KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Tomas Vileiniškis

Reliacinės DB turinio publikavimo pasauliniame semantiniame tinkle galimybių tyrimas

Magistro darbas

Darbo vadovas:

doc. dr. R. Butkienė

Kaunas, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Tomas Vileiniškis

Reliacinės DB turinio publikavimo pasauliniame semantiniame tinkle galimybių tyrimas

Magistro darbas

Recenzentas
doc. dr. G. Vilutis
2012-05-

Vadovas
doc. dr. R. Butkienė
2012-05-

Atliko
IFM-0/4 gr. stud.
Tomas Vileiniškis
2012-05-23

Kaunas, 2012

A Research Into Opportunities Of Publishing Relational Data On The Semantic Web

Summary

Currently the World Wide Web consists of billions of interlinked hypertext documents, most of which are backed by relational databases. While these documents can be perfectly read by humans, computers find it hard to understand what those pages are about and how they relate to each other.

The idea of Semantic Web is to create a web of data by giving a well-defined meaning to all the documents on the web. The core of Semantic Web is Resource Description Framework (RDF) – a data model for conceptual description of web resources. Using RDF and ontologies, information can be semantically structured, which further gives the ability for computers to process it.

The problem, however, lies in the way data is currently stored on the web. In order for the idea of Semantic Web to become a reality, semantically marked up data has to exist in the first place. Since a vast amount of current data resides in relational databases, the most convenient way would be to map relational data into RDF.

This work is dedicated to analyze existing techniques of mapping and publishing relational data on the semantic web. By taking a practical approach, a methodology is proposed for using one of the chosen tools. The methodology outlines necessary steps for dealing with the most common relational database design patterns while mapping it into RDF.

Further on the work, a semantic web application is developed by following the steps of proposed methodology. The end results show that methodology can be successfully applied when publishing relational data on the semantic web and for developing rich web applications.

Terminų ir santrumpų žodynėlis

Pasaulinis semantinis tinklas (*angl. Semantic Web*) – naujos kartos tinklas, kuriame duomenims suteikiama prasmė, susiejant juos semantiniais ryšiais.

RDB (*Reliacinė Duomenų Bazė*) – duomenų rinkinys, organizuotas pagal reliacinį duomenų modelį.

DBVS (Duomenų Bazių Valdymo Sistema) – programinės įrangos paketas, skirtas duomenų bazės valdymui.

IS (Informacinė Sistema) – sistema, kurios tarpusavyje susiję komponentai dirbdami kartu surenka, apdoroja, saugo ir platina informaciją.

MVC (*angl. Model-View-Controller*) – programinių sistemų architektūros paradigma, atskirianti duomenų logiką nuo jų pateikimo logikos.

RDF (*angl. Resource Description Framework*) – standartizuota resursų aprašymo kalba semantiniame tinkle.

OWL (*angl. Web Ontology Language*) – standartizuota ontologijų užrašymo kalba semantiniame tinkle.

XML (*angl. Extensible Markup Language*) – duomenų struktūrizavimo kalba.

SPARQL – protokolas ir užklausų kalba, skirta manipuliavimui RDF duomenimis.

URI (*angl. Uniform Resource Identifier*) – standartas resursų identifikavimui.

URL (*angl. Uniform Resource Locator*) – standartas resursų vietos tinkle nusakymui.

RDB2RDF – procesas, kurio metu RDB duomenys transformuojami į RDF trejetus.

ETL (*angl. Extract-Transform-Load*) – procesas, kurio metu duomenys išgaunami, transformuojami ir perkeliami iš vieno duomenų šaltinio į kitą.

Virtualus atvaizdavimas (*angl. Mapping*) – procesas, skirtas sąsajos sukūrimui tarp dviejų skirtingų duomenų modelių.

Turinys

ĮVADAS **8**
1. PASAULINIO SEMANTINIO TINKLO ANALIZĖ **9**
1.1. ANALIZĖS TIKSLAS 9
1.2. TYRIMO OBJEKTAS, SRITIS IR PROBLEMA 9
1.3. ANALIZĖS METODAI 9
1.4. VARTOTOJŲ ANALIZĖ 10
1.4.1. Vartotojų aibė, tipai ir savybės *10*
1.4.2. Vartotojų tikslai ir problemos *10*
1.5. SEMANTINIO TINKLO TECHNOLOGIJŲ ANALIZĖ 11
1.5.1. Semantinio tinklo technologijų stekas *11*
1.5.2. XML ir RDF analizė *12*
1.5.3. URI analizė *13*
1.5.4. OWL analizė *15*
1.5.5. SPARQL užklausų kalbos analizė *15*
1.6. TYRIMO OBJEKTO ANALIZĖ 16
1.6.1. Reliacinių DB analizė *16*
1.6.2. RDB2RDF metodų analizė *18*
1.7. EGZISTUOJANČIŲ RDB2RDF TECHNOLOGIJŲ ANALIZĖ 20
1.7.1. D2RQ platformos analizė *20*
1.7.2. SquirrelRDF analizė *21*
1.7.3. Openlink Virtuoso RDF Views analizė *23*
1.7.4. Triplify analizė *24*
1.8. ANALIZĖS REZULTATAI IR IŠVADOS 26
1.9. TYRIMO TIKSLAS IR UŽDAVINIAI 26
2. KURIAMOS METODIKOS IR IS REIKALAVIMŲ ANALIZĖ IR SPECIFIKACIJA **28**
2.1. REIKALAVIMŲ SPECIFIKACIJA KURIAMAI METODIKAI 28
2.2. REIKALAVIMŲ SPECIFIKACIJA IS PROGRAMINIAM PROTOTIPUI 29
3. IS PROGRAMINIO PROTOTIPO PROJEKTAS **35**
3.1. SISTEMOS ARCHITEKTŪROS PROJEKTAS 36
3.1.1. Sistemos loginė architektūra *36*
3.1.2. Sistemos posistemių projektai *37*
3.2. DUOMENŲ BAZĖS SCHEMA 42
3.3. IS REALIZACIJOS MODELIS 45
4. D2RQ ĮRANKIO PRAKTINIO TAIKYMO METODIKOS SUDARYMAS **46**
4.1. *D2RQ* VIRTUALAUS ATVAIZDAVIMO KALBOS ANALIZĖ 46
4.1.1. Database *47*
4.1.2. ClassMap *47*
4.1.3. PropertyBridge *49*
4.2. *D2RQ* PLATFORMOS SUDEDAMOSIOS DALYS 54
4.3. FORMALIZUOTA *D2RQ* ĮRANKIO PRAKTINIO TAIKYMO METODIKA 55
5. IS PROTOTIPO REALIZACIJA **60**
5.1. FILMŲ NUOMOS DALYKINĖS SRITIES ONTOLOGIJOS SCHEMA 60
5.2. IS PROTOTIPO RDB ELEMENTŲ ATVAIZDAI Į ONTOLOGIJOS ELEMENTUS 61
5.3. IS PROTOTIPO REALIZACIJA JAVA PLATFORMOJE 62
5.3.1. Java klasių generavimas Schemagen įrankiu *63*
5.3.2. Duomenų logikos sluoksnio realizacija *63*
5.3.3. MVC komponentų realizacija *67*
6. EKSPERIMENTINIS SPRENDIMO TYRIMAS **70**
6.1. I EKSPERIMENTO ETAPAS 71
6.2. II EKSPERIMENTO ETAPAS 73
6.3. III EKSPERIMENTO ETAPAS 76
6.4. EKSPERIMENTO IŠVADOS 79
7. IŠVADOS **80**
8. LITERATŪRA **81**
9. PRIEDAS **82**
9.1. *D2RQ* VIRTUALAUS ATVAIZDAVIMO TAISYKLIŲ FAILO TURINYS 82
9.2. DALYKINĖS SRITIES *OWL* ONTOLOGIJOS IŠEITIES TEKSTAS 90
9.3. INFORMACINĖS SISTEMOS IŠEITIES TEKSTAI (*CD*) 94

Lentelių turinys

1 lent. Lyginamosios RDB2RDF įrankių analizės apibendrinti rezultatai 26
2 lent. PA „Peržiūrėti filmų pasiūlą“ specifikacija 30
3 lent. PA „Peržiūrėti filmo aprašą“ specifikacija 31
4 lent. PA „Valdyti užsakymus“ specifikacija 32
5 lent. PA „Valdyti klientus“ specifikacija 33
6 lent. PA „Gauti filmų nuomos informaciją“ specifikacija 34
7 lent. DB lentelių atributų aprašas 43
8 lent. *d2rq:Database* elemento savybės 47
9 lent. *d2rq:ClassMap* elemento savybės 48
10 lent. *d2rq:PropertyBridge* elemento savybės 50
11 lent. RDB elementų atvaizdai į ontologijos elementus 61

Paveikslų turinys

1 pav. Semantinio tinklo technologijų stekas 11
2 pav. RDF trejetas 12
3 pav. RDF grafas, sudarytas iš kelių RDF trejetų 13
4 pav. Pavyzdinė RDB lentelių schema 17
5 pav. Pavyzdinis duomenų saugojimo būdas RDB lentelėse 18
6 pav. RDB2RDF konteksto schema 19
7 pav. D2RQ platformos architektūra [7] 21
8 pav. SquirrelRDF struktūra 22
9 pav. *Triplify* struktūra ir sąsaja su egzistuojančiomis *WEB* technologijomis [11] 24
10 pav. Panaudojimo atvejų diagrama D2RQ praktinio taikymo metodikai 28
11 pav. Panaudojimo atvejų diagrama programiniam IS prototipui 29
12 pav. PA „Peržiūrėti filmų pasiūlą“ sekų diagrama 30
13 pav. PA „Peržiūrėti filmo aprašą“ sekų diagrama 31
14 pav. PA „Valdyti užsakymus“ sekų diagrama 32
15 pav. PA „Valdyti klientus“ sekų diagrama 33
16 pav. PA „Gauti filmų nuomos informaciją“ sekų diagrama 34
17 pav. IS dalykinės srities esybių klasių diagrama. 35
18 pav. IS loginė architektūra, paremta MVC paradigma. 36
19 pav. PA „Valdyti užsakymus“ realizavimas projektinėmis klasėmis 37
20 pav. Užsakymų valdymo posistemio elgsena 37
21 pav. PA „Valdyti klientus“ realizavimas projektinėmis klasėmis 38
22 pav. Klientų valdymo posistemio elgsena 38
23 pav. PA „Peržiūrėti filmų pasiūlą“ realizavimas projektinėmis klasėmis 39
24 pav. Filmų valdymo posistemio elgsena PA „Peržiūrėti filmų pasiūlą“ vykdymo metu 39
25 pav. PA „Peržiūrėti filmo aprašą“ realizavimas projektinėmis klasėmis 40
26 pav. Filmų valdymo posistemio elgsena PA „Peržiūrėti filmo aprašą“ vykdymo metu.....40
27 pav. Duomenų valdymo variklio operacijų vykdymo logika 41
28 pav. Detali duomenų valdymo variklio elgsena pagrindinių operacijų vykdymo metu 41
29 pav. Duomenų bazės schema 42
30 pav. Pagrindiniai IS realizuojantys komponentai 45
31 pav. IS diegimo diagrama 45
32 pav. Resurso fragmentas HTML interfeise 53
33 pav. *D2RQ* platformos panaudos atvejai 54
34 pav. *D2RQ* įrankio praktinio taikymo metodika (I d.) 55

35 pav. *D2RQ* įrankio praktinio taikymo metodika (II d.) 58
36 pav. Dalykinės srities ontologijos schema 60
37 pav. Eksperimento kontekstinė schema 70
38 pav. Pradinis D2R serverio puslapis 71
39 pav. RDF resursas aprašytas HTML formatu 72
40 pav. Pradinis IS puslapis 73
41 pav. Konkretaus filmo aprašo puslapis 74
42 pav. Duomenų integravimo procesas kliento pusėje 74
43 pav. Filmo aprašas RDF formatu 75
44 pav. Duomenų integravimo procesas klientų posistemyje 75
45 pav. Eksperimentinės IS pagrindinis puslapis 76
46 pav. Semantinės paieškos rezultatai 77
47 pav. *SPARQL* užklausos rezultatai (I) 78
48 pav. SPARQL užklausos rezultatai (II) 79

Įvadas

Šiuo metu informacija pasauliniame (*angl. World Wide Web*) tinkle tarpusavyje susieta nuorodomis, pagal kurių kontekstinę reikšmę informacijos šaltinių tarpusavio sąryšis dažniausiai lengvai suprantamas žmogui, tačiau nesuprantamas kompiuteriams. Pastaruosius keletą metų vis plačiau kalbama apie pasaulinį semantinį tinklą (*angl. Semantic Web*) – tinklą, kuriame kompiuteriai geba suvokti ir apdoroti informaciją iš esamo pasaulinio tinklo. Semantinio tinklo vizija aprašo standartizuotus metodus bei technologijas, kurie padėtų kompiuteriams atsakyti į klausimus: Apie ką yra informacija? Koks jos sąryšis su kitais informacijos šaltiniais?

Informacijos aprašymo ir pateikimo pagrindu semantiniame tinkle laikomas *RDF* (*Resource Description Framework*) standartas, paremtas grafo tipo duomenų modeliu. Šio standarto dėka, aprašomiems duomenims suteikiami semantiniai ryšiai, kurie yra naujosios kartos tinklo esminis bruožas.

Siekiant semantinio tinklo viziją paversti realybe, visų pirma būtina viena sąlyga – *RDF* formatu aprašytų ir tarpusavyje susietų duomenų egzistavimas. Pirmosios kartos pasauliniame tinkle duomenys pateikiami įvairiausiais formatais. Tačiau remiantis 2007 m. atliktu tyrimu[1], neindeksuojamose duomenų bazėse laikoma beveik 500 kartų daugiau duomenų, negu indeksuojamame statiniame tinkle, taip pat, net 70% visų tinklalapių yra paremti reliacinėmis duomenų bazėmis. Taigi, galima daryti išvadą, kad semantinio tinklo sėkmė didžiąja dalimi priklauso nuo galimybių reliacinėse duomenų bazėse esančius duomenis publikuoti naujosios kartos tinkle. Nors šiam tikslui yra sukurta nemažai įrankių, jų galimybės vis dar nėra pakankamai ištirtos.

Šiame darbe atliekama egzistuojančių įrankių bei technologijų, skirtų reliacinių duomenų bazių turinio publikavimui semantiniame tinkle lyginamoji analizė bei jų galimybių tyrimas. Pagal iškeltus kriterijus pasirenkamas vienas įrankis, kurio atžvilgiu atliekamas tolimesnis išsamus jo praktinio panaudojimo galimybių tyrimas. Tyrimo rezultatai perteikiami sukuriant formalizuotą įrankio praktinio taikymo metodiką bei pasiūlant architektūrinį sprendimą įrankio panaudojimui informacinių sistemų kūrime. Sukurti sprendimai įvertinami eksperimentiškai, realizuojant demonstracinę informacinę sistemą bei išpublikuojant mokomosios reliacinės duomenų bazės duomenis *RDF* formatu.

[1] Bin He, Mitesh Patel, Zhen Zhang, Kevin Chen-Chuan Chang (2007). "Accessing the Deep Web: A Survey"

1. Pasaulinio semantinio tinklo analizė

1.1. Analizės tikslas

Analizės etape siekiama šių tikslų:

- Susipažinti su Semantinio tinklo standartais ir technologijomis, įsisavinant kiekvienos iš jų paskirtį ir taikymą naujosios kartos tinklo kūrimo procese.
- Susipažinti su tyrimo objektu ir iš jo kylančiomis problemomis bei sudaryti tyrimo srities esminių kriterijų sąrašą tolimesniam analizės žingsniui.
- Remiantis sudarytais kriterijais, išanalizuoti egzistuojančias reliacinių duomenų bazių (RDB) turinio publikavimo Semantiniame tinkle technologijas.
- Apibendrinus RDB turinio publikavimo Semantiniame tinkle technologijų analizės rezultatus, pasirinkti vieną iš technologijų / įrankių tolimesniam tyrimui.

1.2. Tyrimo objektas, sritis ir problema

Tyrimo objektas

Realiacinės DB turinio publikavimas semantiniame pasauliniame tinkle.

Tyrimo sritis

Realiacinės DB turinio publikavimo semantiniame pasauliniame tinkle technologijos, jų galimybės ir jų praktinis taikymas.

Problema

Vis labiau plėtojantis semantinio pasaulinio tinklo idėjai, didėja poreikis realiacinėse DB esančią informaciją padaryti prieinama semantinio tinklo taikomosioms programoms bei kitiems klientams. Tačiau tam skirtos technologijos nėra pakankamai ištirtos, neaiškios ir jų praktinio taikymo galimybės.

1.3. Analizės metodai

Pagrindiniu tyrimo objekto analizės metodu pasirinkta mokslinės literatūros šaltinių analizė ir dokumentų turinio analizė. RDB turinio publikavimo Semantiniame tinkle

technologijų analizė atliekama lyginamosios analizės principu, iš kontekstinės informacijos išskiriant esminę, atitinkančią iškeltus kriterijus.

1.4. Vartotojų analizė

1.4.1. Vartotojų aibė, tipai ir savybės

Vartotojų aibė – tai asmenys, vienokiu ar kitokiu būdu susiję su Semantinio tinklo vizija. Skiriami šie pagrindiniai vartotojų tipai:

- Semantinio tinklo kūrėjai – asmenys, plėtojantys Semantinio tinklo technologijas.
- Semantinio tinklo vartotojai – asmenys, naudojantys Semantinio tinklo technologijas.
    - Tinklalapių bei RDB savininkai – asmeninių tinklalapių, reliacinių duomenų bazių savininkai (administratoriai), turintys į jas pilnavertes teises.
    - Interneto vartotojai – atsitiktiniai internetą naršantys žmonės.

1.4.2. Vartotojų tikslai ir problemos

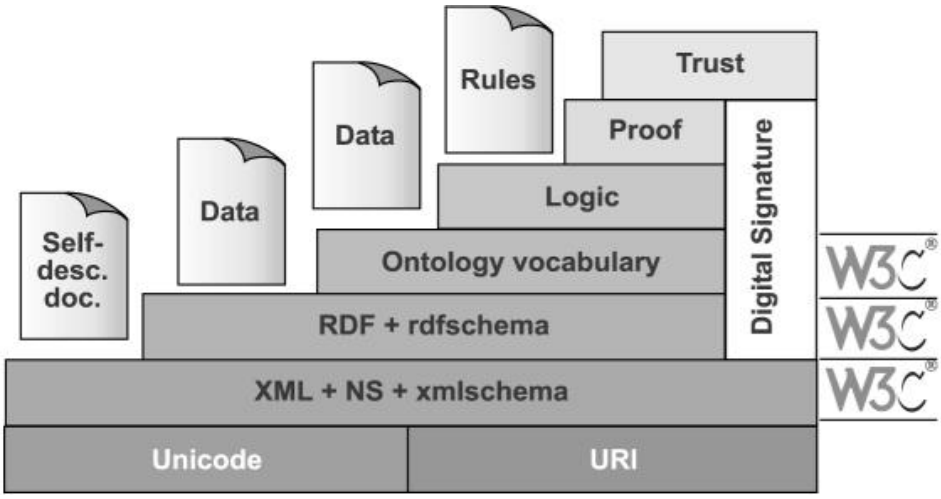
- Kūrėjų tikslas – vystyti Semantinio tinklo technologijas, atsižvelgiant į jų silpnąsias vietas ir trūkumus.
- Tinklalapių bei RDB savininkų tikslas – paversti turimus duomenis prieinamais Semantiniame tinkle ir taip prisidėti prie jo vystymo.
- Interneto vartotojų tikslas – gauti didesnę naudą iš interneto: tikslesnius, greitesnius, struktūrizuotus informacijos paieškos rezultatus, efektyvesnį informacijos filtravimą, jos apdorojimą, įsisavinimą ir pan.

1.5. Semantinio tinklo technologijų analizė

1.5.1. Semantinio tinklo technologijų stekas

Semantinio tinklo technologijų stekas atspindi naujosios kartos tinklo architektūrą. Joje hierarchiškai, skirtinguose sluoksniuose pateikiamos technologijos ir standartai būtini semantinio tinklo įgyvendinimui (*žr. 1 pav.*). Kiekvienas sluoksnis išnaudoja hierarchiškai žemesniame sluoksnyje esančių technologijų teikiamas galimybes. Visus steko sluoksnius, pradedant nuo žemiausio, galima išskaidyti į tris pagrindines dalis:

- WWW tinklo technologijos – dabartiniame pasauliniame tinkle naudojamos technologijos ir standartai (*Unicode, URI, XML*).
- semantinio tinklo technologijos – tai W3C konsorciumo standartizuotos technologijos, naudojamos specifiškai naujos kartos tinklo taikomiesiems uždaviniams spręsti (*RDF, RDFS, OWL, SPARQL*).
- koncepcinės semantinio tinklo technologijos – tai technologijos, kurios taip pat būtinos semantiniam tinklui įgyvendinti, tačiau, kolkas nėra standartizuotos ir siekia tik idėjinį lygmenį (*Logic, Proof, Trust*).



1 pav. Semantinio tinklo technologijų stekas [2]

Svarbiausios ir aktualiausios tyrimo sričiai semantinio tinklo technologijos detaliai apžvelgiamos sekančiuose darbo skyreliuose.

[2] http://www.w3.org/2001/12/semweb-fin/w3csw

1.5.2. *XML* ir *RDF* analizė

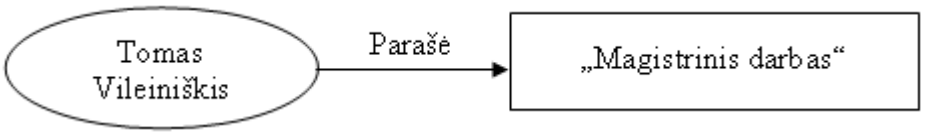
XML (*Extensible Markup Language*) yra duomenų struktūrizavimo kalba, suteikainti vieningą formatą duomenų apsikeitimui internete. Deja, *XML* nėra pajėgus aprašyti duomenų semantinius ryšius – neaišku, ką gauta struktūrizuota informacija reiškia. Tuo tarpu *RDF* [1] (*Resource Description Framework*) siekia išspręsti iš *XML* kylančias problemas, aprašydamas duomenis trilypiais (*angl. Triple*) duomenų rinkiniais. Pastarieji susideda iš:

- aprašomojo subjekto (*angl. Subject*)
- atributo (*angl. Predicate*)
- objekto (*angl. Object*)

Tokiu būdu galima interpretuoti, kad tam tikri subjektai yra susiję tam tikrais ryšiais su tam tikrais objektais. Pavyzdžiui, natūralia kalba užrašytas teiginys - „Tomas Vileiniškis parašė magistrinį darbą“:

- *Tomas Vileiniškis* – šiuo atveju aprašomas magistrinio darbo autorius, todėl jis vadinamas aprašomuoju subjektu.
- *Parašė* – šis teiginys identifikuoja aprašomojo subjekto savybę ir yra vadinamas atributu.
- *Magistrinį darbą* – aprašomojo subjekto identifikuojamos savybės reikšmė vadinama objektu.

