KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Algirdas Šukys

Semantinės informacijos išrinkimo iš reliacinių duomenų bazių metodas taikant ontologijas

Magistro darbas

Darbo vadovas

prof. Lina Nemuraitė

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Algirdas Šukys

Semantinės informacijos išrinkimo iš reliacinių duomenų bazių metodas taikant ontologijas

Magistro darbas

Recenzentas
doc. dr. Stasys Maciulevičius

Darbo vadovas
prof. Lina Nemuraitė

2010-05-28

Atliko
IFM-4/4 gr. Stud.
Algirdas Šukys
2010-05-28

Kaunas, 2010

Turinys

Summary

Method of Semantic Information Retrieval from Relational Databases Using Ontologies

Ontologies are becoming increasingly popular because they allow organizations to describe their problem domains in a more flexible manner and to search information from multiple sources giving semantically significant results for users. However, the increasing amount of information in ontology makes its storing in a text file not effective. The aim of this research is to improve possibilities of querying large ontologies when these are kept in relational databases. The method was created for executing SPARQL queries in ontology, stored in a relational database created by OWL2RDB algorithm. Experiments have shown that the method improves query performance time in comparison with existing query engine especially for large ontologies having many individuals.

Key words: ontology, relational database, OWL, SPARQL.

Terminų ir santrumpų žodynėlis

- Ontologija – dalykinės srities konceptų aprašymas;
- *OWL* (*Web Ontology Language*) - ontologijos aprašymo kalba;
- *OWL Reasoner* – ontologijų analizės ir užklausų vykdymo programa;
- *OWL2RDB* – ontologijos konvertavimo į reliacinės duomenų bazės schemą algoritmas;
- *RDF* (*Resource Description Framework*) - resursų aprašymo karkasas;
- *RDFS* (*RDF Schema*) – *RDF* išplėtimas;
- *SPARQL* (*Simple Protocol and RDF Query Language*) - *OWL* užklausų kalba;
- *URI* (*Uniform Resource Identifier*) – interneto resursų identifikatorius.

1. Įvadas

Ontologija yra tikslus dalykinės srities konceptų, jų ryšių aprašymas. Dalykinės srities ekspertai vis dažniau naudoja ontologijas dalykinei sričiai aprašyti. Daugybėje sričių kuriamos standartizuotos ontologijos, kurios dalykinės srities ekspertų gali būti panaudotos ir pritaikytos specifiniams poreikiams. Tarkime standartinė vyno ontologija, skirta klasifikuoti ir aprašyti vynus, gali būti naudojama tiek vyno gamintojų, kuriems reikia aprašyti savo asortimentą, tiek restoranų, norinčių pasiūlyti klientams vyną pagal tam įvairius skonio spalvos ar kitus kriterijus. Ontologijos tai pat plačiai naudojamos žiniatinklyje – interneto puslapių, produktų kategorizavimui, jų savybių bei sąryšių aprašymui.

Ontologijos aprašymui naudojamas resursų aprašymo karkasas *RDF*. *RDF* idėja yra išskaidyti dalykinės srities aprašą į elementarias formuluotes, kurios aprašomos trigubu formatu – veiksnys, tarinys, objektas. Šios formuluotės jungiasi tarpusavyje ir sudaro *RDF* grafą. Didelis tokios duomenų struktūros privalumas yra tas, kad ją gali suprasti ne tik žmonės, bet ir kompiuteriai, todėl kuriamos duomenų paieškos sistemos (agentai), kurie automatiškai ieško informacijos internete. Toliau pateiktos pagrindinės priežastys, kodėl naudinga kurti ontologijas [14]:

- bendros informacijos struktūros naudojimas – sakykime, keli skirtingi duomenų šaltiniai teikia informaciją apie vynus. Jeigu ta informacija yra vienodos struktūros, t. y. visi terminai naudojami tie patys, automatinės paieškos sistemos gali sujungti informaciją iš skirtingų šaltinių ir pateikti vartotojui kaip visumą.
- leisti dalintis ontologijomis skirtingų sričių ekspertams – įvairiose dalykinėse srityse naudojama laiko sąvoka, į kurią įeina laiko intervalų, matavimo vienetų apibrėžimai ir kita. Jeigu kas nors sukuria detalią laiko ontologiją, ji vėliau gali būti daug kartų panaudota įvairių dalykinių sričių ekspertų.
- aiškiai aprašyti dalykinės srities žinias ontologijoje – toks aprašymas yra daug aiškesnis negu įrašymas į programinį kodą, nes palengvina dalykinės srities žinių modifikavimą ir padeda vartotojams lengviau suprasti dalykinę sritį, jos apribojimus;
- atskirti žinių ir operacijų lygmenis – ontologija naudinga konfigūravimo uždaviniuose, nes leidžia sukurti daiktų konfigūravimo iš elementų algoritmus, nepriklausomus nuo ontologijos, kuriuos vėliau galima pritaikyti įvairioms sritims;
- analizuoti dalykinės srities žinias - tikslus dalykinės srities aprašymas ontologijomis padeda ekspertams lengviau į ją įsigilinti.

Tiksliai aprašyti dalykinės srities žinias ontologijomis leidžia ontologijos aprašymo kalbos. Jeigu *RDF* leidžia aprašyti sąvokas, jų sąryšius, tai vėliau pristatytos kalbos *RDFS* bei *OWL*

suteikia galimybę grupuoti resursus į klases, kurti klasių hierarchiją, nustatyti įvairius kardinalumo, sąryšių ir kitus apribojimus, todėl dalykinės srities aprašymas tampa labai tikslus.

Tačiau taikant ontologijas praktikoje, susiduriama su sunkumais. Ontologijos paprastai saugomos tekstiniame faile. Yra sukurta nemažai įrankių ontologijoms tekstiniuose failuose kurti bei užklausoms vykdyti jose. Saugant ontologiją tekstiniame faile, užklausos vykdomos naudojant specialias paieškos programas (angl. *Reasoner*), įkėlus visą ontologiją į kompiuterio darbinę atmintį. Toks sprendimas yra paprastas ir efektyvus mažesnėms ontologijoms, tačiau ontologijai didėjant, darbo su ja efektyvumas mažėja [15], nes daug laiko užtrunka tekstinio *XML* failo nuskaitymas, ontologijos įkėlimas į atmintį. *RDF* grafas tokiu atveju būna labai didelis, todėl ir užklausos vykdymas užtrunka ilgai, todėl dideles ontologijas siekiama saugoti reliacinėse duomenų bazėse.

Taigi **darbo tikslas** – pagerinti duomenų paieškos ontologijoje procesą, sukuriant ontologijos užklausų vykdymo metodą, suderinantį reliacinių duomenų bazių ir ontologijų galimybes. **Tyrimo objektas** – duomenų išrinkimo iš ontologijos procesas, kai ontologija saugoma reliacinėje duomenų bazėje. Tikslui pasiekti reikėjo atlikti šiuos **uždavinius**:

1. Išanalizuoti:
   - duomenų paieškos procesus;
   - ontologijas, jų kūrimo taisykles ir principus;
   - ontologijų kalbas ir kūrimo įrankius;
   - ontologijų užklausų kalbas, jų saugojimo ir užklausų vykdymo metodus;
2. sukurti ir realizuoti algoritmus, kurie leistų atlikti paiešką bendrai taikant ontologijų ir reliacinių duomenų bazių technologijas;
3. sukurti ir ištestuoti pavyzdinę sistemą, kuri leistų išbandyti sukurtą metodiką ir algoritmą;
4. atlikti eksperimentą ir įvertinti darbo rezultatus.

Tyrimo metodika. Darbe buvo taikomas informacinių sistemų tyrimams skirtas konstruktyviojo tyrimo metodas, kuriame sukuriamas problemos sprendimas ir jį realizuojantis artefaktas, atliekamas šio sprendimo įvertinimas ir sukuriamos naujos žinios. Buvo taikoma literatūros ir lyginamoji analizė, ontologijų kalbos; sprendimo konstravimui buvo taikomas evoliucinis kūrimo procesas, *UML* ir objektinė metodologija; sprendimui įvertinti buvo atliekamas eksperimentas.

Atlikus analizę, ontologijai saugoti reliacinėje duomenų bazėje buvo pasirinktas *OWL2RDB* algoritmas, kuris transformuoja ontologijos schemą į reliacinės duomenų bazės schemą, nes jis

geriausiai išnaudoja reliacinės duomenų bazės galimybes, saugant joje ontologijas. Tačiau *SPARQL* konvertavimo metodo į *SQL* pagal *OWL2RDB* taisykles nėra sukurta, todėl buvo kuriamas naujas metodas. Sukurtas metodas paremtas užklausų vykdymu *Pellet OWL Reasoner* bibliotekoje.

Metodą sudaro du algoritmai. Pirmasis skirtas atkurti ontologiją iš reliacinės duomenų bazės schemos. Jis nuskaito reliacinės duomenų bazės struktūrą ir pagal ją sukuria *RDF* grafą kompiuterio darbinėje atmintyje. Algoritmo idėja – nerašyti ontologijos klasių individų, kurie saugomi duomenų lentelėse į kompiuterio darbinę atmintį, o rašyti tik klasių hierarchiją bei metaduomenis. Tokiu būdu ontologija gali turėti neribotą skaičių individų ir ontologijos įkėlimo į atmintį laikas bei atminties apkrovimas nepriklauso nuo ontologijos individų skaičiaus.

Antrasis algoritmas skirtas vykdyti *SPARQL* užklausas. Jis analizuoja kiekvieną užklausos sąlygos dalį atskirai. Užklausos vykdomos dviem etapais – pirmiausia vykdomas klasių išrinkimas, vėliau – klasių individų. Klasių išrinkimas vykdomas pirmuoju algoritmu *Pellet OWL Reasoner* bibliotekoje sudarytame ontologijos grafe. Šiame etape gali būti randama klasių hierarchija, ontologijos apribojimai, persikertančios klases ir kita. Vėliau randami individai, konvertavus *SPARQL* užklausą į *SQL*.

Buvo sukurtas šio metodo projektas, realizuotas ir ištestuotas programinis prototipas. Atliktas eksperimentas parodė, kad metodas leidžia efektyviau vykdyti užklausas ontologijose, kai jos saugomos reliacinėje duomenų bazėje. Ypač tai išryškėja augant ontologijos individų skaičiui.

Darbo struktūra:

- analizės dalyje analizuojamas dabartinis informacijos išrinkimo įprastose reliacinėse duomenų bazėse procesas, bei kaip jis būtų pagerintas pritaikant ontologijas bei saugojant jas reliacinėje duomenų bazėje. Vėliau paaiškinama ontologijos sąvoka, aprašomas ontologijos kūrimo procesas, išanalizuojamas ontologijos kūrimo įrankis *Protégé,* ontologijos saugojimo duomenų bazėje algoritmai, užklausų vykdymo kalba *SPARQL*, *Pellet OWL Reasoner* užklausų vykdymo biblioteka.
- Projekto dalyje apibrėžiami pagrindiniai kuriamo metodo apribojimai ir reikalavimai, detaliai aprašomi sukurti ontologijos atstatymo iš reliacinės duomenų bazės schemos ir *SPARQL* užklausos vykdymo algoritmai.
- Realizacijos dalyje aprašoma programinio prototipo realizacija, naudoti įrankiai ir programiniai paketai, testavimas ir kontrolinis pavyzdys.
- Eksperimento dalyje buvo siekiama nustatyti, ar sukurtas metodas tikrai veiksmingas – skaičiuojami ir lyginami užklausų vykdymo laikai įprastoje tekstiniame faile

saugojamoje ontologijoje su vykdymo laikais, kai ontologija saugojama reliacinėje duomenų bazėje.

Sukurtas metodas buvo aprašytas dviejuose straipsniuose, kurie pateikiami prieduose. Vienas straipsnis buvo pristatytas konferencijoje Information Technologies‘ 2010, kitas – IVUS‘ 2010.

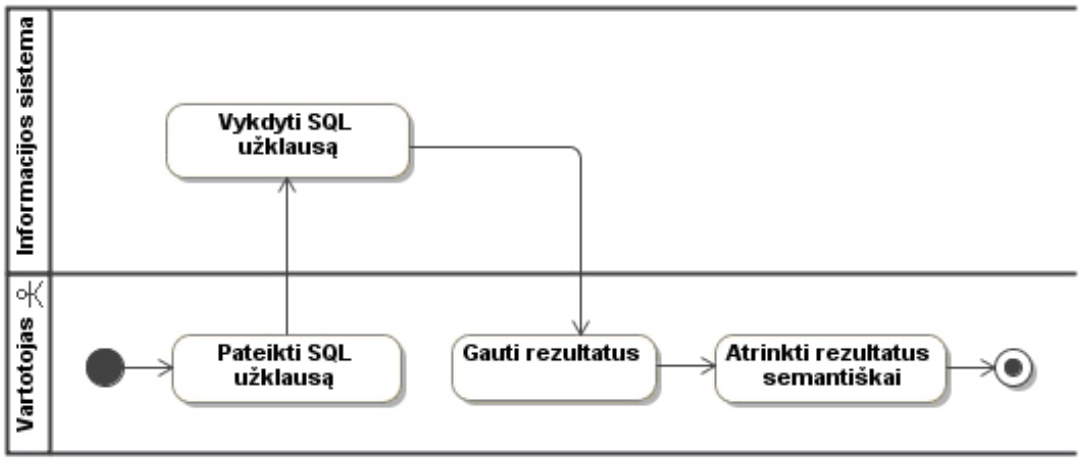
2. Ontologijų saugojimo ir užklausų vykdymo metodų analizė

Šio darbo **tyrimo sritis** yra duomenų bazių ir ontologijų integravimo metodų ir technologijų taikymas, **tyrimo objektas** – duomenų išrinkimo iš duomenų bazių procesas, taikant ontologijas.

Reliacinės duomenų bazės turi dideles informacijos apdorojimo galimybes, jose gali būti saugomi dideli informacijos kiekiai, tačiau vartotojui dažnai sudėtinga susirasti reikiamą informaciją. Įprastose reliacinėse duomenų bazėse duomenų sąryšiai įgyvendinami pirminio-išorinio rakto ryšiais, kurie leidžia struktūrizuoti duomenis, tačiau kompiuteriai gali analizuoti duomenis tik sintaksiniu aspektu, o jų reikšmę gali suprasti tik žmogus. Sukūrus dalykinės srities ontologiją, kuri nusako duomenų semantinius sąryšius, kompiuteris gali atlikti ir semantinę duomenų analizę [19], vartotojui pateikiami tikslesni rezultatai. Tačiau saugant ontologiją įprastu būdu (tekstiniame faile), užklausas vykdyti yra sunku, kai duomenų apimtis išauga, nes visą ontologiją reikia kelti į operatyviąją atmintį, apkraunamas kompiuteris ir užklausų vykdymo laikas labai išauga.

2.1. Duomenų paieškos proceso analizė

Reliacinėse duomenų bazėse įvykdžius užklausą kompiuteris pateikia pagal įvairius sintaksinius parametrus iš duomenų bazės išrinktus duomenis. Norėdamas panaudoti šiuos duomenis, vartotojas vėliau pats juos rūšiuoja pagal prasminę reikšmę, tai yra pagal tam tikrus kriterijus, kurių kompiuteris suprasti negali, taigi tam tikrą duomenų paieškos darbo dalį atlieka pats žmogus. Šis duomenų išrinkimo procesas pavaizduotas 1 paveikslėlyje.

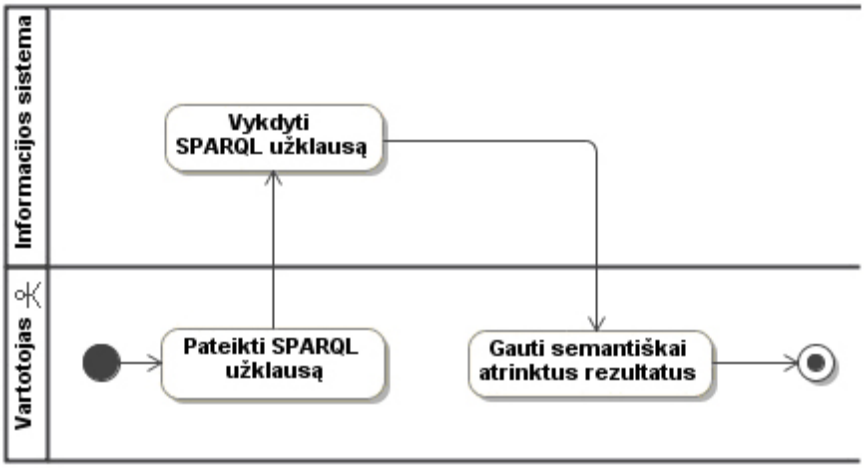


1 pav. Duomenų išrinkimo procesas iš reliacinių duomenų bazių

Pirmoji proceso veikla yra vartotojo užklausos pateikimas. Tai atliekama kompiuterio ekrane pasirenkant tam tikrus užklausos parametrus (įvedamas raktinis žodis, pasirenkamos reikšmės iš sąrašo ir kt.). Tada pagal vartotojo pateiktus parametrus formuojama *SQL* užklausa (žinoma, jeigu

vartotojas yra pakankamai kvalifikuotas, jis gali pats tiesiogiai rašyti užklausas), kuri išrenka iš duomenų bazės norimus įrašus. Pavyzdžiui, paieškos sistema *Google* pagal pateiktą raktinį žodį pateikia iš anksto indeksuotų interneto resursų, kuriuose yra pateiktas raktinis žodis, sąrašą. Taigi rezultatai atrenkami pagal sintaksinį atitikimą pateiktiems paieškos parametrams. Galiausiai paieškos sistema suformuoja ir išveda rezultatus vartotojui.

Šio duomenų išrinkimo proceso privalumas yra didelis informacijos saugojimo kiekis, t.y. duomenų bazėse gali būti saugojama daug duomenų įrašų. Tačiau minusas (palyginti su ontologijomis) – užklausos grąžina ne tokius tikslius rezultatus, kokių norėtų vartotojas. Gavęs rezultatų sąrašą, vartotojas pats turi juose ieškoti loginių sąryšių tarp įrašų, analizuoti, kokias savybes jie turi ir pagal tai iš daugybės rezultatų atsirinkti sau tinkamus. Tarkime, analizuojant vyno ir maisto dalykines sritis, neatsakytume į klausimą, koks vynas prie kokių valgių tinka. Tokiu atveju tektų analizuoti vyno ir maisto sąrašus ir atrinkti rezultatus. Tuo tarpu aukštesnio lygmens duomenų aprašymo modelis – ontologija leidžia klasifikuoti dalykinės srities objektus, nustatyti įvairius jų parametrus bei apribojimus. Be to, ontologija leidžia vykdyti paiešką skirtinguose šaltiniuose, jeigu ontologijos struktūra yra vienoda, ir pateikti vartotojui bendrus rezultatus – visa tai paiešką padaro prasmingesnę. Duomenų išrinkimo proceso taikant ontologijas schema pateikta 2 paveikslėlyje.



2 pav. Duomenų išrinkimo procesas, taikant ontologijas

Tačiau šį duomenų išrinkimo procesą taikant praktikoje susiduriama su sunkumais, kai ontologijos apimtis yra didelė. Šią problemą galima išspręsti saugant ontologiją reliacinėje duomenų bazėje, tam naudojami specialūs algoritmai. Tokiu atveju ontologijos užklausa turi būti transformuojama į *SQL*.

Atlikus duomenų paieškos proceso analizę išsiaiškinta, kad jį galima pagerinti taikant ontologijas, nes jos leidžia detaliau aprašyti dalykinę sritį, tai pat galima automatiškai vykdyti užklausas skirtinguose duomenų šaltiniuose, todėl vartotojui pateikiami tikslesni rezultatai. Taigi kitame skyrelyje bus aprašytos ontologijos, jų kūrimo procesas. Tačiau dideles ontologijas saugoti

tekstiniame faile nėra efektyvu, todėl norint efektyviai vykdyti užklausas, jos turėtų būti saugojamos reliacinėse duomenų bazėse. 2.5 poskyryje analizuojami ontologijų saugojimo reliacinėje duomenų bazėje metodai.

2.2. Ontologijos sąvokų ir kūrimo procesų analizė

Dirbtinio intelekto literatūroje galima rasti daug ontologijos apibūdinimų, tačiau mūsų atveju geriausiai tiktų toks apibūdinimas: ontologija yra formalus ir tikslus dalykinės srities konceptų aprašymas [14]. Ontologija aprašo dalykinės srities esybes, jų sąryšius ir savybes. Klasės yra svarbiausia ontologijos dalis, jos skirtos grupuoti resursus, turinčius panašių požymių [2]. Pavyzdžiui, klasė *Žemynas* aprašo žemynus, todėl visi žemynai priklauso šiai klasei.

Ontologijos kūrimo procesą galima suskirstyti į 7 etapus [14]:

- ontologijos paskirties nustatymas;
- pakartotinio ontologijų panaudojimo galimybių analizė;
- svarbių dalykinės srities terminų nustatymas;
- klasių nustatymas ir klasių hierarchijos formavimas;
- klasių atributų nustatymas;
- atributų apribojimų nustatymas;
- klasių objektų kūrimas.

Nurodyta metodologija yra rekomendacinio pobūdžio, nes nėra vienintelio teisingo ontologijų kūrimo metodo, Toliau šie žingsniai bus apžvelgti detaliau.

- Ontologijos paskirties nustatymas. Nustatoma ontologijos sritis ir apimtis. Galima sudaryti daug tos pačios dalykinės srities ontologijų variantų (pasirinkti skirtingas klases, klases grupuoti į skirtingas hierarchijas ir kt.), todėl svarbu žinoti, kas ir kokiems tikslams naudos ontologiją į kokius klausimus ji turės atsakyti. Naudinga sudaryti pavyzdinių ontologijos klausimų sąrašą, kuriuo remiantis bus galima lengviau nustatyti reikalingas klases, klasių atributus. Pavyzdžiui, jei kuriama filmų ontologija, klausimai galėtų būti tokie: kokiose komedijose vaidina Bruce Willis, kokie filmai, pastatyti nuo 2000 metų yra skirti vaikams, kada filmuotis pradėjo Jim Carrey? Iš šių paprastų klausimėlių galima daug pasakyti apie kuriamą ontologiją: filmų klasifikaciją, filmų atributus, filmų sąsajas su aktoriais ir kt.
- Pakartotinio ontologijų panaudojimo galimybių analizė. Ištiriama, ar jau yra sukurta ontologija, tinkama dalykinei sričiai aprašyti. Rastoji ontologija gali būti panaudojama arba kaip dalis kuriamos ontologijos arba ji gali visiškai aprašyti dalykinę sritį.
- Svarbių dalykinės srities terminų nustatymas. Išvardijamos svarbios dalykinės srities sąvokos, terminai. Kuriant filmų ontologiją, galima išskirti apibrėžti tam tikras esmines

sąvokas: filmas, filmo žanrai (komedija, trileris, siaubo, drama), filmo savybės (sukūrimo metai, vaidinantys aktoriai, auditorijos amžiaus grupė), aktoriai. Šie terminai bus baziniai, jais remiantis vėliau bus nustatomos pagrindinės ontologijos klasės.

- Klasių nustatymas ir klasių hierarchijos formavimas. Ontologijoje dalykinė sritis modeliuojama klasėmis ir klasės egzemplioriais. Klasė yra formalus aprašas, skirtas grupuoti objektus, turinčius panašius požymius. Klasės objektai yra realūs dalykinės srities egzemplioriai. Jei objektas priklauso klasei, jis turi tos klasės savybes. Klasės tarpusavyje gali jungtis ryšiais, sudaryti klasių hierarchiją, todėl objektas, priklausantis žemesniojo lygmens klasei, turi ir aukštesnės klasės savybes. Nustatyti klasių hierarchiją galima vienu iš trijų principų:
    - iš viršaus žemyn – pirmiausia randamos aukščiausio lygmens klasės, kurios vėliau detalizuojamos žemesniojo lygmens klasėmis.
    - iš apačios į viršų – pirmiausia randamos žemiausiojo lygmens klasės, vėliau jos sujungiamos į apibendrinančias aukštesniojo lygmens klases.
    - mišrus – naudojami abu aukščiau aprašyti principai kartu, šiuo atveju pirmiausia gali būti paimama klasė, esanti hierarchijos viduryje, ir vienu metu tiek detalizuojama, tiek apibendrinama.

Nė vienas iš šių trijų principų nėra geresnis už kitus ir rekomenduotinas naudotis, nes klasių hierarchijos sudarymas yra specifinis procesas, priklausantis nuo to, kokia dalykinė sritis modeliuojama. Jeigu iš anksto sunku nustatyti žemesniojo lygio klases, patogiau naudoti principą iš viršaus žemyn.

Sudarant klasių hierarchiją remiamasi ankstesniuose etapuose nustatytais faktais – svarbiausiomis dalykinės srities sąvokomis bei ontologijos paskirtimi. Kuriant filmų ontologiją, aukščiausio lygmens klasė bus *Filmas*, o hierarchinį skirstymą galima atlikti keliais būdais: filmus skirstyti pagal žanrus, tada kuriamos vaikinės klasės *Komedija*, *Trileris*, *Siaubo filmas*, *Drama* ir kitos. Šios klasės turi visas tėvinės klasės *Filmas* savybes, tačiau turi ir papildomų (pavyzdžiui *Siaubo filmas* turi trukmę, pastatymo metus, vaidinančius aktorius ir tai, kas jo egzempliorius skiria nuo kitų – draudžiama žiūrėti vaikams).

Sukūrus dalykinės srities klases, galima iš jų sudaryti išvestines klases. Toks klasių formavimas remiasi aibių teorija. *OWL* kurti išvestines klases galima šiais būdais [15]:

- išvardijant klasės objektus;
- paimant tik tam tikrus kitos klasės objektus;
- dviejų klasių sankirta – tik tie objektai, kurie priklauso ir vienai ir kitai klasei;

- dviejų klasių sąjunga – visi objektai, priklausantys vienai arba kitai klasei;
- klasės inversija – visi objektai, nepriklausantys pasirinktai klasei.

Toks išvestinių klasių formavimas reikalingas norint realizuoti tam tikras sudėtines užklausas, kurioms vykdyti reikalingų klasių ontologijoje nėra, bet jas galima gauti esamų pagrindu, pavyzdžiui, jei reikia išrinkti visus filmus, kurie nėra komedijos, sukuriama klasės *Komedija* inversija, kuriai ir priklauso minėti filmai. Kitas pavyzdys – vartotojui reikia išrinkti filmus, kurie yra arba *Siaubo* arba *Trileriai.* Sukuriama nauja klasė, kuriai priklauso visi *Siaubo* ir *Trileriai* klasių egzemplioriai.

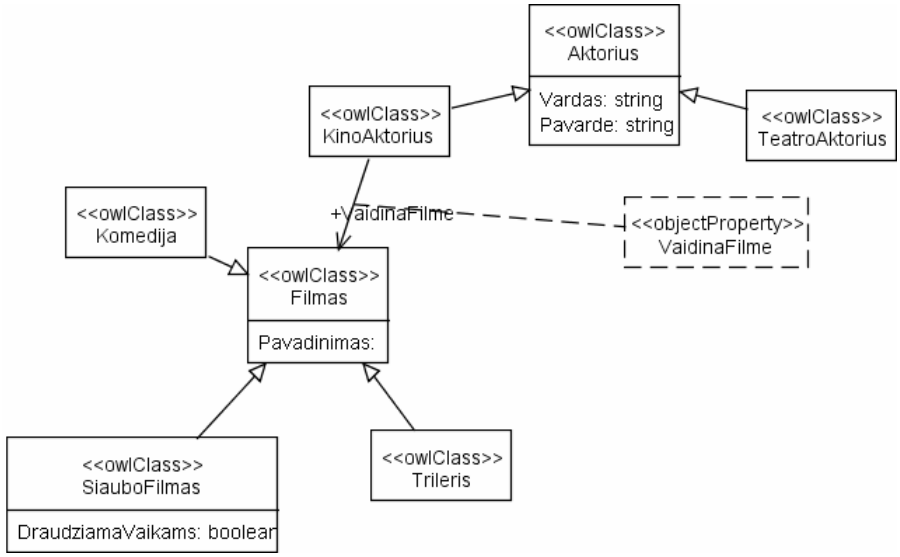
- Klasių atributų nustatymas. Sukūrus klases reikia sukurti kiekvienos klasės vidinę struktūrą – atributus, kurie aprašo klasę, nes vien klasių hierarchija nesuteikia pakankamai informacijos apie dalykinę sritį. Atributas gali būti paprasta reikšmė - tekstinė eilutė gali saugoti filmo pavadinimą, sveikasis skaičius nurodyti pastatymo metus ir kt. Klasės atributas gali būti ir kita klasė, pavyzdžiui, jeigu turime aktorių ir filmų ontologijas, norint klasės *Filmai* egzemplioriui (konkrečiam filmui) priskirti vaidinančius aktorius, galima sukurti ryšį tarp filmo ir aktoriaus.

