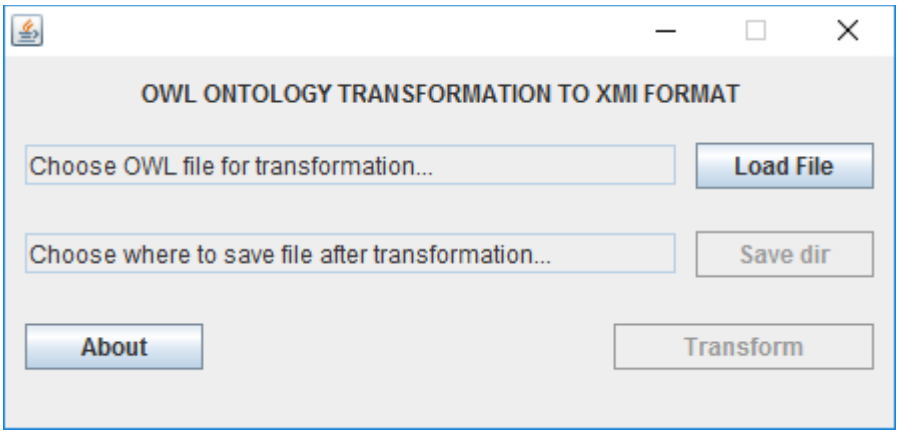
52 pav. Projektuojamos sistemos diegimo modelis

4. SPRENDIMO REALIZACIJA IR TESTAVIMAS

4.1. Sprendimo realizacijos ir veikimo aprašas

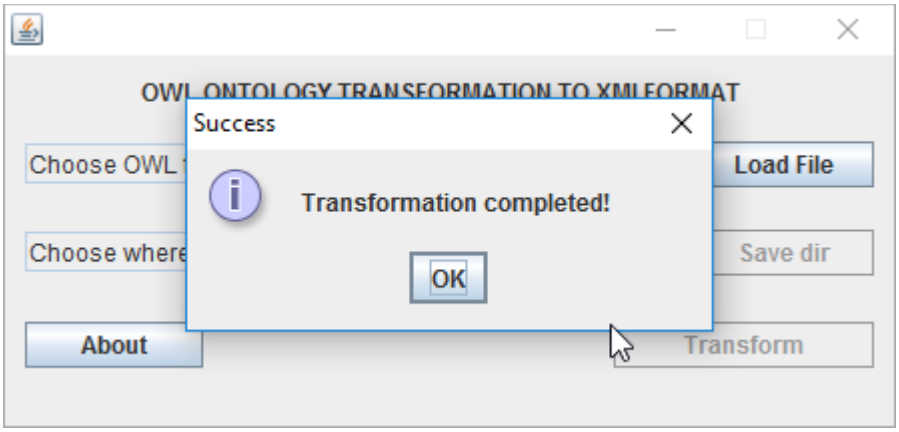
OWL ontologijų transformavimo į *XMI* formatą programa realizuota panaudojant Java programavimo kalbą bei *OWLAPI* biblioteką, skirtą šiai programavimo kalbai. Programa veikia *Windows, Linux* ir *OSX* operacinėse sistemose, tačiau norint naudotis programa operacinėje sistemoje turi būti įdiegtas *Java Runtime Environment (JRE)* programinės įrangos paketas.

Realizuota vartotojo sąsaja (53 pav.) yra labai paprasta, kuria naudotis yra ypač paprasta: pasileidžiate programą, pasirenkate ontologijų failą, pasirenkate direktoriją, kurioje išsaugoti rezultatų failą ir spaudžiate transformacijos mygtuką.



53 pav. OWL ontologijų transformacijos į XMI formatą programos vartotojo sąsaja

Po sėkmingai atliktos transformacijos programa parodo tai patvirtinantį pranešimą, o nurodytoje išsaugojimo vietoje atsiranda *.xmi* formato failas su transformuota ontologija.



54 pav. OWL ontologijų transformacijos į XMI formatą programos pranešimas apie sėkmingą transformaciją

4.2. Testavimo modelis, duomenys, rezultatai

Sukurto algoritmo veikimo teisingumas turi būti patikrintas panaudojus jau sukurtą programą, kuri atlieka atvirkštinę transformacija, t. y. transformuoja *XMI* dokumentą į *OWL* ontologijų aprašus. Atlikus atvirkštinę transformaciją, jei mūsų algoritmas veikia teisingai, turime gauti tokį patį ontologijos *OWL* aprašą, kaip ir prieš transformaciją, be to ontologija turi būti sėkmingai atidaryta *Protege* ontologijų kūrimo įrankyje.

Atvirkštinei transformacijai atlikti panaudosime *SBVR* žodynų ir taisyklių konvertavimo į *OWL2* ontologijas programos (*s2o*) komponentą *„Transform OWL 2 XMI to OWL 2“* (8 pav.). Šio komponento veikimas pagrįstas *XSLT* transformavimo taisyklėmis ir jis transformuoja OWL2 XMI į OWL2 ontologijas.

Veiksmų seka, kurią naudosime atliekant testavimą atvaizduota ***55 pav***. Pirmiausia atliekama *OWL2* ontologijos transformacija panaudojant realizuotą sprendimą. Gautam rezultatų *XMI* failui atliekama atvirkštinė transformacija panaudojant „*Transform OWL 2 XMI to OWL 2“* komponentą. Gautą *OWL2* ontologijos failą lyginame su pradiniu ontologijos failu, kuris buvo naudojamas transformacijai atlikti. Palyginimas atliekamas lyginant pradinį ontologijos failo kodą su ontologijos failo, gauto po atvirkštinės transformacijos, kodu naudojant tekstinį redaktorių.

55 pav. Testavimo atlikimo veiksmų seka

Testavimui buvo pasirinkta *Photo equipment* ontologija, kuri pateikta kaip vienas iš pavyzdžių *s2o* programos svetainėje [11].

Palyginimo rezultatai pateikti lentelėje (*25 lentelė),* kurioje kiekvieno tipo pradinės ontologijos elementas buvo palygintas su elementu, gautu po atvirkštinės transformacijos. Iš lentelėje pateiktų palyginimų galima daryti išvadą, jog realizuotas sprendimas veikia teisingai.