Grafiškai šis trilypis duomenų rinkinys (toliau - *RDF* trejetas) atrodo taip:



2 pav. RDF trejetas

RDF trejetai gali būti aprašomi keletu formų, viena iš pagrindinių ir bene plačiausiai naudojamų yra *XML* kalba:

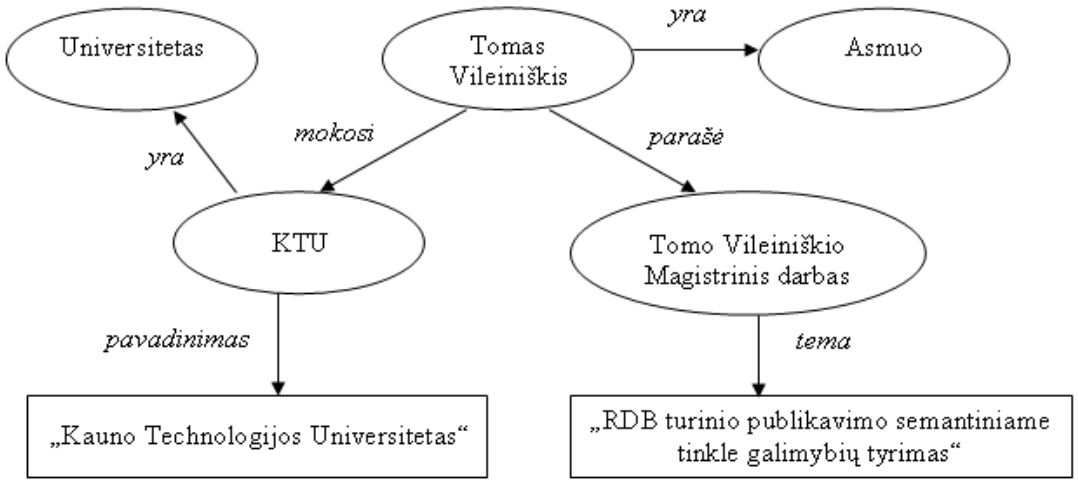
```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
     <rdf:Description rdf:about="http://pavyzdys.com/Asmuo/Tomas_Vileiniškis">
               <foaf:made>Magistrinis darbas</foaf:made>
     </rdf:Description>
</rdf:RDF>
```

Kompaktiškesne forma *RDF* trejetai aprašomi *N3*[3] notacijoje:

```
@prefix foaf: < http://xmlns.com/foaf/0.1/>.

<http://pavyzdys.com/Asmuo/Tomas_Vileiniškis>
  foaf:made "Magistrinis darbas".
```

RDF trejetų rinkiniai, tarpusavyje sujungti semantiniais ryšiais sudaro *RDF* grafus:



3 pav. RDF grafas, sudarytas iš kelių RDF trejetų

RDF grafas, taip pat kaip ir atskiri *RDF* trejetai gali būti aprašomas viename faile (arba keliuose, susiejant juos nuorodomis) jau minėtais būdais. Tarpusavyje jungiant kelis *RDF* grafus, gaunami vis platesni aprašomojo subjekto semantiniai sąryšiai ne tik su jį apibūdinančiais objektais, bet ir su kitais, giminingais subjektais.

1.5.3. *URI* analizė

URI [2] (*angl. Uniform Resource Identifier*) yra standartas, skirtas resursų (tokių kaip tinkalalapiai ar įvairūs dokumentai) identifikavimui ir lokalizavimui internetinėje erdvėje ir ne tik. *URI* nėra vien tik naujosios kartos semantinio tinklo standartas, jis yra viena iš svarbiausių ir dabartinio tinklo dalių.*URI* klasifikuojamas pagal skirtingas, tačiau glaudžiai tarpusavyje susijusias prasmes:

- *URN* (*angl. Uniform Resource Name*) – tai *URI*, kurie naudojami konkrečiam resursui suteikiant vardą. Šis resursas nebūtinai turi egzistuoti internetinėje erdvėje.

[3] http://www.w3.org/DesignIssues/Notation3.html

- *URL* (*angl. Uniform Resource Locator*) – tai *URI,* kurie naudojami nusakant konkretaus resurso vietą.

Dabartiniame *WWW* tinkle URI naudojami kaip *URL* tinklo dokumentams adresuoti. Tačiau semantiniame tinkle atsiranda poreikis ne tik resursų adresavimui, bet ir vardų jiems suteikimui. Toks poreikis gali iššaukti keletą neaiškumų, pavyzdžiui: tarkime magistrinio darbo autorių reprezentuoja tinklalapis, kurio *URL* yra *http://www.pavyzdys.com/TomasVileiniškis*, tačiau koks *URI* aprašo patį resursą „Tomas Vileiniškis“, o ne jo tinklalapį, lieka neaišku. Taip pat neaiškus ir teiginio „Tomas Vileiniškis turi asmeninį tinklalapį adresu *http://www.pavyzdys.com/TomasVileiniškis“* aprašymas *RDF* formatu. Siekiant išvengti tokių kliūčių, semantiniame tinkle įvedamos dvi resursų sąvokos:

- informaciniai resursai – tai tinkle esantys dokumentai (tinklalapiai, paveikslai ir pan.)
- neinformaciniai resursai – tai realaus pasaulio objektai, nebūtinai egzistuojantys internete (žmonės, daiktai, spalvos ir pan.)

To paties *URI* identifikatoriaus suteikimas tiek informaciniam, tiek neinformaciniam resursui sukeltų dviprasmybę, todėl neinformaciniai resursai identifikuojami dviem metodais:

- *URI* su diezo simboliu – šiuo atveju, *URI* *htttp://www.pavyzdys.com/TomasVileiniškis* nurodytų Tomą Vileiniškį reprezentuojantį tinklalapį, o *URI* *htttp://www.pavyzdys.com/TomasVileiniškis#apie* reprezentuotų Tomą Vileiniškį kaip asmenį (objektą).
- *URI* nukreipimu pagal *HTTP 303* kodą – šiuo atveju, kreipiantis į neinformacinį resursą, kurio *URI* *htttp://www.pavyzdys.com/id/TomasVileiniškis*, atitinkamai sukonfigūruotas *WEB* serveris atsakytų nukreipimo kodu 303 ir nukreipimo adresu *htttp://www.pavyzdys.com/apie/TomasVileiniškis*, nurodančiu neinformacinį resursą aprašantį tinklalapį. Sėkmingam kreipiniui į informacinį resursą, standartiškai suteikiamas *HTTP* atsako kodas 200.

Naudotino metodo pasirinkimą lemia tokie kriterijai kaip aprašomų resursų dydis, serverio konfigūravimo galimybės, serverio apkrovos ribojimai bei tvarkingesnių *URI* poreikis.

1.5.4. OWL analizė

OWL [3] *(angl. Web Ontology Language)* yra semantinio tinklo ontologijų užrašymo kalba, praplečianti *RDF* formatu pateikiamos struktūrizuotos informacijos išraiškingumą. *OWL* siekia detaliai apibrėžti *RDF* žodynuose naudojamų sąvokų reikšmes ir jų tarpusavio sąryšį – šis sąryšis bendriniu atveju vadinamas ontologija. Pagal sudėtingumą ir ekspresyvumą skiriami 3 pagrindiniai *OWL* kalbos lygiai:

- *OWL Lite* – skirtas vartotojams, suinteresuotiems bazinėmis klasifikavimo hierarchijų ir kardinalumo apribojimų funkcijomis.
- *OWL DL* (*angl. Description Logics*) – skirtas vartotojams, siekiantiems maksimalių išraiškingumo galimybių, tuo pačiu užtikrinant galutinį skaičiavimų įvykdymą. *OWL DL* apima visas *OWL* kalbos koncepcijas, tačiau jos gali būti naudojamos tik iki tam tikro lygio.
- *OWL Full* – skirtas vartotojams, taip pat siekiantiems maksimalių išraiškingumo galimybių, tačiau šiuo atveju neužtikrinamas galutinis skaičiavimų įvykdymas.

W3C konsorciumas rekomenduoja[4] naudoti naujai sukurtą *OWL* kalbos plėtinį *OWL 2*. Abiejų kalbų struktūra išlieka ganėtinai panaši, todėl *OWL* kalba sukurtos ontologijos pilnai suderinamos naujoje *OWL 2* versijoje, kuri praplečia pirmykštę naujomis funkcijomis ir galimybėmis.

1.5.5. SPARQL užklausų kalbos analizė

SPARQL [4] yra užklausų protokolas ir kalba, skirta informacijos išrinkimui bei įterpimui *RDF* grafuose. Užklausos, paremtos šia kalba gali būti vykdomos tarp skirtingų duomenų šaltinių. Nors *SPARQL* turi daug panašumų su reliacinėse duomenų bazėse naudojama *SQL* kalba, kol kas ji negali prilygti pastarąjai nei galimybėmis, nei užklausų vykdymo greičiu. Tačiau tai natūralu, nes *SPARQL* yra pradinėje vystymosi stadijoje.

Pateikiamas *SPARQL* užklausos pavyzdys pagal 1.5.2 skyrelyje matomą *RDF* trejeto aprašą:

[4] http://www.w3.org/TR/owl2-overview/

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?darbai
FROM <http://pavyzdys.com/Asmuo/Tomas_Vileiniškis.rdf>
WHERE {
  ?x foaf:made ?darbai.
}
```

Šios paprasčiausios užklausos tikslas yra surasti visus autoriaus parašytus darbus jį apibūdinančiame *RDF* faile. Šiuo atveju rezultatas būtų – „Magistrinis darbas“. Esant daugiau rezultatų, jie gali būti rikiuojami, grupuojami, limituojami kaip *SQL* kalboje, taip pat pateikiami skirtingais formatais.

1.6. Tyrimo objekto analizė

1.6.1. Reliacinių DB analizė

Reliacinės duomenų bazės (toliau – RDB) pagrindu laikomas reliacinis duomenų modelis (E. F. Codd, 1970m.)[5], paremtas aibių teorija bei predikatų logika. RDB duomenys pateikiami dvimatėse lentelėse, kurias sudaro:

- Eilutės – dar kitaip vadinamos RDB įrašais, kuriose saugomi duomenys apie tam tikrus objektus .
- Stulpeliai – dar kitaip vadinami RDB atributais, kurie nurodo eilutėse saugomų duomenų tipus.

Stulpelių ir eilučių susikirtimo taškai (laukeliai) atitinka konkrečias saugomų duomenų reikšmes.

Vienas iš svarbiausių RDB elementų yra lentelių raktai (*angl. keys*) – atributai ar jų rinkiniai. Raktų paskirtis – indentifikuoti unikalius laukelių įrašus bei užtikrinti duomenų integralumą, užmezgant sąryšius tarp lentelių. Atributas, unikaliai identifikuojantis bet kurią eilutę, vadinamas pirminiu raktu (*angl. primary key*). Pirminį raktą tam tikrais atvejais gali sudaryti ne vienas, o keli atributai – tokiu atveju jis vadinamas sudėtiniu raktu (*angl. composite key*). Naudojamas dar vienas rakto tipas – tai išorinis raktas (*angl. foreign key*). Jis nurodo lentelės laukus ar jų rinkinius, kurių reikšmės sutampa su kitoje lentelėje esančio pirminio rakto reikšmėmis.

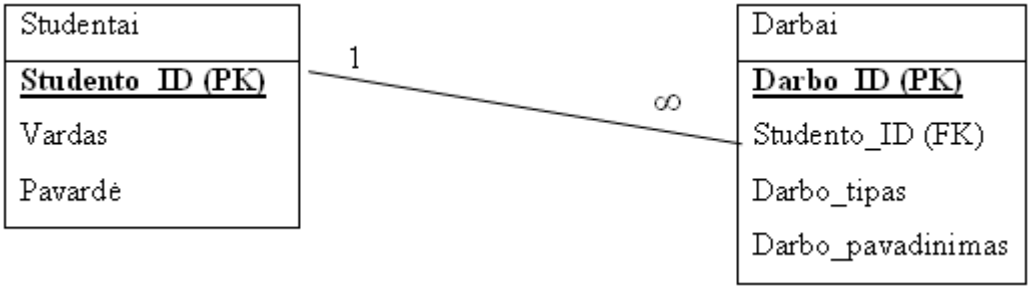
RDB lentelės susiejamos ryšiais, apibrėžiant jų tarpusavio duomenų santykius. Skiriami trys ryšių tipai:

- 1:1 (vienas su vienu) – vienas lentelės įrašas atitinka vieną kitos lentelės įrašą. Šis ryšys praktikoje retai naudojamas dėl pertekliškumo. Pastaroji problema sprendžiama sujungiant dvi lenteles į vieną.

[5] „A Relational Model of Data for Large Shared Databanks“ (Communications of the ACM, June 1970)

- 1:M (vienas su daug) – vienas lentelės įrašas gali atitikti daug įrašų kitoje lentelėje. Šio ryšio tipas yra dažniausiai sutinkamas RDB praktikoje.
- M:N (daug su daug) – daug įrašų vienoje lentelėje atitinka daug įrašų kitoje lentelėje. M:N ryšio taikymas nerekomenduotinas, siekiant išvengti duomenų anomalijų ir dublikuotų įrašų – kitaip tariant, laikantis DB norminių formų. Esant būtinybei tokio ryšio naudojimui, įterpiama tarpinė lentelė, kurios pagalba M:N ryšys išskaidomas į 1:M ir 1:N.

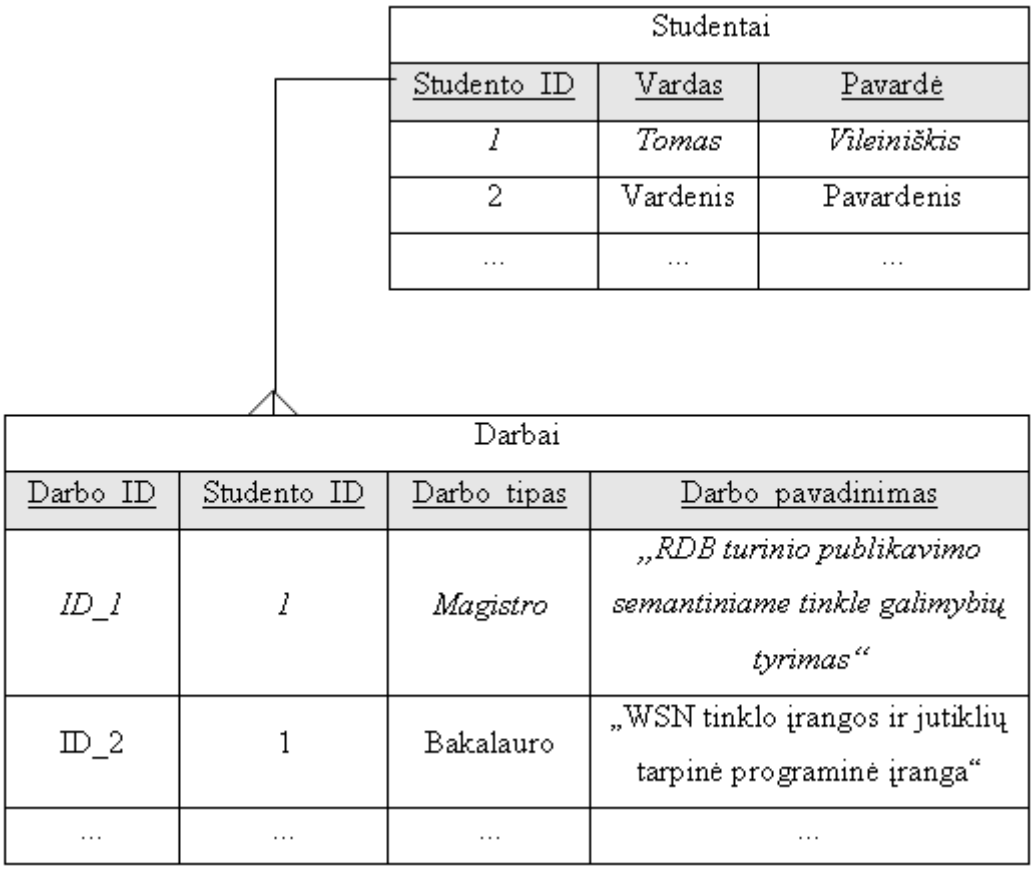
Pavyzdinė studentų asmeninių duomenų ir jų parašytų darbų saugojimui skirta RDB lentelių schema pateikta 4 pav.



4 pav. Pavyzdinė RDB lentelių schema

DB lentelėje, pavadinimu *Studentai*, saugomus studentų asmeninius duomenis aprašo atributai: *Studento_ID* (pirminis raktas), *Vardas* bei *Pavardė*. Lentelėje, pavadinimu *Darbai*, saugomi duomenys apie įvairius studentų darbus, o juos aprašo atributai: *Darbo_ID* (pirminis raktas), *Studento_ID* (išorinis raktas), *Darbo_tipas* bei *Darbo_pavadinimas*. Kiekvienas studentas gali būti parašęs po kelis skirtingo tipo ir pavadinimų darbus, todėl lenteles jungia ryšys 1:M. Lentelėje *Studentai*, kaip unikalus identifikatorius naudojamas atributas *Studento_ID*, nes egzistuoja žmonių su vienodomis pavardėmis ir, be abejo, vardais. Neatmetama galimybė, kad keli skirtingi studentai bus parašę atskirus to paties tipo ir to paties pavadinimo darbus, todėl lentelėje *Darbai* unikalaus identifikatoriaus funkciją atlieka atributas *Darbo_ID*.

Analogiškai 1.5.2 skyriuje pateiktam pavyzdžiui, teiginio „Tomas Vileiniškis parašė magistrinį darbą tema „RDB turinio publikavimo semantiniame tinkle galimybių tyrimas““ saugojimas RDB lentelėse pagal 4 pav. schemą atrodo taip:



Studentai

Studento_ID	Vardas	Pavardė
1	*Tomas*	*Vileniškis*
2	Vardenis	Pavardenis
...	...	...

Darbai

Darbo_ID	Studento_ID	Darbo_tipas	Darbo_pavadinimas
ID_1	*1*	*Magistro*	*„RDB turinio publikavimo semantiniame tinkle galimybių tyrimas“*
ID_2	1	Bakalauro	„WSN tinklo įrangos ir jutiklių tarpinė programinė įranga“
...	...	...	...

5 pav. Pavyzdinis duomenų saugojimo būdas RDB lentelėse

Duomenų, aprašytų reliaciniu modeliu valdymo funkcijos atliekamos RDBVS (*trump. Reliacinių DB Valdymo Sistemos*) pagalba. Populiariausios DBVS, palaikančios didžiąją dalį reliacinio modelio taisyklių yra *Oracle, MS SQL Server, MySQL, PostgreSQL, MS Access*. Daugelis paminėtų DBVS manipuliavimui RDB duomenimis naudoja standartinę SQL (*angl. Structured Query Language)* užklausų kalbą.

1.6.2. RDB2RDF metodų analizė

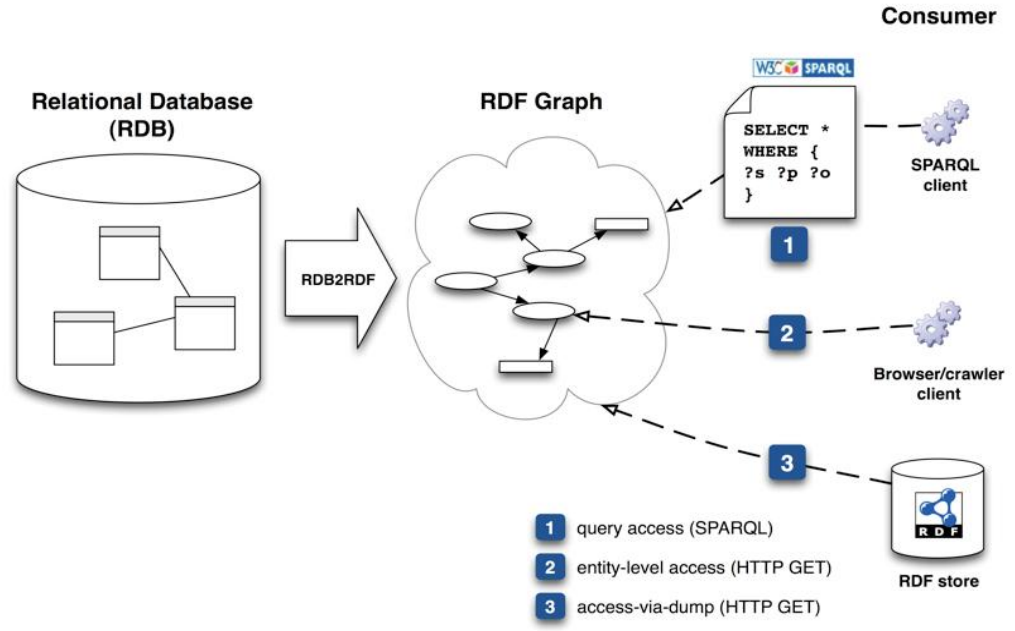
Didėjant poreikiui RDB turinį padaryti prieinamu Semantiniame tinkle, 2007 m. įkurta *W3C* konsorciumo darbo grupė *RDB2RDF*[6], kurios tikslas buvo apžvelgti esamus RDB turinio atvaizdavimo *RDF* formatu būdus, ir nuspręsti, ar reikalinga standartizuota *RDB2RDF* atvaizdavimo kalba. 2009 m. buvo priimtas sprendimas kurti standartizuotą *RDB2RDF* atvaizdavimo kalbą *R2RML*[7]. Grupės darbas šiuo metu aktyviai tęsiamas.

[6] http://www.w3.org/2005/Incubator/rdb2rdf/
[7] http://www.w3.org/TR/2010/WD-r2rml-20101028/

Skiriami du pagrindiniai RDB2RDF įgyvendinimo metodai [5]:

- Esamos RDB turinys gali būti fiziškai transformuojamas į *RDF* formatą, pasitelkiant *ETL* (*angl. Extract-Transform-Load*) metodiką, ir patalpinamas į *RDF* saugyklą. *RDF* saugykla gali saugoti daugybę *RDF* duomenų, taip palengvindama jų tarpusavio integravimo uždavinius. Tačiau bene didžiausias šio metodo trūkumas yra visiškas RDB atsiejimas nuo *RDF* kopijos – dinamiškai kintant RDB turiniui, *RDF* kopija išlieka nepakitusi, t.y. ji atvaizduoja pirminę, seną RDB kopiją. Be to, atsiranda tiesioginė priklausomybė nuo *RDF* saugyklų, kurių galimybės ir našumas kol kas yra pakankamai nauja tyrimo sritis.
- Alternatyvus metodas, nereikalaujantis fizinio RDB transformavimo į *RDF* formatą, vadinamas virtualiu atvaizdavimu (*angl. Mapping*). Šis metodas paremtas atvaizdavimo taisyklių sukūrimu, kurių pagrindu SPARQL užklausos vykdymo metu išverčiamos į standartines RDB *SQL* užklausas. RDB turinio atvaizdavimas į virtualius *RDF* grafus realiu laiku išsprendžia pirmojo metodo trūkumą – *RDF* visuomet atvaizduos „šviežiausius“ RDB duomenis. Tačiau išlieka duomenų integravimo problema – kaip vykdyti *SPARQL* užklausas, kurios tarpusavyje apimtų kelis *RDF* šaltinius? Tai taip pat pakankamai nauja tyrimo sritis.

Abu aprašyti metodai iliustruojami bendra RDB2RDF konteksto schema, pateikta 6 pav.



6 pav. RDB2RDF konteksto schema[8]

[8] http://www.w3.org/TR/2010/WD-rdb2rdf-ucr-20100608/

RDF grafo (tiek realizuoto fiziškai, tiek virtualaus) duomenys turėtų būti pasiekiami keliais skirtingais būdais:

1. *SPARQL* užklausomis – vartotojas (agentas) *SPARQL* priegos taške įvykdo užklausą ir apdoroja gaunamus duomenis *XML* ar kitu formatu;
2. naršykle – vartotojas (agentas) vykdydamas *HTTP Get* komandą norimu *URI* adresu apdoroja RDF formatu gaunamus rezultatus;
3. *RDF* saugyklos prieiga – vartotojas (agentas) įvykdo *HTTP Get* komandą *RDF* grafo atžvilgiu ir gautą fizinę jo išraišką patalpina į *RDF* saugyklą.

Atsižvelgiant į aukščiau išvardintų *RDB2RDF* metodų dabartinį aktualumą, tolimesniuose šio darbo etapuose bus nagrinėjamas antrasis, virtualaus atvaizdavimo metodas bei jį realizuojantys įrankiai.

Atliekant tyrimo objekto analizę, išryškėjo pagrindiniai *RDB2RDF* virtualizavimo įrankių pasirinkimo kriterijai:

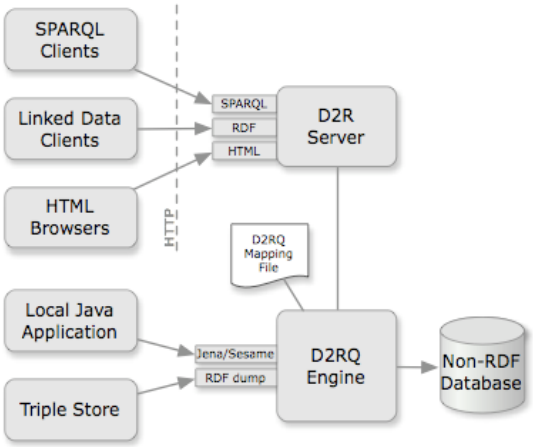
1. suderinamumas su skirtingomis DBVS (Duomenų Bazių Valdymo Sistemos),
2. automatinis virtualaus atvaizdavimo taisyklių failo generavimas,
3. galimybė modifikuoti sugeneruotą virtualaus atvaizdavimo taisyklių failą,
4. SPARQL užklausų kalbos palaikymas,
5. programinių karkasų API prieiga,
6. laikomasi atviro kodo filosofijos.

1.7. Egzistuojančių RDB2RDF technologijų analizė

1.7.1. D2RQ platformos analizė

D2RQ [6] [7] platforma skirta RDB turinio prieigai virtualių *RDF* grafų pavidalu. Ši platforma suteikia kelis skirtingus prieigos prie RDB mechanizmus:

- *SPARQL* užklausų vykdymą reliacinių duomenų atžvilgiu,
- *Jena* ir *Sesame API* (*angl. Application Programming Interface*) prieigą prie virtualių *RDF* grafų,
- fizinių *RDF* grafų generavimą *RDF* saugykloms,
- semantinio tinklo naršyklių prieigą prie semantiniais ryšiais susietų duomenų,
- tradicinių *HTML* naršyklių prieigą prie semantiniais ryšiais susietų duomenų.



7 pav. D2RQ platformos architektūra [7]

D2RQ platformą (*žr. 7 pav.*) sudaro šios pagrindinės dalys:

- *D2RQ* virtualaus atvaizdavimo kalba – skirta nusakyti ryšiams bei taisyklėms tarp virtualizuojamų RDB elementų ir RDFS schemų/*OWL* ontologijų,
- *D2RQ* varikliukas – įskiepis Semantinio tinklo programiniams karkasams *Jena* ir *Sesame*, kurio paskirtis naudojant *D2RQ* virtualaus atvaizdavimo taisyklių failą, perrašyti *Jena*[9] ir *Sesame*[10] *API* programinius kreipinius į RDB *SQL* užklausas,
- *D2R* serveris – *HTTP* serveris, suteikiantis prieigą prie RDB turinio Semantinio tinklo naršyklėms, *HTML* naršyklėms ir *SPARQL* klientams.

D2RQ platforma suderinama su daugeliu reliacinių DBVS, turinčių JDBC ir ODBC tvarkyklių prieigą: *Oracle*, *MySQL*, *PostgreSQL*, *Microsoft SQL Server*. *D2RQ* platforma taip pat siūlo mechanizmą automatiniam virtualaus atvaizdavimo failų generavimui, su galimybe atlikti tolimesnes jo modifikacijas.

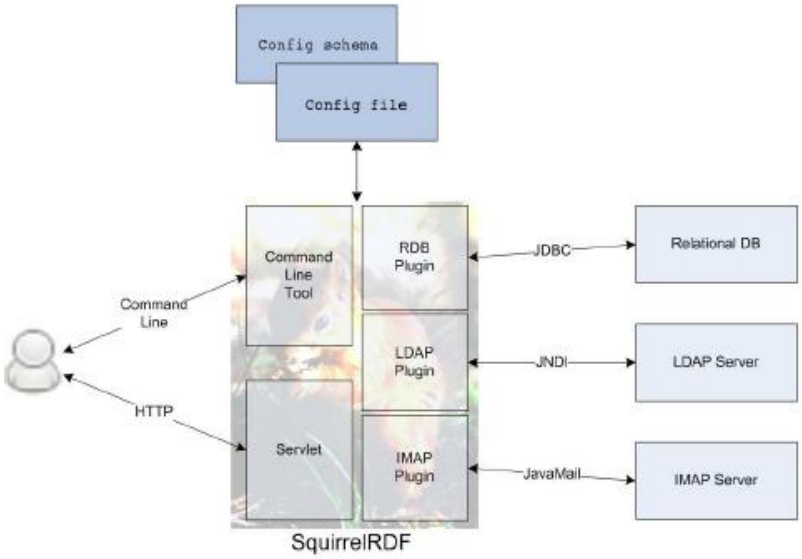
Nors *D2RQ* specifikacijoje pateikiami daugelio virtualizavimo kalbos elementų aprašai, nėra iki galo aiškus jų praktinis taikymas: neįvertinamas taisyklių pertekliškumas, nestandartiniai virtualizavimo atvejai, būtinų/nebūtinų sub-elementų naudojimas ir pan.

1.7.2. SquirrelRDF analizė

SquirrelRDF [8] įrankis skirtas ne tik RDB turinio, bet ir *LDAP* (*angl. Lightweight Directory Access Protocol*) direktorijų bei *IMAP* (*angl. Internet Message Access Protocol*) elektronio pašto dėžučių prieigai virtualių *RDF* grafų pavidalu. *SquirrelRDF* struktūrą, pateiktą 8 pav. sudaro:

[9] http://jena.sourceforge.net/
[10] http://www.openrdf.org/doc/sesame2/2.3.2/users/index.html

- konfigūravimo failas – aprašo visus būtinus parametrus, reikalingus prisijungimui prie serverio ir virtualaus atvaizdavimo taisykles,
- konfigūravimo schema – nusako konfigūravimo failo generavimo logiką (naudojama ne visuose įskiepiuose).,
- įskiepiai - skirti prieigai prie skirtingo tipo resursų (RDB, *LDAP* ir *IMAP*),
- kKomandinė eilutė – skirta automatiniam konfigūravimo failo generavimui, užklausų vykdymui,
- *JAVA* Servletas – skirtas *SPARQL* užklausų vykdymui, naudojant *HTTP* protokolą.



8 pav. SquirrelRDF struktūra[11]

Vykdant *SPARQL* užklausas tiek komandinėje eilutėje, tiek *JAVA* servlete per *HTTP* protokolą, visuomet kreipiamasi į konfigūravimo failą. Kuomet pastarasis failas užkraunamas, pagal parametrų tipą *SquirrelRDF* parenka konkrečiu atveju naudotiną įskiepį. Vykdomos *SPARQL* užklausos išverčiamos atitinkamai pagal taisykles, nurodytas konfigūravimo faile. Verta paminėti, kad automatinio virtualizavimo failo generavimo galimybė egzistuoja tik RDB įskiepiui.

SquirrelRDF turi kelis esminius trūkumus:

- virtualizavimo procesas vyksta ganėtinai paviršutiniškai, neįtraukiant nei žodynų, nei ontologijų vaidmens. To pasekoje, duomenys gautuose virtualiuose *RDF* grafuose pasižymi itin silpnais semantiniais ryšiais;

[11] www.hpl.hp.com/techreports/2007/HPL-2007-161.pdf

- SquirrelRDF tik dalinai suderinamas su programiniais karkasais – realizuota *API* prieiga tinka tik *SPARQL* užklausų vykdymui *Jena* programinio karkaso pagrindu;
- ribotos *SPARQL* užklausų galimybės neleidžia vykdyti `{ ?s ?p ?o }` užklausų *RDF* atžvilgiu.

Primityvi *SquirrelRDF* virtualaus atvaizdavimo taisyklių kūrimo metodika ir iki galo neišspęstas suderinamumo su programiniais karkasais uždavinys stipriai apriboja šio įrankio praktinio taikymo galimybes.

1.7.3. Openlink Virtuoso RDF Views analizė

Virtuoso universalus serveris (*angl. Virtuoso Universal Server*) yra sistema, apjungianti reliacinių, objektinių-reliacinių DBVS, *RDF*, *XML*, *WEB* serverio ir failų serverio funkcijas į bendrą visumą. *Virtuoso RDF Views* [9] įrankis yra hibridinio *Virtuoso* serverio sudedamoji dalis, skirta RDB duomenų virtualizavimui į *RDF* grafus. Skirtingai, nei kiti analizuojami įrankiai, *Virtuoso RDF Views* praktiškai nereikalauja iš vartotojo specifinių virtualizavimo kalbos [12] žinių – virtualizavimo procesas vyksta interaktyvaus *HTML* vedlio pagalba:

- jeigu norima virtualizuoti ne *Virtuoso* RDBVS esančius duomenis, nurodomi nutolusių duomenų bazių prisijungimo duomenys per *JDBC* ar *ODBC* tvarkykles;
- Įvedamas bazinis *URI* adresas, kurio atžvilgiu bus generuojami virtualizuotų resursų *URI* ir pažymimos DB lentelės, kurias norima virtualizuoti;
- Pasirenkamas arba pilnai automatinis arba dalinai automatinis virtualizavimo taisyklių failo generavimo metodas. Pastaruoju atveju leidžiama kiekvienam iš DB lentelės stulpelių priskirti norimą duomenų formato tipą ar nurodyti specifinę ontologiją, kurios pagrindu ir būtų vykdomas RDB duomenų virtualizavimas į *RDF* grafą;
- Atlikus išvardintus veiksmus, pateikiamas sugeneruotas virtualizavimo taisyklių failo išeities tekstas ir/arba ontologija. Paskutiniuoju žingsniu, sugeneruotos taisyklės įvykdomos pasirinktos RDB atžvilgiu ir kaip galutinis rezultatas

[12] Virtuoso RDF Views virtualizavimo taisyklių aprašymui naudoja *Meta Schema* kalbą – SPARQL ir SPASQL junginį.

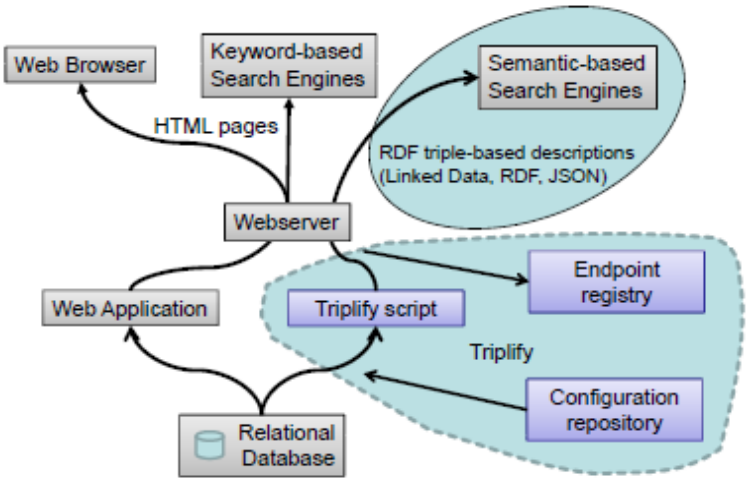
pateikiami gautų duomenų pavyzdiniai *URI* adresai bei virtualizavimo rezultatą apibūdinanti statistika.

Virtuoso serveryje *SPARQL* užklausos gali būti vykdomos ne tik virtualizuotų RDB (saugomų tiek nutolusiose DBVS, tiek vietinėje *Virtuoso* RDBVS) atžvilgiu, bet ir *Virtuoso* fizinių *RDF* saugyklos atžvilgiu. Atskiri *Virtuoso* serverio posistemiai suteikia semantinio tinklo *Java* karkasų *Jena* ir *Sesame API* prieigą prie RDF trejetų rinkinių (tiek fizinių, tik virtualių). Šis įrankis turi dvi versijas: komercinę ir atviro kodo. Pastarojoje versijoje *Virtuoso RDF Views* suteikia galimybę virtualizuoti tik vietinėje *Virtuoso* RDBVS saugomus duomenis. Tuo tarpu komercinė versija, *JDBC* ir *ODBC* tvarkyklių pagalba, leidžia virtualizuoti RDB duomenis, saugomus skirtingose nutolusiose DBVS.

1.7.4. Triplify analizė

Triplify [10] [11] įrankis orientuotas į labiausiai internete paplitusių, RDB paremtų turinio valdymo sistemų (toliau – TVS), skirtų tinklaraščių, nuotraukų galerijų, diskusijų forumų, *Wiki* tipo tinklalapių ir kt. turinio publikavimui semantiniame tinkle. Šis įrankis sukurtas, siekiant kaip įmanoma labiau supaprastinti jo praktinio taikymo metodiką vartotojams, turintiems ribotus techninius įgūdžius. *Triplify* sudarytas iš dviejų pagrindinių sudedamųjų dalių: vykdomojo skripto ir konfigūravimo failo (*žr. 9 pav.*). Abi šios dalys realizuotos *PHP (angl. Hypertext Preprocessor*) pagrindu.

- konfigūravimo failas – naudojamas virtualizavimo taisyklėms aprašyti konkrečios RDB schemos atžvilgiu;
- vykdomasis skriptas – atlieka loginius RDB duomenų virtualizavimo ir transformavimo žingsnius pagal konfigūravimo faile aprašytas taisykles.



9 pav. *Triplify* struktūra ir sąsaja su egzistuojančiomis *WEB* technologijomis [11]

Priklausomai nuo siekiamos virtualizuoti TVS RDB, skiriami du *Triplify* panaudos atvejai:

- *Web* serverio TVS direktorijoje išsaugomas *Triplify* failų katalogas ir atitinkamos TVS RDB schemai sukurtas konfigūravimo failas su virtualizavimo taisyklėmis. Jau paruošti naudojimui konfigūravimo failai apima tokių TVS RDB schemas kaip *WordPress*, *Joomla*, *phpBB* ir kt.[13] Konfigūravimo faile papildomai nurodomi prisijungimo prie RDB duomenys per *PDO* (*angl. PHP Data Objects*) arba standartinę *MySQL* tvarkyklę;
- Antrasis panaudojimo atvejis analogiškas pirmąjam, išskyrus tai, kad naudojama TVS reikalauja unikalaus konfigūravimo failo kūrimo. Konfigūravimo failo pagrindą sudaro *SQL* užklausos su *SELECT AS* teiginiais (pvz.: `SELECT id,vardas AS 'foaf:name' FROM vartotojai`), kurių vykdymo metu atitinkamų RDB lentelių duomenys pateikiami kaip atvaizdai į RDF trejetus. Pastarųjų semantiniai ryšiai nusakomi laisvai pasirenkant naudotinus žodynus.

Abiem atvejais serveryje įdiegus įrankį, *RDF* duomenys iš RDB generuojami arba panaudojant *ETL* metodiką fiziniams RDF trejetams išgauti arba realiu laiku išverčiant *HTTP-URI* kreipinius į standartines RDB *SQL* užklausas. Gauti *RDF* trejetai gali būti eksportuojami *RDF*, *JSON* (*angl. JavaScrip Object Notation*) formatais arba pateikiami kaip tarpusavyje semantiškai susieti duomenys (*angl. Linked Data*). Skirtingai negu kiti analizuoti įrankiai, *Triplify* nepalaiko *SPARQL* užklausų kalbos bei nesuteikia semantinio tinklo programinių karkasų *API* prieigos prie sugeneruotų *RDF* trejetų. Vienintelis būdas viešai pasiekti sugeneruotus *RDF* duomenis yra *Triplify* prieigos taškų registras[14], kuris šio darbo rašymo metu labiau atitinka ne pilno funkcionavimo, bet koncepcinę fazę. Taip pat verta paminėti, jog *Triplify* orientuotas tik į *PHP* ir *MySQL* technologijomis paremtas taikomąsias *Web* programas. Tokie apribojimai komplikuoja įrankio praktinį taikymą pasaulinio semantinio tinklo duomenų tarpusavio integravimui.

[13] http://triplify.org/Documentation?v=13da
[14] http://triplify.org/Registry

1.8. Analizės rezultatai ir išvados

Semantinio tinklo technologijų ir standartų analizė leido įsisavinti jų vaidmenį naujos kartos tinklo kūrime bei susipažinti su bendra semantinio tinklo koncepcija. Išanalizavus tyrimo objektą, sudarytas esminių tyrimo srities (*RDB2RDF* technologijų) kriterijų sąrašas:

1. Suderinamumas su skirtingomis DBVS,
2. Automatinis virtualaus atvaizdavimo taisyklių failo generavimas,
3. Galimybė modifikuoti sugeneruotą virtualaus atvaizdavimo taisyklių failą,
4. *SPARQL* užklausų kalbos palaikymas,
5. Programinių karkasų API prieiga,
6. Laikomasi atviro kodo filosofijos.

Remiantis nustatytais kriterijais, atlikta labiausiai vartotojų tarpe paplitusių *RDB2RDF* įrankių lyginamoji analizė. Analizės rezultatai atskleidė kiekvieno iš jų stipriąsias ir silpnąsias puses bei praktinio taikymo galimybes. Lyginamosios *RDB2RDF* įrankių analizės apibendrinti rezultatai pateikiami 1 lent.

1 lent. Lyginamosios RDB2RDF įrankių analizės apibendrinti rezultatai

Kriterijus / Įrankis	Skirtingos DBVS	Automatinis virt. taisyklių generavimas	Taisyklių failo modifikavimas	SPARQL palaikymas	Programinių karkasų API prieiga	Atviro kodo filosofija
D2RQ	+	+	+	+	+	+
SquirrelRDF	+	+	+	+	-	+
Virtuoso RDF Views	-	+	+	+	+	-
Triplify	-	-	+	-	-	+

Remiantis analizės rezultatais, suformuotos šios išvados:

1. Atlikus pasirinktų RDB2RDF įrankių analizę nustatyta, kad *D2R platforma yra tinkamiausia tolimesniam tyrimui, nes*atitinka visus iškeltus kriterijus.