  Atributai turi būti kuriami kuo aukštesniame abstrakcijos lygmenyje, t.y. jei visos to paties lygmens klasės turi tą patį atributą, tai jį reikia kurti ne kiekvienai atskirai, bet apibendrinančiai klasei.
- Atributų apribojimų nustatymas. Sukūrus klasių atributus, nustatomi jų apribojimai, t.y. kokias reikšmes jie gali įgyti. Pirmoji apribojimų rūšis yra kardinalumo apribojimas, kuris nusako, kiek reikšmių atributas gali turėti, pavyzdžiui filmo pavadinimas arba pastatymo metai gali įgyti tik vieną reikšmę, tačiau atributas, nusakantis filme vaidinančius aktorius, gali turėti daug reikšmių. Kita apribojimų rūšis – atributo reikšmės tipas. Atributai būna tekstinių, skaitinių, loginių reikšmių, objekto tipo (išvardijami kitos klasės objektai, sudarantys ryšį su aprašomu objektu), galima nurodyti, kad atributas įgyja tik vieną iš išvardintų reikšmių (pavyzdžiui, filmo metai gali įgyti reikšmes tik iš tam tikro intervalo).
- Klasių objektų kūrimas. Paskutinis ontologijos kūrimo etapas – klasių objektų sukūrimas. Sukuriami realūs dalykinės egzemplioriai (filmai, aktoriai), nustatomos jų atributų reikšmės.

Sukurta ontologija yra detalus dalykinės srities modelis.

Toliau pateikiama pavyzdinė ontologija (3 pav.). Joje dalykinės srities objektai suskirstyti į dvi klases: *Filmas* ir *Aktorius.* Šios klasės taip pat detalizuojamos į žemesnio lygio klases, turinčias papildomų atributų: klasė *Filmas* į *Komedija*, *Trileris* ir *Siaubo filmas*, *Aktorius* į *Kino aktorius* ir *Teatro aktorius*. Klasė filmas turi jungiantį klasės tipo atributą *vaidina aktoriai,* kuris sujungia

klases *Filmas* ir *Kino aktorius*. Jungimo apribojimas yra *allValuesFrom*, tai reiškia, kad atributo reikšmės gali būti tik klasės *Kino aktorius* objektai (filme gali vaidinti tik kino aktoriai). Todėl kompiuterio programa, suradusi, kad filme vaidina aktorius, gali daryti išvadą, kad šis aktorius priklauso klasei *Kino aktorius*.



3 pav. Filmų ir aktorių ontologija

Šiame skyrelyje buvo supažindinta su ontologijomis, jų kūrimo principais, toliau bus analizuojamos ontologijų aprašymo kalbos *RDF*, *RDFS* ir *OWL*, kokias ontologijų aprašymo galimybes jos turi, bei ontologijos kūrimo įrankio *Protégé* analizė.

2.3. Ontologijos kalbų analizė

Pagrindinė ontologijos aprašymo kalba yra *RDF*. *RDF* yra metaduomenų apdorojimo karkasas, aprašantis resursus bei jų formuluotes. *RDF* pagrindas yra modelis, skirtas atvaizduoti savybės ir savybių reikšmes. *RDF* modelis remiasi skirtingų duomenų vaizdavimo bendruomenių principais. *RDF* savybės gali būti laikomos resursų atributais, jos taip pat reiškia ryšius tarp resursų, tuo *RDF* primena *ER* diagramą. Objektinio projektavimo požiūriu, resursai atitinka objektus, o savybės objektų kintamuosius [13].

RDF modelis sudarytas iš trijų pagrindinių dalių:

- resursas – bet koks daiktas, aprašytas *RDF*, yra resursas. Tai gali būti visas interneto puslapis, dalis interneto puslapio ar, tarkime, knyga. Kiekvienas resursas turi turėti *URI* - adresą, kuris jį identifikuoja.
- savybė – charakteristika, atributas, ryšys, skirtas apibūdinti resursą.
- formuluotė – resursas, kartu su savybe ir jos reikšme vadinamas *RDF* formuluote.

Šios trys formuluotės dalys yra vadinamos veiksniu (*subject*), tariniu (*predicate*) ir objektu (*object*). Objektas (savybės reikšmė) gali būti kitas resursas arba duomenų tipo reikšmė.

Tokia duomenų struktūra yra tinkama aprašyti didžiąją daugumą informacijos, apdorojamos kompiuterių [3]. Išnagrinėkime pavyzdį „Dangus yra mėlynos spalvos“. Šioje formuluotėje veiksnys yra „Dangus“, jo savybė – „turi spalvą“, kurios reikšmė – „mėlyna“. Formulutė pateikta 1 lentelėje.

1 lentelė. RDF formuluotės pavyzdys

Veiksnys (resursas)	http://www.pavyzdys.lt/Dangus
Tarinys (savybė)	turiSpalvą
Objektas (reikšmė)	„Mėlyna“

Vienas iš *RDF* modelio privalumų yra atviro pasaulio prielaida [5]. Tiesa yra tai, kas modelyje pasakyta, tačiau tų dalykų, kurie nepaminėti, mes nelaikome netiesa, todėl modelį lengva papildyti naujais faktais, nepažeidžiant esamų faktų teisingumo.

Siekiant išplėsti semantines *RDF* galimybes, buvo sukurtos dvi papildomos kalbos: *RDF Schema* (*RDFS*) ir *OWL*. *RDFS* yra paremta *RDF* kalba, turi visas jos ir papildomų savybių, iš kurių svarbiausios yra klasių ir savybių apribojimų kūrimas. Jei *RDF* leidžia kurti resursus, tai *RDFS* leidžia juos grupuoti į klases. *RDFS* tai pat leidžia kurti klasių hierarchiją, tam naudojama savybė *rdfs:subClassOf* kiekviena klasė gali turėti neribojamą skaičių tėvinių klasių. Hierarchiją galima kurti ir savybėms, tam skirtas *rdfs:subPropertyOf* savybė. Kita *RDFS* galimybė – savybių apribojimas, kai savybę leidžiama naudoti tik tam tikrų klasių egzemplioriams, nurodant *domain* ir *range*.

OWL yra *RDFS* pagrindu sukurta kalba, kuri dar labiau išplečia ontologijos aprašymo galimybes. *OWL* papildymus galima skirstyti į kelias grupes [7]:

- ekvivalentiškumas – leidžia nustatyti, kad du resursai (klasės, individai ar savybės) aprašo tą patį realaus pasaulio objektą. Taip pat leidžiama apibrėžti, kad dvi klasės yra skirtingos (*distinct*), t.y., jei objektas priklauso vienai klasei, jis negali priklausyti kitai.
- savybių charakteristikos, apribojimai – *OWL* suteikia galimybes plačiau aprašyti savybes (tranzityvumą, simetriškumą, invertiškumą), nustatyti jų tarpusavio sąryšius, kardinalumą.
- aibių savybės – naujų klasių kūrimo galimybė, pasinaudojant esamomis klasėmis, sudarant jų sąjungas, sankirtas.

Ontologijos aprašymo kalbos yra pakankamai išvystytos, jos leidžia detaliai aprašyti ontologiją, jos apribojimus, klasifikuoti informacijos resursus.

2.4. Ontologijos kūrimo įrankio *Protégé* analizė

Protégé yra ontologijų redagavimo įrankis, padedantis lengviau kurti dalykinės srities ontologijas [11]. *Protégé* suteikia grafinę sąsają kurti klases, klasių savybes, nustatyti savybių

apribojimus. Šis įrankis leidžia patikrinti ontologijos suderinamumą bei pilnumą, turi daug naudingų priedų (grafinis ontologijos vaizdavimas, *DB* schemos ir duomenų importavimas iš reliacinių bazių į *OWL* ir kt.).

Kuriant ontologijas *Protégé* įrankiu, naudojamos trys svarbiausios sąvokos:

- objektas – bet koks realus objektas. Skirtumas tarp *OWL* ir *Protégé* objektų yra toks, kad *Protégé* vienas objektas gali turėti tik vieną pavadinimą, tuo tarpu *OWL* tas pats objektas gali turėti skirtingus pavadinimus.
- savybė – dvinaris ryšys tarp dviejų objektų, t.y. savybė sujungia du objektus ryšiu. Pavyzdžiui savybė *turiBrolį* gali sujungti objektus *Jonas* ir *Tomas.* Savybės gali būti hierarchinės, tarkime, savybė *turiGiminaitį* galėtų būti tėvinė savybei *turiBrolį.* Savybė sujungia dviejų klasių objektus, todėl ji turi turėti domeną (*domain*) ir sritį (*range*). Savybės jungia klasės, priklausančios domenui objektus su klasės, priklausančios sričiai objektais.
- klasė – objektų aibė.

Protégé savybės gali turėti įvairius apribojimus. Vienas iš pavyzdžių, kada prireikia nustatyti, kad ryšys tarp objektų būtų vienos krypties (savybė *turiTėvą* tarp *Jonas* ir *Petras* gali būti tik vienkryptė, kadangi *Petras* yra *Jono* tėvas), arba dviejų krypčių (savybė *turiBrolį* tarp objektų *Jonas* ir *Tomas* privalo būti dvikryptė, nes abu yra broliai vienas kitam). Todėl savybės yra skirstomos:

- funkcinės – jeigu objektas turi funkcinę savybę, reiškia šia savybe jis gali būti susietas daugiausia su vienu objektu, todėl objektą *a* sujungę su *b*, *a* ta pačia funkcine savybe daugiau negalėsime sujungti su jokiu kitu objektu.
- Atvirkštinės funkcinės – atvirkštinė jos savybė yra funkcinė. Jei tarp *a* ir *b* galioja atvirkštinis funkcinis ryšys, reiškia *b* šia savybe galime sujungti tik su vienu *a*.
- Tranzityviosios – jeigu tranzityvioji savybė sujungia objektus *a* su *b* ir *b* su *c*, tada ta savybė sujungia ir objektus *a* su *c*.
- Simetrinės – jeigu simetrinė savybė sujungia objektus *a* su *b*, tai ji sujungia ir *b* su *a*.
- Nesimetrinės – jeigu nesimetrinė savybė sujungia objektus *a* su *b*, tai ji negali sujungti ir *b* su *a*.
- Refleksyviosios – savybės, sujungiančios objektą su juo pačiu.
- Nerefleksyviosios – savybės galinčios sujungti tik du atskirus objektus, bet ne tą patį.

Kad ontologija būtų tiksliau aprašyta, ta pati savybė gali būti kelių tipų, pavyzdžiui, savybė *turiMamą* yra funkcinė, nes asmuo gali turėti tik vieną mamą, ši savybė taip pat yra nesimetrinė, nes atvirkštinė priklausomybė negalioja.

Dažnai savybėms prireikia nurodyti įvairius apribojimus. Pavyzdžiui, norint rasti asmenis, turinčius tris brolius, reikia klasės, kurios objektai turi tris ryšius *turiBrolį.* Tokiais atvejais klasėms gali būti nustatomi trijų tipų apribojimai:

- *Kiekio apribojimai*. Dar skirstomi į egzistencinius ir universalius apribojimus. Egzistenciniai apribojimai dažniausiai naudojami *OWL* apribojimai, jie aprašo klasę, kuri turi bent vieną ryšį (atitinka *OWL someValuesFrom*) su kitos nurodytos klasės objektais. Tarkime, norime pasakyti, kad kiekvienas automobilis turi žibintą. Savybe *turiDetalę* sujungiame klases *Automobilis* ir *Žibintas:*

  Automobilis turiDetalę some Žibintas

Taigi bet kuris klasės *Automobilis* objektas dabar turi turėti sąryšį su bent vienu klasės *Žibintas* objektu.

Universalūs (atitinka *OWL allValuesFrom*) apribojimai aprašo klasę, kuri aprašoma savybe jungiasi tik su nurodytos klasės elementais. *Protégé* toks apribojimas aprašomas taip:

Žmogus turiBrolį only Vyras

Šis teiginys reiškia, kad brolis gali būti tik vyriškos giminės.

- *Kardinalumo apribojimai*. *OWL* galima pažymėti, kad klasės objektai gali jungtis su kitos klasės objektais nustatytą kiekį kartų (mažiausiai, daugiausiai, lygiai tiek). Pavyzdžiui norėdami pasakyti, kad automobilis turi daugiausiai 4 ratus, sukuriame savybę *turiDetalę*, kuri jungia klases *Automobilis* ir *Ratas* su apribojimu *max 4*. *Protégé* tai aprašoma taip:

  Automobilis turiDetalę max 4 Ratas

- *hasValue apribojimai* naudojami, kada norima klasei priskirti reikšmę (objektą). Šis ryšys ypatingas tuo, kad jungia klasę su objektu (prieš tai aptarti ryšiai jungė dviejų klasių objektus). *Protégé* tai aprašoma taip:

  OpelAutomobilis yraPagamintas hasValue Vokietija

  Čia klasė *OpelAutomobilis* sujungta su objektu *Vokietija*.

Protégé įrankis leidžia lengvai kurti ontologijas ir išsaugoti jas tekstiniame faile. Šis įrankis išnaudoja visas *OWL* ontologijos aprašymo galimybes. Šiuo įrankiu sukurta ontologija turi būti konvertuojama į reliacinės duomenų bazės schemą, todėl kitame skyrelyje bus analizuojami ontologijos konvertavimo į reliacinę duomenų bazę algoritmai.

2.5. Ontologijos saugojimo reliacinėje duomenų bazėje metodų analizė

Ontologija gali būti efektyviai saugojama reliacinėje duomenų bazėje [1]. Saugojimui gali būti naudojami įvairūs algoritmai, kurie tarpusavyje skiriasi *RDF* grafo saugojimo metodu. Vieni jų ontologiją saugo pastovioje duomenų bazės schemoje, kitų schema priklauso nuo ontologijos

schemos. Algoritmo pasirinkimas priklauso nuo ontologijos paskirties ir struktūros. Ontologijoms, kurių klasių hierarchija yra didelė, sudėtinga bei nepastovi, geriau naudoti nuo struktūros nepriklausomą schemą, tuo tarpu ontologijoms, kurių klasių hierarchija nekinta, geriau naudoti nuo struktūros priklausomą algoritmą, nes jis padeda paprasčiau atskirti skirtingų klasių individus. Toliau bus aprašyti skirtingi ontologijos saugojimo metodai [10].

2.5.1. Horizontali lentelė

Vienas iš būdų, saugoti ontologiją reliacinėje duomenų bazėje – naudoti horizontalią schemą. Šiuo atveju visai ontologijai saugoti naudojama viena lentelė. Lentelėje saugomi individai. Lentelė turi stulpelius individo *ID*, klasei, savybėms saugoti. Šis metodas turi trūkumų, nes kiekvienos savybės reikšmei saugoti naudojamas atskiras stulpelis, todėl savybė gali įgyti tik vieną reikšmę, taip pat individai dažnai turi nulinių savybių reikšmių. Taip pat yra ištirta, kad didelės apimties ontologijų įkėlimo laikas yra didesnis negu naudojant kitus metodus. Šio metodo duomenų pavyzdys pateiktas 2 lentelėje.

2 lentelė. Horizontalios lentelės pavyzdys

Individo ID	Tipas	Savybė_1	Savybė_2	...	Savybė_n
...#1	Klasė_1	Reikšmė_a	Reikšmė_b		Reikšmė_c
...#2	Klasė_2	Reikšmė_d	Reikšmė_e		Reikšmė_f

2.5.2. Vertikali lentelė

Kitas metodas saugoti ontologiją reliacinėje duomenų bazėje – naudoti vertikalią lentelę. Šiuo atveju taip pat naudojama tik viena lentelė, bet jos stulpelių skaičius yra pastovus. Stulpelių yra trys, *RDF subject*, *predicate* ir *object* elementams. Kadangi metodas turi pačią paprasčiausią duomenų bazę, ontologiją į gali būti įkeliama labai paprastai. Šio metodo minusas yra toks, kad *SQL* užklausos turi būti vykdomos visoje duomenų bazėje, o tai užtrunka gana ilgai. Rasti klasių hierarchiją tokioje lentelėje taip pat sudėtinga, tam reikia įvykdyti ne vieną užklausą. Lentelės pavyzdys pateiktas 3 lentelėje.

3 lentelė. Vertikalios lentelės pavyzdys

Subject	*Predicate*	*Object*
…#1	Tipas	Klasė_1
...#2	Savybė_a	Reikšmė_a

2.5.3. Horizontali klasė

Šis metodas yra panašus į horizontalios lentelės metodą, tačiau skiriasi nuo jo tuo, kad atskira lentelė kuriama kiekvienai klasei, todėl nulinių atributų reikšmių skaičius sumažėja. Šis metodas

leidžia efektyviai vykdyti individų paieškos užklausas. Šio metodo lentelių struktūra tokia pati, kaip pateikta 3 lentelėje.

2.5.4. Lentelė kiekvienai savybei

Naudojant šį metodą, atskira lentelė kuriama kiekvienai savybei (*predicate*). Tarkime, visos *rdf:type* arba objektų savybės bus saugojamos atskirose lentelėse. Paprastoms užklausoms metodas tinkamas, tačiau sudėtingesnes vykdyti yra sunkiau, nes reikia naudoti daug *JOIN* operacijų. 4 lentelėje pateiktas savybės *rdf:type* lentelės pavyzdys.

4 lentelė. Savybės rdf:type lentelės pavyzdys

Subject	*Object*
...#1	Klasė_A
...#2	Klasė_B

2.5.5. OWL2RDB algoritmas

Šis algoritmas kiekvienai klasei sukuria po lentelę, klasės individai saugojami kaip lentelės įrašai. Šiuo požiūriu jis panašus į horizontalios klasės metodą. Tačiau už jį *OWL2RDB* [21] algoritmas pranašesnis dėl to, kad naudoja po atskirą lentelę ir objektų savybėms, todėl nelieka nulinių savybių reikšmių problemos, kaip horizontalios lentelės atveju. Šis metodas taip pat leidžia išsaugoti klasių hierarchiją reliacinės duomenų bazės schemoje, nes tarp klasių lentelių sukuria hierarchinius 1:0..1 ryšius. Ontologijos apribojimai saugojami atskiroje meta duomenų bazėje. Tačiau šio metodo minusas – sudėtingas ontologijos išplėtimas, nes ontologijoje atsiradus naujai klasei, reikia kurti naują lentelę ir jos sąryšius su kitomis klasių lentelėmis.

Algoritmas susideda iš šių žingsnių:

- *OWL* klasių transformavimas į reliacinės *DB* lenteles. Analizuojama dalykinės srities ontologija ir kiekvienai klasei sukuriama reliacinės *DB* lentelė. Nustatomi ryšiai vienas-su-vienu tarp tėvinių ir dukterinių lentelių.
- Klasių objektų savybių transformavimas į reliacinių *DB* sąryšius. *OWL* savybė yra dvinaris ryšys. Objektų savybė yra ryšys tarp dviejų klasių objektų. Žemiau pateikta pavyzdinis sąryšis tarp dviejų klasių:

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

Čia nustatytas ryšys tarp klasių *Wine* ir *WineGrape.* Algoritmas tokiam ryšiui pavaizduoti reliacinėje *DB*, sukurtų tarpinę lentelę *madeFromGrape*.

- Duomenų tipų savybių transformavimas į reliacinių *DB* lentelių atributus. Duomenų tipo savybė sujungia klasės objektą su duomenų tipu. Algoritmas klasės lentelei sukuria naują atributą.

```
<owl:Class rdf:ID="VintageYear" />
<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear" />
  <rdfs:range  rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

- Ontologijos metaduomenų bazės užpildymas apribojimais. Apribojimams (*allValuesFrom*, *someValuesFrom*, *cardinality*, *hasValue, equivalentClasses, disjointClasses, propertyChain, equivalentGroup, disjointGroup, disjointUnion*) naudojama atskira meta duomenų bazė.
- Lentelių užpildymas ontologijos individais.

Išanalizavus ontologijos saugojimo reliacinėje duomenų bazėje algoritmus, nuspręsta naudoti *OWL2RDB* algoritmą, nes jis geriau išnaudoja reliacinės duomenų bazės privalumus ontologijai saugoti – ne tik sukuria klasę kiekvienai lentelei, bet ir nustato klasių hierarchiją - tai palengvina individų ir klasių paiešką. Be to šis algoritmas ontologijos apribojimus saugo atskiroje meta duomenų bazėje todėl yra išplečiamas ir suderinamas su nuolat besivystančia *OWL2* ontologijų kalba.

Pasirinktas ontologijų išsaugojimo metodas neturi užklausų vykdymo mechanizmo, todėl kituose skyreliuose bus analizuojamos užklausų vykdymo ontologijoje galimybės ir egzistuojantys užklausų vykdymo sprendimai.

2.6. Užklausų vykdymas ontologijoje

RDF yra duomenų modelis, skirtas žiniatinklio resursų aprašymui, tai yra pats tinkamiausias standartas duomenų aprašymui ir mainams semantiniam žiniatinkliui [9]. *RDF* buvo pristatytas 1998 metais. Tuomet iškilo duomenų išrinkimo problema. Buvo pasiūlyta įvairių užklausų kalbų (*RQL*, *SeRQL*, *TRIPLE*, *RDQL*, *N3*, *Versa*) [9], tačiau 2004 metais buvo pristatytas *SPARQL* prototipas [16], o 2008 metais *SPARQL* tapo oficialia *W3C* rekomendacija. Taigi *SPARQL* dabar yra tipinė ontologijų užklausų kalba [18].

SPARQL paremta *RDF* grafo atitikimo mechanizmu [17]. Šis grafas yra *RDF* formuluočių aibė. Užklausa taip pat susideda iš formuluočių, kurios yra lyginamos su duomenų šaltinio formuluotėmis, šio palyginimo rezultatai grąžinami vartotojui. Taigi *SPARQL* užklausa susideda iš trijų pagrindinių dalių, kurios aprašytos žemiau:

- sąlyginė dalis (angl. *pattern matcing part*) - užklausos *where* dalis. Joje aprašomas grafo atitikimo šablonas, kuris, vykdant užklausą bus lyginamas su *RDF* grafo struktūra ir taip

gaunami rezultatai. Šioje dalyje, norint gauti tikslesnius rezultatus, gali būti naudojami papildomi raktiniai žodžiai:

- *optional* – suteikia galimybę pridėti prie rezultatų eilutės informaciją tik tada, jeigu ji egzistuoja ir neatmesti visos eilutės, jei tik viena užklausos sąlygos dalis neatitinka *RDF* grafo. Pavyzdžiui, vartotojo elektroninio pašto adreso paieška, jeigu jis jį turi. Nenaudojant *optional*, vartotojas, neturintis elektroninio pašto adreso apskritai nebūtų surastas:

```
{?x foaf:name  ?name . OPTIONAL { ?x  foaf:mbox  ?mbox }}
```

- *union* – skirtingų rezultatų rinkinių sujungimas į vieną. Tai naudinga ieškant informacijos skirtingose ontologijose, tarkime knygų paieškos skirtingose ontologijose rezultatai gali būti sujungti ir vartotojui pateiktas bendras sąrašas:

```
{?book dc10:title  ?title } UNION {?book dc11:title  ?title}
```

- *filter* – užklausos sąlygos dalyje parašytas *filter* modifikatorius leidžia išrinkti rezultatus pagal norimas sąlygas. Leidžia filtruoti įvairių duomenų tipų rezultatus. Tekstiniams įrašams gali būti naudojamos ir reguliariosios išraiškos. Tai leidžia iš knygų sąrašo išrinkti knygas, kurios nėra brangesnės už 50 Lt., arba rasti knygų sąrašą, kurio pavadinime būtų žodis „animals“:

```
{?x dc:title ?title FILTER regex(?title, "^animals")}
```

- Sprendinio modifikavimas (angl. *solution modifiers*) – leidžia modifikuoti jau surastos informacijos išvedimą vartotojui. Lentelės pavidalu išvesti rezultatai gali būti modifikuojami šiomis operacijomis:
  - *projection* – leidžiama vartotojui pateikti tik tą informaciją, kuri nurodyta *select* dalyje. *where* dalyje gali būti naudojama daug kintamųjų, tačiau ne visi jie turi būti pateikiami vartotojui išvedant rezultatus. Žemiau pateiktame pavyzdyje sąlygos dalyje naudojami *?x* ir *?name* kintamieji, tačiau vartotojui pateikiamas tik *?name*. Tai atliekama formuojant naują rezultatų rinkinį pagal *select* dalyje nurodytus kintamuosius.