25 lentelė. Atliktos transformacijos rezultatų palyginimas

Pradinis ontologijos elementas
Declaration(AnnotationProperty(<ns:Photo_equipment#label_sbvr>))
Gautas elemento XMI aprašas
<owl2:AnnotationProperty entityIRI="/1441" declaration="/1440"/>
<owl2:Declaration ontology="/0" entity="/1439"/>
<owl2:IRI lexicalValue="ns:Photo_equipment#label_sbvr"/>
Atvirkštinės transformacijos rezultato elementas
Declaration(AnnotationProperty(<ns:Photo_equipment#label_sbvr>))

Pradinis ontologijos elementas

AnnotationAssertion(<http://www.w3.org/2000/01/rdf-schema#label> <ns:Photo_equipment#Jim> "Jim"@en)

Gautas elemento XMI aprašas

<owl2:AnnotationProperty entityIRI="/1344"/>

<owl2:IRI lexicalValue="http://www.w3.org/2000/01/rdf-schema#label"/>

<owl2:AnnotationAssertion ontology="/0" annotationProperty="/1343" annotationSubject="/142" annotationValue="Jim" annotationLanguage="@en"/>

Atvirkštinės transformacijos rezultato elementas

AnnotationAssertion(<http://www.w3.org/2000/01/rdf-schema#label> <ns:Photo_equipment#Jim> "Jim"@en)

Pradinis ontologijos elementas

AsymmetricObjectProperty(<ns:Photo_equipment#has__accessory>)

Gautas elemento XMI aprašas

<owl2:AsymmetricObjectProperty ontology="/0" objectPropertyExpression="/199"/>

Atvirkštinės transformacijos rezultato elementas

AsymmetricObjectProperty(<ns:Photo_equipment#has__accessory>)

Pradinis ontologijos elementas

ClassAssertion(<ns:Photo_equipment#person> <ns:Photo_equipment#Jim>)

Gautas elemento XMI aprašas

<owl2:ClassAssertion ontology="/0" individual="/142" classExpression="/51"/>

Atvirkštinės transformacijos rezultato elementas

ClassAssertion(<ns:Photo_equipment#person> <ns:Photo_equipment#Jim>)

Pradinis ontologijos elementas

DataPropertyAssertion(<ns:Photo_equipment#first_name> <ns:Photo_equipment#G._Gudas> "Gytis"^^<http://www.w3.org/2001/XMLSchema#string>)

Gautas elemento XMI aprašas

<owl2:DataPropertyAssertion ontology="/0" targetValue="/1483" sourceIndividual="/170" dataPropertyExpression="/430"/>

Atvirkštinės transformacijos rezultato elementas

DataPropertyAssertion(<ns:Photo_equipment#first_name> <ns:Photo_equipment#G._Gudas> "Gytis"^^<http://www.w3.org/2001/XMLSchema#string>)

Pradinis ontologijos elementas

DataPropertyDomain(<ns:Photo_equipment#bag_weight> <ns:Photo_equipment#camera_bag>)

Gautas elemento XMI aprašas

<owl2:DataPropertyDomain ontology="/0" dataPropertyExpression="/444" domain="/129"/>

Atvirkštinės transformacijos rezultato elementas

DataPropertyDomain(<ns:Photo_equipment#bag_weight> <ns:Photo_equipment#camera_bag>)

Pradinis ontologijos elementas

DataPropertyRange(<ns:Photo_equipment#bag_weight> <http://www.w3.org/2001/XMLSchema#nonNegativeInteger>)

Gautas elemento XMI aprašas

<owl2:DataPropertyRange ontology="/0" dataPropertyExpression="/444" range="/442"/>

Atvirkštinės transformacijos rezultato elementas

DataPropertyRange(<ns:Photo_equipment#bag_weight>

<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>)

Pradinis ontologijos elementas

Declaration(Class(<ns:Photo_equipment#accessory>))

Gautas elemento XMI aprašas

<owl2:Class entityIRI="/46" declaration="/47"/>

<owl2:IRI lexicalValue="ns:Photo_equipment#accessory"/>

<owl2:Declaration ontology="/0" entity="/45"/>

Atvirkštinės transformacijos rezultato elementas

Declaration(Class(<ns:Photo_equipment#accessory>))

Pradinis ontologijos elementas

Declaration(DataProperty(<ns:Photo_equipment#bag_weight>))

Gautas elemento XMI aprašas

<owl2:DataProperty entityIRI="/446" declaration="/445" domains="/440" ranges="/441"/>

<owl2:Declaration ontology="/0" entity="/444"/>

<owl2:IRI lexicalValue="ns:Photo_equipment#bag_weight"/>

Atvirkštinės transformacijos rezultato elementas

Declaration(DataProperty(<ns:Photo_equipment#bag_weight>))

Pradinis ontologijos elementas

Declaration(NamedIndividual(<ns:Photo_equipment#Jim>))

Gautas elemento XMI aprašas

<owl2:NamedIndividual entityIRI="/144" declaration="/143"/>

<owl2:Declaration axiomAnnotations="/145" ontology="/0" entity="/142"/>

<owl2:IRI lexicalValue="ns:Photo_equipment#Jim"/>

Atvirkštinės transformacijos rezultato elementas

Declaration(NamedIndividual(<ns:Photo_equipment#Jim>))

Pradinis ontologijos elementas

Declaration(ObjectProperty(<ns:Photo_equipment#captured__photo>))

Gautas elemento XMI aprašas

<owl2:ObjectProperty entityIRI="/290" declaration="/289" domains="/286" ranges="/287"/>

<owl2:Declaration ontology="/0" entity="/288"/>

<owl2:IRI lexicalValue="ns:Photo_equipment#captured__photo"/>

Atvirkštinės transformacijos rezultato elementas

Declaration(ObjectProperty(<ns:Photo_equipment#captured__photo>))

Pradinis ontologijos elementas

DifferentIndividuals(<ns:Photo_equipment#Jim> <ns:Photo_equipment#John>)

Gautas elemento XMI aprašas

<owl2:DifferentIndividuals ontology="/0" individuals="/142 /174"/>

Atvirkštinės transformacijos rezultato elementas

DifferentIndividuals(<ns:Photo_equipment#Jim> <ns:Photo_equipment#John>)