2. Nors D2RQ platforma teoriškai gali būti panaudojama daugelio *RDB2RDF* kontekste pateikiamų funkcijų realizavimui, tačiau trūksta nuoseklios jo praktinio taikymo metodikos, yra poreikis atlikti išsamesnį šio įrankio galimybių tyrimą.

1.9. Tyrimo tikslas ir uždaviniai

Remiantis analizės išvadomis, numatoma sudaryti D2RQ platformos praktinio taikymo metodiką ir jos pagrindu realizuoti IS prototipą, kurie semantinio tinklo kūrėjams

suteiktų daugiau aiškumo apie RDB publikavimo semantiniame tinkle procesą ir technologijų panaudojimą. Šiam tikslui pasiekti iškelti tokie uždaviniai:

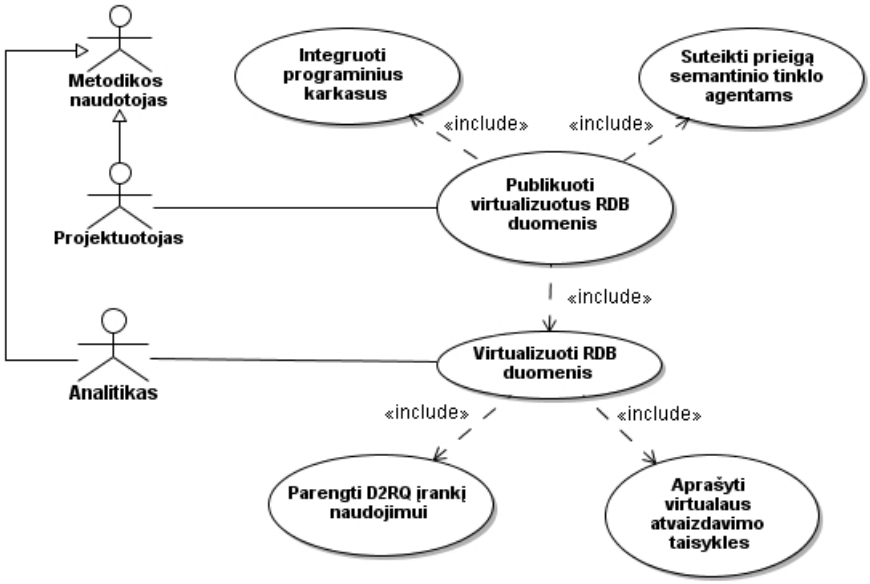
1. Specifikuoti reikalavimus D2RQ praktinio taikymo metodikai ir IS prototipui,
2. Suprojektuoti IS prototipą,
3. Atlikti detalią D2RQ virtualizavimo kalbos specifikacijos analizę,
4. Suformuoti D2RQ platformos praktinio taikymo metodiką,
5. Remiantis D2RQ praktinio taikymo metodika, sukurti mokomosios RDB virtualizavimo taisykles,
6. Remiantis D2RQ praktinio taikymo metodika ir IS projektu realizuoti programinį IS prototipą,
7. Įvertinti metodikos veiksmingumą ir IS funkcionalumą eksperimentiškai.

2. Kuriamos metodikos ir IS reikalavimų analizė ir specifikacija

2.1. Reikalavimų specifikacija kuriamai metodikai

Šiame skyriuje aprašomi bendri reikalavimai kuriamai metodikai. Jie formaliai pateikiami *UML* panaudojimo atvejų diagrama, matoma 10 pav. Skiriami du pagrindiniai reikalavimai:

- virtualizuoti RDB duomenis – metodika turi nusakyti RDB duomenų virtualizavimo žingsnius nuo įrankio diegimo iki taisyklių šablono generavimo bei redagavimo;
- publikuoti virtualizuotus RDB duomenis – metodika turi nusakyti virtualizuotų RDB duomenų panaudojimo taikomuosiuose IS sprendimuose principus. Tai apima programinių karkasų naudojimo aspektus, pagalbinių įrankių ir serverių konfigūravimą bei prieigos prie išpublikuotų duomenų suteikimą išoriniams klientams.

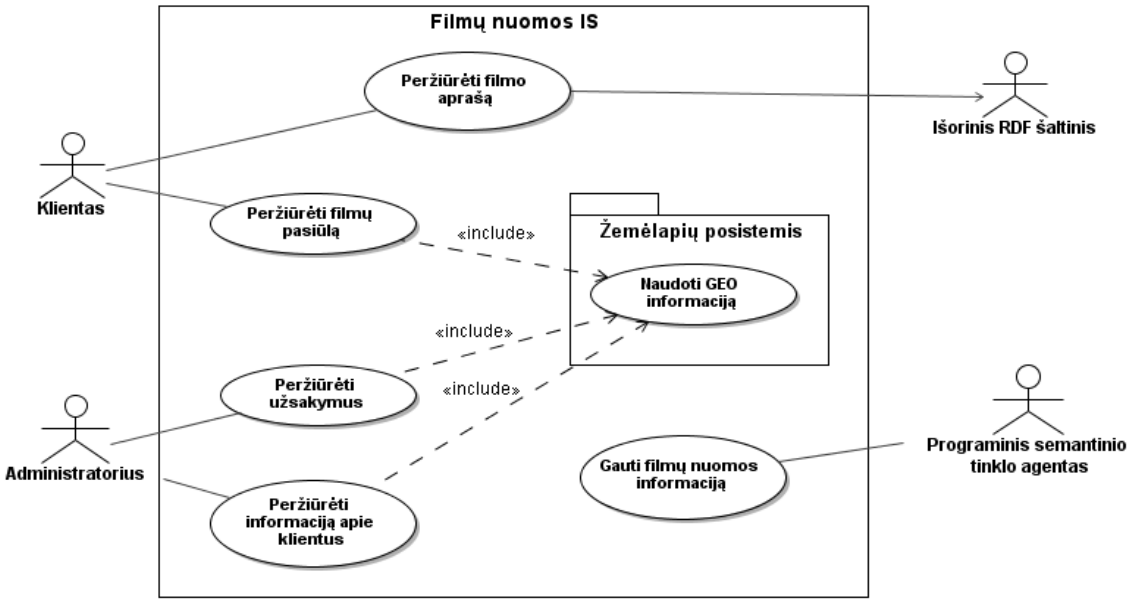


10 pav. Panaudojimo atvejų diagrama D2RQ praktinio taikymo metodikai

2.2. Reikalavimų specifikacija IS programiniam prototipui

Šiame skyriuje aprašomi bendri reikalavimai pagal *D2RQ* įrankio praktinio taikymo metodiką kuriamam programiniam IS prototipui. IS prototipo kūrimo tikslas yra pademonstruoti įrankio taikymo galimybes filmų nuomos dalykinėje srityje. Dalykinė sritis pasirinkta, remiantis mokomąja *MySQL* reliacine duomenų baze - „Sakila“[15].

Kaip matoma 11 pav., IS iškelti reikalavimai yra orientuoti ne į dalykinės srities verslo transakcijų valdymą, o į integruotos, struktūrizuotos informacijos pateikimą. Tokiu būdu siekiama akcentuoti semantinio tinklo technologijų teikiamą naudą eilinio interneto vartotojo atžvilgiu.



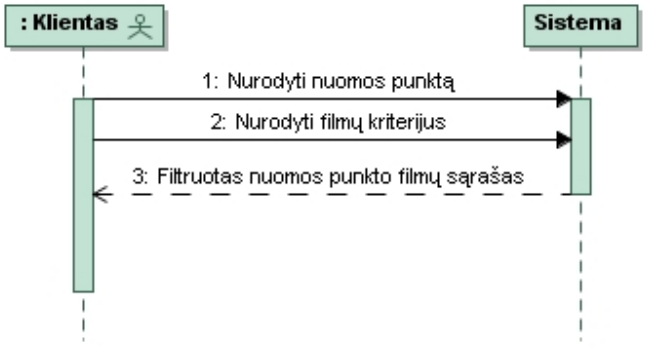
11 pav. Panaudojimo atvejų diagrama programiniam IS prototipui

Siekiant didesnio detalumo lygio, kiekvienas iš 11 pav. diagramoje pateiktų panaudojimo atvejų atskirai specifikuojamas sekų diagramomis.

[15] http://dev.mysql.com/doc/sakila/en/sakila.html

2 lent. PA „Peržiūrėti filmų pasiūlą“ specifikacija

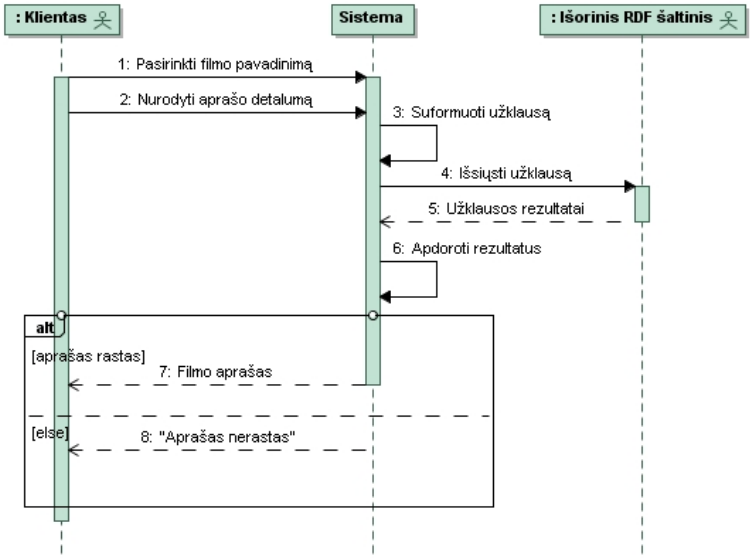
PA „*Peržiūrėti filmų pasiūlą*“		
Tikslas. Pateikti vartotojui pasirinkto nuomos punkto filmų pasiūlą.		
Aprašymas. Šis PA vartotojui suteikia galimybę susipažinti su nuomos punkte esančiais filmais		
Aktorius		Nuomos punkto klientas
Prieš sąlyga		-
Sužadinimo sąlyga		Vartotojas atidaro atitinkamą tinklalapį
Susiję panaudojimo atvejai	**Išplečia PA**	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Klientas nurodo nuomos punktą		1.1 Sistema užkrauna visų nuomos punkto filmų sąrašą
2. Klientas nurodo filmų kriterijus		2.1 Sistema išfiltruoja filmų sąrašą pagal kriterijus
Po sąlyga		-
Alternatyvūs scenarijai		
-		



12 pav. PA „Peržiūrėti filmų pasiūlą“ sekų diagrama

3 lent. PA „Peržiūrėti filmo aprašą“ specifikacija

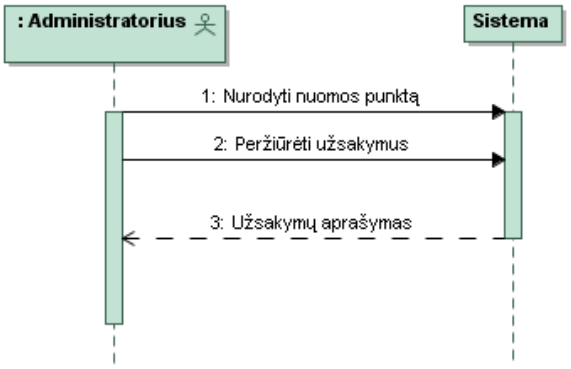
PA „*Peržiūrėti filmo aprašą*“		
Tikslas. Pateikti vartotojui detalų pasirinkto filmo aprašą.		
Aprašymas. Šio PA metu suformuojamas detalus filmo aprašas, panaudojant ne tik turimus duomenis RDB, bet ir duomenis iš išorinio RDF šaltinio.		
Aktorius		Nuomos punkto klientas.
Prieš sąlyga		Vartotojas peržiūri filmų pasiūlą.
Sužadinimo sąlyga		Vartotojas paspaudžia nuorodą filmų pasiūlos peržiūros lange.
Susiję panaudojimo atvejai	**Išplečia PA**	
	Apima PA	Peržiūrėti filmų pasiūlą.
	Specializuoja PA	
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Vartotojas pasirenka filmo pavadinimą		1.1 Sistema suformuoja užklausą
		1.2 Sistema išsiunčia užklausą į išorinį RDF šaltinį
		1.3 Sistema integruoja gautus RDF duomenis su RDB duomenimis
		1.4 Sistema pateikia rezultatus
Po sąlyga:		-
Alternatyvūs scenarijai		
1. Išorinis RDF šaltinis neatsako		Vartotojas informuojamas pranešimu



13 pav. PA „Peržiūrėti filmo aprašą“ sekų diagrama

4 lent. PA „Valdyti užsakymus“ specifikacija

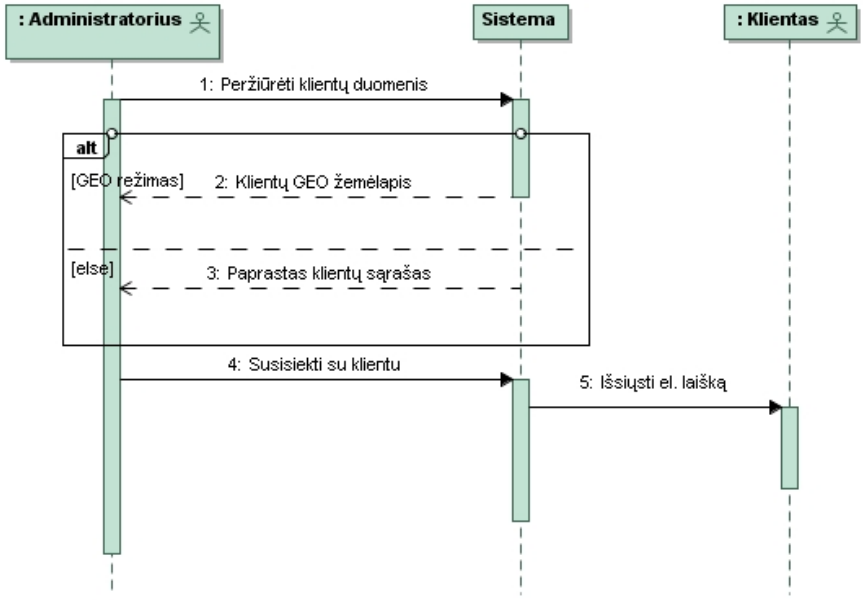
PA „*Valdyti užsakymus*“		
Tikslas. Pateikti administratoriui informaciją apie užsakymus.		
Aprašymas. Šio PA metu sistemos administratoriui pateikiama nuomos punktų užsakymų informacija.		
Aktorius		Administratorius
Prieš sąlyga		Administratorius atidaro užsakymų posistemio langą
Sužadinimo sąlyga		Administratorius nurodo nuomos punktą.
Susiję panaudojimo atvejai	**Išplečia PA**	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Administratorius pasirenka nuomos punktą		1.1 Sistema pateikia nuomos punkto užsakymus
2. Admnistratorius pasirenka vieną iš užsakymų		2.1 Sistema pateikia konkretaus užsakymo duomenis
Po sąlyga:		-
Alternatyvūs scenarijai		
-		



14 pav. PA „Valdyti užsakymus“ sekų diagrama

5 lent. PA „Valdyti klientus“ specifikacija

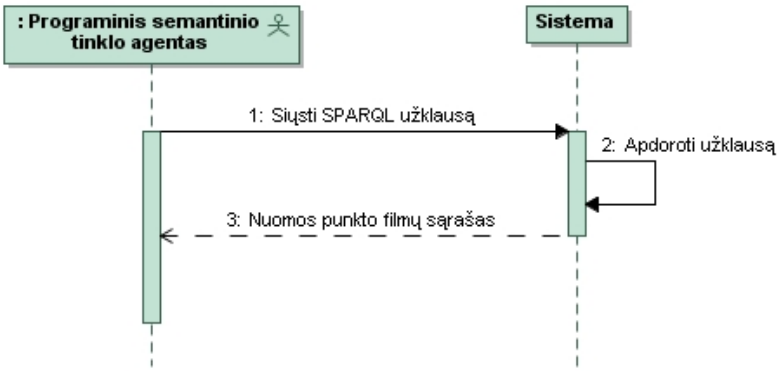
PA „*Valdyti klientus*“		
Tikslas. Pateikti administratoriui klientų duomenis		
Aprašymas. Šio PA metu sistemos administratoriui pateikiami klientų asmeniniai duomenys, suteikiama galimybė su jais susisiekti.		
Aktorius		Administratorius
Prieš sąlyga		Administratorius atidaro klientų posistemio langą
Sužadinimo sąlyga		Administratorius nurodo nuomos punktą.
Susiję panaudojimo atvejai	**Išplečia PA**	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Administratorius paspaudžia klientų peržiūros nuorodą		1.1 Sistema pateikia klientų sąrašą arba klientų žemėlapį
2. Administratorius sukuria žinutę klientui		2.1 Sistema išsiunčia el. laišką klientui
Po sąlyga:		-
Alternatyvūs scenarijai		
-		



15 pav. PA „Valdyti klientus“ sekų diagrama

6 lent. PA „Gauti filmų nuomos informaciją“ specifikacija

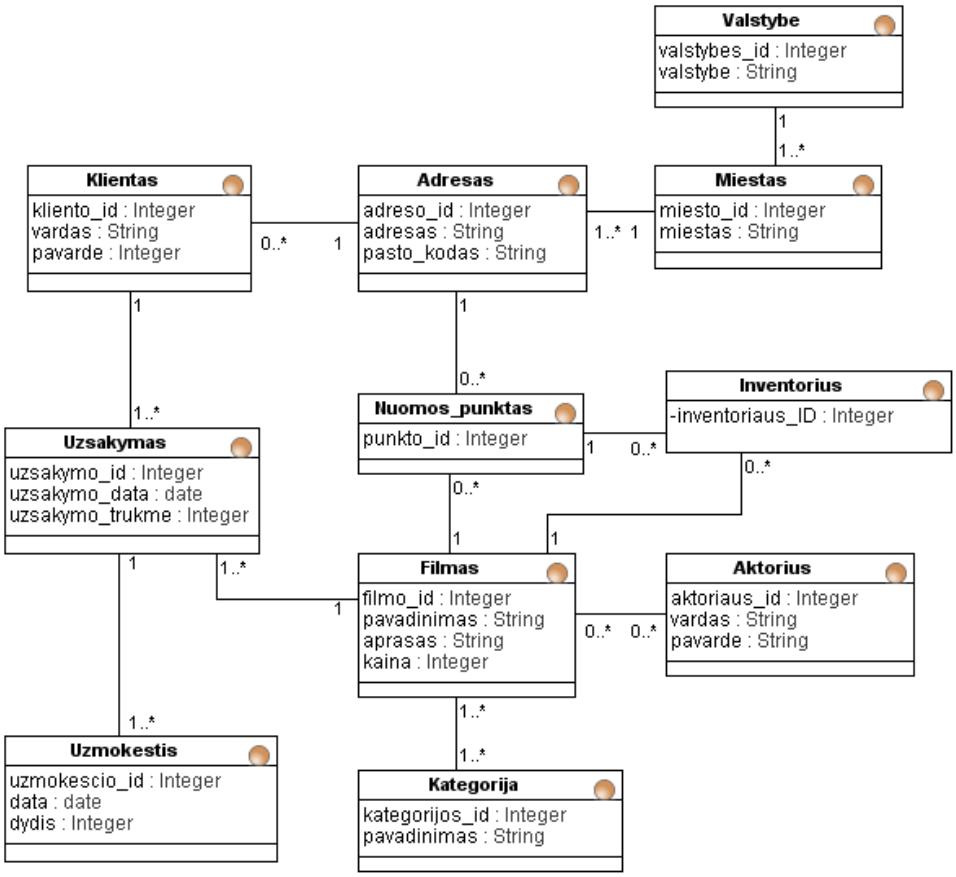
PA „*Gauti filmų nuomos informaciją*“		
Tikslas. Suteikti nuomos punktų informaciją programiniams agentams		
Aprašymas. Šis PA yra dalinai realizuotas D2RQ įrankyje. Programinis agentas siunčia užklausą į IS SPARQL prieigos tašką bei gauna atsakymą – nuomos punkto filmų pasiūlą bei jų nuomos kainą.		
Aktorius		Programinis semantinio tinklo agentas
Prieš sąlyga		IS RDB duomenys virtualiai atvaizduoti RDF grafais
Sužadinimo sąlyga		Programinis agentas kreipiasi į IS
Susiję panaudojimo atvejai	**Išplečia PA**	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Programinis agentas išsiunčia SPARQL užklausą		1.1 Sistema apdoroja gautą užklausą
		2.1 Sistema grąžina rezultatus RDF formatu
Po sąlyga:		-



16 pav. PA „Gauti filmų nuomos informaciją“ sekų diagrama

Programinio prototipo dalykinės srities esybių klasių diagrama

17 pav. pateikiamos IS dalykinės srities esybės ir jų tarpusavio ryšiai.



17 pav. IS dalykinės srities esybių klasių diagrama.

3. IS programinio prototipo projektas

Šio etapo metu siekiama sukurti architektūrinį sprendimą, taikytiną semantinių web aplikacijų kūrimui, panaudojant *D2RQ* RDB virtualizavimo įrankį. IS projektas rengiamas atsižvelgiant į reikalavimų specifikacijos metu išskirtus reikalavimus konkrečios dalykinės srities, filmų nuomos, IS programiniam prototipui.

Kadangi pagrindinė semantinio tinklo vertė eiliniam interneto vartotojui yra struktūrizuotos, integruotos informacijos pateikimas, IS projektiniame lygmenyje, o po to ir realizacijos metu, įgyvendinami funkciniai reikalavimai, orientuoti būtent į informacijos integravimą, struktūrizavimą ir išvedimą.

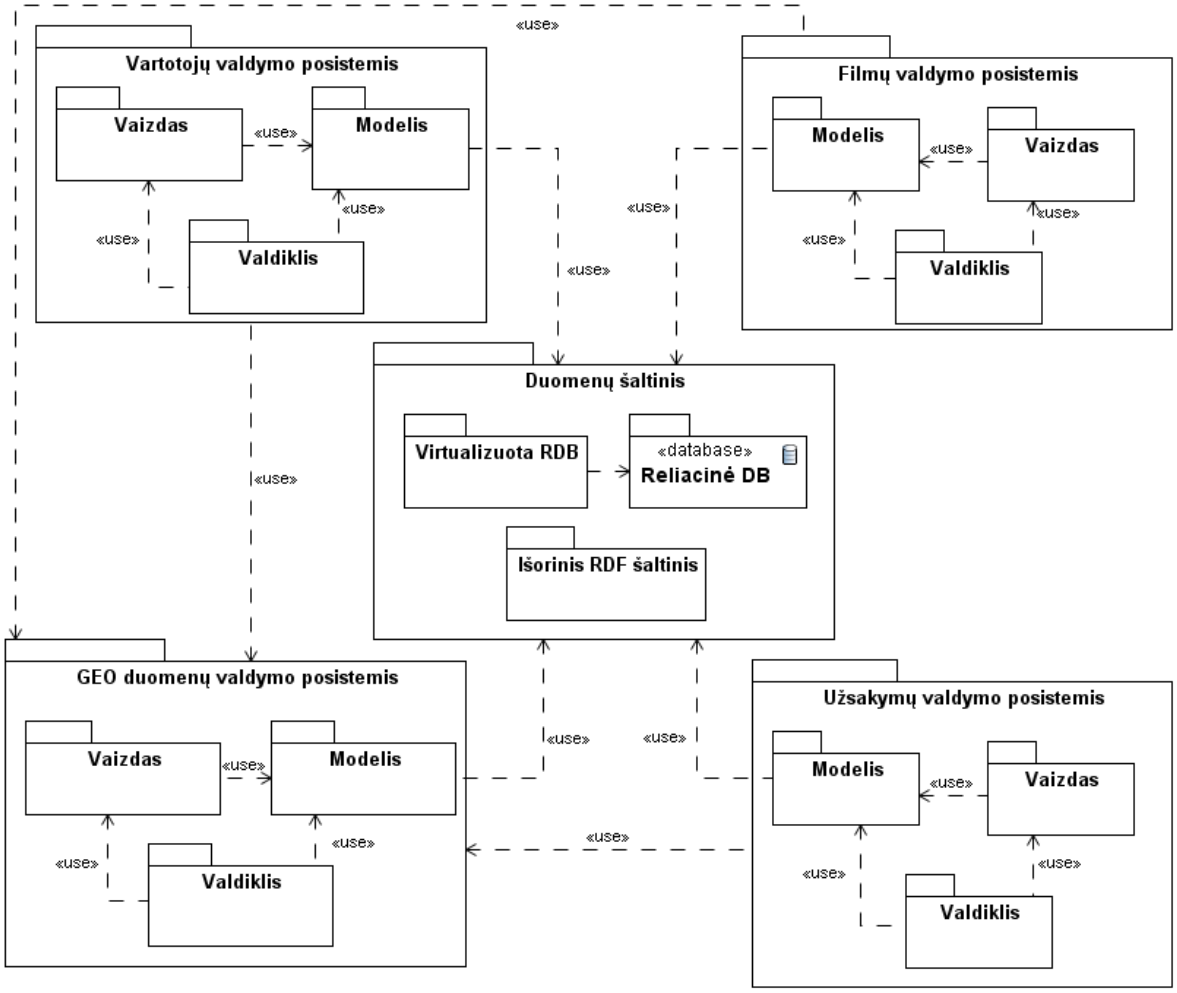
3.1. Sistemos architektūros projektas

3.1.1. Sistemos loginė architektūra

Sistemos loginės architektūros (*žr. 18 pav.*) pagrindu pasirinkta *MVC* (*angl. Model View Controller*) paradigma. Šį pasirinkimą lėmė kelios priežastys:

1. *MVC* architektūra leidžia tarpusavyje atskirti vaizdo ir modelio komponentus – tokiu būdu sistema tampa lankstesne ir lengviau prižiūrima bei atnaujinama;
2. semantinio tinklo idėja iš esmės keičia vienintelį *MVC* architektūros komponentą – modelį;
3. plečiantis semantiniam tinklui, daugėjant *RDF* šaltinių (jiems keičiantis), sistema gali būti tobulinama semantiškai praturtinant modelio komponentą, t.y. tokiu atveju IS nesugriūna iš pagrindų.

Detali posistemių elgsena pagal *MVC* principus pateikiama sekančiuose skyreliuose.

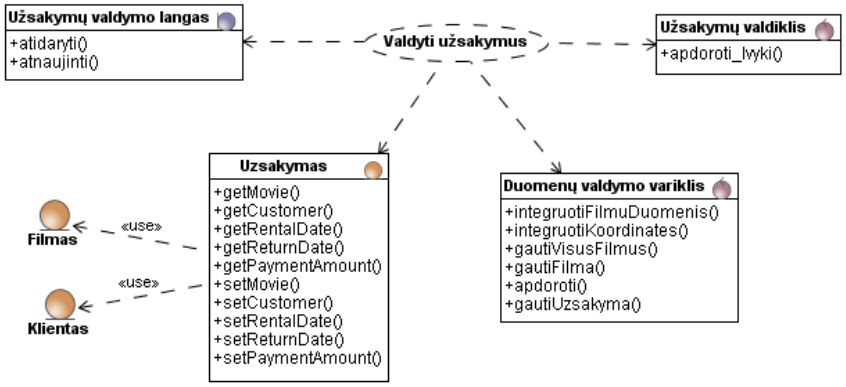


18 pav. IS loginė architektūra, paremta MVC paradigma.

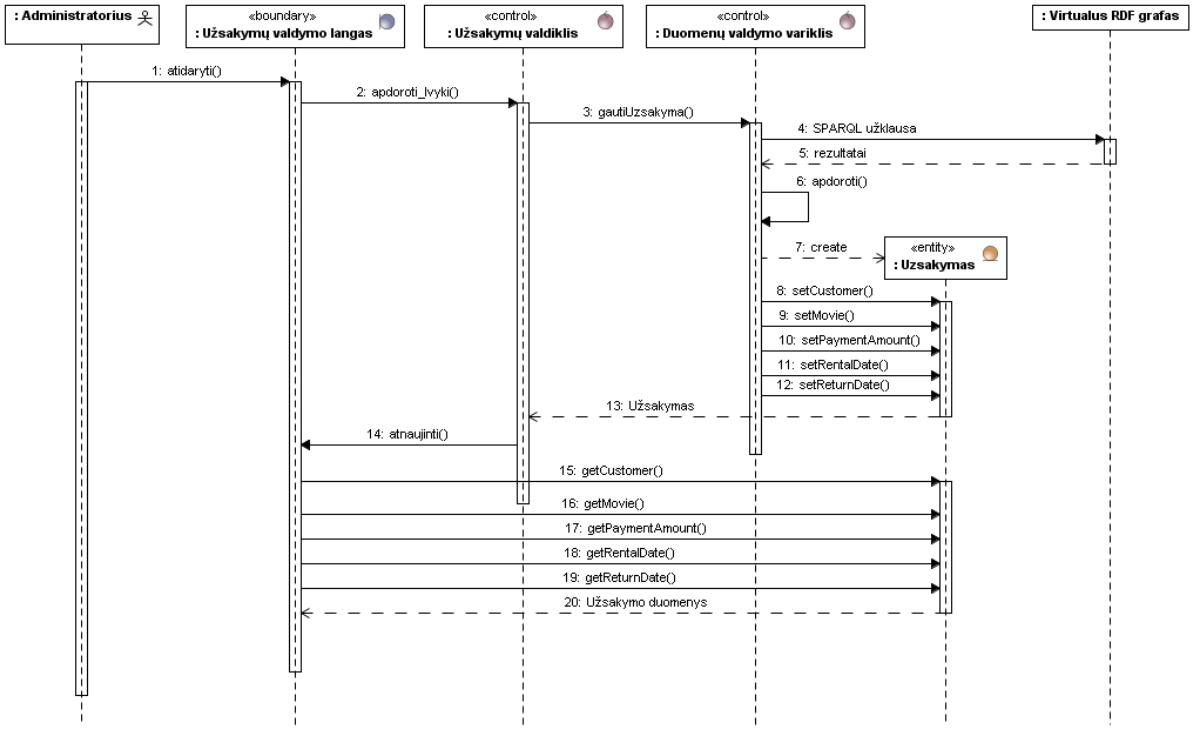
3.1.2. Sistemos posistemių projektai

Šiame skyriuje pateikiamos IS posistemių panaudos atvejus realizuojančios klasės (19 pav., 21 pav., 23 pav., 25 pav.) bei *UML* notacijoje sekų diagramomis aprašoma detali posistemių elgsena (20 pav., 22 pav. 24 pav., 26 pav.). Siekiant projekto detalumo, duomenų valdymo varikliuko veikimo principai perteikti atskirose sekų bei veiklos diagramose.

Užsakymų valdymo posistemio projektas

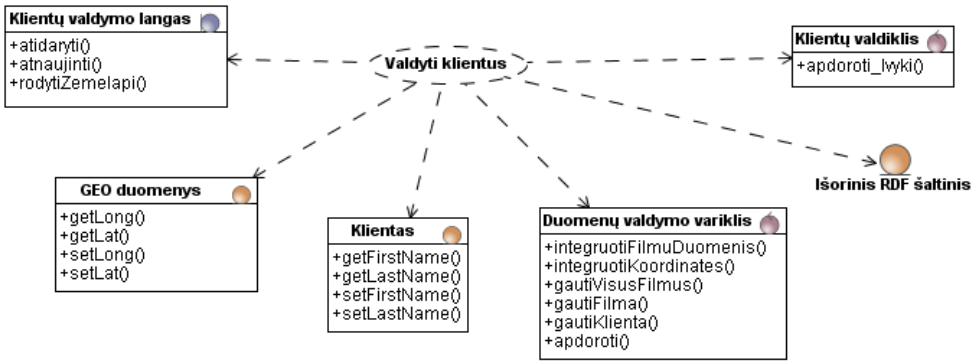


19 pav. PA „Valdyti užsakymus“ realizavimas projektinėmis klasėmis

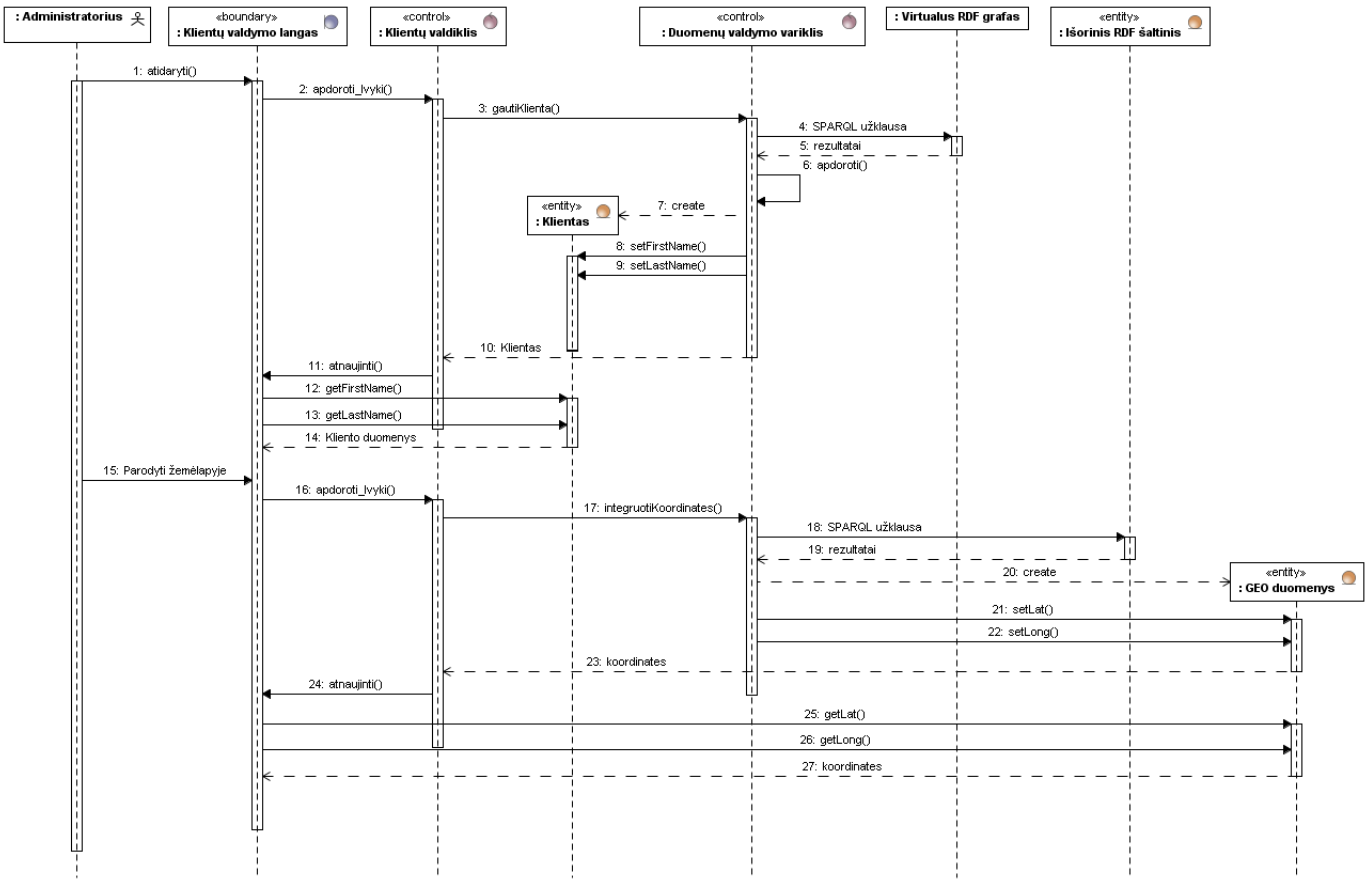


20 pav. Užsakymų valdymo posistemio elgsena

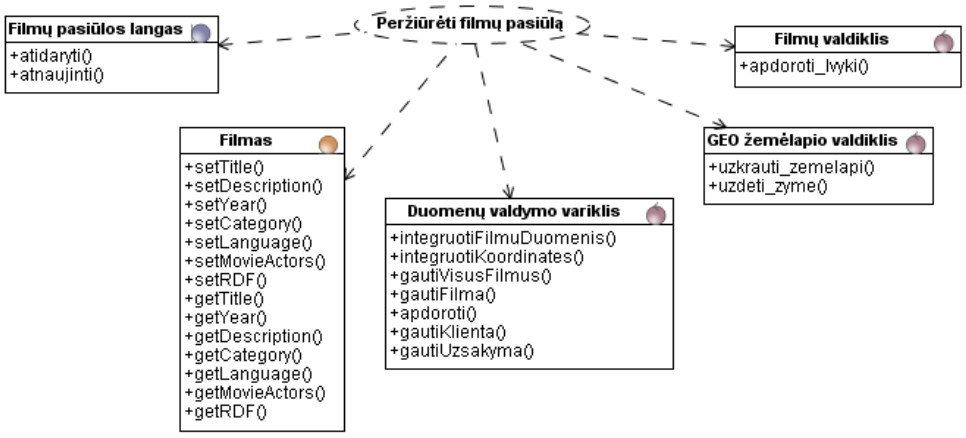
Klientų valdymo posistemio projektas



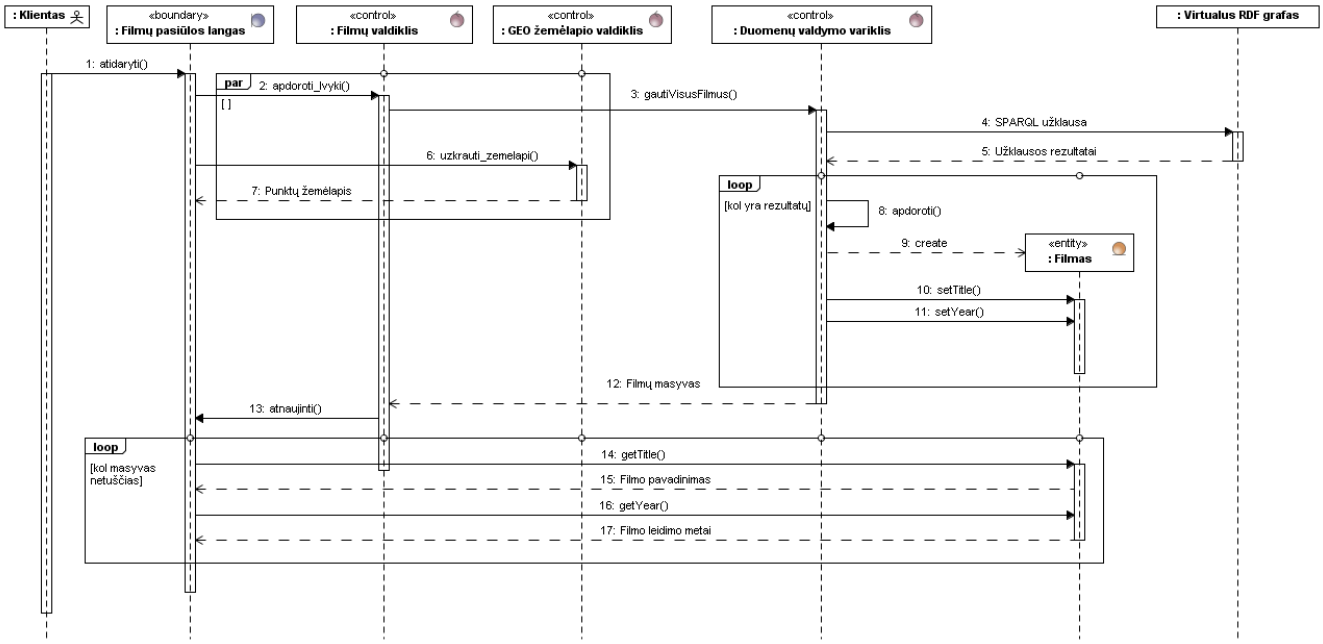
21 pav. PA „Valdyti klientus“ realizavimas projektinėmis klasėmis



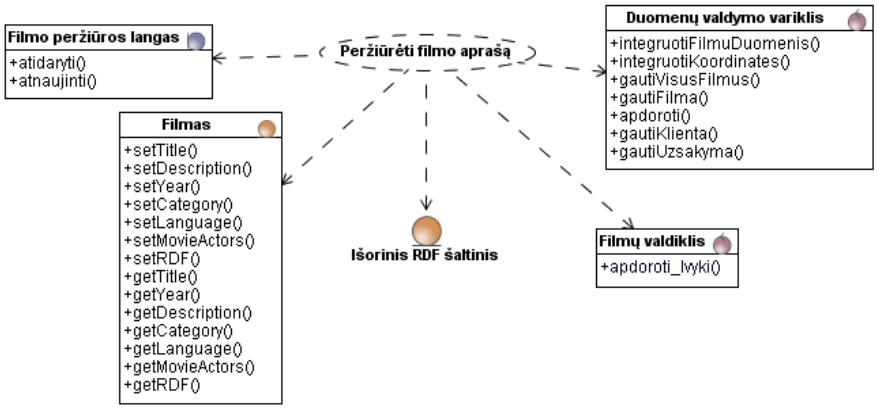
22 pav. Klientų valdymo posistemio elgsena



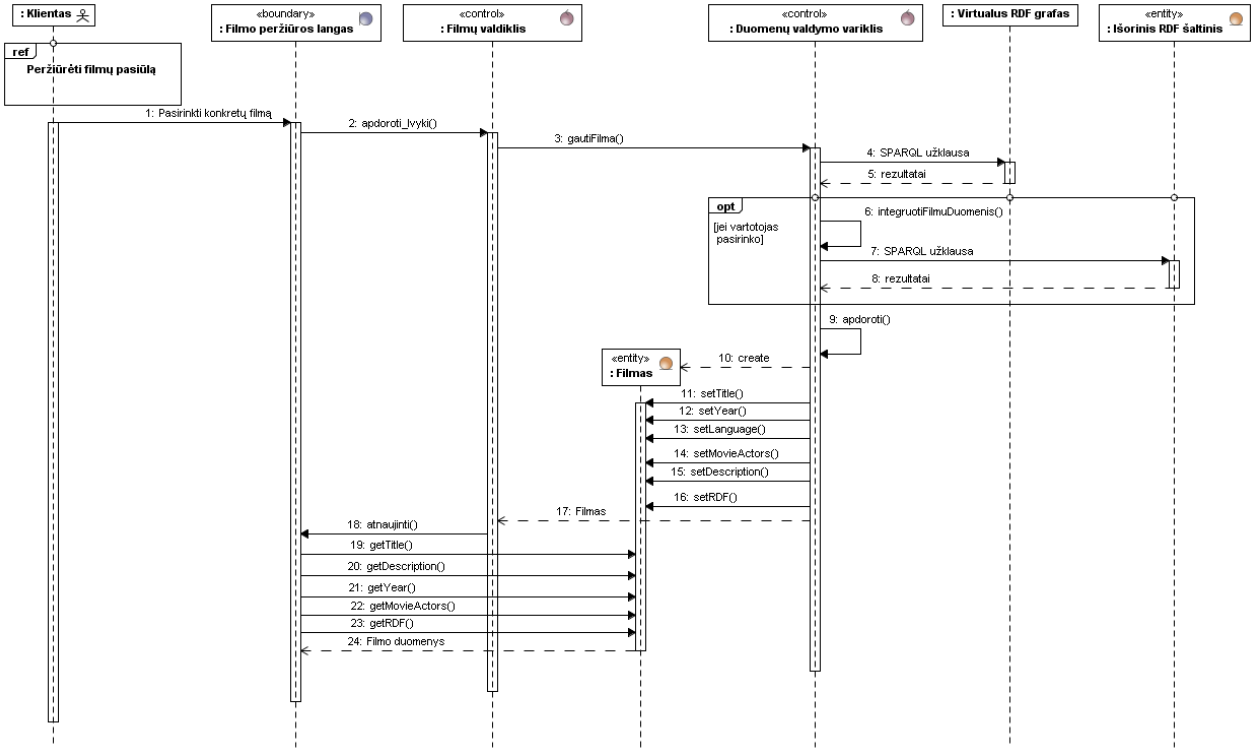
23 pav. PA „Peržiūrėti filmų pasiūlą“ realizavimas projektinėmis klasėmis



24 pav. Filmų valdymo posistemio elgsena PA „Peržiūrėti filmų pasiūlą“ vykdymo metu



25 pav. PA „Peržiūrėti filmo aprašą“ realizavimas projektinėmis klasėmis

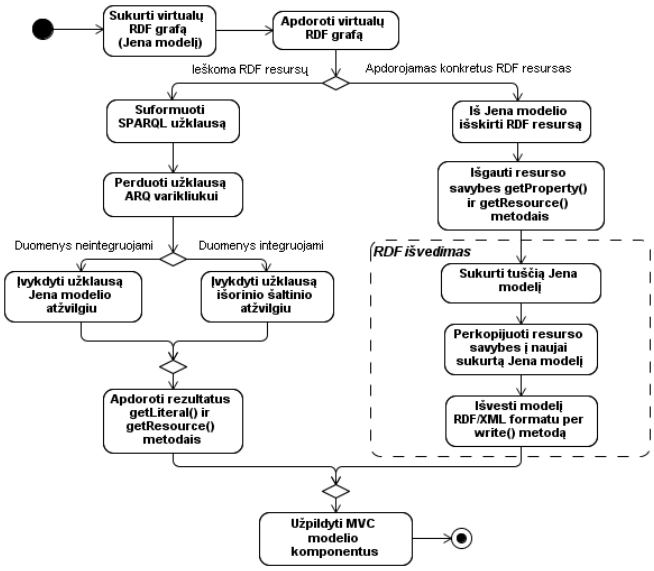


26 pav. Filmų valdymo posistemio elgsena PA „Peržiūrėti filmo aprašą“ vykdymo metu

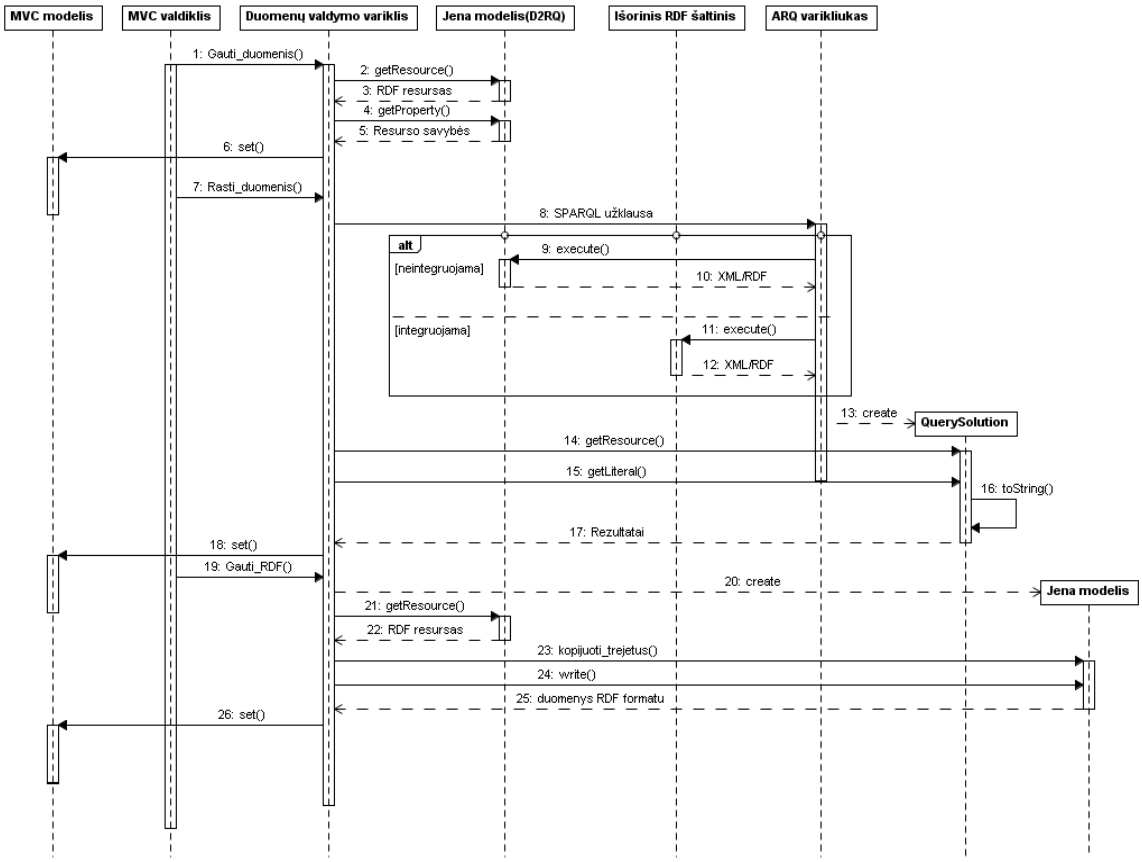
Duomenų valdymo variklio projektas

Vienas pagrindinių komponentų, dalyvaujančių visose PA realizacijose, yra „Duomenų valdymo variklis“. Duomenų valdymo/integravimo variklio paskirtis yra semantiškai praturtinti modelio komponentus duomenimis iš išorinių *RDF* šaltinių – panaudojant *SPARQL* užklausas, gaunami duomenys apie filmo kūrybinę grupę, papildomos nuorodos apie filmą tokiuose šaltiniuose kaip *IMDB*, klientų gyvenamosios vietos geografinės koordinatės ir pan.

Duomenų valdymo variklis taip pat atlieka duomenų iš virtualaus *RDF* grafo nuskaitymo bei apdorojimo funkcijas. Šių, Jena karkaso funkcijų panaudojimo IS principai pateikti 27-28 pav.



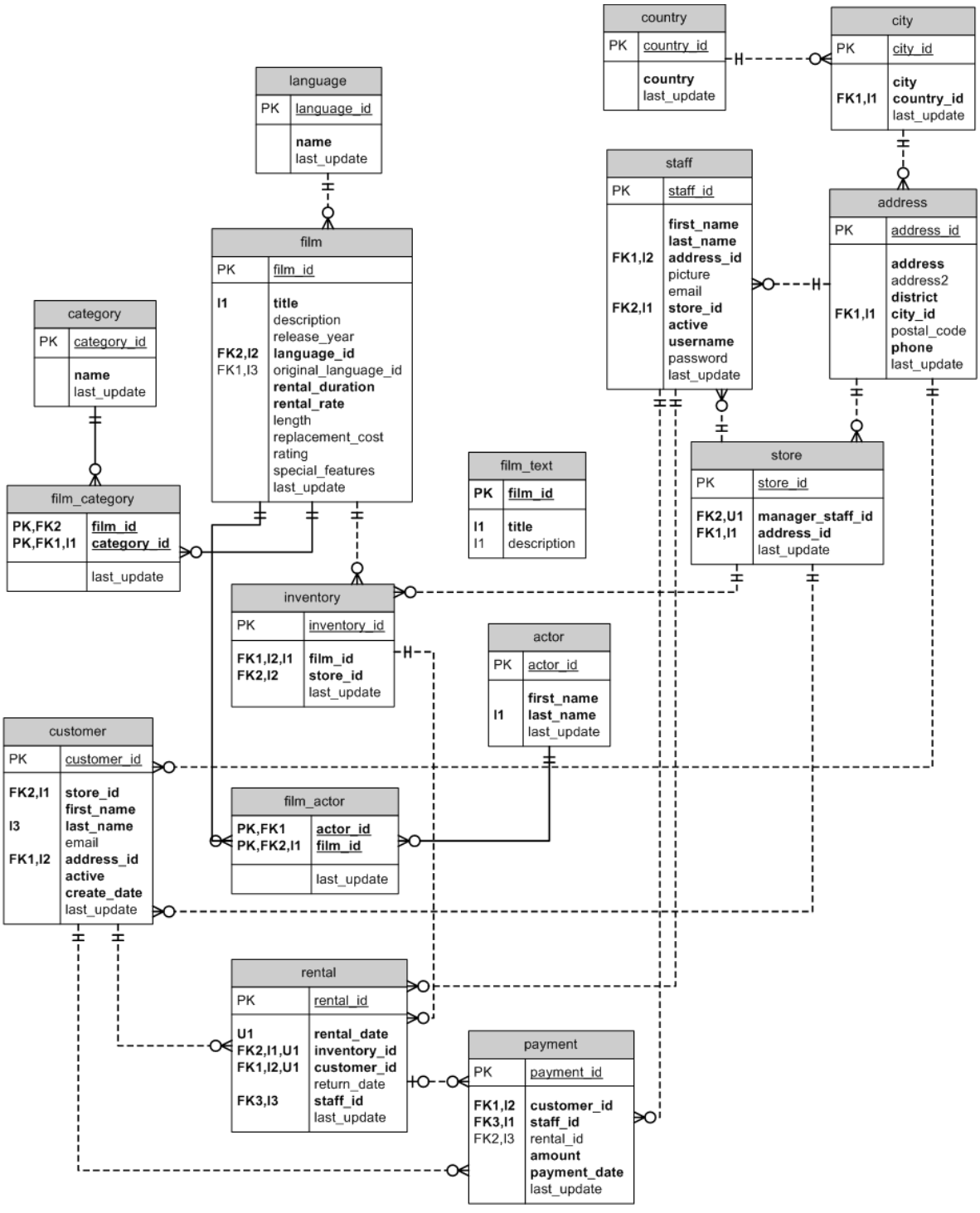
27 pav. Duomenų valdymo variklio operacijų vykdymo logika



28 pav. Detali duomenų valdymo variklio elgsena pagrindinių operacijų vykdymo metu

3.2. Duomenų bazės schema

Kaip minėta reikalavimų specifikacijos IS programiniam prototipui skyrelyje (*žr. 2.2 sk.*), sistemoje naudojama mokomoji *MySQL* reliacinė duomenų bazė „Sakila“, pritaikyta filmų nuomos dalykinei sričiai. Duomenų bazės schema pateikiama žemiau esančiame paveiksle (*žr. 29 pav*.).



29 pav. Duomenų bazės schema

Duomenų bazės lentelių atributų aprašymas pateikiamas žemiau esančioje lentelėje (*žr. 7 lent*.)

7 lent. DB lentelių atributų aprašas

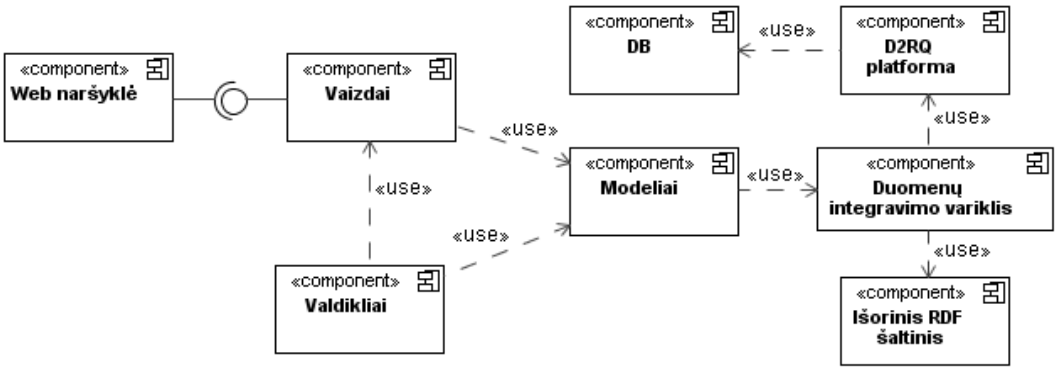
Lentelė	Atributai	Atributo tipas	Apibūdinimas
Actor	actor_id	SMALLINT	Aktoriaus identifikatorius
	first_name	VARCHAR(45)	Aktoriaus vardas
	last_name	VARCHAR(45)	Aktoriaus pavardė
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Film_actor	actor_id	SMALLINT	Aktoriaus identifikatorius
	film_id	SMALLINT	Filmo identifikatorius
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Film	film_id	SMALLINT	Filmo identifikatorius
	title	VARCHAR(255)	Filmo pavadinimas
	description	TEXT	Filmo apibūdinimas
	release_year	YEAR	Filmo išleidimo metai
	language_id	TINYINT	Filmo kalbos identifikatorius
	original_language_id	TINYINT	Originalios kalbos identifikatorius
	rental_duration	TINYINT	Maksimalus nuomos laikotarpis
	rental_rate	DECIMAL	Nuomos kaina
	length	SMALLINT	Filmo trukmė
	replacement_cost	DECIMAL(5,2)	Filmo pakeitimo kaina
	rating	ENUM	Filmo reitingas
	special_features	SET	Papildomi filmo požymiai
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Film_category	film_id	SMALLINT	Filmo identifikatorius
	category_id	SMALLINT	Kategorijos identifikatorius
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Category	category_id	TINYINT	Kategorijos identifikatorius
	name	VARCHAR(25)	Kategorijos pavadinimas
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Language	language_id	TINYINT	Kalbos identifikatorius
	name	CHAR(20)	Kalbos pavadinimas
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Inventory	inventory_id	MEDIUMINT	Nuomos punkto inventoriaus identifikatorius
	film_id	SMALLINT	Filmo identifikatorius
	store_id	TINYINT	Nuomos punkto indentifikatorius
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Film_text	film_id	SMALLINT	Filmo identifikatorius
	title	VARCHAR	Filmo pavadinimas
	description	TEXT	Filmo apibūdinimas
Staff	staff_id	TINYINT	Darbuotojo identifikatorius
	first_name	VARCHAR(45)	Darbuotojo vardas
	last_name	VARCHAR(45)	Darbuotojo pavardė

	address_id	SMALLINT	Adreso identifikatorius
	picture	BLOB	Darbuotojo nuotrauka
	email	VARCHAR(50)	Darbuotojo el. pašto adresas
	store_id	TINYINT	Nuomos punkto identifikatorius
	active	BOOLEAN	Darbuotojo būsena sistemoje
	username	VARCHAR(16)	Darbuotojo slapyvardis
	password	VARCHAR(40)	Darbuotojo slaptažodis
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Store	store_id	TINYINT	Nuomos punkto identifikatorius
	manager_staff_id	TINYINT	Nuomos punkto vadovo identifikatorius
	address_id	SMALLINT	Adreso identifikatorius
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Payment	payment_id	SMALLINT	Apmokėjimo identifikatorius
	customer_id	SMALLINT	Kliento identifikatorius
	staff_id	TINYINT	Darbuotojo identifikatorius
	rental_id	INT	Nuomos identifikatorius
	amount	DECIMAL(5,2)	Apmokėjimo suma
	payment_date	DATETIME	Apmokėjimo data
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Rental	rental_id	INT	Nuomos identifikatorius
	rental_date	DATETIME	Išnuomavimo data
	inventory_id	MEDIUMINT	Nuomos punkto inventoriaus identifikatorius
	customer_id	SMALLINT	Kliento identifikatorius
	return_date	DATETIME	Filmo grąžinimo data
	staff_id	TINYINT	Darbuotojo identifikatorius
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Customer	customer_id	SMALLINT	Kliento identifikatorius
	store_id	TINYINT	Nuomos punkto identifikatorius
	first_name	VARCHAR(45)	Kliento vardas
	last_name	VARCHAR(45)	Kliento pavardė
	email	VARCHAR(50)	Kliento el. pašto adresas
	address_id	SMALLINT	Adreso identifikatorius
	active	BOOLEAN	Kliento būsena sistemoje
	create_date	DATETIME	Kliento įrašo sukūrimo data
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Address	address_id	SMALLINT	Adreso identifikatorius
	address	VARCHAR(50)	Pirminis adresas
	address2	VARCHAR(50)	Antrinis adresas
	district	VARCHAR(20)	Rajonas
	city_id	SMALLINT	Miesto identifikatorius
	postal_code	VARCHAR(10)	Pašto kodas
	phone	VARCHAR(20)	Telefono numeris
	last_update	TIMESTAMP	Paskutinio atnaujinimo data

City	city_id	SMALLINT	Miesto identifikatorius
	city	VARCHAR(50)	Miesto pavadinimas
	country_id	SMALLINT	Valstybės identifikatorius
	last_update	TIMESTAMP	Paskutinio atnaujinimo data
Country	country_id	SMALLINT	Valstybės identifikatorius
	country	VARCHAR(50)	Valstybės pavadinimas
	last_update	TIMESTAMP	Paskutinio atnaujinimo data

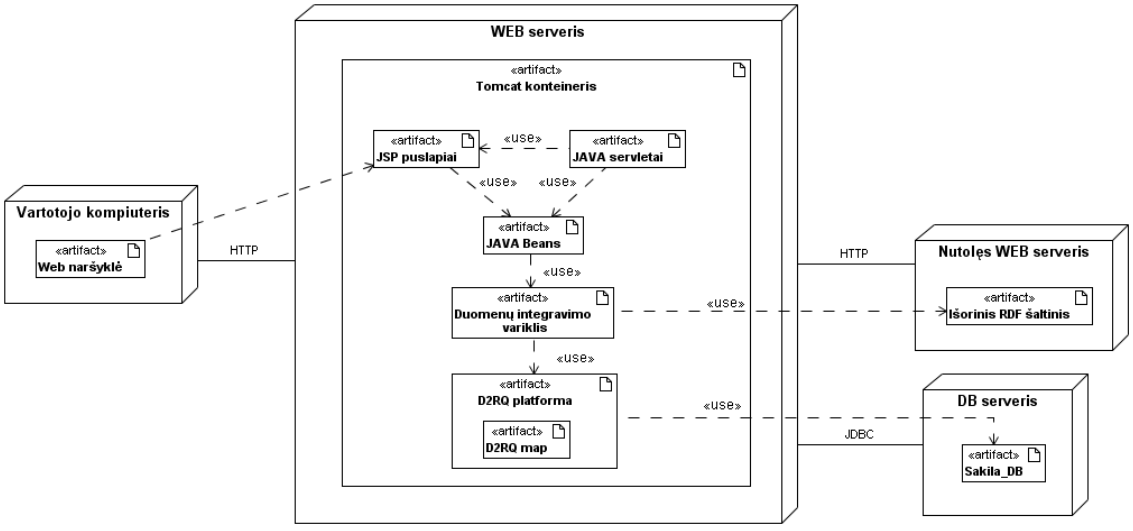
3.3. IS realizacijos modelis

IS realizuojantys komponentai ir jų tarpusavio sąryšiai pateikti 30 pav.



30 pav. Pagrindiniai IS realizuojantys komponentai

Kaip matoma IS diegimo modelyje (*žr. 31 pav.*), projekto programiniam realizavimui pasirinkta *Java* technologija. Šį sprendimą lėmė galimybė lanksčiai realizuoti *MVC* architektūros komponentus diegimo artefaktais: modeliai realizuojami *Java Beans*, valdikliai – *Java* servletais, o vaizdai – *JSP* puslapiais. Diegimo modelyje puikiai išryškėja IS projekto esmė – semantinis *MVC* modelių praturtinimas, panaudojant *D2RQ* įrankį ir išorinius *RDF* šaltinius.



31 pav. IS diegimo diagrama

4. D2RQ įrankio praktinio taikymo metodikos sudarymas

4.1. *D2RQ* virtualaus atvaizdavimo kalbos analizė

D2RQ[16] virtualaus atvaizdavimo kalba skirta realiacinių duomenų bazių (toliau – RDB) schemų ir *RDF* schemų (*RDFS, OWL* bei *DAML+OIL* formatu) tarpusavio sąryšiui nusakyti. *D2RQ* taisyklės rašomos *N3* kalba, kuri buvo pasirinkta kaip alternatyva *RDF/XML* sintaksei, siekiant supaprastinti taisyklių rašymą bei skaitomumą jas kuriantiems žmonėms.

Ontologijų ir RDB schemų tarpusavio sąryšiui nusakyti naudojami du pagrindiniai *D2RQ* kalbos elementai:

- `d2rq:ClassMap` – nusako, kokia ontologijos klasė (*angl. Class*) bus naudojama konkrečios RDB lentelės virtualizavimui,
- `d2rq:PropertyBridge` – nusako, kaip ir kokiems RDB lentelės atributams suteikiamos ontologijos savybės (*angl. Property*).

Virtualaus atvaizdavimo taisyklių rašymas visuomet pradedamas nuo pirmojo elemento, klasių atvaizdo, po to nuosekliai pereinama prie ontologinių savybių priskyrimo lentelės atributams. Apibendrinant, RDB schemos susiejimo su ontologija metu, pagal *D2RQ* taisykles, įgyvendinami šie sąryšiai:

- RDB lentelės susiejamos su ontologijos klasėmis,
- RDB lentelių eilutės su ontologijos klasių egzemplioriais,
- RDB lentelių pirminio rakto atributai virsta resurso URI (RDF subjekto) dalimi.,RDB lentelių kiti atributai susiejami su RDF atributais (predikatais) pagal nurodomas ontologines savybes,
- RDB lentelės įrašai virsta RDF objektais.

Plačiau *D2RQ* kalbos elementai bei pagrindiniai jų akcentai aprašomi sekančiuose skyreliuose.

[16] Šiame ir sekančiuose darbo skyriuose nagrinėjama D2RQ v0.7 versija.

4.1.1. Database

Elementas `d2rq:Database` aprašo visus reikalingus duomenis prisijungimui prie virtualizuojamos RDB. Esant poreikiui, t.y. dirbant su keliomis RDB vienu metu, taisyklių faile galima nurodyti ir kelis šio elemento atvejus.

Šio elemento pagrindinės savybės (sub-elementai)

8 lent. *d2rq:Database* elemento savybės

`d2rq:jdbcDSN`	RDB URL nuoroda kartu su JDBC tvarkyklės prefiksu formate *jdbc:DBprotokolas:Dbadresas*.
`d2rq:jdbcDriver`	JDBC tvarkyklės klasės vardas konkrečiai RDB.
`d2rq:odbcDSN`	RDB URL nuoroda ODBC tvarkyklės atveju.
`d2rq:username`	Prisijungimo prie RDB vardas.
`d2rq:password`	Prisijungimo prie RDB slaptažodis.

Pavyzdys