```
SELECT ?name
WHERE
 { ?x foaf:name ?name }
```

  - *distinct* – pašalina besidubliuojančias rezultatų eilutes. Jeigu randamos dvi eilutės, kurių visos reikšmės sutampa, viena iš jų pašalinama.

- *order* – rezultatų rikiavimas pagal nurodytą kintamąjį. Gali būti naudojami *asc* arba *desc* raktiniai žodžiai, rikiavimo tvarkai nurodyti.
- *limit* – leidžia nurodyti maksimalų pateikiamų rezultatų eilučių skaičių.
- *offset* – leidžia nurodyti, nuo kurios eilutės rodyti rezultatus.

- Išvedimo dalis (angl. *output*). Ši dalis aprašo 4 skirtingas *SPARQL* užklausų rūšis. Jos visos informacijos ieško tuo pačiu principu, t.y. *where* dalyje esančio šablono atitikimu *RDF* grafui, tačiau skiriasi rezultatais ir jų pateikimu:
  - *select* – įprasta *SPARQL* užklausa, rezultatai priskiriami kintamiesiems, pažymėtiems ? ženklu. Žemiau pateikta užklausa išrenka visas Lietuvos kaimynines valstybes:

```
PREFIX geo: <http://www.fao.org/aims/geopolitical.owl#>
select ?x
where {
          ?x geo:hasBorderWith geo:Lithuania
}
```

  - *ask* – grąžina atsakymą, ar pateikta sąlyga atitinka grafą. Šis užklausos tipas gali grąžinti tik dvi reikšmes: *yes* arba *no*. Kadangi informacijos negrąžina, užklausa gali būti naudojami tik testavimo tikslais. Žemiau pateikta užklausa atsako, ar Lietuva turi bent vieną kaimynę:

```
PREFIX geo: <http://www.fao.org/aims/geopolitical.owl#>
ASK
where {
          ?x geo:hasBorderWith geo:Lithuania
}
```

  - *describe* – grąžina informaciją apie norimą resursą. Užklausos klientui nereikia žinoti *RDF* grafo struktūros, kaip įprastos *select* užklausos atveju, struktūrą išgauna *SPARQL* procesorius, kuris *XML* formatu grąžina visus resursus, su kuriais susijęs pateiktas resursas. Pateikta užklausa grąžina išsamų Lietuvos kaimynių aprašymą:

```
PREFIX geo: <http://www.fao.org/aims/geopolitical.owl#>
describe ?x
where {
          ?x geo:hasBorderWith geo:Lithuania
}
```

  - *construct* - formuoja naują grafą pagal pateiktą šabloną. Žemiau pateiktas užklausos pavyzdys suformuoja naują *RDF* grafą, kuriame Lietuva sujungta su kaimyninėmis valstybėmis savybe *hasNeighbour*:

```
PREFIX geo: <http://www.fao.org/aims/geopolitical.owl#>
```

```
construct {
          ?x geo:hasNeighbour geo:Lithuania
}
where {
          ?x geo:hasBorderWith geo:Lithuania
}
```

SPARQL yra gana nauja ir sparčiai besivystanti užklausų kalba. Originali *SPARQL/Query 1.0* versija neturi visų reikalingų savybių, tačiau kūrėjai sulaukia pageidavimų iš vartotojų bendruomenės ir stengiasi papildyti *SPARQL* naujomis savybėmis, todėl naujoje *SPARQL/Query 1.1* versijoje atsirado patobulinimų, kurie palengvina užklausų rašymą, jos tampa paprastesnėmis. Šie atnaujinimai kol kas dar nėra plačiai naudojami, jie įdiegti tik kai kuriose užklausų vykdymo sistemose. Toliau bus apžvelgti svarbiausi atnaujinimai [12].

Pirmiausia siūloma įvesti agregavimo funkcijas, kurios leistų vykdyti skaičiavimo veiksmus, rasti minimalią, maksimalią reikšmę, suskaičiuoti vidurkį. Rezultato rinkinys būtų grupuojamas į agregavimo grupes, vartotojui būtų pateikta tiek eilučių, kiek yra agregavimo grupių, su kiekvienos grupės suskaičiuota tam tikra reikšme (pavyzdžiui, suskaičiuotas kiekvienos valstybės turistų skaičius).

Kitas atnaujinimas – *subqueries*, kurios leidžia vykdyti vieną užklausą kitoje. Tai būna naudinga, kai kiekvienam rezultatų eilutės elementui reikia rasti papildomą informaciją, tarkime žmonių sąraše rasti kiekvienam žmogui reikia surasti vardą.

Paneigimas (*negation*) – tikrinimas, ar *RDF* grafe nėra nurodyto formuluotės. Tai dar vienas patobulinimas, kuris gali būti naudojamas norint rasti žmones, kurie nepažįsta nurodyto žmogaus, arba rasti valstybes, kurios nėra tam tikros valstybės kaimynės. Nors tai įmanoma ir dabartinėmis priemonėmis, naudojant *filter* raktinį žodį, tačiau toks užrašymas nėra pakankamai intuityvus.

2.7. Esamų užklausų vykdymo sprendimų analizė

Šiame skyrelyje analizuojami sprendimai, kurie gali būti suderinami su ontologija, saugoma reliacinėje duomenų bazėje.

Pellet OWL [8] – Java biblioteka, vykdanti užklausas tekstiniuose failuose saugomose ontologijose. Ši biblioteka išanalizavusi tekstinį *OWL* failą sukuria *Java* objektus ir juose vykdo užklausas. Ši biblioteka teikia programavimo sąsają (*API*), ontologijos modelį kurti programiškai.

SPASQL (*SPARQL-inside-SQL*) [20] – *SQL* standarto išplėtimas, gali būti įgyvendinamas *DBVS* viduje arba kaip papildomas *DBVS* modulis. *SPARQL* užklausa nėra perrašoma, kaip kituose sprendimuose, bet paduodama tiesiogiai *DBVS*. Kita savybė – nereikia saugoti *RDF* grafo, užklausos vykdomos reliacinėje duomenų bazėje, todėl veikimo laikas sutrumpėja, tačiau semantinis duomenų interpretavimas abejotinas. Kitas šio sprendimo minusas tas, kad suderinama tik su *MySQL DBVS*.

SparqlEngineDb [4] - *PHP* kalba parašytas *SPARQL-SQL* transliatorius. Grafo viršūnės saugomos ne *OWL* faile, bet duomenų bazės lentelėje, todėl nereikia grafo įrašyti į pagrindinę atmintį - tai pasiteisina esant didelėms duomenų apimtims, tačiau dėl to paieškos negalima vykdyti keliuose šaltiniuose.

SPARQL-to-SQL [6] – *SPARQL* užklausų konvertavimo į *SQL* algoritmas, naudojantis *RDFLib* ontologijos saugojimo mechanizmą, kuris ontologiją saugo reliacinėje duomenų bazėje, trijose lentelėse.

5 lentelėje pateikiami užklausų vykdymo ontologijoje sprendimai.

5 lentelė. Ontologijos užklausų vertimo į SQL įrankių palyginimas

	Veikimo principas	**Suderinamos *DBVS***	***RDF* grafo saugojimas**	**Išplečiamumas bei įmanomas suderinamumas su *OWL2RDB***
Pellet OWL	*Java* biblioteka	-	*Jena* karkasas	+
SPASQL	*DBVS* modulis	*MySQL*	Nėra	-
SparqlEngineDb	Serverio programa	*MySQL*, *MS SQL*, *Oracle* (bet kuri, turinti prisijungimą per *PHP*)	*RDB* lentelė	-
SPARQL-to-SQL	Algoritmas	Nenurodyta	*RDFLib*	-

Visiškai tinkamo metodo vykdyti semantines užklausas reliacinėje duomenų bazėje *OWL2RDB* algoritmu išsaugotoje ontologijoje nėra, tačiau galima pritaikyti *Pellet OWL* bibliotekos funkcionalumą, nes ji yra išplečiamas ir suteikia galimybę ontologijos modelį kurti programiniu būdu. Kitame skyriuje bus plačiau išnagrinėtos *Pellet OWL Reasoner* galimybės.

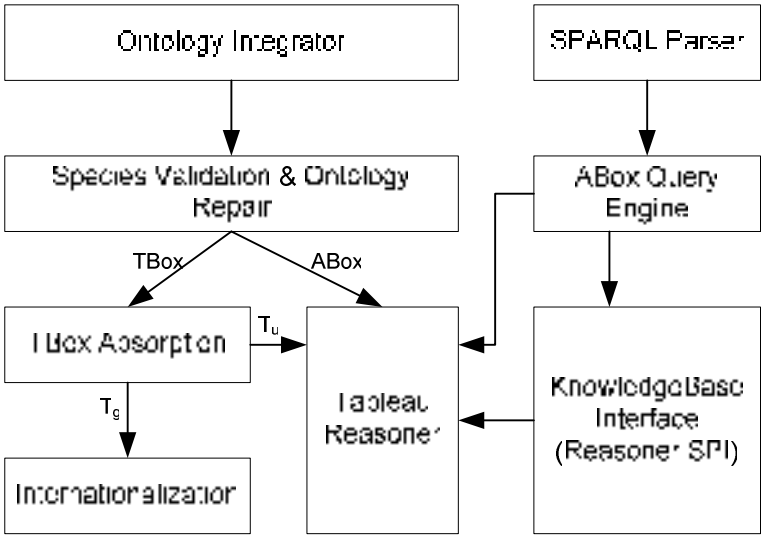
2.8. *Pellet OWL Reasoner* analizė

Pellet OWL Reasoner - *Java* kalba parašyta biblioteka, skirta ontologijoms analizuoti, užklausoms jose vykdyti. Toliau pateikiamos pagrindinės šios bibliotekos atliekamos funkcijos:

- ontologijos pilnumo tikrinimas – užtikrina, kad ontologijoje nebūtų prieštaringų faktų, pavyzdžiui, kad objektas nepriklausytų skirtingoms klasėms;

- klasių objektų galimumo patikrinimas – tikrinama, ar klasė gali turėti objektus. Jei randamas objektas klasės, kuriai negalima jų turėti, vadinasi ontologija nėra suderinta.
- klasifikavimas – nustatomi ryšiai tarp klasių, sukuriama klasių hierarchija, kuri vėliau gali būti naudojama užklausose, pavyzdžiui surasti tiesiogines ar visas klasės poklases;
- realizavimas – randama pati detaliausia objekto klasė, jo tiesioginis tipas. Realizavimo operacija gali būti atlikta po klasifikavimo operacijos, kada sudaryta klasių hierarchija. Naudojant šią klasifikaciją galima rasti ir visas klases, kurioms netiesiogiai priklauso objektas.

Pellet OWL bibliotekos principinė schema pateikta 4 paveikslėlyje.



4 pav. Principinė Pellet OWL Reasoner schema

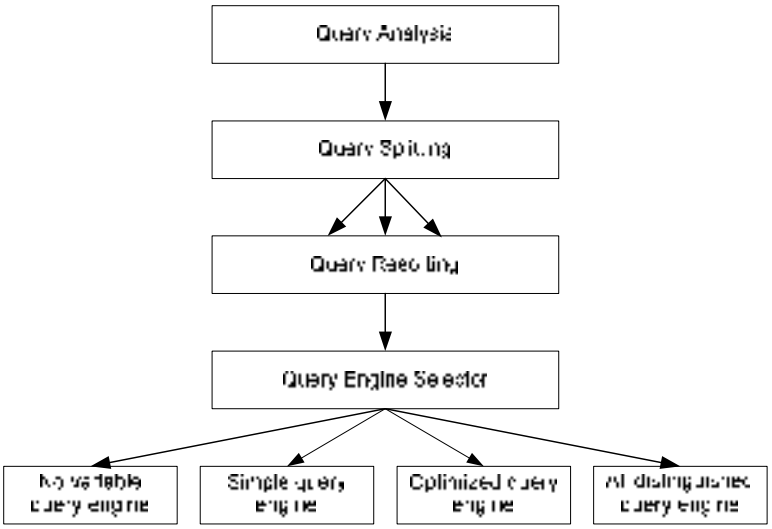
Prieš patekdama į šį komponentą ontologija sintaksiškai patikrinama, kad visi resursai būtų tinkamai aprašyti trigubu formatu: veiksnys (*subject*), tarinys (*predicate*) ir objektas (*object*). Radus klaidų, ontologija pakoreguojama remiantis euristiniais algoritmais, kurie stengiasi atspėti neteisingai aprašyto resurso tipą, pavyzdžiui resursas, naudojamas kaip tarinys yra paverčiamas savybe.

Kitas svarbus uždavinys yra žodyno atskyrimas, kai siekiama, kad klasės, savybės ir objektai nesikirstų. Jeigu biblioteka aptinka, kad resursas naudojamas ir kaip klasė ir kaip objektas ar savybė, vartotojui leidžiama pasirinkti, kaip turi būti apdorojama ontologija: ignoruoti neteisingai aprašytus resursus, visiškai nutraukti ontologijos nagrinėjimą arba priimti viską kaip yra ir resursą, aprašytą kaip klasė ir kaip savybė, skirtingose situacijose, priklausomai nuo užklausos, laikyti klase arba savybe.

Patikrinta ir pakoreguota ontologija patenka į *TBox* ir *ABox* komponentus. Į *TBox* komponentą keliauja aksiomos apie klases, o į *ABox* komponentą – teiginiai apie objektus. Kartu šie komponentai sudaro žinių bazę (*knowledge base*) - pilną OWL ontologiją. Žinių bazės pilnumo tikrinimui skirtas *Tableau Reasoner* komponentas, jis bando sudaryti ontologijos modelį, kuris tenkintų visas ontologijos aksiomas ir faktus.

Žinių bazės sąsajos (*knowledge base interface*) komponentas atsako į paprastas atomines ontologijos užklausas, pavyzdžiui, gauti vaikines ar nesikertančias klases, funkcines savybes ir pan. Jis taip pat gali atsakyti ir į logines užklausas, šiuo atveju sprendžiama nepatenkinamumo problema. Užklausoms, kurioms reikalingi keli atsakymai, atliekami sudėtingi ontologijos tikrinimai. Pavyzdžiui, jei norima rasti visus vienos klasės objektus, pirmiausia randami tiesioginiai tos klasės objektai, vėliau tikrinamas kiekvienas objektas, ar jis nepriklauso klasei per klasių hierarchijos sąryšius.

Žinių bazės sąsaja yra sujungta su *ABox* užklausos varikliu (*ABox query engine*), kuris gali atsakyti ne tik į atomines bet ir sudėtines užklausas (5 pav.). Šį komponentą sudaro keli užklausų varikliai, kurie atsako į užklausas ir pagrindinis užklausų variklis, kuris apdoroja užklausą ir parenka tinkamą užklausų variklį. Pirmasis žingsnis, kurio imasi pagrindinis užklausų variklis – užklausos analizė. Nusprendžiama, ar užklausa sudaryta iš kelių nepriklausomų užklausų, jei taip, ji išskaidoma į kelias atskirai atsakomas užklausas. Rezultatai sujungiami vėliau. Kitas žingsnis - užklausos fragmentų bei kintamųjų rikiavimas naudojant euristinius algoritmus, kad būtų pagerintas vykdymo efektyvumas. Galiausiai užklausa patenka į užklausos variklį, kuris ją įvykdo ir pateikia atsakymą.



5 pav. Užklausų variklio schema

2.9. Siekiamo sukurti užklausų vykdymo metodo formuluotė

Taigi **darbo tikslas** – pagerinti duomenų paieškos ontologijoje procesą, sukuriant ontologijos užklausų vykdymo metodą, suderinantį reliacinių duomenų bazių ir ontologijų galimybes. **Tyrimo objektas** – duomenų išrinkimo iš ontologijos procesas, kai ontologija saugoma reliacinėje duomenų bazėje.

Pagrindinis sistemai keliamas **kokybės kriterijus** yra užklausos vykdymo laikas, kai ontologija yra didelės apimties, t.y. turi daug klasių individų. Todėl tikslas bus pasiektas, jeigu sistema leis reliacinėje duomenų bazėje saugojamoje ontologijoje užklausas vykdyti greičiau, negu jos vykdomos ontologijoje, saugojamoje tekstiniame faile.

Tikslui pasiekti reikia įvykdyti šiuos **uždavinius:**

- sukurti pavyzdinę užklausų vykdymo sistemą, kuri pateiktų vartotojui geriau atrinktus rezultatus ir pagreitintų *SPARQL* užklausų vykdymą ontologijoje, saugojant ją reliacinėje duomenų bazėje pagal *OWL2RDB* algoritmą.
- *Protégé* įrankiu sukurti pavyzdinę ontologiją. Ši ontologija vėliau turi būti transformuota į reliacinės duomenų bazės schemą pagal *OWL2RDB* algoritmo taisykles. Naudojant ontologiją turi būti atliktas sistemos testavimas ir eksperimentinis tyrimas.
- Naudojant pavyzdinę ontologiją, atlikti eksperimentą ir įvertinti darbo rezultatus.

Kuriamai užklausų vykdymo sistemai keliami šie **funkciniai reikalavimai**:

- klasių hierarchijos vykdant užklausą analizė;
- objektų savybių vykdant užklausą analizė;
- klasių individų išrinkimas.

Užklausų vykdymo sistemai keliami šie **nefunkciniai reikalavimai ir apribojimai**:

- sistema turi veikti su bet kokios dalykinės srities ontologija;
- sistema turi būti realizuota *JAVA* aplinkoje;
- sistema turi turėti grafinę vartotojo sąsają;
- turi būti naudojama *W3C* rekomenduojama ontologijų užklausų kalba *SPARQL*;
- sistema neturi visos ontologijos rašyti į operatyvinę atmintį.

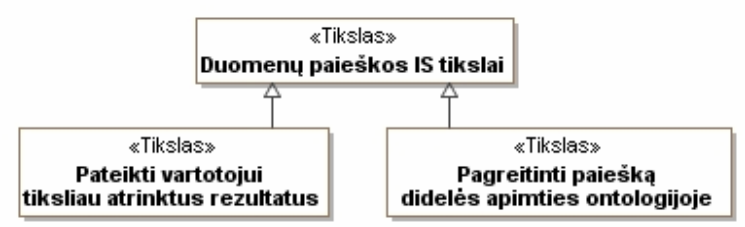
2.10. Analizės išvados

1. Analizuojant duomenų išrinkimo iš reliacinių duomenų bazių procesą nustatyta, kad, sukūrus dalykinės srities ontologiją, duomenys galėtų būti analizuojami semantiškai.
2. Ontologijų apimtis labai plečiasi, todėl neparanku jas saugoti tekstiniame faile. Didelės apimties ontologiją geriau saugoti reliacinėje duomenų bazėje, nes vykdant užklausas, mažiau apkraunama kompiuterio darbinė atmintis.

3. Ontologijos kūrimui pasirinktas įrankis *Protégé*, nes jis turi patogų grafinį klasių kūrimo, manipuliavimo sąsają, leidžia paprastai nustatyti klasių ribojimus, atributus bei sąryšius.
4. Dalykinės srities ontologija į reliacinės duomenų bazės schemą bus transformuojama naudojant *OWL2RDB* algoritmą, sukurtą *ISK* doktoranto. Šis algoritmas pasirinktas todėl, kad geriausiai išnaudoja reliacinės duomenų bazės galimybes ontologijai saugoti.
5. Paieškai ontologijoje nuspręsta naudoti *SPARQL* užklausų kalbą, nes ji yra labiausiai išvystyta bei rekomenduojama *W3C* konsorciumo.
6. Užklausų ontologijoje vykdymui nuspręsta pritaikyti *Pellet OWL* biblioteką, nes ji turi programinę ontologijos kūrimo sąsają. Bus kuriamas algoritmas, kuris pagal reliacinės duomenų bazės schemą atstato ontologiją ir sukuria *Pellet OWL* ontologijos klasių modelį bei apribojimus. Ontologijos individai į *Pellet OWL* biblioteką nebus įrašomi, jie turės būti randami naudojant *SQL*. Tam bus sukurtas *SPARQL* užklausos fragmentų transformavimo į *SQL* algoritmas.

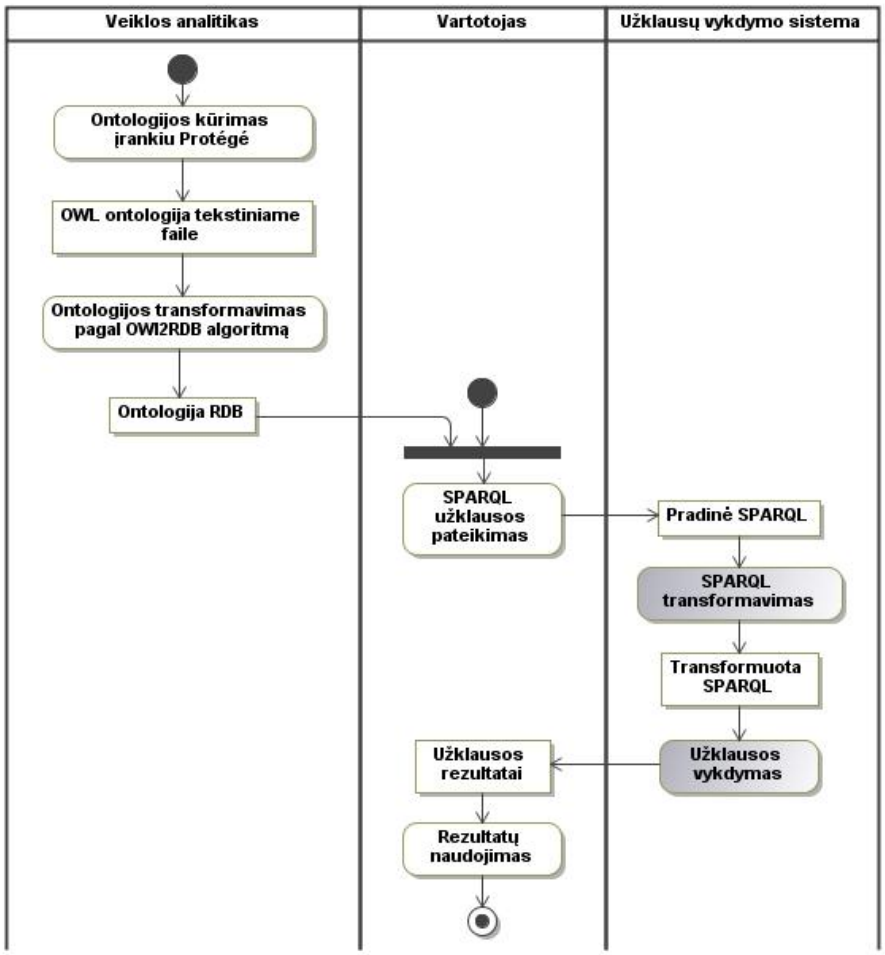
3. Užklausų vykdymo sistemos projektas

Pagrindiniai kuriamos užklausų vykdymo ontologijoje sistemos tikslai yra pateikti vartotojui geriau atrinktus rezultatus ir pagreitinti paiešką didelės apimties ontologijoje (6 pav.).



6 pav. Informacijos sistemos kūrimo tikslai

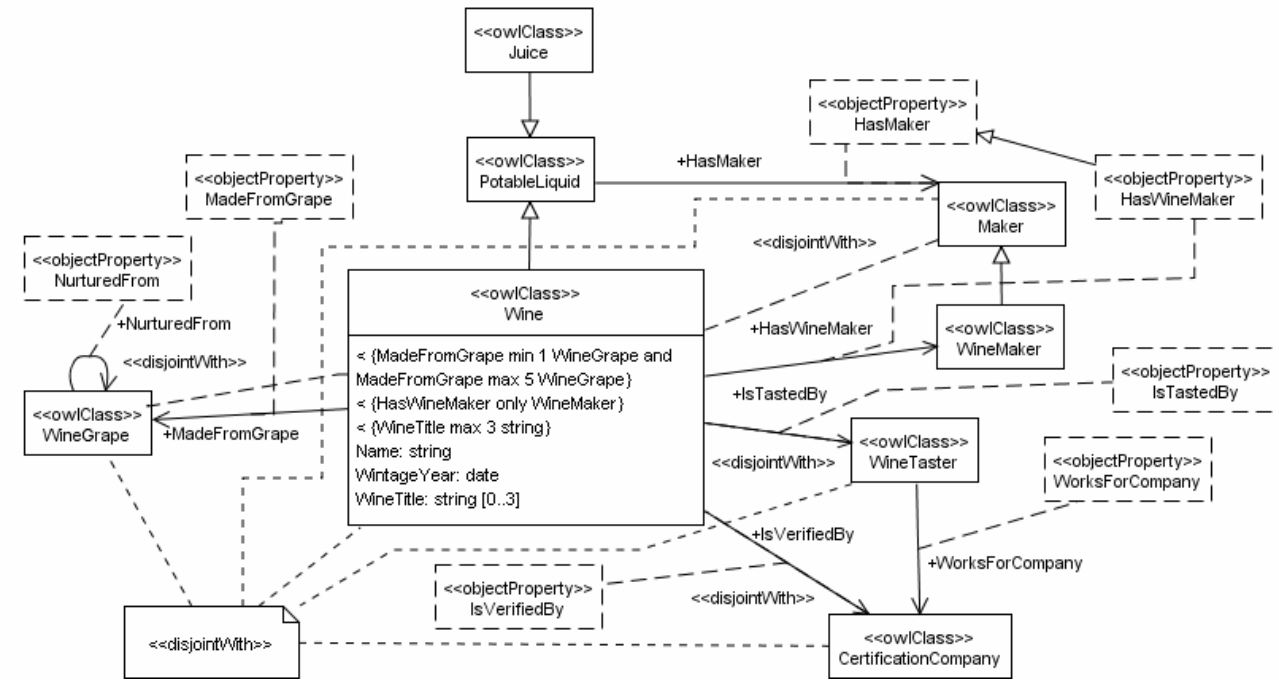
Užklausų vykdymo sistemos kūrimo tikslas – pateikti vartotojui geriau atrinktus rezultatus ir pagreitinti *SPARQL* užklausų vykdymą ontologijoje, saugojant ją reliacinėje duomenų bazėje pagal *OWL2RDB* algoritmą. Pagrindinis uždavinys, kurį reikia atlikti, norint pasiekti šį tikslą - realizuoti *SPARQL* transformavimo į *SQL* ir užklausų vykdymo metodą, kuris pilkai pavaizduotas 7 paveikslėlyje pateiktame ontologijos kūrimo ir užklausų vykdymo joje kontekste.



7 pav. Duomenų paieškos procesas naudojant ontologijas

3.1. Darbe naudojamos vyno ontologijos modelis

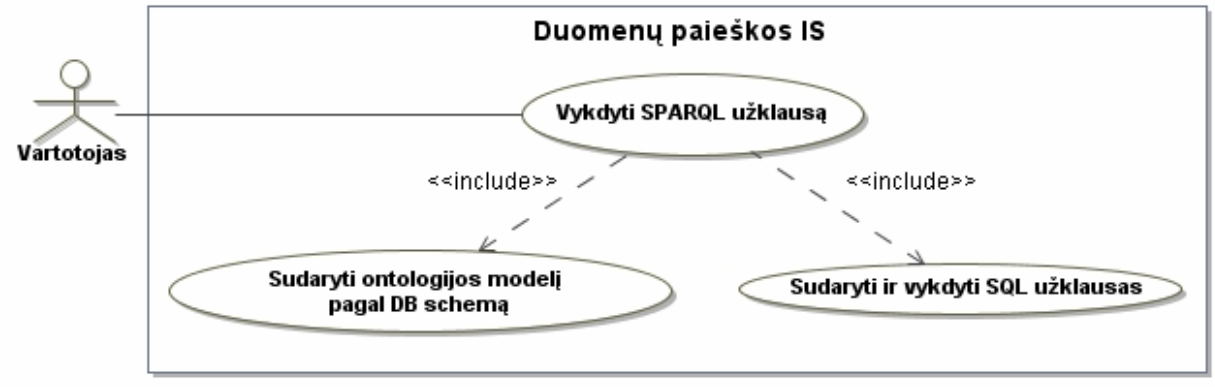
8 paveikslėlyje pateiktas vyno dalykinės srities ontologijos fragmentas. Ontologija sudaryta kūrimo įrankiu *Protégé*. Ji vėliau transformuojama į reliacinės duomenų bazės schemą pagal *OWL2RDB* algoritmo taisykles. Ši ontologija vėliau bus naudojama vykdant testavimą ir eksperimentą.



8 pav. Dalykinės srities ontologijos modelis

3.2. Kompiuterizuojamų panaudojimo atvejų diagrama

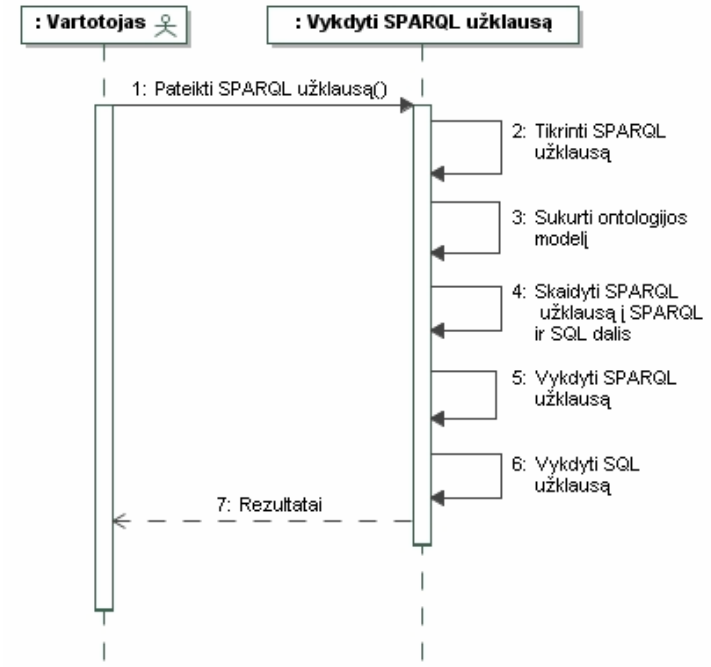
Kompiuterizuojamų panaudojimo atvejų modelyje išskiriamos vartotojų grupės, bei kuriamos informacinės sistemos paslaugos, kuriomis naudosis šių grupių vartotojai. 9 paveikslėlyje pavaizduota panaudojimo atvejų diagrama, 6 lentelėje panaudojimo atvejų specifikacija, o 9 - 11 paveikslėliuose scenarijai.



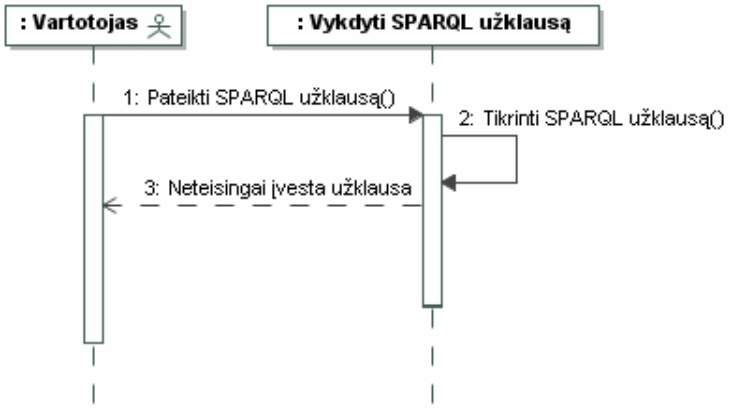
9 pav. Duomenų paieškos IS kompiuterizuojamų panaudojimo atvejų diagrama

6 lentelė. PA „Vykdyti semantinę užklausą“ specifikacija

PA „Vykdyti SPARQL užklausą“		
Tikslas. Gauti rezultatus įvykdžius SPARQL užklausą duomenų bazėje saugojamoje ontologijoje.		
Aprašymas.		
Aktorius		Vartotojas
Sužadinimo sąlyga		Vartotojas įvedė užklausą per vartotojo sąsają ir paspaudė patvirtinimo mygtuką
Susiję panaudojimo atvejai	**Išplečia PA**	
	Apima PA	„Sudaryti ontologijos modelį pagal DB schemą“, „Transformuoti SPARQL užklausą į SPARQL ir SQL“, „Sudaryti ir vykdyti SQL užklausas“.
	Specializuoja PA	
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas pateikia sistemai SPARQL užklausą		1.1. Sistema sukuria ontologijos modelį pagal duomenų bazės schemą 1.2. Sistema išskaido SPARQL užklausą į SPARQL ir SQL dalis. 1.3. Sistema įvykdo visas užklausas ir grąžina vartotojui rezultatus.
2. Vartotojas baigia PA		
Po sąlyga:		
Alternatyvūs scenarijai		
1.1.a. Užklausos sintaksė neteisinga		1.1.a.1 Vartotojui pateikiamas pranešimas, PA vykdymas nutraukiamas.



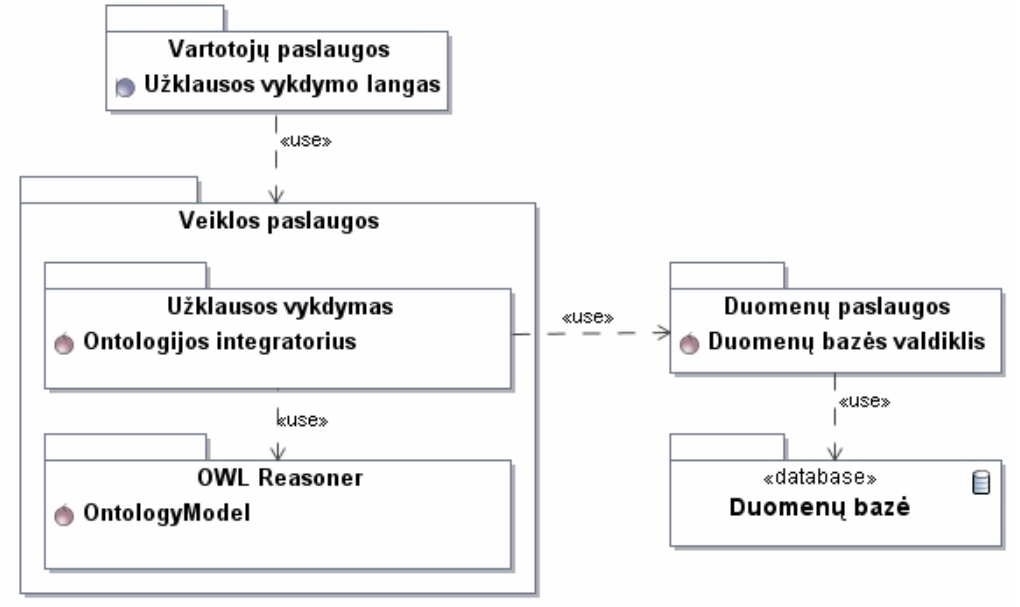
10 pav. PA „Vykdyti SPARQL užklausą“ scenarijus



11 pav. PA „Vykdyti SPARQL užklausą“ alternatyvus scenarijus

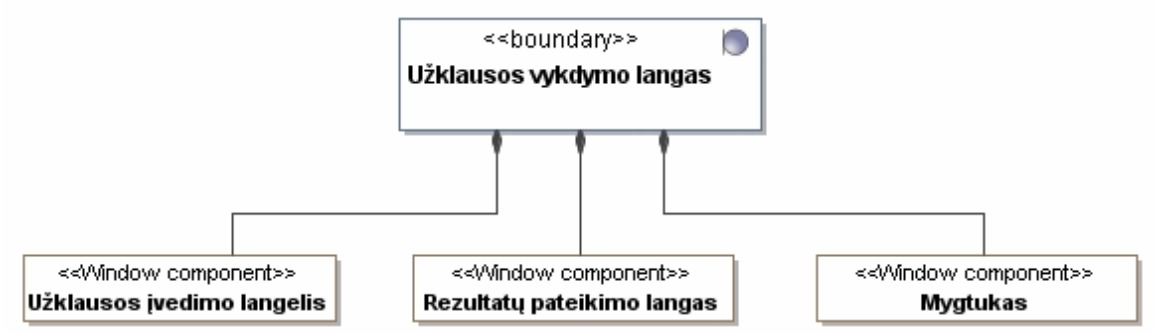
3.3. Loginė užklausų vykdymo sistemos architektūra, klasių modeliai

Sistemos loginė architektūra (12 pav.) parodo jos pagrindinius architektūrinius komponentus. Veiklos logikos posistemį sudaro du komponentai: užklausų vykdymo posistemis, kuris skirtas vykdyti *SPARQL* užklausas ir pagal duomenų bazės struktūrą sudaryti *OWL Reasoner* (semantinių užklausų vykdymo ir ontologijų analizės bibliotekos) ontologijos modelį.



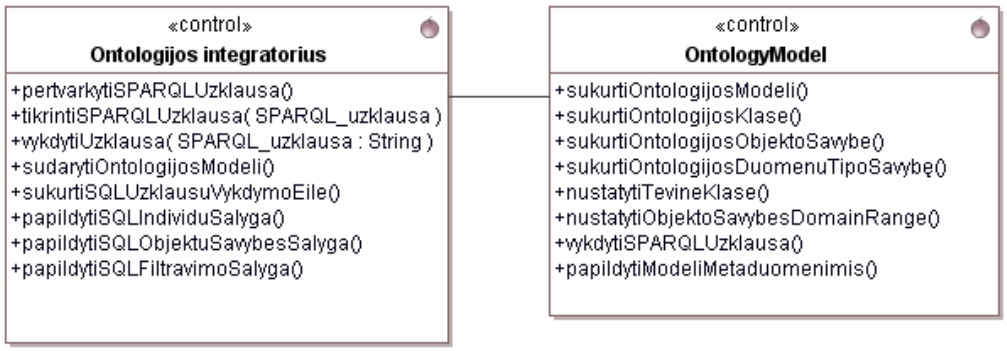
12 pav. Duomenų paieškos IS loginė architektūra

Vartotojo sąsają (13 pav.) sudaro vienintelis užklausos vykdymo langas.



13 pav. Duomenų paieškos IS vartotojo sąsajos planas

Veiklos paslaugų ontologijos integravimo posistemį (14 pav.) sudaro ontologijos konvertavimo valdiklis, kuriame aprašyta reliacinėje duomenų bazėje saugomos ontologijos interpretavimo taisyklės, *SPARQL* fragmentų konvertavimas į *SQL* ir *OWL Reasoner* valdiklis, kuris saugo ontologijos modelį, kuria ontologijos komponentus: klases, klasių ryšius, duomenų tipų ir objektų savybes.



14 pav. Veiklos paslaugų ontologijos integravimo posistemio valdikliai

Duomenų paslaugų posistemyje saugomas duomenų bazės valdiklis (15 pav.), skirtas prisijungti prie reliacinės duomenų bazės, vykdyti joje užklausas ir grąžinti rezultatus.

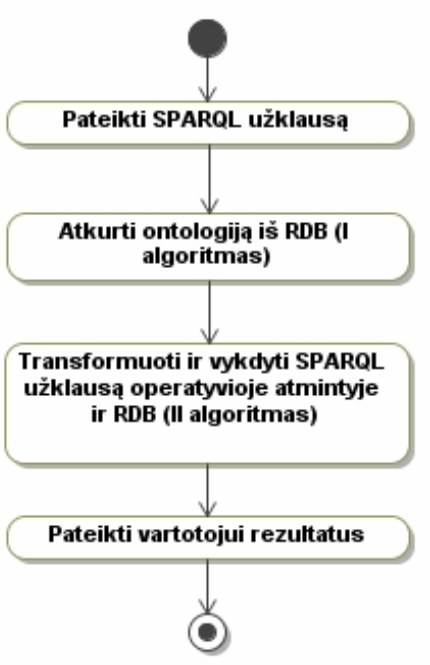
15 pav. Duomenų paslaugų posistemio valdiklis

16 paveikslėlyje pateiktas sistemos klasių modelis.

16 pav. Duomenų paieškos IS projektas

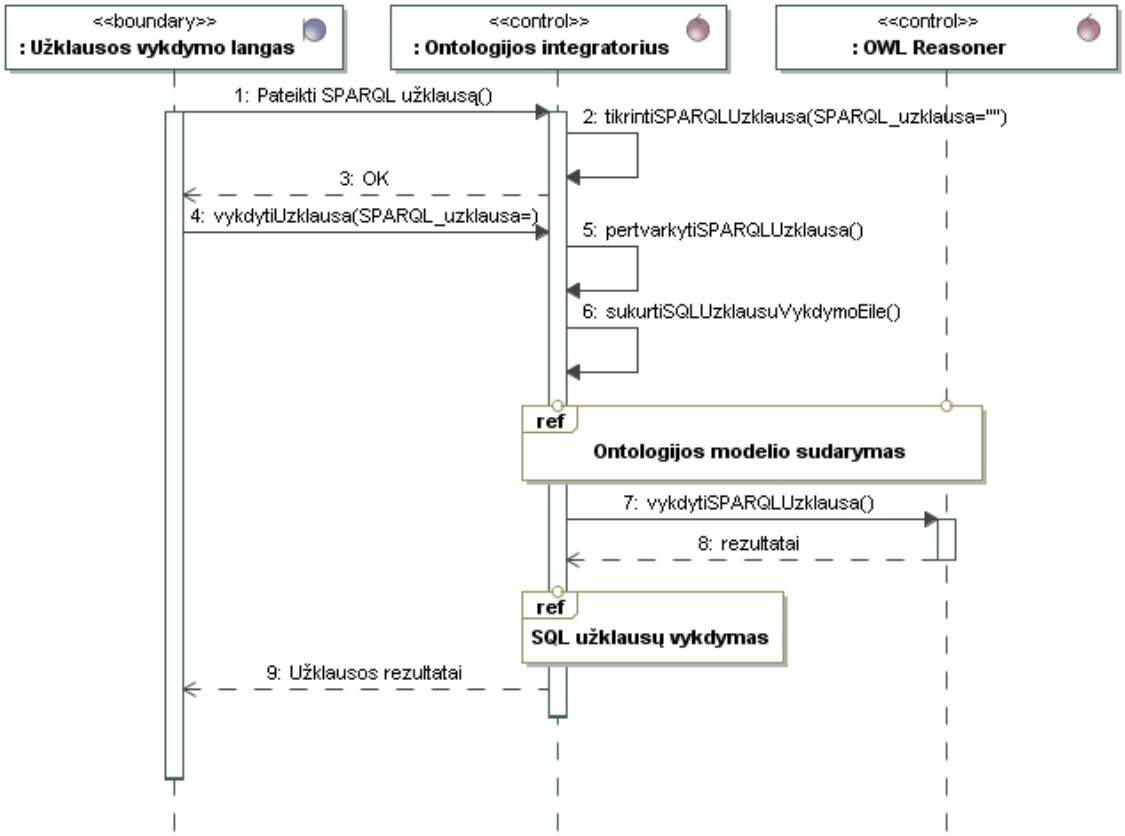
3.4. Užklausų vykdymo sistemos elgsenos modelis

Bendrą sistemos elgseną pavaizduoja 17 paveikslėlyje pateikta diagrama. Joje matyti pagrindiniai sistemos veikimo etapai vykdant užklausą – užklausos pateikimas, ontologijos modelio atkūrimas, užklausos transformavimas ir vykdymas.

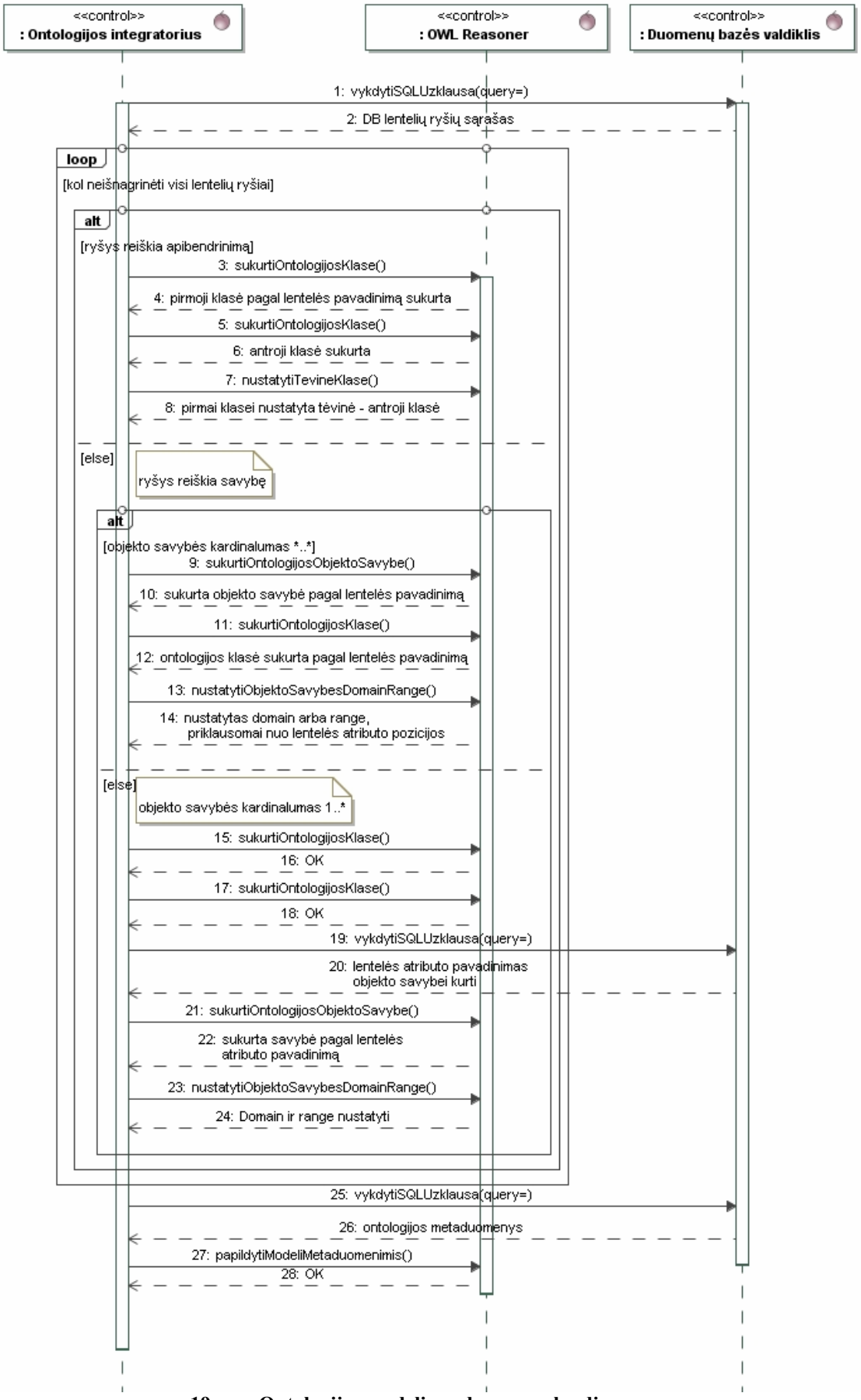


17 pav. Duomenų paieškos IS veiklos diagrama

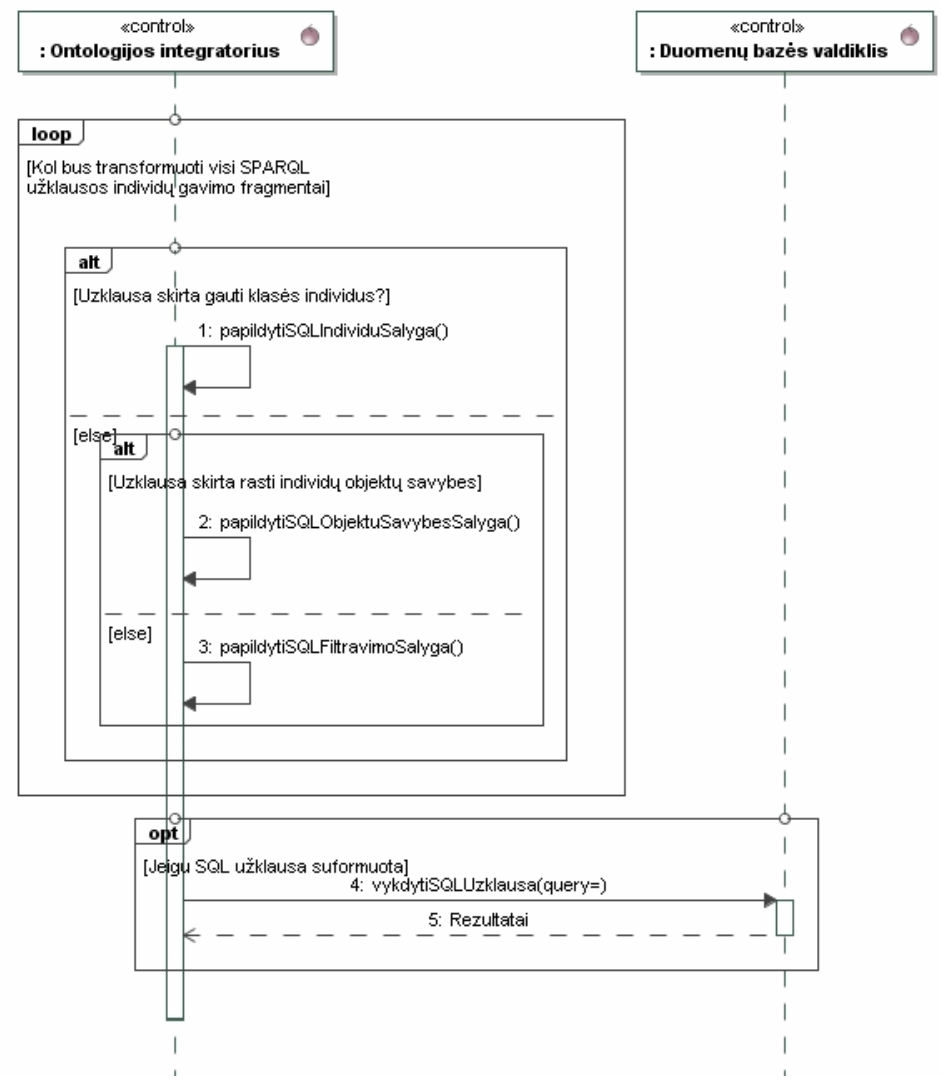
Duomenų paieškos IS komponentų sąveiką aprašo sekų diagramos, pateiktos 18 - 20 paveikslėliuose.



18 pav. Pagrindinė duomenų paieškos IS sekų diagrama



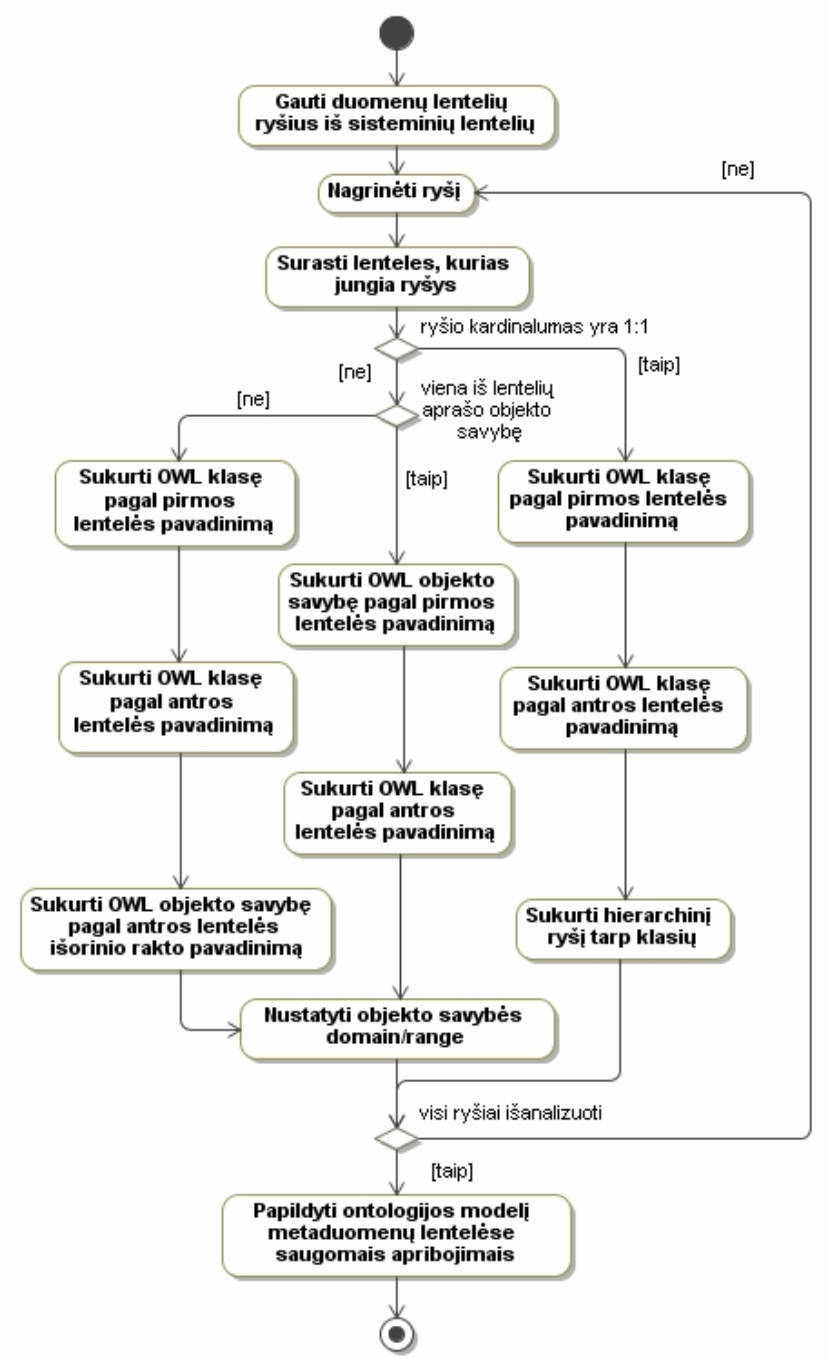
19 pav. Ontologijos modelio sudarymo sekų diagrama



20 pav. SQL užklausos vykdymo sekų diagrama

OWL2RDB algoritmo ontologijos saugojimo idėja remiasi tuo, kad visa ontologija neturi būti saugoma operatyvinėje atmintyje, todėl visos jos rašyti į *Pellet OWL Reasoner* ontologijos modelį negalima. Į šį modelį įrašoma tik klasių hierarchija, objektų, duomenų tipų savybės bei apribojimai iš metaduomenų lentelių.

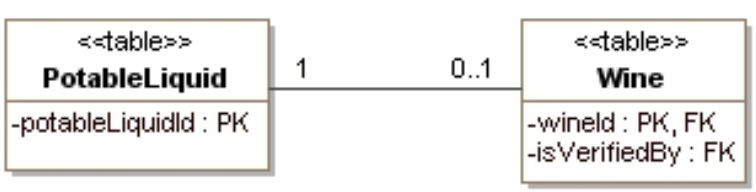
Algoritmas, kuris realizuoja ontologijos modelio kūrimą iš reliacinės duomenų bazės schemos pavaizduotas 21 paveikslėlyje.



21 pav. Ontologijos modelio iš reliacinės DB schemos sudarymo veiklos diagrama

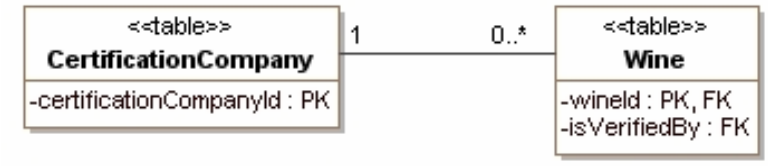
Algoritmas remiasi duomenų bazės sisteminių duomenų analize, t.y. randami ryšiai tarp lentelių, lentelių atributai, pagal juos sukuriamas ontologijos modelis. Žemiau pateikiami algoritmo paaiškinimai:

- Jeigu ryšio kardinalumas yra 1:1, sukuriamos dvi klasės pagal lentelių pavadinimus bei nustatoma hierarchija tarp jų. 22 paveikslėlyje pateiktame pavyzdyje bus sukurtos klasės *PotableLiquid* bei *Wine*, *PotableLiquid* bus nustatyta kaip tėvinė *Wine* klasė.



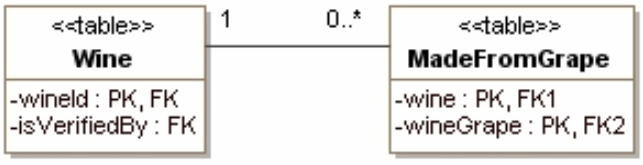
22 pav. Hierarchinis ryšys tarp duomenų lentelių

- Jeigu ryšio kardinalumas yra 1:*, kuriama objekto savybė, kurios pavadinimas nustatomas pagal antrosios klasės išorinio rakto atributo pavadinimą. Pirmoji klasė nustatoma kaip objekto savybės *range*, antroji - *domain*. 23 paveikslėlyje pateiktame pavyzdyje bus sukurta objekto savybė *isVerifiedBy*, jos *domain* – klasė *Wine*, *range* – *CertificationCompany*.

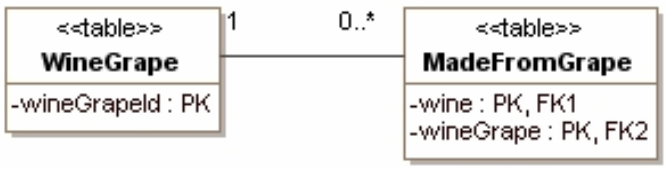


23 pav. Objekto tipo ryšys tarp duomenų lentelių

- Jeigu viena iš ryšio lentelių reiškia objekto savybę (tai nustatoma pagal kardinalumo metaduomenų lenteles), sukuriama objekto savybė pagal jos pavadinimą. Pagal kitą lentelę sukuriama klasė, ji nustatoma kaip objekto savybės *domain* arba *range*, priklausomai nuo išorinio rakto pozicijos. 24 paveikslėlyje pateiktame pavyzdyje bus sukurta objekto savybė *MadeFromGrape*, jos *domain* bus klasė *Wine*. 25 paveikslėlyje pavaizduotame pavyzdyje klasė *WineGrape* nustatoma objekto savybės *MadeFromGrape range* klase.



24 pav. Objekto tipo ryšys tarp duomenų lentelių

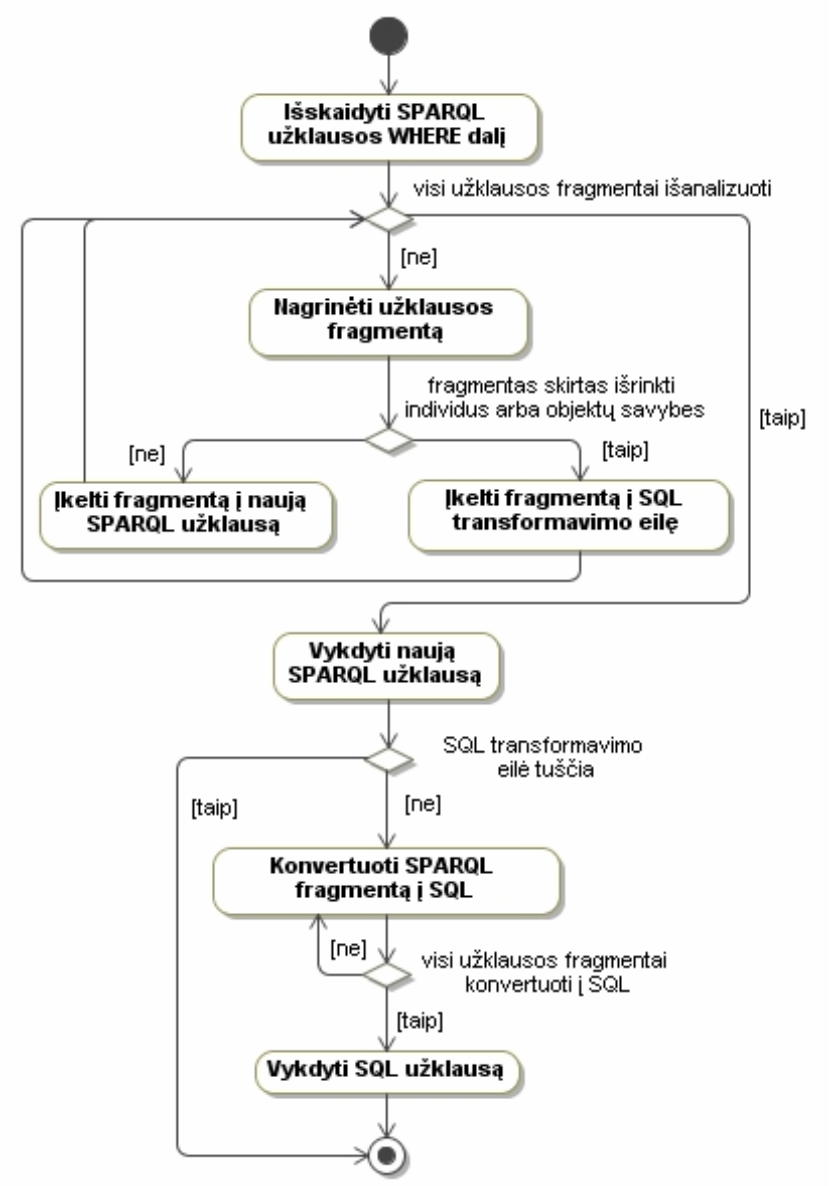


25 pav. Objekto tipo ryšys tarp duomenų lentelių

Ontologijos užklausoms vykdyti naudojama užklausų kalba *SPARQL*. Užklausos vykdomos tiek klasių lygmenyje, tiek rasti klasių individams. Klasių paieškai naudojamas *Pellet OWL Reasoner*, kuriame sukuriamas ontologijos modelis pagal reliacinės duomenų bazės schemą. Tačiau šiame modelyje nesaugoma informacija apie individus, jie gaunami iš duomenų bazės lentelių,

vykdant *SQL* užklausą, todėl sukurtas algoritmas (26 pav.), kuris pirmiausia įvykdo *SPARQL* užklausą, susijusią su klasių hierarchijos gavimu, vėliau atskirai vykdo individų gavimo sąlygas *SQL* užklausų kalba. Šis algoritmas pateiktos *SPARQL* užklausos sąlyginę dalį skaido į dvi dalis:

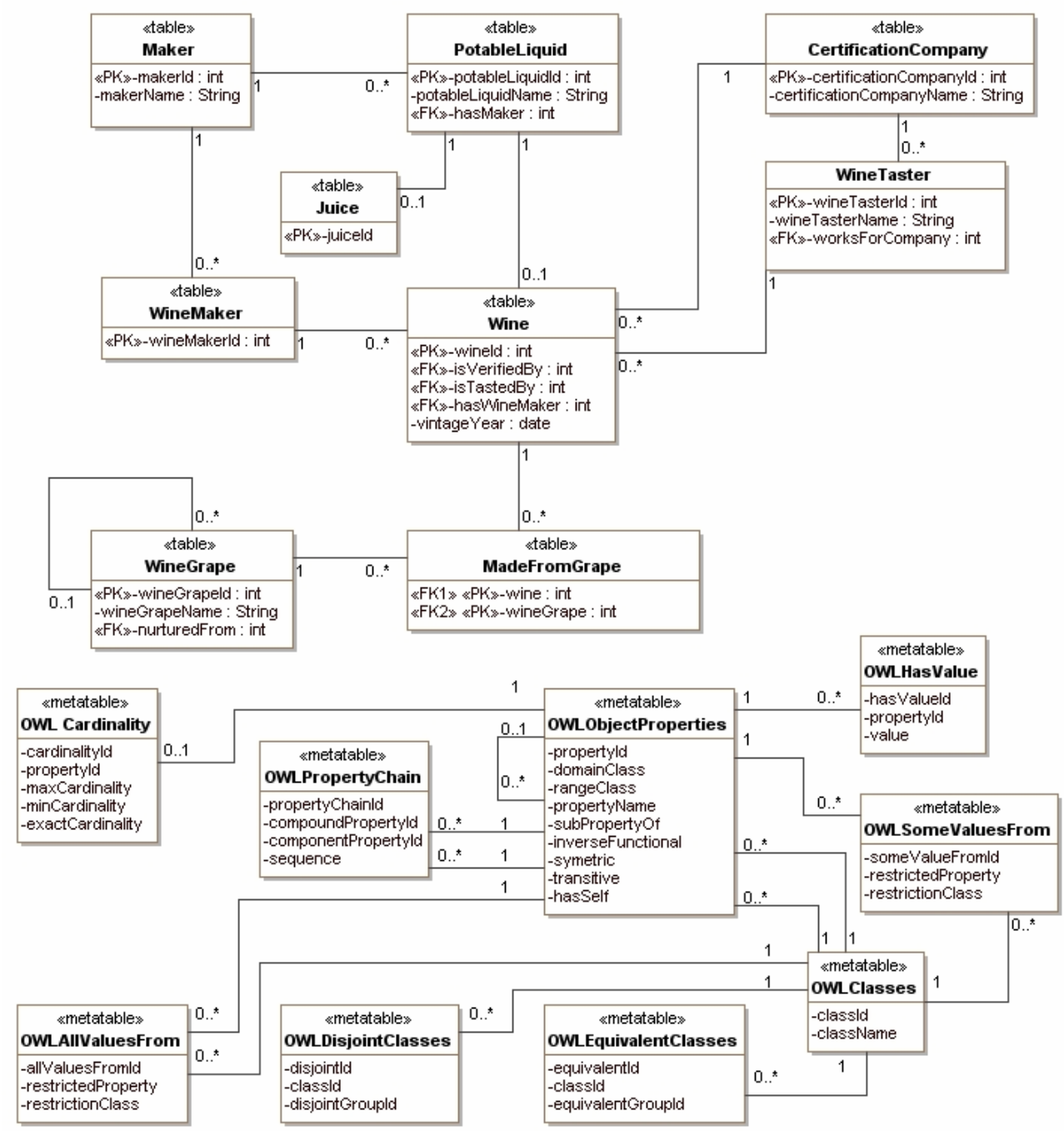
- sudaroma nauja *SPARQL* užklausa, kurioje lieka tik sąlygos dalys, susijusios su klasių gavimu;
- sudaroma užklausos sąlygų, susijusiu su individų gavimu, eilė. Šios sąlygos atskiriamos pagal *predicate* dalį. Į ją patenka sąlygos su *predicate* dalimi *rdf:type* arba ontologijos objektų savybėmis.



26 pav. Užklausos vykdymo veiklos diagrama

3.5. Vyno ontologijos duomenų bazės schema

Duomenų bazės schema (27 pav.) sudaryta iš dalykinės srities ontologijos modelio (8 pav.) pagal *OWL2RDB* algoritmą. Ši schema aprašo vyno dalykinės srities ontologiją.

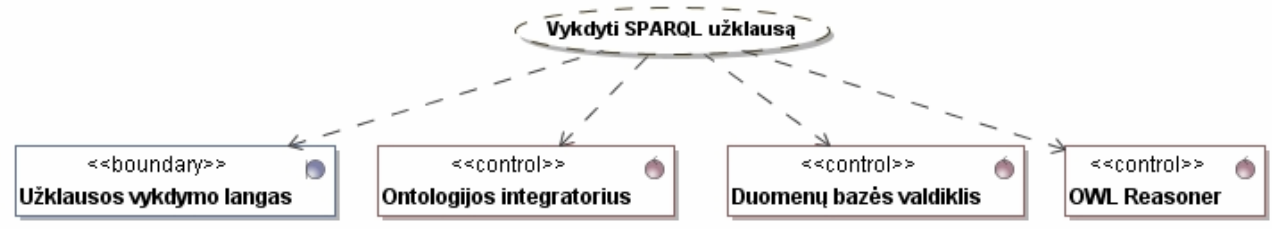


27 pav. Pavyzdinės ontologijos duomenų bazės schema

3.6. Užklausų vykdymo sistemos realizacijos modelis

28 paveikslėlyje vaizduojamas panaudojimo atvejo „Vykdyti *SPARQL* užklausą“ realizacijos panaudojimo atvejis. 29 paveikslėlyje parodyta, kokiomis klasėmis šis panaudojimo atvejis bus realizuotas.

28 pav. Duomenų paieškos IS realizacijos diagrama

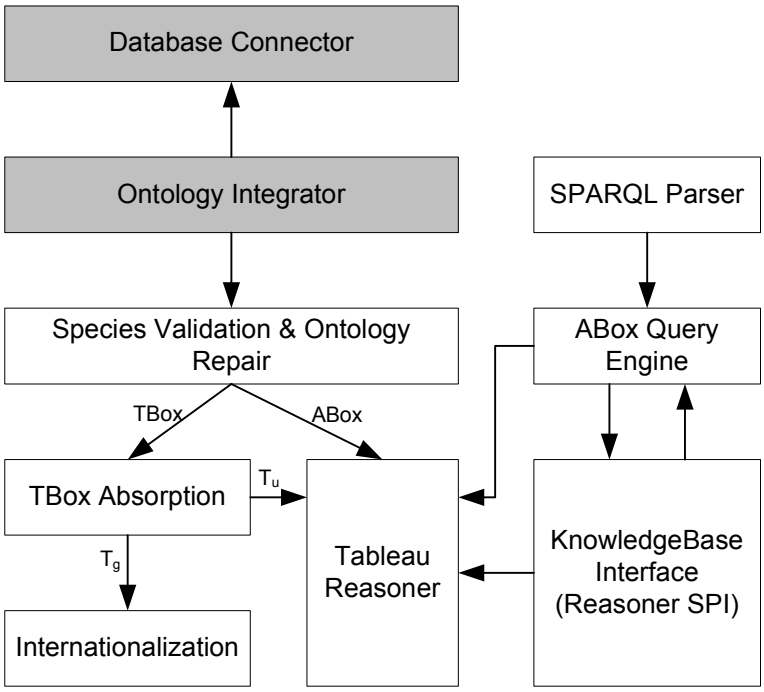


29 pav. Panaudojimo atvejo „Vykdyti *SPARQL* užklausą“ realizacijos diagrama

4. Realizacija

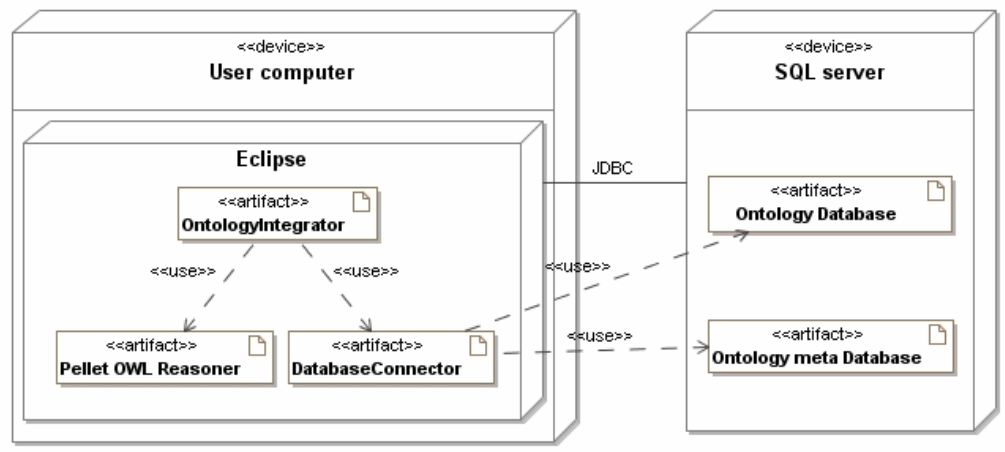
4.1. Sistemos veikimo aprašymas

Užklausų vykdymo sistemai kurti buvo pritaikta *Pellet OWL Reasoner* biblioteka. Ontologijos pateikimui joje standartiškai naudojamas skaitytuvas (*Parser*), kuris nagrinėja *OWL* failą ir pateikia jame saugomą ontologiją *Pellet OWL Reasoner* bibliotekai. Šis variantas nėra tinkamas darbo tikslui pasiekti, nes ontologija turi būti saugoma reliacinėje duomenų bazėje, tačiau bibliotekoje suteikiama galimybė kurti ontologijas programiškai. Todėl bus realizuotas naujas komponentas *Ontology Integrator* (30 pav.), kuris analizuos duomenų bazės struktūrą ir pagal ją sukurs *Pellet OWL Reasoner* ontologiją. Pagalbinis komponentas *Database Connector* skirtas prisijungti prie duomenų bazės ir vykdyti joje užklausas. Visi kiti bibliotekos komponentai išliko nepakitę - pakeistas tik ontologijos pateikimo metodas.



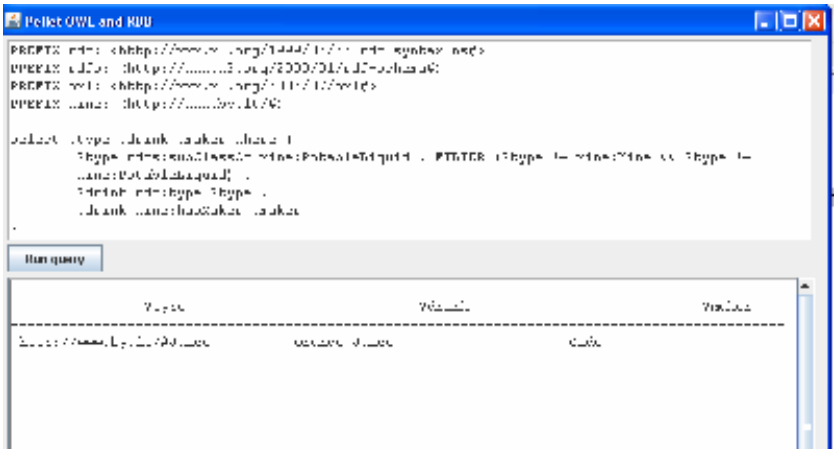
30 pav. Pellet OWL Reasoner schema, kai ontologija sudaroma iš duomenų bazės schemos

Sistema realizuota *Eclipse* aplinkoje, ontologijai saugoti pasirinkta *MS SQL Server* DBVS. Naudojamos dvi atskiros duomenų bazės – viena ontologijai, kita ontologijos metaduomenims, tam, kad korektiškai veiktų ontologijos kūrimo pagal duomenų bazės schemą algoritmas. 31 paveikslėlyje pateikta sistemos diegimo diagrama.



31 pav. Sistemos diegimo diagrama

Užklausoms pateikti ir rezultatams vaizduoti bus sukurta vartotojo sąsaja, ji pateikta 32 paveikslėlyje.



32 pav. Vartotojo sąsajos langas

4.2. Užklausų vykdymo sistemos testavimas, duomenys bei kontrolinis pavyzdys

Siekiant nustatyti sistemos atrenkamų duomenų teisingumą, jie bus lyginami su rezultatais, gautais naudojant tekstiniame faile saugomą ontologiją, užklausą vykdant *Pellet OWL Reasoner*. Vykdant užklausą bus rodomos ir tarpinės sugeneruotos *SQL* užklausos bei jų vykdymo rezultatai, nes metodas išskaido *SPARQL* užklausą ir *SQL* generuoja etapais. Taip bus išsiaiškinta, ar korektiškai veikia užklausos išskaidymo ir SQL sudarymo mechanizmai.

Užklausa bus vykdoma 8 paveikslėlyje pateiktoje vyno ontologijoje. Nuo 7 lentelės iki 12 lentelės pateikiami duomenų bazės lentelių duomenys.

7 lentelė. Lentelės *PotableLiquid* duomenys

potableLiquidId	potableLiquidName	hasMaker
1	FoxenCheninBlank	1
2	MariettaZinfandel	2
3	MariettaPetiteSyrah	2

8 lentelė. Lentelės *Wine* duomenys

wineId	isTastedBy	isVerifiedBy	hasWineMaker	vintageYear
1	2	1	1	2000
2	1	2	2	2001
3	2	1	2	1995

9 lentelė. Lentelės *Juice* duomenys

juiceId
4

10 lentelė. Lentelės *Maker* duomenys

makerId	makerName
1	Foxen
2	Marietta
3	Cido

11 lentelė. Lentelės *WineMaker* duomenys

wineMakerId
1
2

12 lentelė. Lentelės *CertificationCompany* duomenys

certificationCompanyId	certificationCompanyName
1	French Certification Company
2	Itallian Certification Company

Toliau pateikta užklausa, kuri bus vykdoma testuojant. Ši užklausa ontologijoje randa visus gėrimus ir jų gamintojus, sertifikuotus kompanijos *French_Certification_Company*:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wine: <http://www.by.lt/#>

select ?type ?drink ?maker where
{
  ?type rdfs:subClassOf wine:PotableLiquid . FILTER (?type != wine:PotableLiquid) .
  ?drink rdf:type ?type .
  ?drink wine:isVerifiedBy wine:French_Certification_Company .
  ?drink wine:hasWineMaker ?maker
}
```