Pradinis ontologijos elementas

DisjointClasses(<ns:Photo_equipment#commercial_organization> <ns:Photo_equipment#other_organization>)

Gautas elemento XMI aprašas

<owl2:DisjointClasses ontology="/0" classExpressions="/90 /27"/>

Atvirkštinės transformacijos rezultato elementas

DisjointClasses(<ns:Photo_equipment#commercial_organization> <ns:Photo_equipment#other_organization>)

Pradinis ontologijos elementas

DisjointUnion(<ns:Photo_equipment#camera_by_weight> <ns:Photo_equipment#lightweight_camera> <ns:Photo_equipment#moderate_weight_camera> <ns:Photo_equipment#heavy_camera>)

Gautas elemento XMI aprašas

<owl2:DisjointUnion ontology="/0" disjointClassExpressions="/78 /63 /12" class="/15"/>

Atvirkštinės transformacijos rezultato elementas

DisjointUnion(<ns:Photo_equipment#camera_by_weight> <ns:Photo_equipment#lightweight_camera> <ns:Photo_equipment#moderate_weight_camera> <ns:Photo_equipment#heavy_camera>)

Pradinis ontologijos elementas

EquivalentClasses(<ns:Photo_equipment#camera_by_weight> <ns:Photo_equipment#photo_camera>)

Gautas elemento XMI aprašas

<owl2:EquivalentClasses ontology="/0" classExpressions="/15 /123"/>

Atvirkštinės transformacijos rezultato elementas

EquivalentClasses(<ns:Photo_equipment#camera_by_weight> <ns:Photo_equipment#photo_camera>)

Pradinis ontologijos elementas

FunctionalObjectProperty(<ns:Photo_equipment#has__camera_model>)

Gautas elemento XMI aprašas

<owl2:FunctionalObjectProperty ontology="/0" objectPropertyExpression="/325"/>

Atvirkštinės transformacijos rezultato elementas

FunctionalObjectProperty(<ns:Photo_equipment#has__camera_model>)

Pradinis ontologijos elementas

EquivalentClasses(<ns:Photo_equipment#heavy_camera> ObjectIntersectionOf(DataSomeValuesFrom(<ns:Photo_equipment#camera_weight> DatatypeRestriction(<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> xsd:minExclusive "800"^^xsd:integer)) <ns:Photo_equipment#camera_by_weight>))

Gautas elemento XMI aprašas

<owl2:FacetRestriction restrictionValue="800" restrictionDatatypeValue="xsd:integer" constrainingFacet="/1503"/>

<owl2:IRI lexicalValue="xsd:minExclusive"/>

<owl2:DatatypeRestriction arity="1" restrictions="/1502" datatype="/1505"/>

<owl2:Datatype entityIRI="/1506"/>

<owl2:IRI lexicalValue="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>

<owl2:DataSomeValuesFrom dataPropertyExpressions="/437" dataRange="/1504"/>

<owl2:ObjectIntersectionOf classExpressions="/15 /1507"/>

<owl2:EquivalentClasses ontology="/0" classExpressions="/78 /1508"/>

Atvirkštinės transformacijos rezultato elementas

EquivalentClasses(<ns:Photo_equipment#heavy_camera> ObjectIntersectionOf(DataSomeValuesFrom(<ns:Photo_equipment#camera_weight> DatatypeRestriction(<http://www.w3.org/2001/XMLSchema#nonNegativeInteger> xsd:minExclusive "800"^^xsd:integer)) <ns:Photo_equipment#camera_by_weight>))

Pradinis ontologijos elementas

InverseObjectProperties(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#is_captured_by__person>)

Gautas elemento XMI aprašas

<owl2:InverseObjectProperties ontology="/0" objectPropertyExpression1="/288" objectPropertyExpression2="/229"/>

Atvirkštinės transformacijos rezultato elementas

InverseObjectProperties(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#is_captured_by__person>)

Pradinis ontologijos elementas

ObjectPropertyAssertion(<ns:Photo_equipment#has_produced__product> <ns:Photo_equipment#Jim> <ns:Photo_equipment#Photo_1>)

Gautas elemento XMI aprašas

<owl2:ObjectPropertyAssertion ontology="/0" targetIndividual="/146" sourceIndividual="/142" objectPropertyExpression="/308"/>

Atvirkštinės transformacijos rezultato elementas

ObjectPropertyAssertion(<ns:Photo_equipment#has_produced__product> <ns:Photo_equipment#Jim> <ns:Photo_equipment#Photo_1>)

Pradinis ontologijos elementas

ObjectPropertyDomain(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#person>)

Gautas elemento XMI aprašas

<owl2:ObjectPropertyDomain ontology="/0" objectPropertyExpression="/288" domain="/51"/>

Atvirkštinės transformacijos rezultato elementas

ObjectPropertyDomain(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#person>)

Pradinis ontologijos elementas

ObjectPropertyRange(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#photo>)

Gautas elemento XMI aprašas

<owl2:ObjectPropertyRange ontology="/0" objectPropertyExpression="/288" range="/3"/>

Atvirkštinės transformacijos rezultato elementas

ObjectPropertyRange(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#photo>)

Pradinis ontologijos elementas

SameIndividual(<ns:Photo_equipment#G._Gudas> <ns:Photo_equipment#Gytis_Gudas>)

Gautas elemento XMI aprašas

<owl2:SameIndividual ontology="/0" individuals="/170 /162"/>

Atvirkštinės transformacijos rezultato elementas

SameIndividual(<ns:Photo_equipment#G._Gudas> <ns:Photo_equipment#Gytis_Gudas>)

Pradinis ontologijos elementas

SubClassOf(<ns:Photo_equipment#accessory> <ns:Photo_equipment#role>)

Gautas elemento XMI aprašas

<owl2:SubClassOf ontology="/0" subClassExpression="/45" superClassExpression="/30"/>

Atvirkštinės transformacijos rezultato elementas

SubClassOf(<ns:Photo_equipment#accessory> <ns:Photo_equipment#role>)

Pradinis ontologijos elementas

SubClassOf(<ns:Photo_equipment#digital_photo_camera> ObjectMinCardinality(2 <ns:Photo_equipment#has__shooting_mode> <ns:Photo_equipment#shooting_mode>))