```
map:database a d2rq:Database;
       d2rq:jdbcDriver "com.mysql.jdbc.Driver";
       d2rq:jdbcDSN "jdbc:mysql://localhost/sakila";
       d2rq:username "root";
       d2rq:password "pass";
       jdbc:autoReconnect "true";
       jdbc:zeroDateTimeBehavior "convertToNull";
       .
```

Pavyzdyje matomi ir `jdbc` vardų srities nustatymai. Jie papildomai parenkami, atsižvelgiant į norimus DB serverio nustatymus, t.y. automatinį prisijungimą, aktyvios sesijos laiką ir pan.

4.1.2. ClassMap

Kaip jau minėta anksčiau, `d2rq:ClassMap` nusako, kokia ontologijos klasė ir kaip bus naudojama, suteikiant RDB lentelei semantinius ryšius. Visos atitinkamos RDB lentelės eilutės, jas virtualizavus, tampa priskirtosios ontologijos klasės egzemplioriais.

Šio elemento pagrindinės savybės (sub-elementai)

9 lent. *d2rq:ClassMap* elemento savybės

d2rq:dataStorage	Nuoroda į 4.1.1 skyrelyje aprašytą d2rq:Database elementą, skirtą prisijungimo duomenims prie RDB.
d2rq:class	Nurodoma ontologijos klasė, kuri vėliau priskiriama konkrečiai RDB lentelei.
d2rq:uriPattern	Aprašoma klasės egzempliorių URI nuorodos struktūra.
d2rq:uriColumn	Nurodomas RDB stulpelis, kuriame saugomi įrašai patys yra kaip resursų URI nuorodos. Tokiu atveju, d2rq:uriPattern naudoti nebereikia.
d2rq:translateWith	Nuoroda į d2rq:TranslationTable elementą, skirtą tam tikrų RDB stulpelių reikšmių pavertimui į kitas reikšmes. Patogu naudoti, siekiant akronimams suteikti jų pilnus vardus.
d2rq:containsDuplicates	Naudojamas dirbant su nepilnai normalizuotomis RDB lentelėmis, kuriose gausu pasikartojančių įrašų. Kreipinių išvertimo į SQL užklausas metu, pridedama DISTINCT sąlyga, tačiau tai gali turėti įtakos kreipinių įvykdymo laikui.
d2rq:condition	Nusako sąlyginio duomenų atvaizdavimo taisykles. Naudojant šį sub-elementą, galima filtruoti, kokie įrašai bus virtualizuojami. Kreipiniai perrašomi į analogiškas SQL WHERE užklausas.
d2rq:classDefinitionLabel	Nusako klasės aprašo žymę.

Pavyzdys