Užklausa turi 4 sąlygos dalis, kurios toliau bus aprašytos kiekviena atskirai, pateikti tarpiniai etapų rezultatai.

- **sąlygos `?type rdfs:subClassOf wine:PotableLiquid.FILTER(?type!=wine:PotableLiquid)` vykdymas**

Ši užklausos dalis vykdoma *Pellet OWL Reasoner*, ji grąžina visas *PotableLiquid* poklases, atmetant pačią *PotableLiquid* (tam naudojama *FILTER* komanda), nes *Pellet OWL Reasoner* laiko, kad klasė yra savo paties poklasė. Šiame pavyzdyje gaunamas vienintelis įrašas *Wine*, jis priskiriamas kintamajam *?type*:

```
select ?type where
{
   ?type rdfs:subClassOf wine:PotableLiquid . FILTER (?type != wine:PotableLiquid)
}
```

Gauti rezultatai pateikti 13 lentelėje:

13 lentelė. Pirmojo užklausos vykdymo etapo rezultatai

?type
Wine

- **sąlygos `?drink rdf:type ?type` vykdymas**

Šioje užklausos dalyje randami visi ankstesnėje užklausoje gautų klasių egzemplioriai. *SQL* užklausa formuojama pagal klasių pavadinimus. Kadangi klasė *Wine* turi tėvinę klasę *PotableLiquid* (tai sužinoma iš *Pellet OWL Reasoner* ontologijos modelio), vynų pavadinimai ir *id* gaunami kreipiantis į ją, naudojant *SQL* komandą *JOIN*:

```
SELECT
       'Wine' as '?type',
       PotableLiquidName as '?drink'
FROM
       Wine LEFT JOIN PotableLiquid ON WineId = PotableLiquidId
```

Įvykdžius šią užklausą gaunami visi *Wine* klasės egzemplioriai, jie priskiriami kintamajam *?drink*. Užklausos vykdymo rezultatai pateikiami 14 lentelėje:

14 lentelė. Antrojo užklausos vykdymo etapo rezultatai

?type	?drink
Wine	FoxenCheninBlank
Wine	MariettaZinfandel
Wine	MariettaPetiteSyrah

- **sąlygos `?drink wine:isVerifiedBy wine:French_Certification_Company` vykdymas**

Filtruojamas vynų sąrašas, išrenkami tik tie vynai, kuriuos sertifikavo nurodyta kompanija. Čia taip pat iš *Pellet OWL Reasoner* ontologijos modelio sužinoma, kad savybės *isVerifiedBy domain* klasė yra *Wine*, o *range* – *CertificationCompany*. Pagal šias reikšmes papildoma ankstesniame etape sudaryta *SQL* užklausa:

```
SELECT
       'Wine' as '?type',
       PotableLiquidName as '?drink'
FROM
       Wine LEFT JOIN PotableLiquid ON WineId = PotableLiquidId
       LEFT JOIN CertificationCompany ON Wine.isVerifiedBy =
                  CertificationCompany.CertificationCompanyId
WHERE  CertificationCompanyName  = 'French_Certification_Company'
```

Užklausa papildoma komanda *WHERE*, kuri išrenka tik tuos vynus, kurie yra sertifikuoti kompanijos *French_Certification_Company*. 15 lentelėje pateikti trečiojo užklausos vykdymo etapo rezultatai.

15 lentelė. Trečiojo užklausos vykdymo etapo rezultatai

?type	?drink
Wine	FoxenCheninBlank
Wine	MariettaPetiteSyrah

- **sąlygos `?drink wine:hasMaker ?maker` vykdymas**

Vykdomas savybės išrinkimas. Pirmiausia randama savybės *hasMaker domain* ir *range* klasės iš *Pellet OWL Reasoner* ontologijos modelio (*domain* - *PotableLiquid*, *range* – *Maker*). *SQL* užklausa formuojama pagal šias reikšmes:

```
SELECT
       'Wine' as '?type',
       PotableLiquidName as '?drink' ,
       MakerName as '?maker'
FROM
       Wine LEFT JOIN PotableLiquid ON WineId = PotableLiquidId
       LEFT JOIN CertificationCompany ON Wine.isVerifiedBy =
       CertificationCompany.CertificationCompanyId
       LEFT JOIN WineMaker ON Wine.hasWineMaker = WineMaker.WineMakerId
       LEFT JOIN Maker ON WineMaker.WineMakerId = Maker.MakerId
WHERE  CertificationCompanyName  = 'French_Certification_Company'
```

Gautas gamintojų sąrašas priskiriamas kintamajam *?maker*. 16 lentelėje pateiktame sąraše rodomi galutiniai užklausos vykdymo rezultatai.

16 lentelė. Galutinio užklausos vykdymo etapo rezultatai

?type	?drink	?maker
Wine	FoxenCheninBlank	Foxen
Wine	MariettaPetiteSyrah	Marietta

Rezultatai sutampa su rezultatais, gautais vykdant užklausą tekstiniame faile saugojamoje analogiškoje ontologijoje. Testavimo rezultatai parodė, kad sistema gerai veikia vykdant klasių hierarchijos, objektų savybių paiešką, filtravimą pagal objektų savybes.

5. Eksperimentinis užklausų vykdymo sistemos tyrimas

Sukurtas ontologijos užklausų reliacinėje duomenų bazėje vykdymo metodas turėtų leisti vykdyti užklausas didelės apimties ontologijose. Jo pagrindinis privalumas palyginti su tekstiniame faile saugoma ontologija – *SPARQL* užklausų vykdymo greitis, kai ontologija turi daug egzempliorių.

5.1. Tiriamų parametrų aprašymas

Vykdant eksperimentą, buvo siekiama patvirtinti hipotezę, kad sukurtas metodas leidžia vykdyti *SPARQL* užklausas reliacinėje duomenų bazėje greičiau negu jos įvykdomos tekstiniame faile, kai ontologijos egzempliorių skaičius didelis.

Siekiant patvirtinti šią hipotezę, buvo atliktas eksperimentas – užklausos vykdomos analogiškose tekstiniame faile ir reliacinėje duomenų bazėje saugomose ontologijose, lyginami jų vykdymo laikai su skirtingu ontologijos egzempliorių skaičiumi, buvo stebima, kaip užklausos vykdymo ir ontologijos įkėlimo į atmintį laikas priklauso nuo individų kiekio. Siekiant išvengti atsitiktinių rezultatų, tose pačiose ontologijose buvo vykdomos dvi skirtingos užklausos. Taigi nepriklausomi eksperimento kintamieji yra:

- užklausos tipas;
- ontologijos saugojimo metodas (*XML* faile arba reliacinėje duomenų bazėje saugoma ontologija);
- ontologijos *Wine* klasės individų kiekis, visų kitų klasių individų kiekis buvo pastovus.

Tuo tarpu priklausomi eksperimento kintamieji yra šie:

- ontologijos įkėlimo į darbinę atmintį laikas;
- užklausos vykdymo laikas ontologijoje.

Buvo tiriama ontologijos įkėlimo laiko į darbinę atmintį priklausomybė nuo ontologijos saugojimo metodo ir individų skaičiaus bei užklausos vykdymo laiko priklausomybė nuo užklausos tipo, saugojimo metodo ir individų skaičiaus.

Eksperimentas buvo atliktas nešiojamuoju kompiuteriu su *1,6 GHz* procesoriumi ir *1 GB* laisvos darbinės atminties. Ontologijos duomenų bazei saugoti buvo naudojama *Microsoft SQL Server 2005* duomenų bazių valdymo sistema, užklausų vykdymo programa veikė *Eclipse* aplinkoje. Eksperimento metu buvo naudojama vyno ontologija, pateikta 8 paveikslėlyje. Ontologijos schema nebuvo keičiama viso eksperimento metu – buvo naudojama ta pati klasių hierarchija, tačiau buvo keičiamas klasės *Wine* individų skaičius, todėl buvo sukurtas programinis individų generatorius – įterpti įrašus į duomenų bazės lenteles, kai ontologija saugoma duomenų

bazėje ir generuoti *OWL* failus, kai ontologija saugoma tekstiniame faile. Generatorius sugeneruodavo individus ir nustatydavo jo savybes. Toliau pateiktas sugeneruotas *OWL* individo pavyzdys:

```
<Wine rdf:about="MountadamChardonnay">
            <rdf:type rdf:resource="&owl;Thing"/>
            <VintageYear>2000</VintageYear>
            <hasWineMaker rdf:resource="Foxen"/>
            <isVerifiedBy rdf:resource="French_Certification_Company"/>
            <isTastedBy rdf:resource="John_Smith"/>
</Wine>
```

5.2. Ontologijos įkėlimas į operatyviąją atmintį

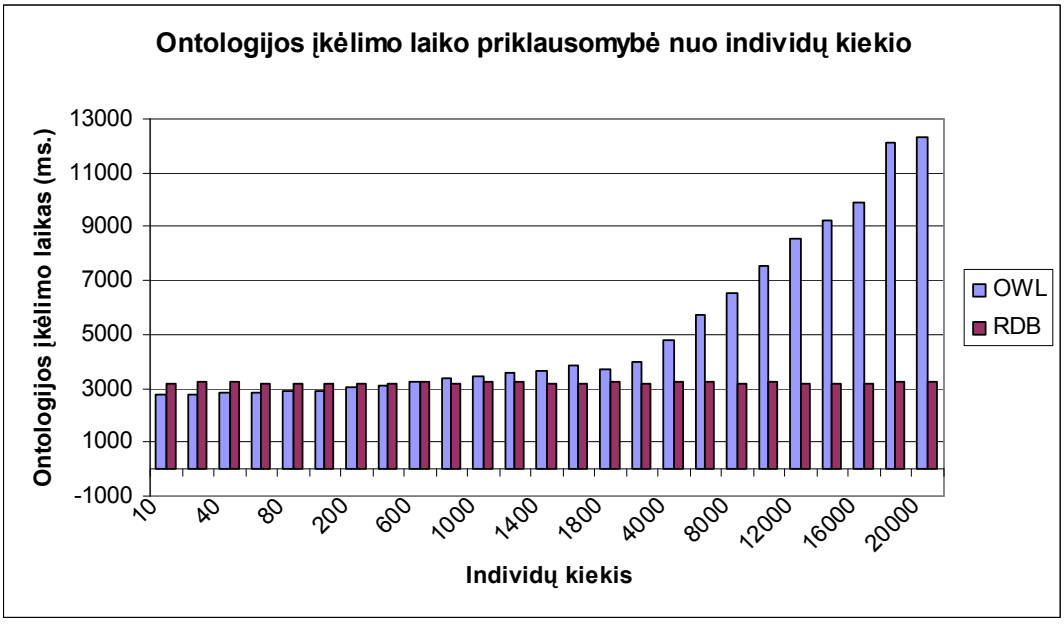
Įrašant į atmintį ontologiją, atliekami tam tikri veiksmai, kurie yra skirtingi abiem ontologijos saugojimo atvejais. Kai ontologija saugoma tekstiniame faile, įkėlimo laiką sudaro:

- tekstinio failo nuskaitymas;
- ontologijos klasių, savybių, individų sukūrimas;
- ontologijos patikrinimas, kiekvieno individo klasės nustatymas.