Gautas elemento XMI aprašas

<owl2:ObjectMinCardinality cardinality="2" classExpression="/36" objectPropertyExpression="/189"/>

<owl2:SubClassOf ontology="/0" subClassExpression="/87" superClassExpression="/1556"/>

Atvirkštinės transformacijos rezultato elementas

SubClassOf(<ns:Photo_equipment#digital_photo_camera> ObjectMinCardinality(2 <ns:Photo_equipment#has__shooting_mode> <ns:Photo_equipment#shooting_mode>))

Pradinis ontologijos elementas

SubDataPropertyOf(<ns:Photo_equipment#bag_weight> <ns:Photo_equipment#product_weight>)

Gautas elemento XMI aprašas

<owl2:SubDataPropertyOf ontology="/0" superDataPropertyExpression="/409" subDataPropertyExpression="/444"/>

Atvirkštinės transformacijos rezultato elementas

SubDataPropertyOf(<ns:Photo_equipment#bag_weight> <ns:Photo_equipment#product_weight>)

Pradinis ontologijos elementas

SubObjectPropertyOf(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#has_created__creative_product>)

Gautas elemento XMI aprašas

<owl2:SubObjectPropertyOf ontology="/0" superObjectPropertyExpression="/244" subObjectPropertyExpression="/288"/>

Atvirkštinės transformacijos rezultato elementas

SubObjectPropertyOf(<ns:Photo_equipment#captured__photo> <ns:Photo_equipment#has_created__creative_product>)

Pradinis ontologijos elementas

TransitiveObjectProperty(<ns:Photo_equipment#contains__product>)