```
map:actor a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "actor/@@actor.actor_id@@";
      d2rq:class movie:Actor;
      d2rq:classDefinitionLabel "actor";
      .
```

Pavyzdyje pateikiamas ontologijos klasės „*Actor*“ priskyrimas `d2rq:ClassMap` elementui per `d2rq:class` sub-elementą. Vėliau šis elementas bus susietas su virtualizuojama RDB lentele. Čia „*movie*“ prefiksas naudojamas kaip nuoroda į konkrečią ontologiją, kurios adresas pateikiamas pačioje taisyklių failo pradžioje, lygiai taip pat kaip ir kitų ontologijų/*RDFS* schemų vardų srities adresai, pvz.:

```
@prefix movie: <http://localhost:8080/d2r/movieRental.owl#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix order: <http://www.purl.org/net/ontology/order.owl#> .
@prefix movie: <http://139.91.183.30:9090/RDF/VRP/Examples/moviedatabase.rdf#>.
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
...
...
```

Klasės egzempliorių *URI* nuorodų struktūra apsprendžiama `d2rq:uriPattern` sub-elemento apraše. Jame lentelės stulpelių reikšmės įterpiamos tarp „@@” simbolių. Kadangi virtualizuotos RDB lentelės eilutės virsta ontologijos klasės egzemplioriais, unikalumui užtikrinti *URI* struktūroje vertėtų naudoti RDB lentelių pirminio rakto atributus. Kaip matoma pateiktame pavyzdyje, `"actor/@@actor.actor_id@@"` struktūros pagrindu sugeneruotos *URI* nuorodos atrodys taip:

```
http://bazinisURI/actor/1
http://bazinisURI/actor/2
http://bazinisURI/actor/3
...
...
```

Čia bazinis *URI* – reliatyvi nuoroda, kurios atžvilgiu generuojami resursų *URI*. Fiksuotą bazinę nuorodą galima nurodyti pačiame `d2rq:uriPattern` sub-elemente (`d2rq:uriPattern "http://konkretusDomenas.com/actor/@@actor.actor_id@@"`), tačiau tokiu atveju prarandamas taisyklių failo portatyvumas.

4.1.3. PropertyBridge

Elementas `d2rq:PropertyBridge` skirtas RDB lentelių atributams suteikti semantinę prasmę. Kitaip tariant, jau sukurtiems klasių egzemplioriams priskiriamos ontologinės savybės, kurių dėka suformuojami RDF trejetų rinkiniai.

Šio elemento pagrindinės savybės (sub-elementai)

10 lent. *d2rq:PropertyBridge* elemento savybės

d2rq:belongsToClassMap	Pateikiama nuoroda į klasę aprašantį d2rq:ClassMap elementą, kurio atžvilgiu kuriamas konkretus d2rq:PropertyBridge elementas.
d2rq:property	Ontologijos savybė (RDF atributas), aprašanti klasės egzempliorių (RDF subjektą) bei sujungianti jį su RDB lentelės įrašu (RDF objektu).
d2rq:column	Lentelės stulpelis (atributas), kuriam priskiriama ontologijos savybė. Nurodoma formatu lentelė.stulpelis.
d2rq:pattern	Naudojamas modifikuojant ar apjungiant lentelių įrašus tarpusavyje. Aktualu, siekiant RDF objektams suteikti prasmingesnius vardus.
d2rq:datatype	RDF objekto duomenų tipas.
d2rq:lang	RDF objekto kalbos koduotė.
d2rq:uriColumn	Naudojamas tokiu atveju, jei lentelės įrašas yra ne tekstinėje formoje, o kaip URI adresas.
d2rq:uriPattern	Jei lentelės įrašas yra URI formate, suteikiama galimybė jį taip pat modifikuoti kaip ir d2rq:pattern atveju.
d2rq:refersToClassMap	Naudojamas nuorodai į kitą d2rq:ClassMap elementą. Šios nuorodos atsiranda dėl išorinio rakto priklausomybių RDB lygmenyje.
d2rq:constantValue	Šis elementas leidžia aprašyti klasių egzempliorių savybes, kurių reikšmės visuose egzemplioriuose yra tos pačios, kitaip tariant - konstanta.
d2rq:join	Naudojamas, siekiant prie tam tikros klasės

	egzemplioriaus prijungti savybes, kurios randasi kitoje RDB lentelėje bei kuri tuo pačiu priklauso ir nuo kito `d2rq:ClassMap` elemento.
`d2rq:condition`	Analogiškas sub-elementas minėtam 4.1.2 skyrelyje, tačiau šiuo atveju taikomas filtruojant ne klasių egzempliorius, o jų savybes.
`d2rq:propertyDefinitionLabel`	Nusako egzemplioriaus savybės žymę.

Pavyzdys nr. 1

```
map:actor_first_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:actor;
      d2rq:property foaf:firstName;
      d2rq:column "actor.first_name";
      d2rq:propertyDefinitionLabel "actor first_name";
      .
```

Pavyzdyje pateikiamas RDB lentelės „*actor*“ stulpelio „*first_name*“ semantinių savybių aprašas *D2RQ* kalba. Visų pirma nurodoma, kurios ontologijos klasės aprašas galioja šio konkretaus stulpelio aprašui. Sekančiu žingsniu nurodoma ontologijos savybė, į kurią lentelės stulpelis atvaizduojamas (šiuo atveju tai `FOAF` žodyno atributas `firstName`). Po to belieka nurodyti RDB lentelės stulpelį, kuriam ontologijos savybė priskiriama.

Pavyzdys nr. 2

```
map:actor__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:actor;
      d2rq:property rdfs:label;
      d2rq:pattern "@@actor.first_name@@ @@actor.last_name@@";
      .
```

Šiame pavyzdyje iliustruojamas `d2rq:pattern` panaudojimo atvejis. *D2RQ* įrankio sugeneruotiems klasių egzemplioriams, kitaip tariant resursams, pagal nutylėjimą suteikiami RDB lentelių pirminio rakto atributų vardai. Dažnu atveju, tai įvairūs skaitmenys ar jų ir kitų simbolių kombinacijos. Minėtieji resursai identifikuojami unikaliais *URI* adresas, kurių struktūra paremta tais pačiais pirminio rakto atributų vardais. Šiuo atveju *URI* adreso struktūros estetika nėra itin aktuali, svarbu išlaikyti resursų unikalumą, tačiau patiems

resursams vertėtų suteikti ir prasmingą pavadinimą. *RDF* trejeto skaitomumas žmogui taps lengvesnis, jeigu *RDF* subjektas bus ne simbolių kratinys, o turės prasmingą vardą. Pateiktame pavyzdyje `rdfs:label` žymė priskiriama visiems aktoriaus klasės egzemplioriams formatu „Vardas Pavardė“. Rezultate, visi aktorių *RDF* trejetai turės subjekto žymę „Vardas Pavardė“, o ne identifikacinį numerį, kuris lieka tik *URI* adrese.

Pavyzdys nr. 3

```
map:aktoriaus_filmai a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:hasActor;
      d2rq:refersToClassMap map:actor;
      d2rq:join "film_actor.film_id => film.film_id";
      d2rq:join "film_actor.actor_id => actor.actor_id";
      .
```

Šis taisyklių fragmentas demonstruoja sub-elementų `d2rq:refersToClassMap` ir `d2rq:join` panaudojimą. Nagrinėjamoje RDB schemoje egzistuoja dvi lentelės: „film“ ir „actor“. Kadangi pagal kontekstą tarp jų yra ryšys M:N, jam panaikinti sukurta tarpinė lentelė „film_actor“. Jos dėka RDB nepageidaujamas M:N ryšys skyla į dvigubus 1:M ir 1:N ryšius. Tačiau *RDF* trejetai nėra paremti reliacine teorija, todėl tarpinės ontologijos klasės, pritaikytos „film_actor“ lentelei šiuo atveju nereikia – ji sąlygotų taisyklių pertekliškumą ir ateityje apsunkintų *SPARQL* užklausų rašymą. Nors *D2RQ* automatinio taisyklių failo generavimo skriptas ir sukuria lentelės „film_actor“ virtualaus atvaizdavimo taisyklių šabloną, jį galima panaikinti ir taisykles perrašyti taip, kaip nurodyta aukščiau esančiame pavyzdyje (tai galioja daugeliui RDB schemų su skaidytais M:N ryšiais, tačiau tik tuo atveju, jei tarpinėje lentelėje nėra papildomų atributų, identifikuojančių ryšio savybę, pvz., ši metodika negaliotų „film_actor“ lentelei, jei ji turėtų dar vieną atributą – „aktoriaus_rolė“).

Taisyklių aprašo fragmento pradžia nesiskiria nuo jau nagrinėtų atvejų: nurodomas ontologijos klasės aprašas, kuris galioja šio stulpelio savybių aprašui bei konkreti ontologijos savybė. Sekančiu žingsniu kompiliatoriui pasakoma, kad egzistuoja nuoroda į kitą ontologijos klasės aprašą, kuris naudojamas tos klasės egzemplioriams sukurti. Toliau nurodomos taisyklės, kaip apjungti abiejų klasių egzempliorius į vieną klasę. Rodyklių kryptis rodo išorinio rakto priklausomybę (galima rašyti ir paprastą „=“). Rezultate, aktoriai ir filmai tarpusavyje susiejami be tarpinių lentelių. Kiekvienas aktoriaus egzempliorius turi *URI* nuorodas į filmų egzempliorius, kuriuose jis vaidino ir atvirkščiai – filmų egzemplioriai

susieti su aktorių egzemplioriais *URI* nuorodomis per atitinkamą `movie:hasActor` ontologijos savybę.

Šioje vietoje verta paminėti, jog *D2RQ* įrankis neturi ontologijų interpretavimo varikliuko (*angl. reasoning*). Todėl nors ir `movie` ontologijoje savybė `hasActor` aprašyta kaip priešinga savybei `isActor`, *D2RQ* įrankis sugeneruotose aktorių egzemplioriuose vietoj savybės `isActor`, nurodys savybę „`is movie:hasActor of`“:

Property	Value
foaf:firstName	PENELOPE
is movie:hasActor of	<http://localhost:8080/d2r/resource/film/1>
is movie:hasActor of	<http://localhost:8080/d2r/resource/film/106>
is movie:hasActor of	<http://localhost:8080/d2r/resource/film/140>
is movie:hasActor of	<http://localhost:8080/d2r/resource/film/166>
is movie:hasActor of	<http://localhost:8080/d2r/resource/film/23>

32 pav. Resurso fragmentas HTML interfeise

Pavyzdys nr. 4