Tuo tarpu, kai ontologija saugoma reliacinėje duomenų bazėje, įkėlimo laiką sudaro:

- duomenų bazės schemos analizė (ontologijos atstatymo iš reliacinės duomenų bazės schemos algoritmas);
- ontologijos klasių, savybių sukūrimas;
- ontologijos patikrinimas.

33 paveikslėlyje pateiktame grafike vaizduojam abiejų saugojimo metodų ontologijos įkėlimo į atmintį laiko priklausomybė nuo individų kiekio.



33 pav. Ontologijos įkėlimo į atmintį laiko priklausomybė nuo ontologijos individų skaičiaus

Esant nedideliam individų kiekiui (apytiksliai iki 1000), ontologijos įkėlimo laikai abiem atvejais yra labai panašūs, tačiau nuo 1000 individų, tekstiniame faile saugomos ontologijos įkėlimo laikas pradeda sparčiai didėti. Taip yra dėl to, kad saugant ontologiją duomenų bazėje, individų įkelti į atmintį nereikia, todėl įkėlimo laikas išlieka pastovus, nepriklausomai nuo to, kiek ontologija turės individų.

5.3. Užklausų vykdymas ontologijoje

Kadangi metodas skirtas ontologijoms, kurios sudarytos ne tik iš klasių, bet turi ir individų, vykdant eksperimentą buvo naudojamos užklausos, kurios turi išrinkimo sąlygas tiek klasių, tiek individų lygmeniu. Siekiant nustatyti užklausų vykdymo laikų priklausomybes, ontologijose buvo vykdomos trys skirtingo sudėtingumo užklausos:

- surasti *PotableLiquid* klasės vaikines klases, išrinkti jų individus:

```
select ?x ?y where
{
	?x rdfs:subClassOf wine:PotableLiquid . FILTER (?x != wine:PotableLiquid) .
	?y rdf:type ?x
}
```

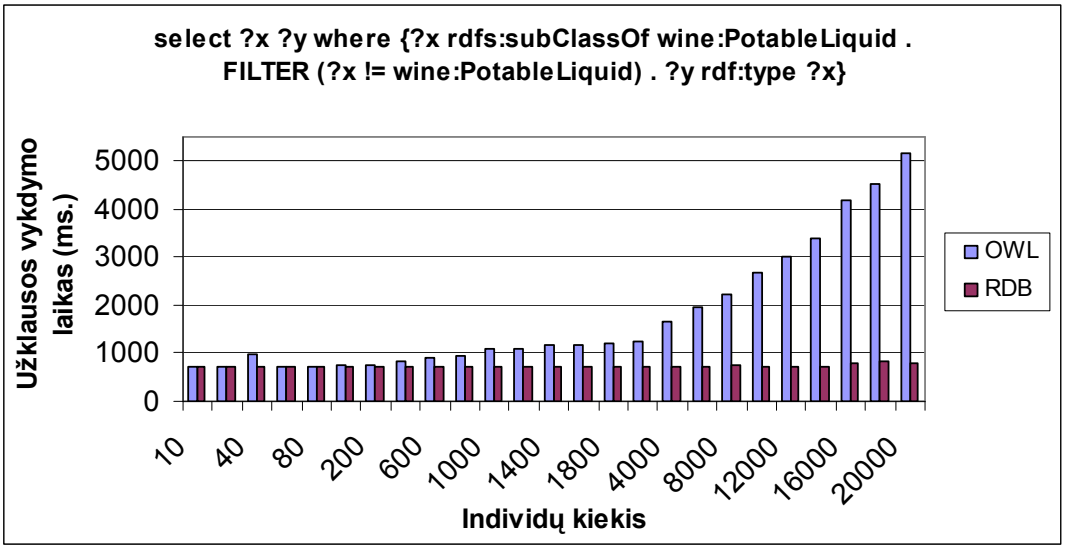
- surasti kiekvieno individo gamintoją:

```
select ?x ?y ?z where
{
	?x rdfs:subClassOf wine:PotableLiquid . FILTER (?x != wine:PotableLiquid) .
	?y rdf:type ?x .
	?y wine:hasWineMaker ?z
}
```

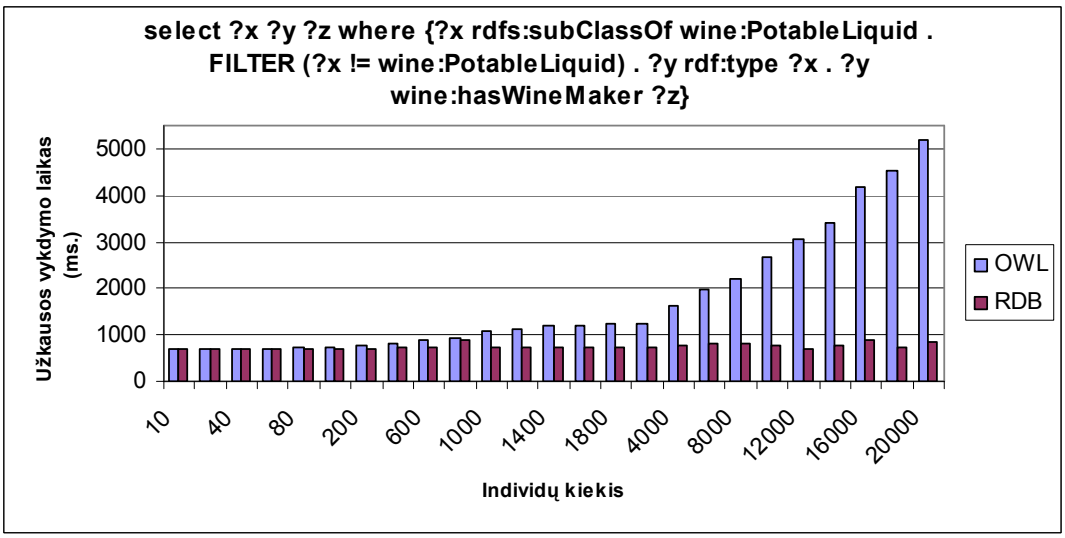
- filtruoti individus pagal sertifikavimo kompaniją:

```
select ?x ?y ?z where
{
	?x rdfs:subClassOf wine:PotableLiquid . FILTER (?x != wine:PotableLiquid) .
	?y rdf:type ?x .
	?y wine:hasWineMaker ?z .
	?y wine:isVerifiedBy wine:French_Certification_Company
}
```

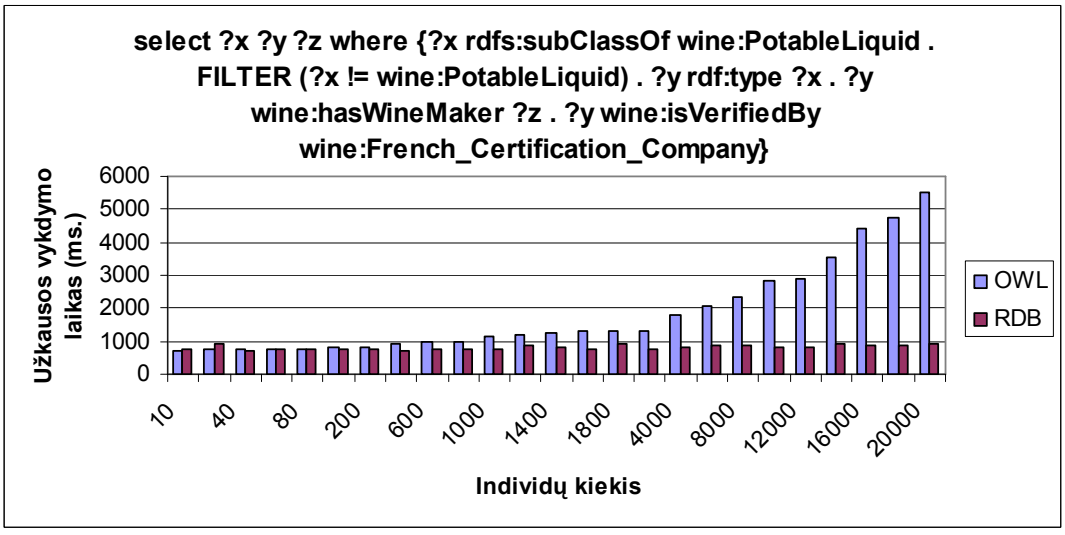
Tos pačios užklausos buvo vykdomos tekstiniame faile ir reliacinėje duomenų bazėje saugomose ontologijose su įvairiais individų kiekiais. 34 - 36 paveikslėliuose pateikti užklausų priklausomybės nuo individų skaičiaus grafikai.



34 pav. Pirmosios užklausos vykdymo laiko priklausomybė nuo individų skaičiaus



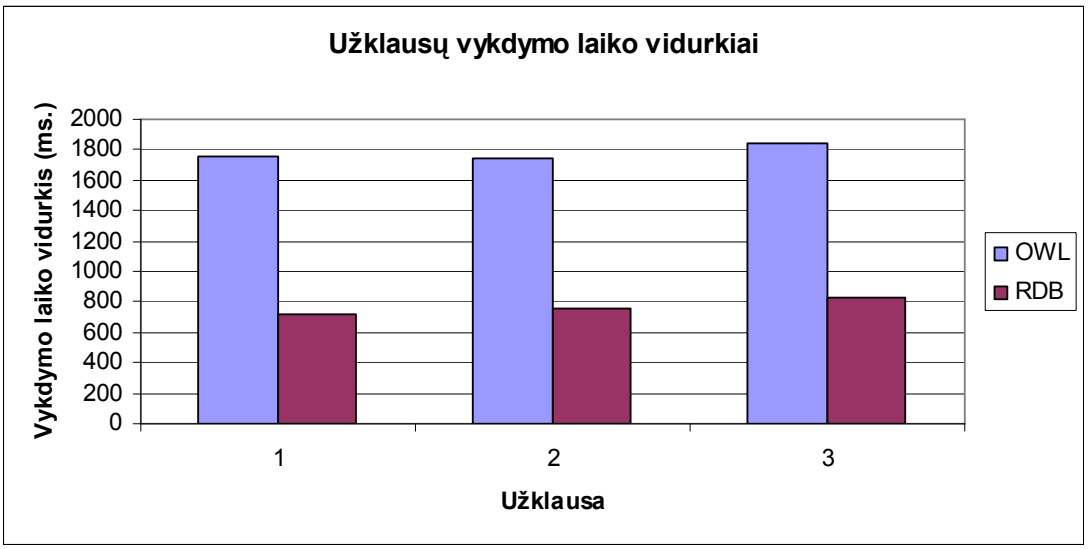
35 pav. Antrosios užklausos vykdymo laiko priklausomybė nuo individų skaičiaus



36 pav. Trečiosios užklausos vykdymo laiko priklausomybė nuo individų skaičiaus

Iš grafikų matyti, kad esant mažam individų skaičiui, užklausų vykdymo laikai yra labai panašūs abiem ontologijos saugojimo atvejais. Tačiau pasiekus tam tikrą ribą (maždaug 500 individų), ontologijos, saugomos tekstiniame faile užklausos vykdymo laikas pradeda smarkiai didėti, tuo tarpu reliacinėje duomenų bazėje saugomos ontologijos išlieka pastovus. Reikia pabrėžti, kad šiuose grafikuose pateikti tik užklausos vykdymo laikai, neįskaitant ontologijos įkėlimo į atmintį laiko, todėl galima teigti, kad didelės apimties ontologijose (turinčiose daug individų) užklausos greičiau vykdomos, kai ontologija saugoma reliacinėje duomenų bazėje.

37 paveikslėlyje pateikti bendri visų trijų užklausų vykdymo su visais individų kiekiais laikų vidurkiai. Šis grafikas parodo, kad egzistuoja skirtingų užklausų vykdymo laikų skirtumai, tačiau šiuo atveju jie yra minimalūs, nes kiekviena užklausa turėjo tik po vieną papildomą sąlygą.



37 pav. Užklausų vykdymo vidurkiai

Atliktas eksperimentas patvirtino hipotezę, kad esant dideliam ontologijos individų skaičiui, efektyviau vykdyti užklausas tada, kai ontologija saugoma reliacinėje duomenų bazėje, nes individai greičiau išrenkami iš reliacinės duomenų bazės vykdant *SQL* užklausas, negu ieškant jų didelės apimties darbinėje atmintyje saugomame *RDF* grafe.

6. Išvados

1. Analizuojant duomenų išrinkimo iš reliacinių duomenų bazių ir ontologijų procesus nustatyta, kad paieška ontologijose duoda prasmingesnius rezultatus, kadangi jose duomenis galima analizuoti semantiškai.
2. Didelės apimties ontologijas saugoti įprastu būdu (*XML* faile) nėra racionalu, jas efektyviau saugoti reliacinėje duomenų bazėje, kurios turi struktūrizuotas duomenų saugojimo priemones ir sudaro prielaidas vykdyti užklausas ontologijose, turinčiose daug egzempliorių.
3. Ontologijų saugojimui reliacinėje duomenų bazėje pasirinktas *OWL2RDB* algoritmas, nes jis geriausiai išnaudoja ontologijos saugojimo galimybes reliacinėje duomenų bazėje.
4. Buvo sukurti du algoritmai, vienas iš jų atstato ontologiją iš reliacinės duomenų bazės, kitas – transformuoja *SPARQL* į *SQL*. Šie algoritmai leidžia vykdyti ontologijos užklausas, kai ji saugojama reliacinėje duomenų bazėje, neįkeliant visos ontologijos į kompiuterio darbinę atmintį.
5. Realizuotų algoritmų testavimas su pavyzdine ontologija parodė, kad algoritmai gerai veikia atrenkant klases, klasių individus ir objektų savybes.
6. Atliekant eksperimentą su kelių tipų užklausomis ir didinant individų skaičių buvo nustatyta, kad sukurtas užklausų vykdymo metodas leidžia vykdyti užklausas greičiau negu jos vykdomos tekstiniame faile saugojamoje ontologijoje, nes į atmintį nėra įkeliami individai. Sukurto metodo privalumai ypač išryškėja tada, kai individų skaičius auga.
7. Sukurtas metodas buvo aprašytas dviejuose straipsniuose, kurie pateikiami prieduose. Vienas straipsnis buvo pristatytas konferencijoje Information Technologies‘ 2010, kitas – IVUS‘ 2010.

7. Literatūra

[1] Baker C. J. O., Cheung K. H. Semantic Web Revolutionizing Knowledge Discovery in the Life Sciences. *2007* Springer Science + Business Media, I ir II dalys, ISBN-13: 978-0-387-48436-5

[2] Bechhofer S., Harmelen F., Hendler J. ir kt. OWL Web Ontology Language Reference. Prieiga per internetą: http://www.w3.org/TR/owl-ref/.

[3] Bernes-Lee T., Hendler J., Lassila O.. The Semantic Web. Scientific American, [žiūrėta 2008-09-20]. Prieiga per internetą: http://www.jeckle.de/files/tblSW.pdf

[4] Bizer C., Cyganiak R.. D2R Server – Publishing Relational Databases on the Semantic Web. Freie Universitat Berlin.

[5] BodenReider O., Lussier Y.A. Semantic Webs for Life Sciences. School of Computer Science, University of Manchester, U.S. National Library of Medicine, Departments of Biomedical Informatics and Medicine Columbia University.

[6] Elliot B., Cheng E., Thomas-Ogbuji C., Meral Ozsoyoglu Z.. A Complete Translation from SPARQL into Efficient SQL.

[7] Golbeck J., Mannes A., Hendler J.. Semantic Web Technologies and Terrorist Network Analysis.

[8] Grau B.C., Kalyanpur A., Parsia B., Katz Y., Sirin E. Pellet: A Practical OWL-DL Reasoner. University of Maryland, MIND Lab, 8400 Baltimore Ave, College Park MD 20742, USA, Departamento de Informatica, Universidad de Valencia Av. Vicente Andres Estelles, s/n, 46100 Burjassot, Valencia, SPAIN

[9] Haase P., Broekstra J., Eberhart A., Volz R. A Comparison of RDF Query Languages, In SWC 2004, pages 502-517.

[10] Heflin J., Pan Z. DLDB: Extending Relational Databases to Support Semantic Web Queries.

[11] Horridge M., Jupp S., Moulton G., Rector A., Stewens R., Wroe C.. A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE tools. *2007* The University Of Manchester.

[12] Kjernsmo K., Passant A.. SPARQL New Features and Rationale. Prieiga per internetą: http://www.w3.org/TR/2009/WD-sparql-features-20090702/

[13] Lassila O., Swick R.R. Resource Description Framework (RDF) Model and Syntax Specification. Prieiga per internetą: http://www.w3.org/TR/PR-rdf-syntax/

[14] Noy N. F., McGuinness D. L.. Ontology Developement 101: A Guide to Creating Your First Ontology. Stanford University, Stanford, CA, 94305

[15] OWL Web Ontology Language Reference, [žiūrėta 2008-11-09]. Prieiga per Internetą: http://www.w3.org/TR/owl-ref/.

[16] Perez J., Arenas M., Gutierrez C.. Semantics and Complexity of SPARQL.

[17] Prud'hommeaux E., Seaborne A. SPARQL Query Language for RDF. Prieiga per internetą: http://www.w3.org/TR/rdf-sparql-query/.

[18] Son J., Jeong D., Baik D.. Practical Approach: Independently Using SPARQL-to-SQL Translation Algorithms on Storage.

[19] SparqlEngineDb [žiūrėta 2008-10-25]. Prieiga per internetą: http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/tutorial/usingtheSparqlEngine.htm

[20] SPASQL [žiūrėta 2008-10-25]. Prieiga per internetą: http://esw.w3.org/topic/SPASQL

[21] Vyšniauskas E., Nemuraitė L. Transforming Ontology Representation from OWL to Reliatonal Database. Kaunas University of Technology, Department of Information Systems Studentų st. 50-308, LT-51368 Kaunas, Lithuania.

8. Priedai

1 priedas. Straipsnis konferencijoje Information Technologies' 2010

ENHANCING CONNECTION BETWEEN ONTOLOGIES AND DATABASES WITH OWL 2 CONCEPTS AND SPARQL

Ernestas Vysniauskas, Lina Nemuraite, Algirdas Sukys, Bronius Paradauskas

Kaunas University of Technology, Department of Information Systems, Studentu 50-308a, Kaunas, Lithuania, Ernestas.Vysniauskas@stud.ktu.lt, Lina.Nemuraite@ktu.lt, Sukys.Algirdas@gmail.com, BoniusParadauskas@ktu.lt

Abstract. The goal of the paper is to present the enhanced database schema for storing ontologies considering new features of OWL 2 and possibilities of querying these ontologies using SPARQL. The growing size of ontologies and the scope of their applications require the effective means for storing ontology data that relational databases already have approved. Many existing ontology reasoning tools are using relational databases for this purpose. However, in practice almost all of them are using the straightforward approach restricted to representing instances whereas the effectiveness of processing ontological data may be considerably improved by keeping information about ontology classes, object properties and more advanced concepts in database tables. Previously we have presented the method and tool for transforming OWL ontologies to relational database. Currently, we have extended our representation with novel concepts of OWL 2, the recent Recommendation of W3C. Also, we present a prototype of a tool for extracting ontologies from relational databases and thus allowing the step-wise processing of SPARQL queries where SPARQL is used for querying ontology structures in a main memory and SQL is used for querying instances in the database.

Keywords. Ontology, relational database, OWL 2, SPARQL, SQL, mapping, transformation.

1 Introduction

Ontology descriptions, in particularly Web Ontology Language OWL, become more and more widely used in the World Wide Web and other fields as common information systems, data integration and software engineering. Currently, many areas are becoming knowledge-based [12]. Ontologies allow creating of better information systems by empowering them with advanced possibilities for operation and interoperability with

In our previous work [29, 30] we have proposed the mapping between OWL ontology and relational database and a tool for transforming ontologies into databases. Such an approach is needed because ontology based systems are growing in scope and storages of ontology reasoners are becoming unsuitable. While there are other solutions and tools for keeping ontologies in databases, the most of them are storing only RDF data. Lee and Goodwin, who have proposed the database-centric approach to ontology management support, notice that such methodology is still in its infancy [20]. In our approach, some concepts, e.g. ontology classes and properties are mapped to relational tables, relations and attributes, other (constraints) are stored like metadata in special tables. Using both direct mapping and metadata, it is possible to obtain appropriate relational structures and do not lose the ontological data. In connection with new features of OWL 2 and its supporting tools as Protégé [18] and Pellet [27], we have revised the previous representation and supplemented it with new concepts[I].

Our approach is well-suited for creating new databases from ontologies, however, in practice ontologies often are used for accessing already existing databases that usually are heterogeneous and distributed. Therefore, methodologies are even more needed for extracting ontologies from existing databases. It is a hard task to automatically obtain meaningful knowledge from such legacy systems without human intervention, so it is worth to ready beforehand ontology structures and to story them in databases for ontology management purposes. In cases when ontologies are being created from databases our approach fits as well, because the database schema obtained by our transformation is capable of the lossless[II] representation of ontological

[I] The research is pursued according the project proposal "Methodology and Technology Foundations for Semantically-Based Information System Design (SEMIS)"

[II] We have in mind the lossless transformation of OWL ontologies formulated using sufficient subset of its concepts because criteria of completeness and performance require for some compromise. Complete representation of OWL in a database is even undesirable as inference in OWL FULL is undecidable [14].

information in databases and the lossless retrieval of this information from databases into ontology reasoning tools. We present a prototype of a tool for extracting ontologies from relational databases, satisfying our schema, and allowing the step-wise processing of SPARQL queries where SPARQL is used for querying ontology structures in a main memory and SQL is used for querying instances in the database.

The rest of the paper is organized as follows. Section 2 presents related works. Section 3 is devoted to mapping of OWL 2 concepts to RDB concepts. Section 4 presents our approach to querying OWL 2 ontologies stored in relational databases. Section 5 draws conclusions and outlines the future work.

2 Related Work

OWL is different from conceptual modelling languages as ER or UML class diagrams as it has richer capabilities to describe classes and to handle incomplete knowledge. OWL 2, the emerging new version of OWL, is more expressive and still allows for complete and decidable computing [11]. Significant improvement in ontology management and reasoning tools has been achieved due to enhancement and additional functionality provided in OWL 2. For example, the widely used Protégé and Pellet systems and graphical OWL notation were extended with additional constructs of OWL 2 [18, 27, 16]. Consequently, we are aiming for the extending and improving our previous OWL2RDB transformation in accordance with new possibilities of OWL 2.

The new features of OWL 2 aim at increasing the relational expressivity of OWL 1 by allowing propagation of constraints along properties: transitivity of properties, subproperty and property chain axioms [10, 11, 23]. Object property axioms now can define reflexivity and symmetry, and various property restrictions: all values from, some values from, restrictions on values. The set of built-in OWL datatypes was extended from strings and integers in OWL 1 to XML schema datatypes and various datatype restrictions. As the lack of keys in OWL 1 was recognized as an important limitation in expressive power keys were introduced into OWL 2. They may be defined on a list of object or data properties. Also, OWL 2 adds syntactic sugar to make some common patterns easier to write. Since these constructs are simply shorthands, they do not change the expressiveness, semantics, or complexity of the language.

There were three dialects defined in OWL 1: OWL DL, OWL Full and the syntactic subset OWL Lite. These dialects exposed their insufficiency for implementing tools working with OWL ontologies [10]. To

We already have discussed existing approaches for representing ontologies in databases [1–3, 8, 19, 20, 22] in [30] and concluded in the rationale of creating bidirectional, lossless, model-based transformations between ontologies and database schemas. Metamodels of ontology language and database schema serve for this purpose. We are following the OWL 2 metamodel [23] for representing ontology. For a relational database, we use a part of Common Warehouse Metamodel (CWM) [4] that currently is under extension to more powerful CWM 2.x named as "Information Management Metamodel" (IMM) [15].

Also, we have studied the approaches for inverse mapping – i.e. from relational databases to ontology [5, 6, 9, 13, 26] (the survey is given in [28]). We analyze aspects of RDB2OWL transformations for lossless OWL2RDB transformation as most of relational concepts may be mapped to ontology structures, but not every ontology concept may be directly mapped to a relational database.

Among all these methods we can distinguish two ultimate information-lossless cases: storing ontology and its instances in the same manner (one fact table) or storing ontology and its instances in different schemas in order to improve access to instances while retaining the capacity of reasoning over the ontology. The first transformation method does not lose information, but it uses advantages of relational databases just for saving many records and does not preserve the real relational structure. The schema is in a low normal form and the performance of using transformed information normally should be slow e.g. [20]. The similar method is highly powered in Oracle Semantic Storage as it is supported with the native functionality of the Oracle database [31], where functionality of triggers helps to reasoning in ontology (alike in business rule manipulation techniques e.g. [21]). The second approach is much more promising for ordinary database management systems (e.g. [1, 2]). However, existing methods of that kind do not cover the sufficient subset of ontology concepts.

Our OWL2RDB transformation combines direct mapping of ontology classes, properties and instances with representing axioms and restrictions in metadata tables. Herewith we consider the reverse transformation of ontology from a database for efficient reasoning that may be achieved by joint usage of ontology query language SPARQL [25] and relational database query language SQL [7]. Reasoners that use ontologies represented in

XML files usually extract ontology schema along with its instances into a main memory and perform all inference there [27]. Performing full reasoning in memory ensures the completeness of query results but it is unsuitable for large ontologies having many instances. In our case, only ontology structures are extracted into a memory and processed by the inference engine. Results of inference are used for accessing individuals by SQL queries obtained by converting fragments of SPARQL to SQL.

This process is optimized in PelletDB reasoner for Oracle DB 11g due to the powerfulness of Oracle [16]. If ontology is of the acceptable size, PelletDB loads both the schema and the instance data from Oracle DB, then computes and saves all inferences back to the Oracle Database, which can be queried without additional reasoning. When instance data are too large to fit into memory, PelletDb extracts only the schema, computes additional schema inferences, and saves these inferences in Oracle Database. Then it is possible to perform instance reasoning using the schema inferences. The combination of reasoning in-memory together with instance reasoning in database provides a viable means to achieving more complete inference and query results than either solution can offer alone. While there is no question about competing with Oracle, our approach scales for every size of applications and may be implemented in non-commercial database management systems.

3 OWL 2 Concepts and their Mapping to RDB Concepts

OWL 1 was mainly focused on constructs for expressing information about classes and individuals, and exhibited some weakness regarding expressiveness for properties. OWL 2 offers new constructs for describing properties, a richer set of datatypes and makes some common patterns easier to write. In this paper we analyse these OWL 2 concepts that in our opinion are the most useful for real world applications and can be transformed to relational database schemas. As basic mappings are similar to OWL 1 mappings presented in [29, 30], in this paper we are focusing on mappings of new constructs. As previously, we are combining mappings of OWL 2 concepts with RDB concepts and storing the problematic (in mapping sense) knowledge in metadata tables. For explaining the proposed mapping, we will use the extended excerpt of Wine Ontology as our example (Figure 1) where it is represented using UML OWL 2 profile [24] implemented in Protégé OWL2UML plug-in.