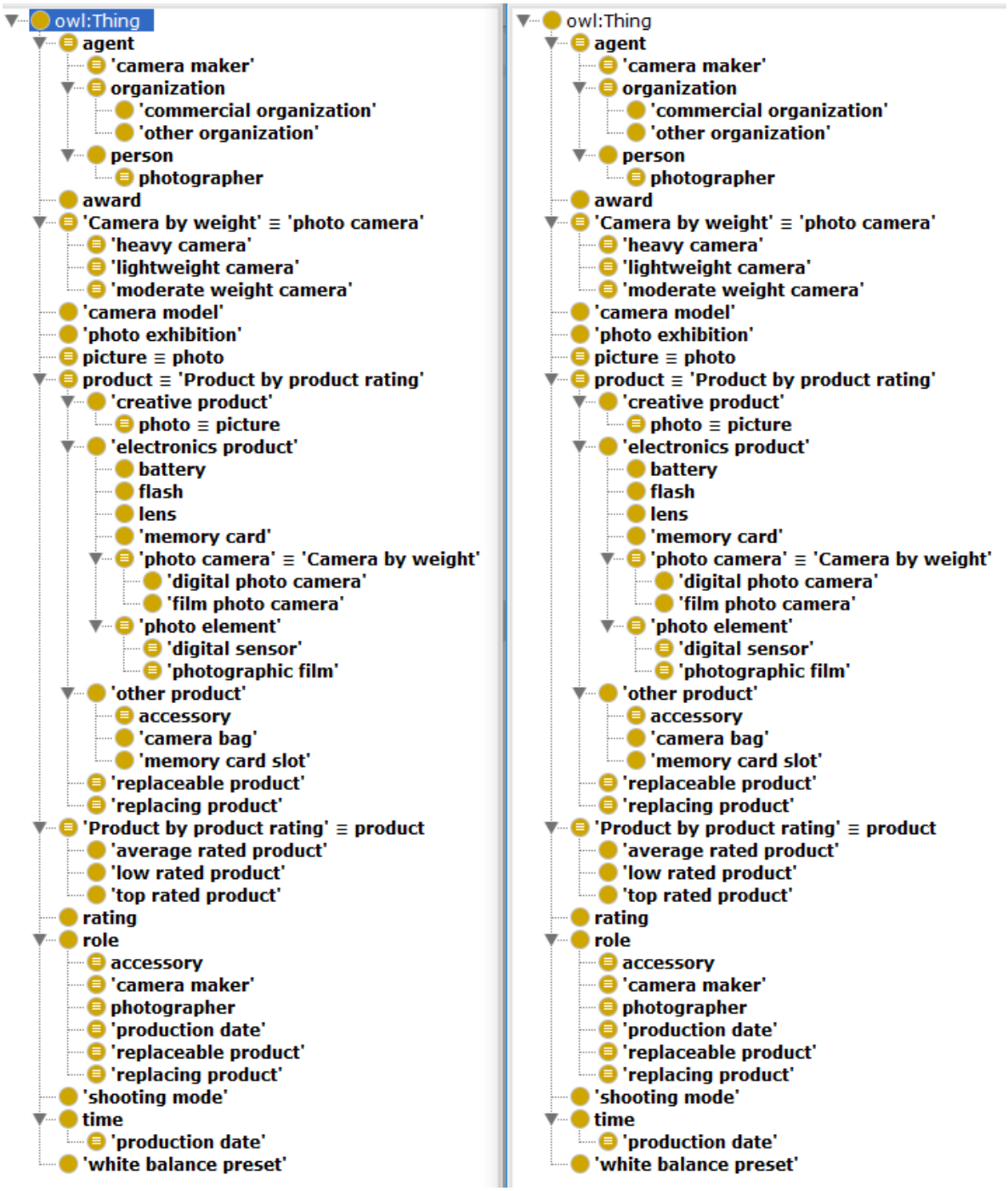
Gautas elemento XMI aprašas

<owl2:TransitiveObjectProperty ontology="/0" objectPropertyExpression="/330"/>

Atvirkštinės transformacijos rezultato elementas

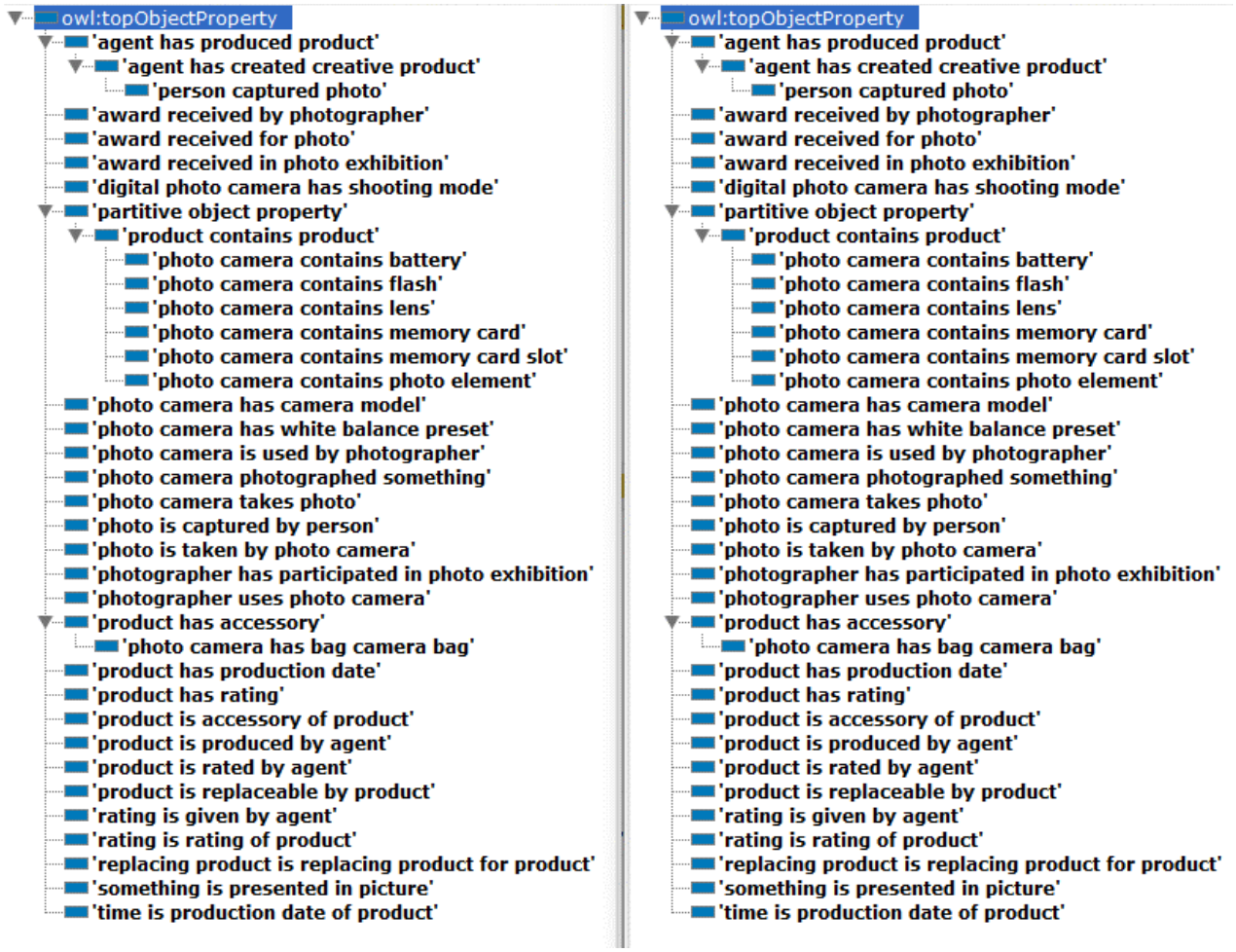
TransitiveObjectProperty(<ns:Photo_equipment#contains__product>)

Po atvirkštinės transformacijos gauta ontologija buvo sėkmingai atidaryta naudojant *Protege* įrankį. Siekiant įsitikinti, jog ontologijos yra identiškos, ontologijos esybės buvo palygintos vizualiai. Pirmiausia buvo lyginamos klasės. Paveikslėlio ***56 pav.*** kairėje pusėje atvaizduotos ontologijos klasės prieš transformaciją, dešinėje – po atvirkštinės transformacijos. Abiejose pusėse klasės ir jų kiekis yra vienodas.



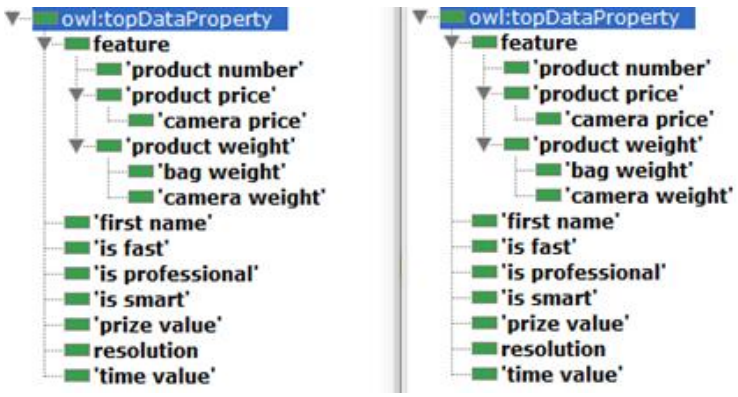
56 pav. Palygintos OWL ontologijų klasės

Tokiu pačiu principu palygintos ir ontologijų objektinės savybės (57 pav.). Kairėje pusėje – prieš transformaciją, dešinėje – po atvirkštinės transformacijos. Abiejose pusėse objektų savybės ir jų kiekis sutampa.



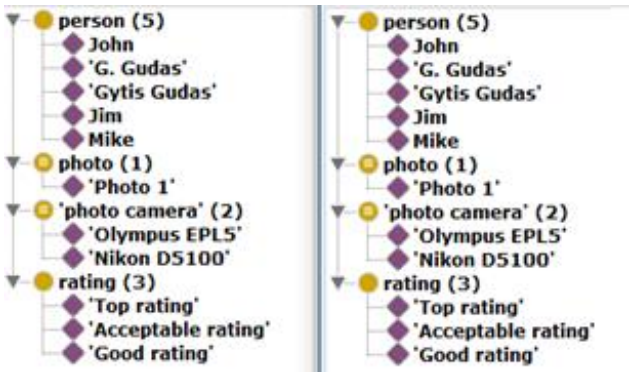
57 pav. Palygintos OWL ontologijų objektų savybės

Palyginus duomenų savybes (58 pav.) akivaizdu, jog jos sutampa. Kairėje pusėje duomenų savybės yra prieš transformaciją, dešinėje – po atvirkštinės transformacijos.