```
map:staff_email a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:propertyDefinitionLabel "staff email";
      d2rq:property foaf:mbox;
      d2rq:uriPattern "mailto:@@staff.email@@";
      .
```

Kaip minėta 4.1.3 skyriaus 10-toje lentelėje, `d2rq:uriPattern` sub-elementas leidžia redaguoti lentelės įrašus, kuriuos norima paversti *URI* formatu (nuorodos, el. pašto adresai ir t.t.). Pavyzdyje nr. 4 pateikiamas atvejis, kai prie el. pašto adreso pridedamas `mailto:` prefiksas. Jo dėka, šio virtualizuoto RDB stulpelio el. pašto adresai virs aktyviomis *URI* nuorodomis.

Pavyzdys nr. 5

```
map:address_address2 a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property vcard:Street;
      d2rq:propertyDefinitionLabel "address address2";
      d2rq:column "address.address2";
      d2rq:condition "address.address2 <> ''";
      .
```

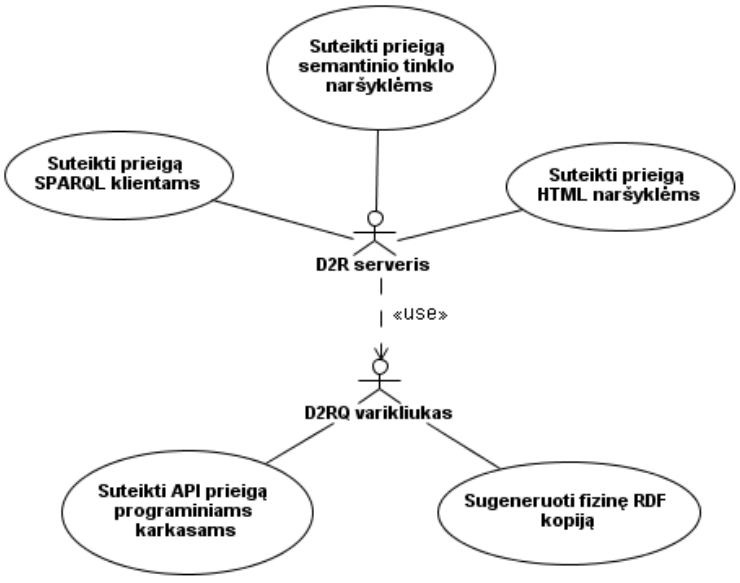
Šiame pavyzdyje demonstruojamas sąlyginio atvaizdavimo atvejis, panaudojant sub-elementą `d2rq:condition`. RDB adreso lentelėje egzistuoja atributas „address2“, kuris naudojamas kaip papildomas adresas. Daugeliu atveju, šis stulpelis lentelėje lieka tuščias. Todėl siekiant išvengti *RDF* trejetų su tuščiais objektais, sąlygos elemente nurodoma taisyklė, teigianti, jog papildomo adreso įrašai turi būti įtraukiami į adreso klasės egzempliorius tik tokiu atveju, jeigu jie nėra tušti.

4.2. *D2RQ* platformos sudedamosios dalys

Visą *D2RQ* platformą pagal panaudos atvejus galima išskirti į dvi pagrindines dalis:

- *D2RQ* varikliukas – jo paskirtis naudojant *D2RQ* virtualaus atvaizdavimo taisyklių failą, perrašyti tiek Jena ir Sesame programinių karkasų *API* kreipinius, tiek *HTTP* kreipinius iš *D2R* serverio į RDB *SQL* užklausas;
- D2R serveris – HTTP serveris, suteikiantis prieigą prie RDB turinio Semantinio tinklo naršyklėms, HTML naršyklėms ir išoriniams SPARQL klientams. D2R serveris gali būti naudojamas kaip web aplikacija, veikianti savo nepriklausomame serveryje arba kaip J2EE aplikacija JAVA servletų konteineryje.

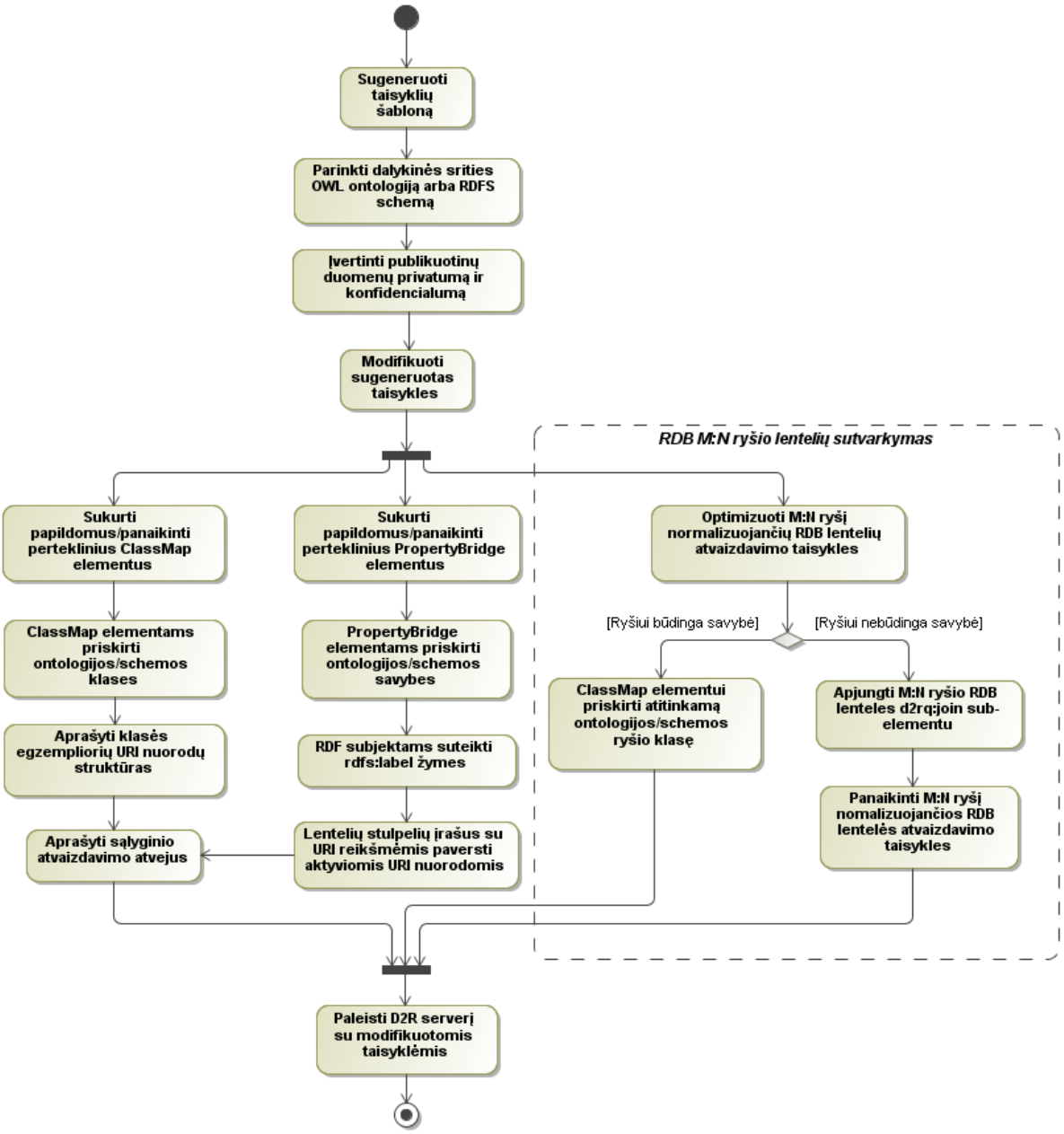
D2RQ platformos sudedamųjų dalių panaudos atvejai pateikti 33 pav.



33 pav. *D2RQ* platformos panaudos atvejai

4.3. Formalizuota *D2RQ* įrankio praktinio taikymo metodika

Atlikus *D2RQ* virtualaus atvaizdavimo kalbos analizę ir praktinius bandymus su mokomąja RDB, sudaryta formali *D2RQ* įrankio praktinio taikymo metodika *UML* notacijoje. Metodika išskaidyta į dvi sudedamąsias dalis – *D2RQ* virtualaus atvaizdavimo taisyklių kūrimą ir modifikavimą (*žr. 34 pav*.) bei *D2RQ* platformos praktinį panaudojimą IS kūrime (*žr. 35 pav*.)



34 pav. *D2RQ* įrankio praktinio taikymo metodika (I d.)

Metodikos žingsnių aprašymas (I dalis):

1. Sugeneruojamas taisyklių šablonas, komandiniame lange įvykdant šią komandą:

```
generate-mapping [-u db_vartotojas] [-p db_slaptažodis] [-d db_tvarkyklė]
[-o taisyklių_failas.n3] [-b bazinis_URI] jdbc_tvarkyklės_nuoroda
```

2. Virtualiam RDB duomenų atvaizdavimui parenkama dalykinės srities ontologija (-os) arba *RDFS* schema (-os). Vertėtų vengti ontologijų, paremtų vien klasėmis (ypač nesusietomis tarpusavyje) bei turinčių ribotą kiekį savybių. Pastarųjų trūkumas apsunkina RDB lentelių atributų atvaizdavimą į ontologijos savybes, o neretai tai tampa neįmanomu uždaviniu. Jeigu dalykinei sričiai pritaikytos ontologijos nėra, vertėtų ją susikurti patiems, iš anksto atsižvelgiant į aukščiau minėtas pastabas.
3. Prieš publikuojant RDB duomenis, reikia nuspręsti, ką norime išpublikuoti. Tai neturėtų būti vartotojų asmeniniai duomenys, slaptažodžiai ar verslo komercinės paslaptys. Tokių duomenų automatiškai sugeneruotos virtualizavimo taisyklės turi būti panaikinamos pagal sekančiuose žingsniuose aprašomus principus.
4. Sugeneruotame taisyklių šablone pateikiamos bazinės RDB virtualizavimo taisyklės, gautos programiškai išanalizavus DB reliacinę struktūrą. Semantiniai ryšiai duomenims suteikiami modifikuojant sugeneruotą taisyklių šabloną pagal sekančius metodikos žingsnius.
5. Panaikinami `ClassMap` ir `PropertyBridge` elementai, aprašantys nenorimų publikuoti duomenų virtualizavimo taisykles. Atlikus šį veiksmą, svarbu įsitikinti, jog taisyklėse nebelieka sąsajų su panaikintais elementais (priklausomybių ieškoti sub-elementuose `d2rq:belongsToClassMap` bei `d2rq:refersToClassMap`). Nors pagal nutylėjimą `ClassMap` elementas skirtas *RDF* resursų, suformuojamų iš RDB lentelės įrašų (eilučių) aprašymui, šis elementas taip pat gali būti naudojamas *RDF* resursų formavimui iš RDB lentelės atributų. Pastaruoju atveju tenka kurti naują (-us) `ClassMap` elementą, kadangi taisyklių šablono generavimo metu nestandartiniai virtualizavimo atvejai nėra įvertinami. Taisyklės, skirtos naujo *RDF* resurso suformavimui ir susiejimui su jau egzistuojančiu, pateikos žemiau esančiame pavyzdyje. Pirmasis `ClassMap` elementas „*actor*“ automatiškai sugeneruojamas taisyklių šablone, o antrasis elementas „*person*“ sukuriamas rankiniu būdu. Abu elementai susiejami tarpusavyje per savybę `movie:hasName`, panaudojant elementą `PropertyBridge`.

```
map:actor a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "actor/@@actor.actor_id@@";
      d2rq:class movie:Actor;
      d2rq:classDefinitionLabel "actor";
      .
map:person a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "actorName/@@actor.first_name@@_@@actor.last_name@@";
      d2rq:class movie:FullName;
      d2rq:classDefinitionLabel "person";
      .
map:actor_person a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:actor;
      d2rq:property movie:hasName;
      d2rq:refersToClassMap map:person;
      .
```

Rezultate gaunamas *RDF* trejetas:

```
<bazinisURI/actor/123> movie:hasName <bazinisURI/actorName/Vardas_Pavarde>
```

Toks virtualizavimo principas taikytinas tuo atveju, jeigu pasirinktoje ontologijoje dalykinės srities objektas, į kurį norime atvaizduoti RDB lentelės atributą yra aprašytas ontologijos klase, o ne savybe. Tačiau ir šiuo atveju ontologija turi būti pilnai išbaigta, t.y. turi egzistuoti ryšys (pvz.: `:hasName`) tarp klasės, kurios egzemplioriai virsta *RDF* subjektu, ir tarp klasės, kurios egzemplioriai virsta *RDF* objektu. Verta paminėti, kad *RDF* resursų generavimas nenaudojant RDB lentelės pirminio rakto atributų gali sąlygoti resursų unikalumo praradimą.

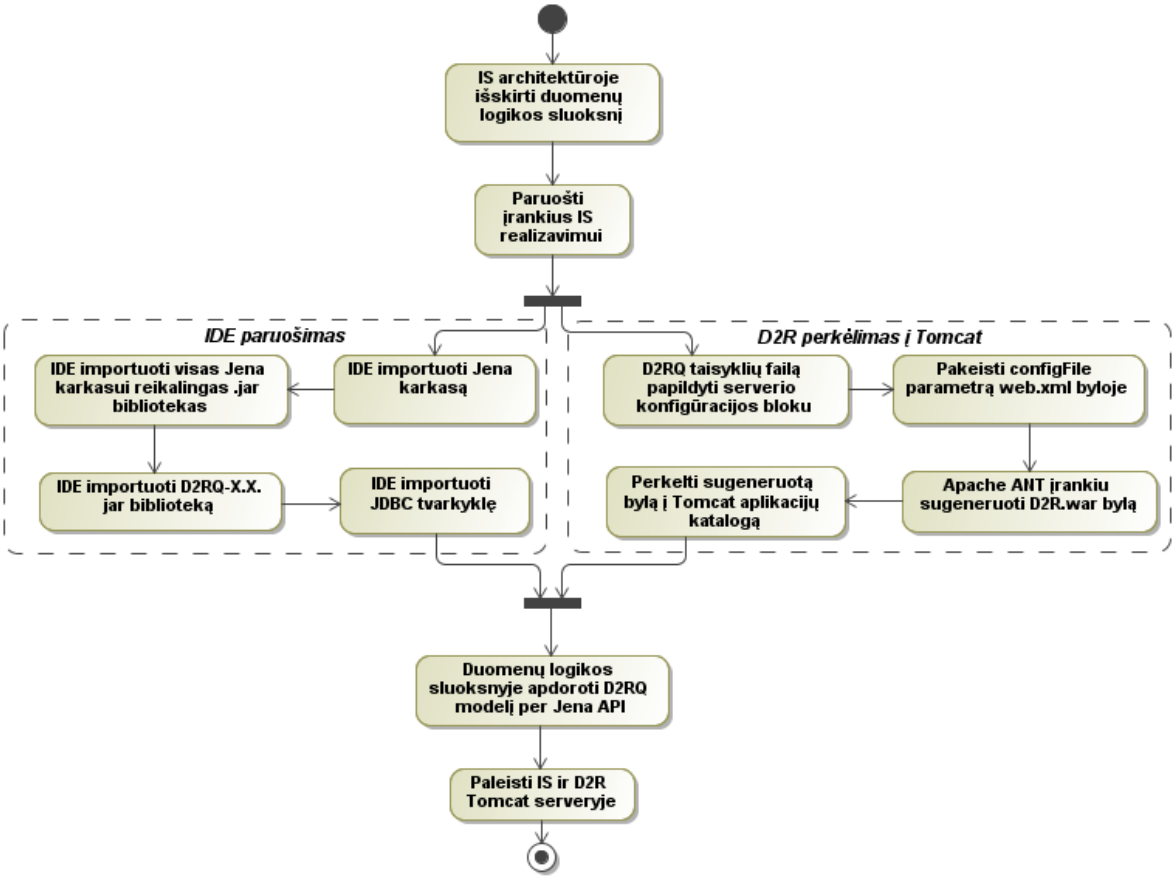
6. `ClassMap` ir `PropertyBridge` elementams ontologijos klasės ir savybės priskiriamos pagal 4.1.2 – 3 skyreliuose aprašytas taisykles. Vienam elementui galima priskirti neribotą kiekį ontologijos klasių/savybių – tai lems papildomų *RDF* trejetų sugeneravimą.
7. Klasės egzempliorių *URI* struktūra aprašoma panaudojant `d2rq:uriPattern` sub-elementą pagal 4.1.2 skyrelyje pateiktą pavyzdį.
8. Pati pirma savybė, kurią reikėtų priskirti generuojamiems *RDF* resursams yra *RDFS* žodyno `rdfs:label` žymė (*plačiau žr. 4.1.3 sk., Pavyzdys Nr. 2*). Tokiu būdu klasių egzemplioriai gauna prasmingus vardus net ir tais atvejais, kuomet dalykinės srities ontologijoje tam numatytų savybių nėra.
9. Jeigu RDB lentelėje saugomi tokie įrašai kaip tinklalapių *URL* adresai arba *RDF* resursų *URI* nuorodos, juos reikėtų paversti aktyviomis nuorodomis, panaudojant elemento `PropertyBridge` sub-elementą `d2rq:uriColumn`.

10. Sąlyginio atvaizdavimo atvejai aprašomi panaudojant `d2rq:condition` sub-elementą, kuris tinka tiek `ClassMap`, tiek `PropertyBridge` elementams (*plačiau žr. 4.1.3 sk., Pavyzdys Nr. 5*).

RDB M:N ryšio lentelių sutvarkymas:

11. Kaip minėta 4.1.3 skyrelyje (*žr. Pavyzdys Nr. 3*), *D2RQ* taisyklių generavimo skriptas automatiškai sukuria virtualizavimo taisykles ir tarpinėms M:N ryšį normalizuojančioms lentelėms. Jeigu pastarosios lentelės neturi jokių kitų atributų, (išskyrus normalizuojamų lentelių pirminio rakto atributus), kitaip tariant, jeigu M:N ryšiui nebūdinga jokia savybė, tarpinės lentelės virtualizavimo taisyklės tampa perteklinės. Tokiu atveju M:N ryšio lentelės apjungiamos `d2rq:join` sub-elementu, o perteklinės taisyklės panaikinamos.
12. Jeigu M:N ryšiui yra būdinga savybė, sugeneruotos tarpinės lentelės virtualizavimo taisyklės paliekamos, o `ClassMap` elementui priskiriama atitinkama ontologijos ryšio klasė.
13. Virtualizavimo taisyklių failas išsaugomas ir jo pagrindu paleidžiamas *D2R* serveris:

```
d2r-server taisyklių_failas.n3
```



35 pav. *D2RQ* įrankio praktinio taikymo metodika (II d.)

Metodikos žingsnių aprašymas (II dalis):

1. IS architektūroje išskirti duomenų logikos sluoksnį – kaip ir standartinėse web aplikacijose, semantinėse aplikacijose taip pat rekomenduotina atskirti duomenų valdymo logiką nuo jų pateikimo.
2. Paruošti įrankius IS realizavimui – *žr. sekančius punktus*.
3. *IDE* paruošimas – į projektą būtina importuoti visas Jena karkaso naudojamas bibliotekas. Siekiant išvengti versijų suderinamumo problemų, vertėtų naudoti bibliotekas, esančias *D2RQ* prog. paketo */lib* kataloge.
4. *D2R* perkėlimas į *Tomcat* – siekiant aukštesnio sistemos patikimumo lygio, rekomenduotina *D2R* serverio aplikaciją perkelti į *Tomcat* serverį. Prieš atliekant šį veiksmą, *D2RQ* taisyklių failas papildomas konfigūracijos bloku:

```
<> a d2r:Server;
    rdfs:label "Serverio pavadinimas";
    d2r:baseURI <http://localhost:8080/d2r/>; //serverio URI
    d2r:port 8080;                            //serverio portas
    .
```

5. *D2R* serverio kataloge, *\webapp\WEB-INF* direktorijoje, *web.xml* byloje pakeičiamas parametras *configFile*, nurodantis virtualizavimo taisyklių failo vietą (Patartina jį perkelti į tą pačią *\webapp\WEB-INF* direktoriją.)