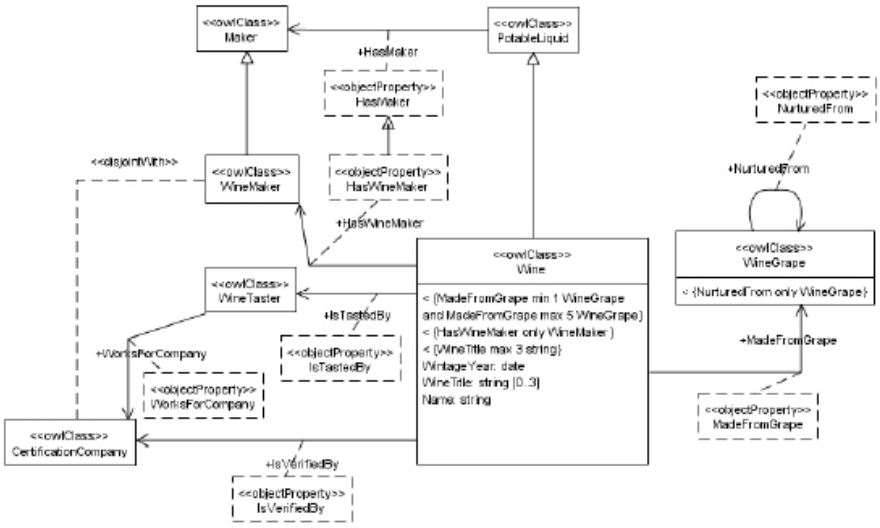


Figure 1. Example of OWL 2 wine ontology

3.1 OWL Classes and Class Axioms

In OWL 2, classes and property expressions are used to construct class expressions that represent sets of individuals by formally specifying conditions on the individuals' properties; individuals satisfying these conditions are said to be instances of the respective class expressions. OWL 2 provides axioms that allow relationships to be established between class expressions (Figure 2).

When we are converting the OWL ontology description to relational database schema, we map each ontology class to a database table. As the name of an ontology class is unique in the whole ontology and instances have unique names, we create a primary key for each table by adding some suffix to the corresponding class name, e.g. "*Id*", and the additional column by adding "*Name*" suffix to the class name for saving names of instances of the class. The fundamental taxonomic construct – the *SubClassOf* axiom, which allows to state that each instance of one class expression is also an instance of another class expression, is mapped to 1:0..1 relation

in RDB. The subclass doesn't need a column for saving names of its instances, because the instance of the subclass is also the instance of the super class and its name is already stated. These mappings for *PortableLiquid*, *Wine*, *WineGrape*, *WineMaker and WineTaster* classes of Wine Ontology example are presented in the upper part of Figure 3. We use our own UML profile for representing database schema where <<PK>>, <<FK>>, <<UK>> stereotypes mark primary keys, foreign keys and unique constraints; tags "id" mark names of foreign keys, and tags "uk" mark names of unique constraints.

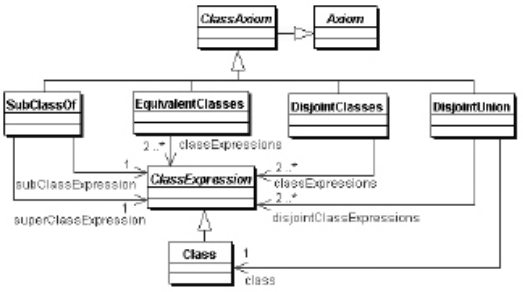


Figure 2. The OWL class descriptions diagram ([23])

The *EquivalentClasses* axiom defines that several class expressions are equivalent to each other, i.e. they have the same instances. The *DisjointClasses* axiom states that several class expressions are pair wise disjoint. The *DisjointUnion* class expression allows to define a class as a disjoint union of several class expressions and thus to express covering constraints. In our example, disjoint classes are the *WineMaker* and the *CertificationCompany*; they comprise one disjoint class group and we could be able to define a superclass of these classes e.g. "Company" as the disjoint union class if there were more disjoint groups. For preserving such information, we suggest saving all classes of the ontology in *OWLClasses* table with two main columns *classId*, which is an auto increment identification number, and *className*, which saves the unique name of the class. This name is also the name of the corresponding table. Information about groups of disjoint and equivalent classes is saved in metatables *OWLDisjointClasses* and *OWLEquivalentClasses*. The groups of equivalent or disjoint classes also are represented in *OWLEquivalentGroup* and *OWLDisjointGroup* tables. Metatables for OWL 2 disjoint, equivalent and disjoint union classes are presented in the lower part of Figure 3.

3.2 OWL 2 Properties and Property Axioms

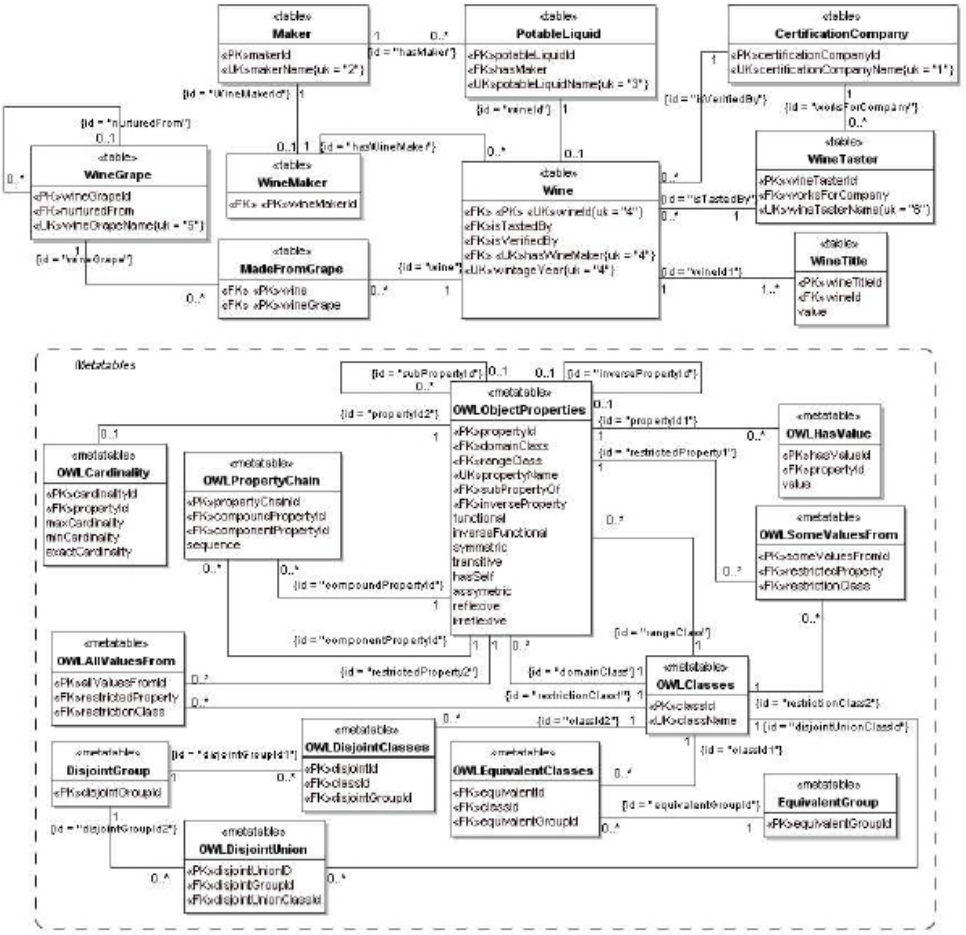
OWL 2 has two main categories of properties – object and data properties, and also annotation properties that may be useful for ontology documentation. Object properties relate individuals to other individuals. Data properties relate individuals to literals. We map the object property to the foreign key. Depending on the local cardinality of some class property and the object property is functional or not, one-to-many or many-to-many relation between tables of classes are created. In a case of many-to-many relation, an intermediate table must be created.

OWL 2 provides axioms for establishing relationships between object property expressions. The *ObjectPropertyDomain* and *ObjectPropertyRange* axioms can be used to restrict the first and the second individual, connected by an object property expression. The *FunctionalObjectProperty* and *InverseFunctionalObjectProperty* axioms define that each individual can have at most one outgoing or incoming connection of the specified object property expression respectively. The *InverseObjectProperties* axiom can be used to state that two object property expressions are the inverse of each other. The *ReflexiveObjectProperty*, *IrreflexiveObjectProperty*, *SymmetricObjectProperty*, *AsymmetricObjectProperty*, and *TransitiveObjectProperty* axioms define that an object property expression is reflexive, irreflexive, symmetric, asymmetric, or transitive. These axioms are represented in metatable "*OWLObjectProperties*" (Figure 3).

In OWL 2 there are two forms of object subproperties axioms. The basic form is *SubObjectPropertyOf*(*OPE*$_1$ *OPE*$_2$). This axiom states that the object property expression *OPE*$_1$ is a subproperty of the object property expression *OPE*$_2$ — that is, if an individual *x* is connected by *OPE*$_1$ to an individual *y*, then *x* is also connected by *OPE*$_2$ to *y*. E.g. in our example the class *PotableLiquid* has the object property *HasMaker*, and the class *Wine* has the object property *HasWineMaker* which is the subproperty of the property *HasMaker*. Information that one property is a subproperty of another property we save in the metatable *OWLObjectProperties*.

Another form of OWL2 object subproperty is *ObjectPropertyChain*. The axiom *SubObjectPropertyOf* (*ObjectPropertyChain*(*OPE*$_1$ ... *OPE*$_n$) *OPE*) states that, if an individual *x* is connected by a sequence of object property expressions *OPE*$_1$, ..., *OPE*$_n$ with an individual *y*, then *x* is also connected with *y* by the object property expression *OPE*. E.g. we have the class *Wine* and the object property *isTastedBy* with the range class

WineTaster. The class *WineTaster* has the object property *worksForCompany* with the range class *CertificationCompany*. We can declare the axiom *SubObjectPropertyOf*(*ObjectPropertyChain*(*a:isTastedBy a:worksForCompany*) *isVerifiedBy*) that means if some wine is tasted by the taster who works for some certification company then this wine is verified by this company. *ObjectPropertyChain* axioms are represented in metatable *OWLPropertyChain*. (Figure 3). This table has links to the compound and component object properties and the sequence number of some component property in the property chain.



At the last stage of transforming domain ontology into relational database, when whole schema is created, we must convert all assertions of classes and properties into records and fill the database. During this process object property chains can be used to gain some missing information about relations between objects. E.g. if we have both object property assertions *isTastedBy* and *worksForCompany* and the axiom *SubObjectPropertyOf*(*ObjectPropertyChain*(*a:isTastedBy a:worksForCompany*) *isVerifiedBy*) on some instance, we can create object property assertion and insert the appropriate value in the column *isVerifiedBy* of the table *Wine* automatically.

Ontology data properties relate individuals to literals. Functional data properties can be mapped to relational database columns of the tables corresponding to the domain classes of these properties. Because the OWL 2 was extended for representing ranges of data properties by the XML schema datatypes, we map XML schema datatypes to corresponding SQL datatypes. In a case of the data property is not functional or it has cardinality more than one, the data property is mapped to the additionally created table named by the data property name. This additional table has three columns – the auto increment identification number, the foreign key to the table of the corresponding domain class of this property and the value. The value column is SQL datatype corresponding to the XML schema datatype of the data property.

OWL 2 provides a new construct "*HasKey*" which allows keys to be defined for a given class. With this construct it is possible to give a list of object or data properties, which together identify resources of a given type. For example, if individuals of the class "*Wine*" are uniquely identified by data properties "*wineName*", "*wintageYear*" and the object property "*hasWineMaker*", then the OWL 2 axiom *HasKey(:Wine :wineName :wintageYear :hasWineMaker)* states that each named instance of the class "*Wine*" is uniquely identified by this set of properties – that is, if two named instances of the class coincide on values for each of key properties, then these two individuals are the same.

For converting the OWL ontology description to the RDB schema, we map the "*HasKey*" axiom on some properties for the certain class to the uniqueness constraint of columns of the corresponding table. Depending on "*HasKey*" properties count (one or many), we create the unique key on the single column, or the multi column (combination of columns) unique index of the table.

3.3 OWL Restrictions

In OWL 2 class expressions can be formed by placing restrictions on object property expressions. The *ObjectSomeValuesFrom*(*OPE CE*) class expression allows for existential quantification over an object property expression *OPE*, and it contains those individuals that are connected through an object property expression *OPE* to at least one instance of a class expression *CE*. The *ObjectAllValuesFrom*(*OPE CE*) class expression allows for universal quantification over an object property expression *OPE*, and it contains those individuals that are connected through an object property expression *OPE* only to instances of a class expression *CE*. The *ObjectHasValue*(*OPE a*) class expression contains those individuals that are connected by an object property expression *OPE* to a particular individual *a*. Finally, the *ObjectHasSelf*(*OPE*) class expression contains those individuals that are connected by an object property expression *OPE* to themselves.

When we are converting the OWL ontology description to the relational database schema we save this information in special metadata tables. *ObjectAllValuesFrom*, *ObjectSomeValuesFrom* and *ObjectHasValue* restrictions have their own metadata tables with column *restrictedProperty* which links to the table *OWLObjectProperties*. Metadata tables for *ObjectAllValuesFrom* and *ObjectSomeValuesFrom* restrictions also have column *restrictionClass*, which points to the table of the corresponding restriction source class (Figure 3). The *ObjectHasValue* restriction metadata table has the column "*Value*" for storing the value of the restricted resource of the corresponding property. Indication that object property has *ObjectHasSelf* restriction is saved in the column *hasSelf* of the *OWLObjectProperties* metatable.

Object property restrictions in OWL 2 can also be formed by placing restrictions on the cardinality of object property expressions *ObjectMinCardinality*, *ObjectMaxCardinality*, and *ObjectExactCardinality* that are saved in the metadata table *OWLCardinality*.

4 Querying OWL 2 Ontologies from Relational Databases

In this section we present the prototype for querying ontology, stored in a relational database according to the representation we have proposed. Usually, ontology reasoner (e.g. Pellet) reads ontology, including individuals, from a XML file (Figure 4). In our case, only ontology classes, their hierarchies, object and data properties, axioms and restrictions are extracted into a memory. Individuals are accessed by SQL queries obtained by converting fragments of SPARQL to SQL. The Ontology Database Integration component creates ontology model for the reasoner. This component analyses the database schema and metatables, builds the ontology model, rewrites SPARQL queries and executes SQL for obtaining results. The algorithm of transforming the database to ontology is based on the features of the previously described transformation from ontology into the database schema.

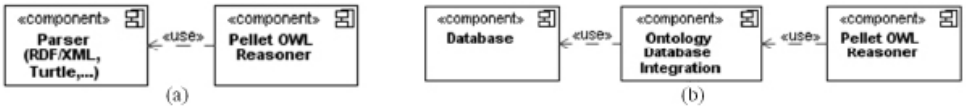


Figure 4. Common (a) versus enhanced (b) OWL reasoner

The following SPARQL query finds all potable liquids and their makers verified by the certain company:

```
select ?type ?drink ?maker where
{?type rdfs:subClassOf wine:PotableLiquid.FILTER(?type!=wine:PotableLiquid).
?drink rdf:type ?type .
?drink wine:hasMaker ?maker .
?drink wine:isVerifiedBy wine:French_Certification_Company}
```

The presented query has four conditional clauses. According to the 1st clause, Pellet OWL Reasoner finds all subclasses of the *PotableLiquid* class according to predicate *?type rdfs:subClassOf wine:PotableLiquid*,

where *FILTER* ensures finding of proper subclasses of the *PotableLiquid* (*Pellet OWL Reasoner* treats every class as a subclass of itself). The first clause of the query (`?type rdfs:subClassOf wine:PotableLiquid.FILTER(?type!= wine:PotableLiquid)`) returns the single record *Wine* and assigns it to the variable `?type`. SQL executes the remaining clauses. The second clause *?drink rdf:type ?type* rewritten to SQL finds all individuals of the class Wine. The 3rd clause *?drink wine:hasMaker ?maker* finds all makers of all previously found individuals and assigns them to the variable *?maker*. The example of SQL query that finds the maker "*Marieta*" of the drink (*Wine*) "*MariettaPetiteSyrah*" (*WineId*=3):

```
SELECT MakerName,MakerId FROM PotableLiquid, Maker, Wine
WHERE  PotableLiquid.hasMaker = Maker.MakerId and
       PotableLiquid.potableLiquidId = Wine.WineId and Wine.WineId = 3
```

The last, 4th clause *?drink wine:isVerifiedBy wine:French_Certification _Company* rewritten into SQL filters selected individuals by checking which of them is certified by the certain certification company. The sample data and results of the query are presented in Figure 5.

Data in Relational Database

PortableLiquid

potableLiquidId	potableLiquidName	hasMaker
1	FoxenCheninBlank	1
2	MariettaZinfandel	2
3	MariettaPetiteSyrah	2
4	Orange Juice	3

Maker

makerId	makerName
1	Foxen
2	Marietta
3	Cido

WineMaker

wineMakerId
1
2

Wine

wineId	isTastedBy	isVerifiedBy	hasWineMaker	vintageYear
1	2	1	1	2000
2	1	2	2	2001
3	2	1	2	1995

CertificationCompany

certificationCompanyId	certificationCompanyName
1	French Certification Company
2	Italian Certification Company

Query results

?type	?drink	?maker
Wine	FoxenCheninBlank	Foxen
Wine	MariettaPetiteSyrah	Marietta

Figure 5. Results from querying relational data representing Wine ontology

5 Conclusions and Future Work

In this paper we presented the mapping for transforming ontologies described in OWL 2 to relational database schemas. These mappings extend our previous transformation, oriented to OWL 1, with new concepts and offer more possibilities for representing the rich knowledge about a problem domain. Our OWL2RDB transformation combines direct mapping of ontology classes, properties and instances to database schema with representing axioms and restrictions in metadata tables. Our transformation is capable of the lossless representation of the chosen subset of ontology concepts in a database and the lossless retrieval of ontology schema from the database into ontology reasoning tools.

Our approach is well-suited for creating new databases from ontologies and creating ontologies for already existing databases. We argue that it is worth to story ontology structures in databases for ontology management purposes. We have tried a prototype of a tool for extracting ontologies from relational databases, satisfying our schema, and allowing the step-wise processing of SPARQL queries where SPARQL is used for querying ontology structures in a main memory and SQL is used for querying instances in the database.

Currently, we are working on the extension of our transformation tool and are willing to provide the transformations of the subset of OWL 2 concepts sufficient for representation of ontologies appropriate for applications of information systems. The OWL 2 QL profile that is oriented towards efficient implementation of tools working with ontologies stored in databases is unsuitable for real needs of information systems. OWL 2 QL profile has strong restrictions and is capable of working only with very simple ontologies when the actual applications require capturing business rules and transforming them to software code, integrating data from distributed resources, effectively communicating on the World Wide Web etc.

References

[1] **Astrova, I., Korda, N., Kalja, A.** Storing OWL Ontologies in SQL Relational Databases. *Engineering and Technology, Vol. 23,* August 2007, 167–172.

[2] **Barranco, C. D., Campana, J. R., Medina, J. M., Pons, O.** On Storing Ontologies Including Fuzzy Datatypes in Relational Databases. *In: Proceedings of Fuzzy Systems Conference 2007, IEEE International,* 2007, 1–6.

[3] **Bechhofer S., Horrocks I., Turi D.** The OWL Instance Store: System Description. *In: Proc. of the 20th Int. Conf. on Automated Deduction (CADE-20), LNAI, Springer,* 2005, 177–181.

[4] Common Warehouse Metamodel Specification. Object Management Group, Document Number: pas/06-04-02, 2006.

[5] **De Laborda, C. P., Conrad, S.** Database to Semantic Web Mapping using RDF Query Languages. *In: Conceptual Modeling - ER 2006, 25th International Conference on Conceptual Modeling, Tucson, Arizona, Springer Verlag, LNCS, Vol. 4215*, November 2006, 241–254.

[6] **De Laborda, C. P., Conrad, S.** Relational OWL – A Data and Schema Representation Format Based on OWL. *In: Proc. Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, Australia, CRPIT, Vol. 43*, 2005, 89–96.

[7] **Elliott, B., Cheng, E., Thomas-Ogbuji, C., Ozsoyoglu, Z. M.** A Complete Translation from SPARQL into Efficient SQL. *In: IDEAS 2009, September 16-18, Cetraro, Calabria,* 2009, 31–42.

[8] **Gali A., Chen, C. X., Claypool, K. T., Uceda-Sosa R.** From Ontology to Relational Databases. *In: Shan Wang et al. (Eds.): Conceptual Modeling for Advanced Application Domains, LNCS, Vol. 3289*, 2005, 278–289.

[9] **Ghawi, R., Cullot, N.** Database-to-Ontology Mapping Generation for Semantic Interoperability. *In: VDBL'07 conference, VLDB Endowment, ACM 978-1-59593-649-3/07/09,* September 23-28, 2007, 1–8.

[10] **Golbreich, C., Wallace, E. K., Patel-Schneider, P. F.** OWL 2 Web Ontology Language New Features and Rationale. *W3C Proposed Recommendation, 2009, http://www.w3.org/TR/2009/PR-owl2-new-features-20090922/.*

[11] **Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.** OWL 2: The next step for OWL. *In: Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 6(4),* November 2008, 309–322.

[12] **Gudas, S.** Enterprise knowledge modelling: domains and aspects. *Technological and economic development of economy, Vol. 15(2),* 2009, 281–293.

[13] **Hillairet, G., Bertrand, F., Yves, J., Lafaye, J. Y.** MDE for publishing Data on the Semantic Web. *In: Workshop on Transformation and Weaving Ontologies and Model Driven Engineering TWOMDE, Vol.395*, 2008, 32–4.

[14] **Horrocks, I.** OWL: A Description Logic Based Ontology Language. *In: Principles and Practice of Constraint Programming - CP 2005, LNCS, Vol. 3709*, 2005, 5–8.

[15] Information Management Metamodel (IMM) Specification. OMG Document Number: ptc/07-08-30, 2007.

[16] Introducing PelletDb. Expressive, Scalable Semantic Reasoning for the Enterprise. *Clark & Parsia LLC, 2009, available at http://clarkparsia.com/whitepapers/.*

[17] **Kendall, E., Bell, R., Burkhart, R., Dutra, M., Wallace, E.** Towards a Graphical Notation for OWL 2. *In: OWLED 2009, OWL: Experiences and Directions. Sixth International Workshop, Chantilly, Virginia, USA 23-24 October 2009,* 2009, 1–8.

[18] **Knublauch, R., Fergerson, W., Noy, N. F., Musen, M. A.** The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications. *In: Semantic Web – ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004, LNCS, Vol. 3298*, 2004, 229–243.

[19] **Konstantinou, N., Spanos, D. M., Nikolas, M.** Ontology and database mapping: a survey of current implementations and future directions. *Journal of Web Engineering, Vol. 7(1),* 2008, 001–024.

[20] **Lee, J., Goodwin, R.** Ontology management for large-scale enterprise systems. *Electronic Commerce Research and Applications, Spring, Vol. 5(1)*, 2006, 2–15.

[21] **Motiejunas, L, Butleris, R.** Business rules manipulation model. *Information Technology and Control, Vol. 36(3)*, 2007, 295–301.

[22] **Motik, B., Horrocks, I., Sattler, U.** Bridging the Gap Between OWL and Relational Databases. *In: WWW 2007, International World Wide Web Conference,* May 8–12, 2007, 807–816.

[23] **Motik, B., Patel-Schneider, P. F., Parsia, B.** OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. *W3C Proposed Recommendation 22 September 2009, available at http://www.w3.org/TR/2009/PR-owl2-syntax-20090922/.*

[24] Ontology Definition Metamodel. OMG Adopted Specification, OMG Document Number: ptc/2007-09-09, 2007.

[25] **Prud'hommeaux, E., Seaborne, A.** SPARQL Query Language for RDF. *W3C Recommendation 15 January 2008, available at http://www.w3.org/TR/rdf-sparql-query/.*

[26] **Seleng, M., Laclavik, M., Balogh, Z., Hluchý, Z.** RDB2Onto: Approach for creating semantic metadata from relational database data. *In: INFORMATICS'2007: proceedings of the ninth international conference on informatics. Bratislava, Slovak Society for Applied Cybernetics and Informatics,* 2007, 113–116.

[27] **Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., Katz, Y.** Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics, Vol. 5,* 2007, 51–53.

[28] **Tirmizi, S. H., Sequeda, J., Miranker, D.** Translating SQL Applications to the Semantic Web. *In: Proceedings of the 19th international conference on Database and Expert Systems Applications, LNCS,* Vol. 5181, 2008, 450–464.

[29] **Vysniauskas, E., Nemuraite, L.** Transforming Ontology Representation from OWL to Relational Database. *Information Technology and Control, Vol. 35(3A)*, 2006, 333–343.

[30] **Vysniauskas, E., Nemuraite, L.** Mapping of OWL ontology concepts to RDB schemas. *In: Information Technologies' 2009: proceedings of the 15th International Conference on Information and Software Technologies, IT 2009, Lithuania, April 23-24, 2009, Kaunas University of Technology, Kaunas, Technologija,* 2009, 317–327.

[31] **Wu, Z., Eadon, G., Das, S., Chong, E. I., Kolovski, V., Annamalai, M., Srinivasan, J.** Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. *In: Proceedings of IEEE 24th International Conference on Data Engineering, Mexico, Cancun,* 2008, 1239–1248.

2 priedas. Straipsnis konferencijoje IVUS' 2010

Semantinių užklausų vykdymas saugant ontologiją reliacinėje duomenų bazėje

Algirdas Šukys
Informacijos sistemų katedra
Kauno Technologijos Universitetas
Kaunas, Lietuva
sukys.algirdas@gmail.com

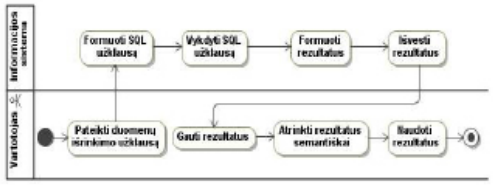
Lina Nemuraitė
Informacijos sistemų katedra
Kauno Technologijos Universitetas
Kaunas, Lietuva
lina.nemuraite@ktu.lt

***Anotacija* — populiarėjant ontologijoms ir didėjant jose saugomos informacijos kiekiui, atsiranda būtinybė saugoti ontologijas reliacinėse duomenų bazėse. Šiame straipsnyje supažindinsime su metodu, kuris leidžia vykdyti semantines *SPARQL* užklausas, kai ontologija saugoma reliacinėje duomenų bazėje.**

Raktiniai žodžiai – ontologija; reliacinė duomenų bazė; OWL; SPARQL; SQL

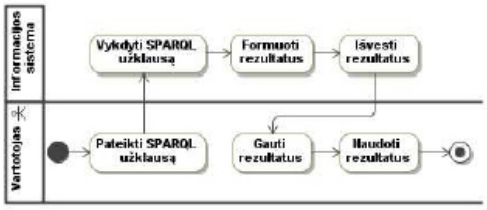
I. ĮVADAS

Ontologija – aukšto semantinio lygmens dalykinės srities aprašymas [3], kuris apima klases, objektus, jų savybes ir ryšius. Ontologija leidžia daryti semantines išvadas, todėl ontologijos paieškos sistema pateikia vartotojui prasmingesnius rezultatus, negu įprasta paieškos sistema. Reliacinėje duomenų bazėje įvykdžius užklausą, sistema pateikia pagal sintaksinius parametrus iš duomenų bazės išrinktus duomenis. Norėdamas suprasti šiuos duomenis, vartotojas pats analizuoja jų prasmę pagal tam tikrus kriterijus, kurių kompiuteris suprasti negali, taigi tam tikrą duomenų paieškos darbo dalį atlieka pats žmogus. Šis duomenų išrinkimo procesas pavaizduotas 1 paveiksle:



Paveikslas 1. Tradicinis duomenų išrinkimo procesas

Pirmoji proceso veikla yra vartotojo užklausos pateikimas. Tai atliekama kompiuterio ekrane pasirenkant tam tikrus užklausos parametrus (įvedami raktiniai žodžiai, pasirenkamos reikšmės iš sąrašo ir kt.). Tada pagal vartotojo pateiktus parametrus formuojama *SQL* užklausa (žinoma, jeigu vartotojas yra pakankamai kvalifikuotas, jis gali pats tiesiogiai rašyti užklausas), kuri išrenka iš duomenų bazės norimus įrašus. Gavęs rezultatų sąrašą, vartotojas pats turi juose ieškoti loginių sąryšių tarp resursų, analizuoti, kokias savybes jie turi ir pagal tai iš daugybės rezultatų atsirinkti sau tinkamus. Tarkime, vartotojas nori internetu užsisakyti viešbutį Ispanijoje liepos mėnesiui. Reliacinės duomenų bazės paieškos sistema su pateiktu sakiniu „Suraskite viešbutį Ispanijoje netoli Viduržemio jūros liepos mėnesiui" nesusidorotų, nes nėra pajėgi suprasti sakinio reikšmės, ji pateiktų visų galimų užsisakyti viešbučių sąrašą, iš kurių vartotojas pats išsirinktų tinkamą. Norint atlikti tokio tipo paiešką (jos schema pateikta 2 pav.) turi būti sudarytas aukštesnio lygmens duomenų aprašymo modelis – ontologija, nes ji suteikia galimybę daug detaliau aprašyti dalykinę sritį bei sukurti saryšius tarp skirtingų dalykinių sričių.