58 pav. Palygintos OWL ontologijų duomenų savybės

Paveikslėlyje ***59 pav.*** Pavaizduoti OWL ontologijų individai. Kairėje pusėje prieš transformaciją, dešinėje – po atvirkštinės transformacijos. Abi pusės yra identiškos.



59 pav. Palyginti OWL ontologijų individai

Pagal atliktus *OWL* ontologijų kodų ir vizualinius palyginimus padaryta išvada, jog realizuota *OWL* ontologijų transformacijos į *XMI* formatą programa veikia teisingai. Programa sėkmingai transformavo *OWL* ontologiją į *XMI* formatą. Transformuotai ontologijai atlikus atvirkštinę transformaciją, iš *XMI* į *OWL* ontologijas, buvo gauta identiška ontologija pradinei ontologijai.

5. EKSPERIMENTINIS OWL ONTOLOGIJŲ TRANSFORMACIJOS Į XMI FORMATĄ TYRIMAS

5.1. Eksperimento planas

Eksperimentinio tyrimo metu yra tiriami šio darbo metu suprojektuotos ir realizuotos *OWL* ontologijų transformavimo į *XMI* formatą programos veikimo rezultatai. Tyrimo metu yra siekiama nustatyti sistemos galimybes, ištirti duomenų nuostolius. Eksperimentui atlikti naudojamas *s2o* programos komponento *„Transform OWL 2 XMI to OWL 2“* (8 pav.) *XSLT* transformavimo taisyklių rinkinys, kuris transformuoja *OWL 2 XMI* į *OWL* ontologijas. Tai yra atvirkštinė transformacija realizuotam sprendimui. Atlikus atvirkštinę transformaciją bus galima palyginti pradinį ontologijos failą su failu, po atvirkštinės transformacijos. Palyginimui atlikti parašytas programinis kodas, kuris suskaičiuoja pateiktoje ontologijoje esančias esybes ir aksiomas, pagal jų tipus. Atlikus ontologijos elementų skaičiavimus prieš ir po transformacijų, bus įvertinama ar visi ontologijos elementai transformuojami teisingai, ar yra patiriami duomenų nuostoliai.

5.2. Eksperimento rezultatai

Eksperimentui atlikti buvo parinkta septynios internete rastos OWL ontologijos, o kiekvienos ontologijos transformavimo rezultatai pateikiami lentelėse.

Lentelėje (26 lentelė) pateikti bandymo rezultatai. Bandymas buvo atliktas panaudojant ontologiją apie paskolas [12]. Pradinė ontologija ir ontologija, gauta po atvirkštinės transformacijos yra identiškos.

26 lentelė. Eksperimento bandymo rezultatai naudojant paskolų ontologiją

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	24	24
Individai	0	0
Duomenų savybės	20	20
Objektų savybės	10	10
Anotacijų savybės	3	3
Duomenų tipai	5	5
Viso aksiomų	303	303
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
AnnotationAssertion	162	162
EquivalentClasses	2	2
ObjectPropertyDomain	10	10
DataPropertyDomain	20	20
DataPropertyRange	20	20
Declaration	56	56
DisjointUnion	1	1

ObjectPropertyRange	10	10
SubClassOf	22	22

Sekantis bandymas buvo atliktas panaudojant ontologiją apie ekonomiką [13]. Lentelėje (27 lentelė) buvo surašyti gauti rezultatai, pagal kuriuos padaryta išvada, jog pradinė ontologija ir ontologija, gauta po atvirkštinės transformacijos yra identiškos.

27 lentelė. Eksperimento bandymo rezultatai naudojant ontologiją apie ekonomiką

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	91	91
Individai	0	0
Duomenų savybės	7	7
Objektų savybės	107	107
Anotacijų savybės	3	3
Duomenų tipai	6	6
Viso aksiomų	1214	1214
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
AnnotationAssertion	615	615
EquivalentClasses	20	20
FunctionalObjectProperty	3	3
ObjectPropertyDomain	107	107
SymmetricObjectProperty	3	3
DataPropertyRange	7	7
DataPropertyDomain	7	7
Declaration	207	207
ObjectPropertyRange	107	107
InverseObjectProperties	49	49
SubClassOf	89	89

Dar vienas bandymas buvo atliktas panaudojant įvykių ontologiją [14]. Bandymo rezultatai surašyti į lentelę (28 lentelė). Nustatyta, jog pradinė ontologija ir ontologija, gauta po atvirkštinės transformacijos yra identiškos.

28 lentelė. Eksperimento bandymo rezultatai naudojant įvykių ontologiją

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	72	72
Individai	0	0
Duomenų savybės	7	7
Objektų savybės	72	72
Anotacijų savybės	3	3
Duomenų tipai	6	6
Viso aksiomų	891	891
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
AnnotationAssertion	453	453
EquivalentClasses	18	18
FunctionalObjectProperty	2	2
ObjectPropertyDomain	72	72

SymmetricObjectProperty	2	2
DataPropertyRange	7	7
DataPropertyDomain	7	7
Declaration	153	153
InverseObjectProperties	35	35
ObjectPropertyRange	72	72
SubClassOf	70	70

Kitas bandymas buvo atliktas panaudojant filmų nuomos ontologiją. Bandymo rezultatai pateikti lentelėje (29 lentelė). Pagal gautus bandymo rezultatus nustatyta, jog pradinė ontologija ir ontologija, gauta po atvirkštinės transformacijos yra identiškos.

29 lentelė. Eksperimento bandymo rezultatai naudojant filmų nuomos ontologiją

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	11	11
Individai	12	12
Duomenų savybės	19	19
Objektų savybės	22	22
Anotacijų savybės	2	2
Duomenų tipai	7	7
Viso aksiomų	277	277
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
EquivalentClasses	2	2
FunctionalObjectProperty	9	9
DataPropertyRange	18	18
ClassAssertion	12	12
ObjectPropertyRange	22	22
SubClassOf	2	2
DataPropertyAssertion	31	31
AnnotationAssertion	15	15
ObjectPropertyDomain	22	22
DisjointClasses	3	3
DataPropertyDomain	19	19
Declaration	64	64
InverseObjectProperties	11	11
InverseFunctionalObjectProperty	11	11
DifferentIndividuals	2	2
ObjectPropertyAssertion	34	34

Lentelėje (30 lentelė) pateikti bandymo rezultatai, kurie gauti atlikus transformaciją su foto įrangos ontologija [15]. Pagal pateiktus rezultatus nustatyta, jog pradinė ontologija ir ontologija, gauta po atvirkštinės transformacijos yra identiškos.

30 lentelė. Eksperimento bandymo rezultatai naudojant foto įrangos ontologiją

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	45	45
Individai	11	11
Duomenų savybės	14	14
Objektų savybės	37	37
Anotacijų savybės	3	3
Duomenų tipai	8	8
Viso aksiomų	649	649
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
EquivalentClasses	17	17
TransitiveObjectProperty	1	1
FunctionalObjectProperty	4	4
SameIndividual	1	1
SubObjectPropertyOf	10	10
SubDataPropertyOf	6	6
DataPropertyRange	14	14
DisjointUnion	1	1
ObjectPropertyRange	35	35
SubClassOf	52	52
ClassAssertion	11	11
AsymmetricObjectProperty	1	1
AnnotationAssertion	324	324
DataPropertyAssertion	1	1
ObjectPropertyDomain	35	35
DisjointClasses	6	6
DataPropertyDomain	14	14
Declaration	109	109
InverseObjectProperties	5	5
DifferentIndividuals	1	1
ObjectPropertyAssertion	1	1

Dar vienas bandymas atliktas naudojant vieną *LKIF core* ontologijos komponentą – išraiškų ontologiją [16]. Bandymo rezultatai surašyti į lentelę (31 lentelė). Analizuojant rezultatus nustatyta, jog transformuota ontologija turėjo daugiau aksiomų. Taip įvyko dėl to, jog pradinėje ontologijoje yra naudojamas *imports* elementas, kuris nurodo, kad šiai ontologijai reikia papildomos ontologijos. Kadangi sprendime panaudota *OWLAPI Java* biblioteka, kuri palaiko kitų ontologijų importą, todėl transformuojant atsirado papildomų deklaracijų į kitus elementus, kurie reikalingi aksiomoms sudaryti. Kadangi šie elementai nepriklauso pačiai ontologijai, o yra importuojami tik teisingiems ryšiams sudaryti, todėl papildomų ontologijų importavimas nekeičia transformuojamos ontologijos esybių skaičiaus.

31 lentelė. Eksperimento bandymo rezultatai naudojant LKIF išraiškų ontologiją

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	38	38
Individai	0	0
Duomenų savybės	0	0
Objektų savybės	34	34
Anotacijų savybės	2	2
Duomenų tipai	2	2
Viso aksiomų	299	**313**
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
AnnotationAssertion	63	63
EquivalentClasses	17	17
ObjectPropertyDomain	28	28
SubObjectPropertyOf	22	22
DisjointClasses	1	1
SymmetricObjectProperty	2	2
Declaration	60	**74**
ObjectPropertyRange	28	28
SubClassOf	63	63
InverseObjectProperties	15	15

Bandant programos galimybes buvo atliktas bandymas naudojant genų ontologiją [17]. Ši ontologija ypatinga tuo, kad ji yra labai didelė: jog dydis 170 MB. Ši ontologija turi apie 50 tūkst. klasių, bei daugiau nei pusę milijono aksiomų. Programa transformaciją atliko greičiau nei per minutę, o rezultatų failas užėmė 210 MB. Bandant atlikti atvirkštinę transformaciją buvo gaunamos klaidos, jog anotacijos reikšmė negali turėti specialių simboliu (<,>,&). Pašalinus specialiuosius simbolius buvo gauta kita klaida: per didelis suvartoto kompiuterio sparčiosios atminties kiekio. Šios problemos pašalinti nepavyko, tačiau sprendžiant iš transformuoto *XMI* failo dydžio, transformacija pavyko. Lentelėje (32 lentelė) pateikti transformuotos ontologijos duomenys.

32 lentelė. Eksperimento bandymo rezultatai naudojant genų ontologiją

Esybės	**Kiekis prieš**	**Kiekis po**
Klasės	48575	-
Individai	0	-
Duomenų savybės	0	-
Objektų savybės	9	-
Anotacijų savybės	56	-
Duomenų tipai	3	-
Viso aksiomų	577836	-
Aksiomų tipai	**Kiekis prieš**	**Kiekis po**
AnnotationAssertion	422730	-

EquivalentClasses	11901	-
TransitiveObjectProperty	4	-
SubAnnotationPropertyOf	23	-
SubObjectPropertyOf	3	-
DisjointClasses	31	-
SubPropertyChainOf	6	-
Declaration	48640	-
SubClassOf	94497	-
InverseObjectProperties	1	-

Pagal gautus ontologijų transformavimo rezultatus matome, jog realizuota *OWL* ontologijų transformacijos į *XMI* standartą programa veikia sklandžiai ne tik su sudėtingas aksiomas turinčiomis ontologijomis, bet ir su ontologijomis, kurios pasižymi ypatingai dideliu esybių ir aksiomų kiekiu. Vykdant bandymus buvo pastebėta, jog po transformacijos padidėdavo aksiomų kiekis. Taip įvyko dėl to, jog transformuojamose ontologijose buvo *includes* sąlyga, kuri pažymi, jog ontologijai funkcionuoti reikia papildomos išorinės ontologijos. Realizuotas sprendimas naudoja *OWLAPI* biblioteką, kuri geba importuoti išorines bibliotekas. Dėl šių papildomai importuotų ontologijų ir padidėdavo bendras transformuotos ontologijos aksiomų skaičius. Programa atlikdama transformaciją nereikalauja didelių kompiuterio resursų net dirbant su ypatingai didelėmis ontologijomis.