Paveikslas 2. Duomenų išrinkimo procesas naudojant ontologijas

II. RDF, RDF SCHEMA IR OWL

Ontologijai saugoti naudojamas *RDF* resursų aprašymo karkasas. *RDF* pagrindas yra modelis, skirtas vaizduoti savybes ir savybių reikšmes. *RDF* modelis remiasi skirtingų duomenų vaizdavimo bendruomenių principais. *RDF* savybės gali būti laikomos resursų atributais, jos taip pat reiškia ryšius tarp resursų, tuo *RDF* primena *ER* diagramą. Objektinio projektavimo požiūriu, resursai atitinka objektus, o savybės – objektų kintamuosius [5].

RDF modelis sudarytas iš trijų pagrindinių dalių:

- *subject* – bet koks daiktas, aprašytas *RDF*. Tai gali būti visas interneto puslapis, jo dalis ar, tarkime, knyga. Kiekvienas resursas turi turėti *URI* – adresą, kuris jį identifikuoja.
- *predicate* – savybė, charakteristika, atributas ar ryšys, skirtas apibūdinti resursą.
- *object* – savybės reikšmė. Tai gali būti kitas resursas arba duomenų tipo reikšmė.

Šios trys pagrindinės *RDF* dalys sudaro formuluotę. Tokia duomenų struktūra yra tinkama aprašyti didžiąją dalį informacijos, apdorojamos kompiuterių [7]. Išnagrinėkime pavyzdį „Dangus yra mėlynos spalvos“. Šioje formuluotėje veiksnys yra „Dangus“, jo savybė – „turi spalvą“, kurios reikšmė – „mėlyna“. Formuluotė pateikta 1 lentelėje.

1 LENTELĖ. RDF FORMULUOTĖS PAVYZDYS

subject	http://www.pavyzdys.lt/Dangus
predicate	turiSpalvą
object	„Mėlyna“

Vienas iš *RDF* modelio privalumų yra atviro pasaulio prielaida [9]. Tiesa yra tai, kas modelyje pasakyta, tačiau tų dalykų, kurie nepaminėti, mes nelaikome netiesa, todėl modelį lengva papildyti naujais faktais, nepažeidžiant esamų faktų teisingumo.

Siekiant išplėsti semantines *RDF* galimybes, buvo sukurtos dvi papildomos kalbos: *RDF* Schema (*RDFS*) ir *OWL*. *RDFS* yra paremta *RDF* kalba, turi visas *RDF* ir papildomų savybių, iš kurių svarbiausios yra klasių ir savybių ribojimai. Jei *RDF* leidžia kurti resursus, tai *RDFS* leidžia juos grupuoti į klases. *RDFS* tai pat leidžia kurti klasių hierarchijas, tam naudojama savybė *rdfs:subClassOf*. Kiekviena klasė gali turėti neribojamą skaičių tėvinių klasių. Hierarchiją galima kurti ir savybėms, tam skirtas *rdfs:subPropertyOf* savybė. Kita *RDFS* galimybė – savybių ribojimas, kai savybę leidžiama naudoti tik tam tikrų klasių egzemplioriams, nurodant *domain* ir *range*.

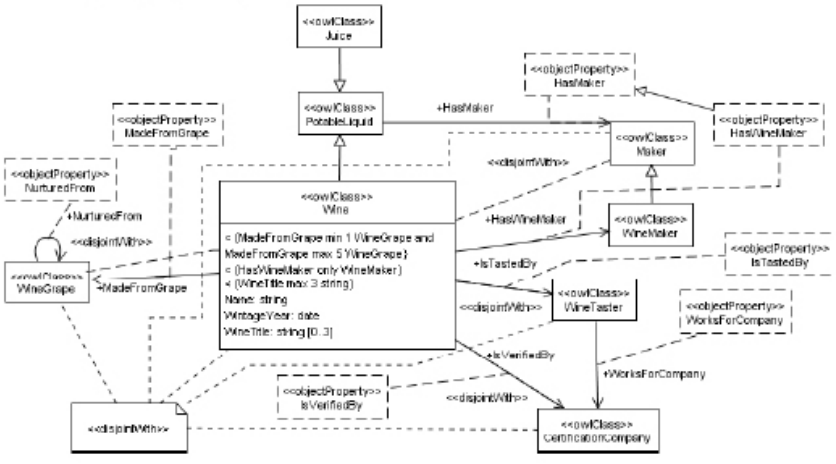
OWL yra *RDFS* pagrindu sukurta kalba, kuri dar labiau išplečia ontologijos aprašymo galimybes. *OWL* papildymus galima skirstyti į kelias grupes [4]:

- ekvivalentiškumas – leidžia nustatyti, kad du resursai (klasės, individai ar savybės) aprašo tą patį realaus pasaulio objektą. Taip pat leidžiama apibrėžti, kad dvi klasės yra skirtingos (*distinct*), t. y. jei objektas priklauso vienai klasei, jis negali priklausyti kitai.
- savybių charakteristikos, ribojimai – *OWL* suteikia galimybes plačiau aprašyti savybes (tranzityvumą, simetriškumą, inversiją), nustatyti jų tarpusavio sąryšius, kardinalumus.
- aibių savybės – naujų klasių kūrimo galimybės, pasinaudojant esamomis klasėmis ir sudarant jų sąjungas, sankirtas ir pan.

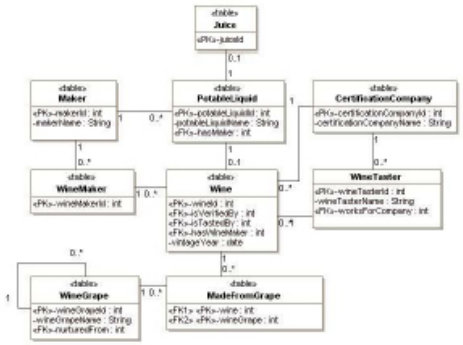
III. *OWL2RDB* ALGORITMAS

RDF aprašytos ontologijos saugomos tekstiniuose failuose, tačiau toks sprendimas turi trūkumą – kai ontologija yra didelės apimties, reikia daug kompiuterio darbinės atminties resursų norint saugoti tokią ontologiją atmintyje ir vykdyti joje užklausas.

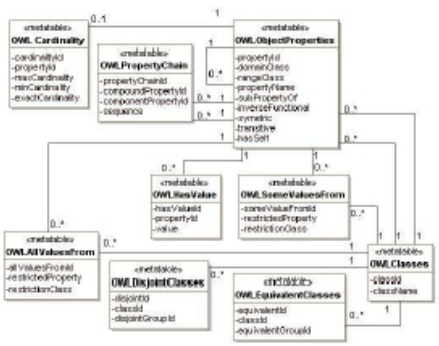
Tačiau ontologija gali būti efektyviai saugoma ir reliacinėje duomenų bazėje [1]. Saugojimui gali būti naudojami įvairūs algoritmai, kurie tarpusavyje skiriasi *RDF* grafo saugojimo metodu. Kai kurie jų yra nepriklausomi nuo ontologijos schemos – t. y. duomenų bazės schema pastovi, todėl jie yra lankstesni, lengviau išplečiami. Vienas tokių - *Chebotko* algoritmas, paremtas grafo saugojimu vienoje lentelėje su trimis stulpeliais (*subject*, *predicate*, *object*) [6]. Algoritmas *SPARQL-to-SQL* [8] naudoja *RDFLib* mechanizmą, kuris saugo ontologiją trijose lentelėse. Norint saugoti sudėtingesnes *OWL* formuluotes, pavyzdžiui, kardinalumo ribojimus, toks saugojimo būdas nėra efektyvus, nes atsiranda daug perteklinių formuluočių [9], todėl pasirinktas algoritmas *OWL2RDB* [2], kuris transformuoja ontologiją į reliacinės duomenų bazės schemą. Šio algoritmo idėja – kiekvienai klasei sukurti po lentelę, kurios jungiasi ryšiais 1:0..1 (jei aprašoma klasių hierarchija) arba 1:* (jei aprašoma objekto savybė). Klasių objektus atitinka lentelių įrašai. Ontologijos apribojimams sukuriamos papildomos metaduomenų lentelės. 3 paveiksle pateikta vyno ontologija, sukurta ontologijos kūrimo įrankiu *Protégé* [3]. 4 ir 5 paveiksluose ši ontologija transformuota į reliacinės duomenų bazės schemą.



Paveikslas 3. Vyno ontologija



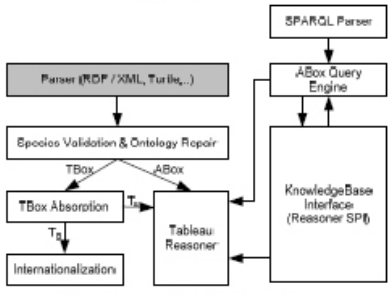
Paveikslas 4. Vyno ontologija, transformuota į reliacinės duomenų bazės schemą



Paveikslas 5. Ontologijos metaduomenų lentelės

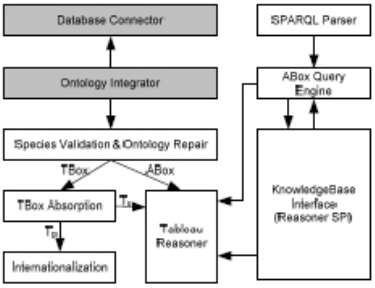
IV. UŽKLAUSŲ VYKDYMAS *OWL2RDB* ALGORITMU SAUGOJAMOJE ONTOLOGIJOJE

Norint įvykdyti *SPARQL* užklausą, kai ontologija saugoma reliacinėje duomenų bazėje pagal *OWL2RDB* algoritmą, užklausą reikia transformuoti. Transformavimui mes siūlome metodą, kuris pradinę *SPARQL* užklausą išskaido į *SPARQL* užklausą, skirtą atrinkti tik klases, ir *SQL* užklausas, skirtas atrinkti individualiems konceptams. Šis metodas paremtas *Pellet OWL Reasoner* ontologijų analizės ir užklausų vykdymo biblioteka (6 pav.).



Paveikslas 6. Pellet OWL architektūra

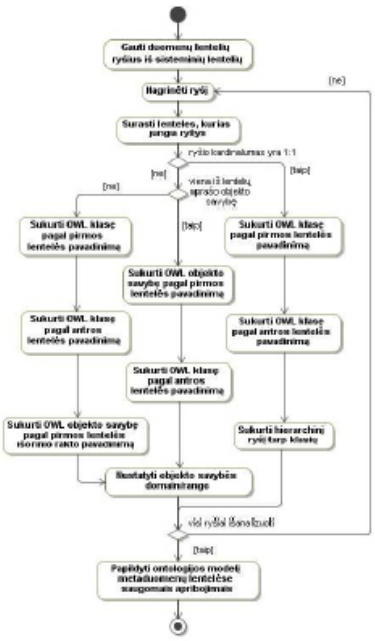
Pellet OWL Reasoner biblioteka pritaikyta vykdyti užklausas, kai ontologija saugoma tekstiniame faile, tuo tarpu *OWL2RDB* algoritmas saugo ontologiją reliacinėje duomenų bazėje, todėl buvo pakeistas vienas iš *Pellet OWL* komponentų. Vietoj komponento *Parser*, kuris apdoroja tekstinį ontologijos failą ir pagal jį sudaro ontologijos modelį, sukurtas kitas komponentas – *Ontology Integrator* (7 pav.). Jo veikimo algoritmas pateikiamas kitame skyrelyje.



Paveikslas 7. Papildyta Pellet OWL architektūra

A. Ontologijos modelio sudarymas pagal reliacinės duomenų bazės schemą

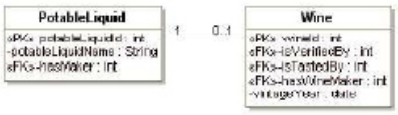
Šis algoritmas skirtas sukurti *Pellet OWL Reasoner* ontologijos modelį pagal reliacinės duomenų bazės schemą. Jo veikimas paremtas *SQL Server* metaduomenų analize. Algoritmas išrenka visus duomenų bazės lentelių ryšius ir pagal juos sukuria klases, nustato klasių hierarchijas, objektų bei duomenų tipų savybes (8 pav.).



Paveikslas 8. Ontologijos modelio iš reliacinės DB schemos sudarymo procesas

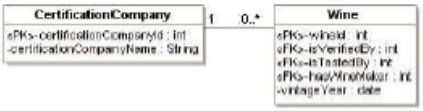
Toliau pateikiamas algoritmo aprašymas.

- Jeigu ryšio kardinalumas yra 1:0..1, sukuriamos dvi klasės (jų vardai atitinka lentelių vadus) ir tarp jų nustatoma hierarchija. 9 paveiksle pateiktame pavyzdyje sukurtos klasės *PotableLiquid* ir *Wine*, *PotableLiquid* nustatyta kaip tėvinė *Wine* klasė.



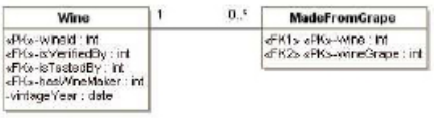
Paveikslas 9. Hierarchinis ryšys tarp duomenų lentelių

- Jeigu ryšio kardinalumas yra 1:*, kuriama objekto savybė, kurios pavadinimas nustatomas pagal antrosios klasės išorinio rakto atributo pavadinimą. Pirmoji klasė nustatoma kaip objekto savybės *range*, antroji – *domain*. 10 paveiksle pateiktame pavyzdyje sukurta objekto savybė *isVerifiedBy*, jos *domain* – klasė *Wine*, *range* – *CertificationCompany*.

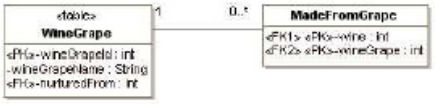


Paveikslas 10. Objekto tipo ryšys tarp duomenų lentelių

- Jeigu viena iš ryšio lentelių reiškia objekto savybę (tai nustatoma iš kardinalumo metaduomenų lentelių), sukuriama objekto savybė. Pagal kitą lentelę sukuriama klasė, ji nustatoma kaip objekto savybės *domain* arba *range*, priklausomai nuo išorinio rakto pozicijos. 11 paveiksle pateiktame pavyzdyje bus sukurta objekto savybė *MadeFromGrape*, jos *domain* bus klasė *Wine*. 12 paveiksle pavyzdyje klasė *WineGrape* nustatoma objekto savybės *MadeFromGrape range* klase.



Paveikslas 11. Objekto tipo domain ryšys tarp duomenų lentelių



Paveikslas 12. Objekto tipo range ryšys tarp duomenų lentelių

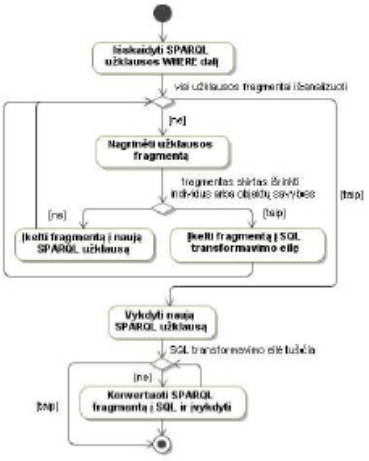
Į ontologijos modelį neįtraukiami lentelių įrašai (individualūs ontologijos konceptai), nes *OWL2RDB* algoritmo idėja remiasi tuo, kad individualūs konceptai neturi būti įkeliami į operatyvinę atmintį. Jie išrenkami vėliau naudojant *SQL*.

B. *Užklausų vykdymas*

Antrasis algoritmas – *SPARQL* užklausų vykdymas ontologijoje. Vartotojas pateikia įprastą *SPARQL* užklausą, kuri vėliau transformuojama (13 pav.). Transformacija reikalinga dėl to, kad dalis ontologijos (klasių hierarchija) saugoma *Pellet OWL Reasoner* bibliotekoje, dalis lieka reliacinėje duomenų bazėje (individualūs konceptai), todėl ir užklausos turi būti vykdomos tiek *Pellet OWL Reasoner* bibliotekoje, tiek reliacinėje duomenų bazėje. Šis algoritmas pateiktos *SPARQL* užklausos sąlygos dalį skaido į dvi dalis:

- sudaroma nauja *SPARQL* užklausa, kurioje lieka tik sąlygos dalys, susijusios su klasių gavimu;
- sudaromos *SQL* užklausos, skirtos išrinkti individualius konceptus.

Užklausos vykdomos etapais, kiekviena užklausa naudoja prieš tai vykdytų užklausų rezultatus – jei reikia rasti klasių individualius konceptus, naudojama *SPARQL* išrinkta klasių hierarchija. Šis procesas detaliau paaiškintas kitame skyrelyje.



Paveikslas 13. Užklausos vykdymo procesas

V. *SPARQL* UŽKLAUSOS PAVYZDYS

Šiame skyriuje detaliai aprašysime pavyzdinės užklausos vykdymą 4 ir 5 paveiksluose pavaizduotoje ontologijoje. Siekiant nustatyti sistemos atrenkamų duomenų teisingumą, jie buvo lyginami su rezultatais, gautais naudojant tekstiniame faile saugomą ontologiją, užklausą vykdant *Pellet OWL Reasoner*. 14 paveiksle pateikti ontologijos lentelių duomenys.

Wine

wineId	isTastedBy	isVerifiedBy	hasWineMaker	vintageYear
1	2	1	1	2000
2	1	2	2	2001
3	2	1	2	1995

PotableLiquid

potableLiquidId	potableLiquidName	hasMaker
1	FoxenCheninBlank	1
2	MariettaZinfandel	2
3	MariettaPetiteSyrah	2
4	Orange Juice	3

Maker

makerId	makerName
1	Foxen
2	Mariette
3	Cido

WineMaker

wineMakerId
1
2

Juice

juiceId
4

CertificationCompany

certificationCompanyId	certificationCompanyName
1	French Certification Company
2	Italian Certification Company

Paveikslas 14. Ontologijos lentelių duomenys

Toliau pateikta užklausa, kuri ontologijoje randa visus gėrimus ir jų gamintojus, sertifikuotus kompanijos „*French_Certification_Company*“:

```
select ?type ?drink ?maker where
{
  ?type rdfs:subClassOf wine:PotableLiquid .
  FILTER (?type != wine:PotableLiquid) .
  ?drink rdf:type ?type .
  ?drink wine:isVerifiedBy
  wine:French_Certification_Company .
  ?drink wine:hasWineMaker ?maker
}
```

Užklausa turi 4 sąlygos dalis, kurių kiekviena toliau aprašyta detaliai.

A. Pirmosios užklausos sąlygos vykdymas

```
?type rdfs:subClassOf wine:PotableLiquid.
FILTER(?type!=wine:PotableLiquid)
```

Ši užklausos dalis vykdoma *Pellet OWL Reasoner*, ji grąžina visas *PotableLiquid* poklases, atmetant pačią *PotableLiquid*, tam naudojamas *FILTER*, nes *Pellet OWL Reasoner* laiko, kad klasė yra savo paties poklase. Šiame pavyzdyje gaunamas vienintelis įrašas *Wine*, jis priskiriamas kintamajam `?type`:

```
select ?type where
{
  ?type rdfs:subClassOf wine:PotableLiquid .
  FILTER (?type != wine:PotableLiquid)
}
```

Gauti rezultatai pateikti 2 lentelėje.

2 LENTELĖ. PIRMOJO UŽKLAUSOS VYKDYMO ETAPO REZULTATAI

?type
Wine

B. Antrosios užklausos sąlygos vykdymas

```
?drink rdf:type ?type
```

Šioje užklausos dalyje randami visi ankstesnėje užklausoje gautų klasių egzemplioriai. *SQL* užklausa formuojama pagal klasių pavadinimus. Kadangi klasė *Wine* turi tėvinę klasę *PotableLiquid* (tai sužinoma iš *Pellet OWL Reasoner* ontologijos modelio), o individualių konceptų pavadinimai saugomi aukščiausioje pagal hierarchiją klasėje, vynų pavadinimai ir *id* gaunami kreipiantis į ją:

```
select
   WineId,
   PotableLiquidName as WineName
from
   Wine,
   PotableLiquid
where
   WineId = PotableLiquidId
```

Gaunami visi *Wine* klasės egzemplioriai, jie priskiriami kintamajam `?drink` (3 lentelėje rodomos *WineName* reikšmės):

3 LENTELĖ. ANTROJO UŽKLAUSOS VYKDYMO ETAPO REZULTATAI

?type	?drink
Wine	FoxenCheninBlank
Wine	MariettaZinfandel
Wine	MariettaPetiteSyrah

C. Trečiosios užklausos sąlygos vykdymas

```
?drink wine:isVerifiedBy
wine:French_Certification_Company
```

Filtruojamas vynų sąrašas, išrenkami tik tie vynai, kuriuos sertifikavo nurodyta kompanija. Čia taip pat iš *Pellet OWL Reasoner* ontologijos modelio sužinoma, kad savybės `isVerifiedBy` *domain* klasė yra *Wine*, o *range* – *CertificationCompany*. Pagal šias reikšmes formuojama *SQL* užklausa:

```
select
   CertificationCompanyName,
   CertificationCompanyId
from
   Wine,
   CertificationCompany
where
   Wine.isVerifiedBy =
   CertificationCompany.
      CertificationCompanyId
   and
   Wine.WineId = 3
```

Gauta *CertificationCompanyName* reikšmė lyginama su „*French_Certification_Company*“, jeigu vynas sertifikuotas šios kompanijos, jis paliekamas sąraše. 4 lentelėje pateikti trečiojo etapo rezultatai:

4 LENTELĖ. TREČIOJO UŽKLAUSOS VYKDYMO ETAPO REZULTATAI

?type	?drink
Wine	FoxenCheninBlank
Wine	MariettaPetiteSyrah

D. Ketvirtosios užklausos sąlygos vykdymas

```
?drink wine:hasMaker ?maker
```

Vykdomas savybės išrinkimas. Pirmiausia randama savybės *hasMaker domain* ir *range* klasės iš *Pellet OWL Reasoner* ontologijos modelio. `hasMaker` domain yra *PotableLiquid*, o range – *Maker*. *SQL* užklausa formuojama pagal šias reikšmes. Ankstesnėje užklausoje išrinkti rezultatai (`?drink`) priklauso klasei *Wine*, todėl įrašoma papildoma sąlyga `PotableLiquid.potableLiquidId = Wine.WineId`, sujungti *Wine* ir *PotableLiquid* klases. Žemiau pateikta vyno, kurio id = 3, gamintojo radimas. Atskira užklausa vykdoma kiekvienam vynui.

```
select
   MakerName,
   MakerId
from
   PotableLiquid,
   Maker,
   Wine
where
   PotableLiquid.hasMaker = Maker.MakerId
   and
   PotableLiquid.potableLiquidId =
   Wine.WineId and
   Wine.WineId = 3
```

Gautas gamintojų sąrašas priskiriamas kintamajam `?maker`. 5 lentelėje pateikti galutiniai užklausos vykdymo rezultatai.

5 LENTELĖ. GALUTINIO UŽKLAUSOS VYKDYMO ETAPO REZULTATAI

?type	?drink	?maker
Wine	FoxenCheninBlank	Foxen
Wine	MariettaPetiteSyrah	Marietta

Galutiniai rezultatai sutapo su rezultatais, kada ontologija saugoma tekstiniame faile.

VI. IŠVADOS

Šiame straipsnyje pristatytas metodas, leidžiantis vykdyti užklausas ontologijose, kurios saugomos reliacinėse duomenų bazėse pagal *OWL2RDB* algoritmą. Metodo veikimo esmė – išskaidyti pradinę *SPARQL* užklausą ir vykdyti ją etapais – pirmiausia išrinkti rezultatus klasių, vėliau individualių konceptų lygmenyje.

Metodui išbandyti buvo realizuotas prototipas, kuriuo gaunami užklausų vykdymo rezultatai palyginti su rezultatais, gaunamais saugant ontologijas įprastu būdu – tekstiniuose failuose. Sukurtas metodas leidžia išrinkti klasių hierarchijas, rasti individualius konceptus ir objektų savybes.

Sukurtas metodas pasižymi tuo, kad individai užklausų vykdymo metu nėra įkeliami į pagrindinę atmintį, o išrenkami *SQL* kalba, todėl šis metodas leidžia analizuoti ontologijas, turinčias daug egzempliorių, kurių neįmanoma patalpinti pagrindinėje atmintyje.

Ateityje numatyta išanalizuoti sudėtingesnius užklausų atvejus ir išsamiai išnagrinėti *SPARQL* transformavimo į *SQL* galimybes, o taip pat atlikti didesnės apimties eksperimentus, kurie leistų įvertinti metodo efektyvumą.

LITERATŪRA

[1] C. J. O. Baker, K. H. Cheung. Semantic Web Revolutionizing Knowledge Discovery in the Life Sciences. Springer Science + Business Media, I ir II dalys, 2007.

[2] E. Vyšniauskas, L. Nemuraitė. Transforming Ontology Representation from OWL to Reliatonal Database. *Information Technology and Control, Vol. 35(3A)*, 2006, 333–343.

[3] M. Horridge, S. Jupp, G. Moulton, A. Rector, R. Stevens, C. Wroe. A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE tools. The University Of Manchester, *2007*.

[4] J. Golbeck, A. Mannes, J. Hendler. Semantic Web Technologies and Terrorist Network Analysis. In: Emergent Technologies and Enabling Policies for Counter Terrorism, IEEE Press, 2005, 1–15.

[5] O. Lassila, R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Prieiga per internetą: http://www.w3.org/TR/PR-rdf-syntax/.

[6] J. Son, D. Jeong, D. Baik. Practical Approach: Independently Using SPARQL-to-SQL Translation Algorithms on Storage. In Fourth International Conference on Networked Computing and Advanced Information Management, 2008, 598–603.

[7] K. Kjernsmo, A. Passant. SPARQL New Features and Rationale. 2009, Prieiga per internetą: http://www.w3.org/TR/2009/ WD-sparql-features-20090702/.

[8] B. Elliot, E. Cheng, C. Thomas-Ogbuji, Z. Meral Ozsoyoglu. A Complete Translation from SPARQL into Efficient SQL. In: Proceedings of the 2009 International Database Engineering & Applications Symposium 2009, 31–42.

[9] J. Bock, S. Grimm, J. Henß, J. Kleb. A Database Backend for OWL. In: Proceedings of the 5th International Workshop on OWL: Experiences and Directions OWLED 2009, Chantilly, VA, United States, October 23-24, 2009 , Vol. 529, 2009, 1–8.