6. IŠVADOS

Sparčiai vystantis informacinėms technologijoms ontologijų naudojimas sistemų kūrime tampa vis dažnesnis, tačiau nėra vieningo standarto keistis jomis tarp taikomųjų programų ar sistemų. Šiame darbe buvo analizuojamos ontologijos, panašūs sprendimai bei procesas, kaip *OWL* ontologijos turėtų būti transformuojamos į *XMI* standartą. Reziumuojant atlikto darbo rezultatus buvo padarytos šios išvados:

1. Atlikus ontologijos analizę buvo išsiaiškintos ontologijos sudedamosios dalys, sudarymo principai, tačiau paaiškėjo, jog nėra vieningo standarto keistis ontologijomis tarp programų;
2. Panašių sprendimų analizė parodė, jog problemą išsprendžiančio sprendimo nėra, o esantys panašūs sprendimai nepriimtini dėl netinkamos transformacijos ar duomenų praradimo;
3. Sukurta *OWL* transformavimo į *XMI* standartą metodika, suprojektuotas bei realizuotas sprendimas išsprendžiantis susidariusią problemą;
4. Programa naudoja *OWLAPI* biblioteką ir yra suprogramuota *Java* programavimo kalba, todėl gali lengvai gali būti tobulinama ar modifikuojama;
5. Atliktas testavimas, kuris panaudojant *s2o* programa ir atliekant atvirkštinę transformaciją parodė, jog realizuotas sprendimas veikia taip, kaip buvo suprojektuotas;
6. Eksperimentinis sprendimo tyrimas atskleidė puikius realizuotos programos veikimo rezultatus. Programa transformuoja ontologijas į *XMI* formatą neprarandant duomenų, gali efektyviai dirbti su labai didelėmis ontologijomis, kurių aksiomų skaičius viršija pusę milijono, palaiko nuotolinių ontologijų importą;

7. LITERATŪRA

[1] „Ontologija (informatika),“ Vikipedija, 27 05 2014. [Tinkle]. Available: https://lt.wikipedia.org/wiki/Ontologija_(informatika).

[2] „Ontologijos,“ 26 5 2008. [Tinkle]. Available: ftp://ausis.gf.vu.lt/kis/Intelektika/ppt/ontologijos.ppt.

[3] „OWL 2 profiles,“ [Tinkle]. Available: https://www.w3.org/People/Sandro/owl2-profiles-doc.

[4] „Wikipedia - Web Ontology Language,“ [Tinkle]. Available: https://en.wikipedia.org/wiki/Web_Ontology_Language.

[5] „s2o: SBVR to OWL converter,“ [Tinkle]. Available: http://s2o.isd.ktu.lt/. [Kreiptasi 13 05 2016].

[6] „OWL2XMI Project,“ Semantic Web Architecture and Performance Group, [Tinkle]. Available: http://owl2xmi.sourceforge.net/. [Kreiptasi 13 05 2016].

[7] „Jena (framework),“ HP Labs, [Tinkle]. Available: https://en.wikipedia.org/wiki/Jena_(framework).

[8] „The OWL API,“ University of Manchester, [Tinkle]. Available: http://owlapi.sourceforge.net/.

[9] „Protege-OWL API Programmer's Guide,“ Protege staff members, [Tinkle]. Available: https://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide.

[10] W3C, „OWL2 syntax,“ [Tinkle]. Available: https://www.w3.org/TR/owl2-syntax/.

[11] „Photo equipment OWL ontologija,“ [Tinkle]. Available: http://s2o.isd.ktu.lt/examples/PhotoEquipment/PhotoEquipment.owl.

[12] „Paskolos OWL ontologija,“ [Tinkle]. Available: http://s2o.isd.ktu.lt/examples/LoanContracts/LoanContracts.owl.

[13] „Ekonomikos ontologija,“ [Tinkle]. Available: http://s2o.isd.ktu.lt/examples/Ekonomika/Ekonomika.owl.

[14] „Įvykių ontologija,“ [Tinkle]. Available: http://s2o.isd.ktu.lt/examples/Ivykiai/Ivykiai.owl.

[15] „Foto įrangos ontologija,“ [Tinkle]. Available: http://s2o.isd.ktu.lt/examples/PhotoEquipment/PhotoEquipment.owl.

[16] „LKIF Core,“ [Tinkle]. Available: http://www.estrellaproject.org/lkif-core/.

[17] „Gene Ontology,“ [Tinkle]. Available: http://www.geneontology.org/ontology/go.owl.

[18] „XML Metadata Interchange,“ 15 7 2015. [Tinkle]. Available: https://en.wikipedia.org/wiki/XML_Metadata_Interchange.

[19] „OWL Web Ontology Language,“ 2004. [Tinkle]. Available: https://www.w3.org/TR/owl-ref/.

[20] „Components of an Ontology,“ 22 01 2010. [Tinkle]. Available: http://ontogenesis.knowledgeblog.org/514.

[21] „Ontology components,“ 2015. [Tinkle]. Available: https://en.wikipedia.org/wiki/Ontology_components.

[22] „What's the use of an XMI to OWL translation?,“ 24 05 2010. [Tinkle]. Available: http://answers.semanticweb.com/questions/918/whats-the-use-of-an-xmi-to-owl-translation.

[23] „XML Metadata Interchange (XMI),“ [Tinkle]. Available: http://www.service-architecture.com/articles/web-services/xml_metadata_interchange_xmi.html.

[24] „OWL 2 Web Ontology Language,“ [Tinkle]. Available: https://www.w3.org/TR/owl2-profiles/. [Kreiptasi 10 10 2015].

[25] „Paslaugos ir įvadas į SOA,“ [Tinkle]. Available: https://moodle.ktu.edu/pluginfile.php/242313/mod_resource/content/2/t120m124%20-%20soa%20-%20teor%20-%2001%20-%20ivadas.pdf. [Kreiptasi 17 05 2016].

[26] J. Karpovič, „TRANSFORMING SBVR BUSINESS SEMANTICS INTO WEB ONTOLOGY LANGUAGE OWL2: MAIN CONCEPTS,“ [Tinkle]. Available: https://www.researchgate.net/publication/228614898_TRANSFORMING_SBVR_BUSINESS_SEMANTICS_INTO_WEB_ONTOLOGY_LANGUAGE_OWL2_MAIN_CONCEPTS.

[27] „LKIF expressions ontology,“ [Tinkle]. Available: https://github.com/RinkeHoekstra/lkif-core/blob/master/expression.owl.