```
<context-param>
    <param-name>configFile</param-name>
    <param-value>taisykles.n3</param-value>
</context-param>
```

6. *D2R* serverio katalogo pagrindinėje direktorijoje, komandiniame lange įvykdoma *Apache Ant* prog. paketo[17] komanda:

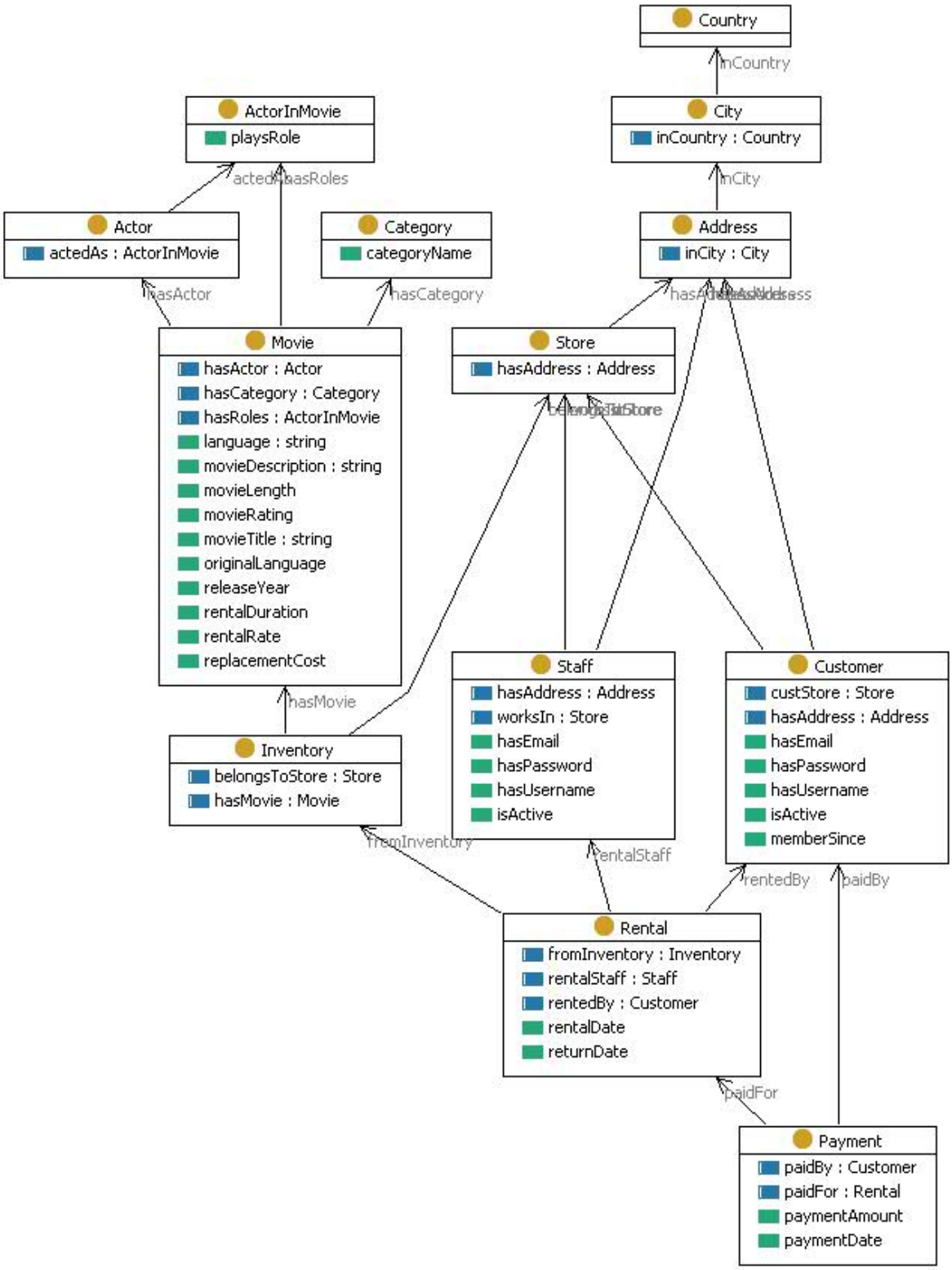
```
ant war
```

7. Sugeneruotas *.war* failas perkeliamas į *Tomcat* aplikacijų katalogą */webapps*.
8. Duomenų logikos sluoksnyje taisyklių failo atžvilgiu sukuriamas virtualus *RDF* grafas, kuris toliau Jena karkaso instrumentinėmis priemonėmis apdorojamas, išpildant numatytus IS funkcinius reikalavimus.
9. Sukurta IS taip pat perkeliama į *Tomcat* aplinką bei paleidžiama kartu su *D2R* serveriu.

[17] http://ant.apache.org/

5. IS prototipo realizacija

5.1. Filmų nuomos dalykinės srities ontologijos schema



36 pav. Dalykinės srities ontologijos schema

36 pav. pavaizduota IS dalykinės srities – filmų nuomos – ontologijos schema, sukurta naudojant TopBraid Composer[18] modeliavimo įrankį. Ontologija buvo kuriama metodiniais tikslais, labiau orientuojantis į specifinę RDB „Sakila“ struktūrą bei siekiant aprašyti visų joje esančių duomenų virtualizavimo taisykles.

5.2. IS prototipo RDB elementų atvaizdai į ontologijos elementus

11 lent. pateikti RDB lentelių elementus atvaizduojantys ontologijos elementai. Dalykinės srities ontologijos elementai vaizduojami be prefikso, tuo tarpu bendrosios paskirties žodynų (FOAF, vCard) elementai su atitinkamais prefiksais. Lentelių pirminių raktų atributai nepateikiami, kadangi jie naudojami tik kaip resursų unikalūs identifikatoriai – *URI* nuorodų dalis.

11 lent. RDB elementų atvaizdai į ontologijos elementus

RDB lentelės elementas	**Ontologijos schemos elementas**
Lentelė *Movie*	
title	:movieTitle
description	:movieDescription
release_year	:realeaseYear
language_id	:language
original_language_id	:language
rental_duration	:rentalDuration
rental_rate	:rentalRate
length	:movieLength
replacement_cost	:replacementCost
rating	:movieRating
Lentelė *Actor*	
first_name	foaf:firstName
last_name	foaf:lastName
Lentelė *Film_Actor**	
actor_id film_id	:hasActor
Lentelė *Category*	
name	:categoryName
Lentelė *Film_Category**	
film_id category_id	:hasCategory
Lentelė *Staff*	
first_name	foaf:firstName
last_name	foaf:lastName
address_id	:hasAddress
email	:hasEmail
store_id	:worksIn
active	:isActive
username	:hasUsername

[18] http://www.topquadrant.com/products/TB_Composer.html

password	:hasPassword
Lentelė *Store*	
address_id	:hasAddress
Lentelė *Inventory*	
film_id	:hasMovie
store_id	:belongsToStore
Lentelė *Rental*	
rental_date	:rentalDate
inventory_id	:fromInventory
customer_id	:rentedBy
return_date	:returnDate
staff_id	:rentedBy
Lentelė *Payment*	
customer_id	:paidBy
rental_id	:paidFor
amount	:paymentAmount
payment_date	:paymentDate
Lentelė *Customer*	
store_id	:custStore
first_name	foaf:firstName
last_name	foaf:lastName
email	:hasEmail
address_id	:hasAddress
active	:isActive
create_date	:memberSince
Lentelė *Address*	
city_id	:inCity
address	vcard:ADR
address2	vcard:ADR
district	vcard:Locality
postal_code	vcard:Pcode
phone	foaf:Phone
Lentelė *City*	
country_id	:inCountry
Lentelė *Country*	
country	vcard:Country

**Pastaba*: RDB M:N ryšį normalizuojančių lentelių atributai atitinkamiems ontologijos elementams priskiriami ne tiesiogiai, o pagal 4.1.3 sk. aprašytas taisykles (žr. Pvz. 3).

IS prototipo RDB virtualizavimo į *RDF* taisyklės pateiktos 9.1*priede*.

5.3. IS prototipo realizacija Java platformoje

Informacinės sistemos realizavimui pasirinkta Java platforma bei *NetBeans IDE* aplinka. Šį pasirinkimą lėmė keli pagrindiniai faktoriai:

- didžioji dalis programinių karkasų bei bibliotekų, skirtų semantinio tinklo sistemoms kurti yra realizuoti ir suderinami tik su Java technologijomis;

- informacinės sistemos projektinėje dalyje pasirinkta architektūra, paremta *MVC* paradigma. *Java* technologijos (servletai, *JavaBeans, JSP*) puikiai tinka pasirinktosios architektūros komponentų realizavimui;
- tiek kuriamą IS, tiek *D2R* serverį galima patalpinti į patikimumu pasižymintį *Tomcat* servletų konteinerį/*web* serverį, tokiu būdu užtikrinant efektyvų abiejų sistemų valdymą bei monitoringą.

Sekančiuose poskyriuose detaliau aprašomi atskirų komponentų realizavimo *Java* programavimo kalba principai bei apžvelgiamas specifinių pagalbinių įrankių taikymas semantinių IS kūrime.

5.3.1. *Java* klasių generavimas *Schemagen* įrankiu

Schemagen[19] įrankis, pateikiamas kartu su *Jena* programiniu karkasu yra skirtas *Java* klasių generavimui iš *OWL, DAML* ontologijų bei *RDFS* žodynų. Klasės generuojamos, komandiniame lange iškviečiant `schemagen.bat` vykdomąjį failą:

```
Jena_katalogas\bat>call schemagen -i ../movieRental.owl -n movieRental
-a http://localhost:8080/d2r/movieRental.owl# -o movieRental.java --owl
```

Schemagen skriptui perduodami šie parametrai:

- `-i` – ontologijos failo vieta sistemoje,
- `-n` – generuojamos Java klasės pavadinimas,
- `-a` – ontologijos vardų srities URI,
- `-o` – sugeneruoto klasės failo pavadinimas,
- `--owl` – nurodoma kalba, kuria parašyta ontologija (šiuo atveju OWL).

Java klasės generavimo metu sukuriami statiniai kintamieji, aprašantys ontologijos klases bei savybes. Šie kintamieji vėliau naudojami virtualaus *RDF* grafo duomenų apdorojimui, pasitelkiant kitas *Jena* instrumentines priemones.

5.3.2. Duomenų logikos sluoksnio realizacija

Pagal sudarytos *D2RQ* įrankio praktinio taikymo metodikos rekomendacijas (*žr. 4.3 sk, 35 pav.*), IS prototipe išskirtas duomenų logikos sluoksnis (failas `DataEngine.java`), atsakingas už duomenų apdorojimą, integravimą su išoriniais šaltiniais, duomenų nuskaitymą iš virtualaus *RDF* grafo bei atitinkamų *MVC* modelių užpildymą.

[19] http://jena.sourceforge.net/how-to/schemagen.html

Visų tolimesnių operacijų ir veiksmų pagrindas duomenų logikos sluoksnyje yra virtualus *RDF* grafas, gaunamas sukuriant Jena karkaso modelį RDB virtualizavimo taisyklių failo atžvilgiu.

```
String baseURI = "http://localhost:8080/d2r/resource/";
ModelD2RQ m = new ModelD2RQ("file:C:/d2r/taisykles.n3",null,baseURI);
```

Jena modelio `ModelD2RQ` klasės konstruktoriui perduodami parametrai: virtualizavimo taisyklių failo vieta diske, taisyklių failo formatas (pagal nutylėjimą *N3*) bei bazinė *RDF* resursų nuoroda, naudojama pilnų modelio resursų *URI* generavimui, prijungiant reliatyvią *URI* dalį, aprašytą virtualizavimo taisyklėse.

Sukurtas virtualus *RDF* grafas toliau apdorojmas dviem būdais: naudojant *Jena* navigavimo *RDF* grafu metodus arba vykdant *SPARQL* užklausas per *ARQ*[20] užklausų valdymo varikliuką.

Navigavimas virtualiu *RDF* grafu (duomenų apie filmą išgavimas):

```
//Per parametrą 'ID ' suformuojamas resurso URI
String resource = "http://localhost:8080/d2r/resource/"+ID;
//Iš Jena modelio išgaunamas suformuotas RDF resursas
Resource movie = m.getResource(resource);
//Get() metodais išgaunamos RDF resursą aprašančios savybės
movie.getProperty(MovieRental.movieTitle).getString();
movie.getProperty(MovieRental.language).getString();
movie.getProperty(MovieRental.movieDescription).getString();
//Ciklinis RDF objektų išgavimas, naudojant tą patį predikatą
StmtIterator iter = movie.listProperties(MovieRental.hasActor);
ArrayList list = new ArrayList();
   while (iter.hasNext()) {
       list.add(iter.nextStatement()
               .getResource()
                  .getProperty(RDFS.label).getString());
            }
```

Toks duomenų išgavimo iš *Jena* modelio principas taikytinas tuo atveju, kuomet žinome pilną resurso *URI* nuorodą ir siekiame rasti jį aprašančias savybes.

SPARQL užklausų vykdymas per *ARQ* varikliuką (nuomos punkto užsakymų paieška):

```
//Parametrizuota SPARQL užklausa aprašoma String formatu
String sparql =
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
        "PREFIX movie: <http://localhost:8080/d2r/movieRental.owl#> " +
        "SELECT DISTINCT ?rentalDate ?returnDate ?movie ?orderURI WHERE {"
+
        " ?URI movie:belongsToStore " + "<" + baseURI + ID + "> ." +
        " ?URI movie:hasMovie ?movieURI ." +
        " ?movieURI rdfs:label ?movie ." +
        " ?orderURI movie:fromInventory ?URI ." +
```

[20] http://incubator.apache.org/jena/documentation/query/index.html

```
        " ?orderURI movie:rentalDate ?rentalDate ." +
        " ?orderURI movie:returnDate ?returnDate ." +
        " } ";
//Užklausa perduodama ARQ užklausų varikliukui
Query q = QueryFactory.create(sparql);
//Užklausa įvykdoma Jena modelio 'm ' atžvilgiu
ResultSet rs = QueryExecutionFactory.create(q, m).execSelect();
//Rezultatų apdorojimas bei priskyrimas MVC modelio komponentui
while (rs.hasNext()) {
     QuerySolution row = rs.nextSolution();
     OrderBean obean = new OrderBean();
     obean.setRentalDate(row.getLiteral("rentalDate").getString();
     obean.setReturnDate(row.getLiteral("returnDate").getString();
     obean.setMovie(row.getLiteral("movie").getString());
     obean.setOrderURI(row.getResource("orderURI").toString();
     list.add(obean);
 }
```

Skirtingai nei navigavimas *RDF* grafu, pasitelkiant *Jena API* metodus, duomenų išrinkimas *SPARQL* užklausomis dažnais atvejais sumažina rašomo kodo kiekį ir sudėtingumą. Tai ypač aktualu, kuomet duomenys, reikalingi realizuojamam funkciniam reikalavimui išpildyti yra išsibarstę skirtingose *RDF* grafo viršūnese, t.y. kelias iki jų eina per kelis ar net keliolika *RDF* trejetų.

IS vienas pagrindinių uždavinių yra pateikti informaciją tiek *HTML*, tiek *RDF* formatais. Jena karkasas suteikia galimybę kiekvieną Jena modelį išvesti kaip baitų srautą (*RDF/XML, N-Triple, Turtle* ar *N3* formoj). Kadangi IS posistemiuose, kuriuose prasminga pateikti informaciją RDF formatu veiksmai atliekami ne su visu Jena modeliu, o su atskirais RDF resursais, iš pastarųjų tenka konstruoti naujus Jena modelius:

```
//Sukuriamas tuščias Jena modelis
Model temp = ModelFactory.createDefaultModel();
//Modeliui ciklišKai priskiriami visi resurso 'movie' RDF trejetai
StmtIterator iter = movie.listProperties();
        while(iter.hasNext()){
            Statement stmt = iter.nextStatement();
            temp.add(stmt);
        }
//Jena modeliui priskiriami vardų srities prefiksai ir adresai
temp.setNsPrefix("movie", "http://localhost:8080/d2r/movieRental.owl#");
temp.setNsPrefix("foaf", "http://xmlns.com/foaf/0.1/");
ByteArrayOutputStream baitai = new ByteArrayOutputStream();
//Jena modelis RDF formate išvedamas į baitų srautą
temp.write(baitai, "RDF/XML");
//Baitų srautas konvertuojamas į String tipą ir priskiriamas MVC modeliui
mbean.setMovieRDF(baitai.toString());
```

Duomenų integravimui sistemoje naudojami du išoriniai RDF šaltiniai, turintys *SPARQL* prieigos tašką: *LinkedMDB*[21] ir *DBpedia*[22]. *LinkedMDB* šaltinis naudojamas papildomai informacijai apie konkretų filmą gauti, o *DBpedia* – kliento gyvenamosios vietos (miesto) koordinačių nustatymui. *SPARQL* užklausų vykdymas išorinių prieigos taškų atžvilgiu panašus į aukčiau minėtą atvejį, kuomet kreipiamasi į virtualų *RDF* grafą. Šiuo atveju skiriasi tik vykdomasis *Jena* metodas.

Kreipimasis į *LinkedMDB* šaltinį (papildomų duomenų apie filmą išgavimas):

```
//Parametrizuota SPARQL užklausa aprašoma String formatu
String sparql=
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>"+
        "PREFIX foaf: <http://xmlns.com/foaf/0.1/>"+
        "PREFIX dc: <http://purl.org/dc/terms/>" +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
        "PREFIX movie: <http://data.linkedmdb.org/resource/movie/>" +
        "SELECT DISTINCT ?page ?director ?editor ?producer WHERE {"+
        " ?res dc:title "+ "\"" +
StringEscapeUtils.escapeSql(movieTitle) + "\"" + "." +
        " ?res dc:date "+ "?data" + "." +
        " ?res foaf:page ?page . " +
        " OPTIONAL { ?res movie:director ?directorURI . " +
        " ?directorURI rdfs:label ?director . }" +
        " OPTIONAL {?res movie:editor ?editorURI . " +
        " ?editorURI rdfs:label ?editor . }" +
        " OPTIONAL { ?res movie:producer ?producerURI . " +
        " ?producerURI rdfs:label ?producer . }" +
        " FILTER regex(?data," + "'" + date + "',"+"'i')" +
        " }" ;
//Užklausa perduodama ARQ užklausų varikliukui
Query query = QueryFactory.create(sparql);
//Užklausa įvykdoma LinkedMDB šaltinio SPARQL prieigos taško atžvilgiu
QueryExecution qexec =
QueryExecutionFactory.sparqlService("http://data.linkedmdb.org/sparql",
query);
   try {
      ResultSet results = qexec.execSelect();
      while (results.hasNext()) {
        QuerySolution row = results.nextSolution();
        ... //Rezultatų apdorojimas bei priskyrimas MVC modelio komponentui
          }
       }
    catch (QueryExceptionHTTP e) {
            System.err.println("<response>: ");
            System.err.println(e.getResponseCode());
            System.err.println("</response> ");
            System.err.flush();
            e.printStackTrace();
        }
```

Vykdant *SPARQL* užklausą išorinio šaltinio atžvilgiu, vertėtų nepamiršti naudoti `try{} catch{}` blokų galimoms *HTTP* klaidoms aptikti (403, 404 ir pan.).

[21] http://linkedmdb.org/
[22] http://dbpedia.org/About

Kreipimasis į *DBpedia* šaltinį (kliento gyvenamosios vietos koordinačių nustatymas):

```
//Parametrizuota SPARQL užklausa aprašoma String formatu
String sparql=
      "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"+
      "PREFIX foaf: <http://xmlns.com/foaf/0.1/>"+
      "PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>" +
      "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
      "PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>" +

      " SELECT DISTINCT ?lat ?long WHERE {"+
      " ?URI rdf:type dbpedia-owl:Place." +
      " ?URI dbpedia-owl:isPartOf <http://dbpedia.org/resource/Florida>." +
      " ?URI foaf:name ?var." +
      " ?URI geo:lat ?lat. " +
      " ?URI geo:long ?long." +
      " FILTER regex(?var," + "'" + city + "',"+"'i')" +
      " }" ;
//Užklausa perduodama ARQ užklausų varikliukui
Query query = QueryFactory.create(sparql);
//Užklausa įvykdoma DBpedia šaltinio SPARQL prieigos taško atžvilgiu
QueryExecution qexec =
QueryExecutionFactory.sparqlService("http://dbpedia.org/sparql", query);
  try {
     ResultSet results = qexec.execSelect();
//Rezultatų apdorojimas bei priskyrimas MVC modelio komponentui
     while (results.hasNext()) {
        QuerySolution row = results.nextSolution();
        gbean.setLat(row.getLiteral("lat").getString());
        gbean.setLong(row.getLiteral("long").getString());
          }
       }
   catch (QueryExceptionHTTP e) {
        System.err.println("<response>: ");
        System.err.println(e.getResponseCode());
        System.err.println("</response> ");
        System.err.flush();
        e.printStackTrace();
      }
```

Neretai *SPARQL* užklausų vykdymas išorinių prieigos taškų atžvilgiu gali užimti nemažai laiko, ypač, jeigu užklausa sudėtinga ar šaltinio serveris neužtikrina reikiamo pralaidumo. Įvertinus šias galimas aplinkybes, visi IS duomenų integravimo uždaviniai vykdomi asinchroniškai, naudojant *AJAX* technologiją.

5.3.3. MVC komponentų realizacija

Remiantis *MVC* paradigma, informacinės sistemos duomenų ir operacijų valdymas bei vartotojo sąsaja realizuoti šiais komponentais:

***MVC* modeliai**

- `CustomerBean.java` – klasė, `set() get()` metodais suteikianti prieigą prie kliento objekto kintamųjų;

- `GeoBean.java` – klasė, `set() get()` metodais suteikianti prieigą prie vietovės koordinačių objekto kintamųjų;
- `MovieBean.java` – klasė, `set() get()` metodais suteikianti prieigą prie filmo objekto kintamųjų;
- `OrderBean.java` – klasė, `set() get()` metodais suteikianti prieigą prie nuomos punkto užsakymo objekto kintamųjų.

Modelio komponento (`MovieBean.java`) programinio kodo fragmentas:

```
public class MovieBean {
   //Modelio kintamųjų aprašymas
   private String title ;
   private String movieRDF;
   ...
   //Pradinių reikšmių kintamiesiems priskyrimas
   public MovieBean()
    {
       title = "";
       movieRDF = "";
       ...
    }
    //Setter ir getter metodai objekto kintamiesiems pasiekti
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getMovieRDF() {
         return movieRDF;
    }
    public void setMovieRDF(String movieRDF) {
        this.movieRDF = movieRDF;
    }
    ...
}
```

***MVC* valdikliai**

- `CustomerServlet.java` – servletas, atsakingas už *HTTP GET* kreipinių ir klientų duomenų valdymą klientų posistemyje;
- `MovieServlet.java` – servletas, atsakingas už *HTTP GET* kreipinių ir filmų duomenų valdymą filmų posistemyje;
- `OrderServlet.java` – servletas, atsakingas už *HTTP GET* kreipinių ir užsakymų duomenų valdymą užsakymų posistemyje.

***MVC* vaizdai**

- `index.jsp` – *JSP* dinaminis puslapis, skirtas nuomos punkto filmų sąrašui išvesti. Tai taip pat pradinis IS puslapis;

- `movie.jsp` – *JSP* dinaminis puslapis, skirtas informacijos apie konkretų filmą pateikimui;
- `orders.jsp` – *JSP* dinaminis puslapis, skirtas nuomos punkto užsakymų sąrašui išvesti;
- `order.jsp` – *JSP* dinaminis puslapis, skirtas informacijos apie konkretų užsakymą pateikimui;
- `clients.jsp` – *JSP* dinaminis puslapis, skirtas nuomos punkto klientų sąrašui išvesti;
- `client.jsp` – *JSP* dinaminis puslapis, skirtas informacijos apie konkretų klientą pateikimui.

Dėl pasikeitimų *Google Maps API*[23] v3 (nėra sukurtos pilnos *Java – Gmaps API wrapper* bibliotekos), IS realizavime buvo atsisakyta atskiro žemėlapių posistemio, o projektavimo dalyje numatytos funkcijos dalinai perkeltos vykdymui kliento pusėje (failas `map.js`), pasitelkiant JavaScript technologiją.

Kaip minėta 5.4.2 sk., duomenų integravimo su išoriniais *SPARQL* šaltiniais uždaviniai atliekami asinchroniškai, naudojant *AJAX* technologiją. Tam sukurtas atskiras failas (`ajax.js`), kuriame aprašytos funkcijos asinchroniniam servletų iškvietimui, atsako iš serverio valdymui, *XML* apdorojimui bei *HTML* elementų per *DOM* (*angl. Document Object Model*) sukūrimui/atnaujinimui/pašalinimui. *AJAX* kreipiniai vykdomi dviejuose posistemiuose – klientų ir filmų. Atitinkamai `client.jsp` ir `movie.jsp` dinaminių puslapių *HTML* elementai per *DOM* pakeičiami atsižvelgiant į vykdomą operaciją, t.y. inicijuojant servleto *GET* kreipinį *HTML* elemente išvedamas „progreso simbolis“, kuris, gavus iš serverio atsaką ir apdorojus duomenis panaikinamas, o *HTML* elementas užpildomas prasminga informacija.

[23] https://developers.google.com/maps/documentation/

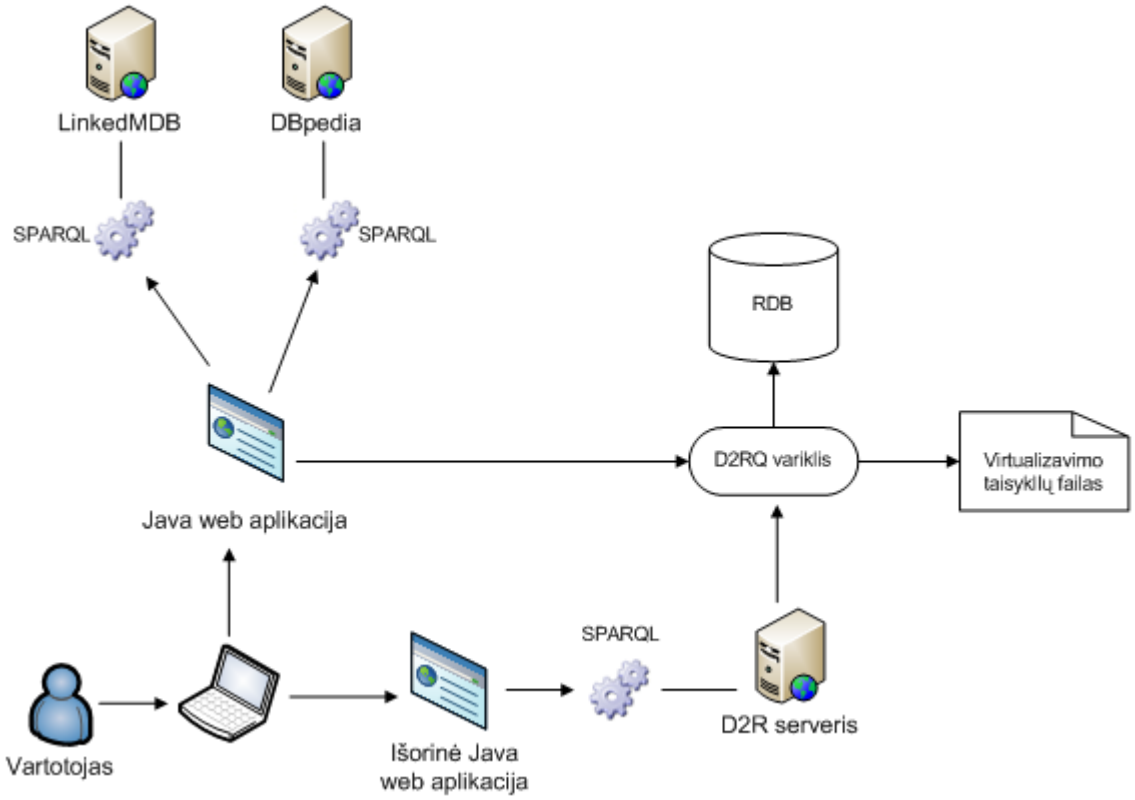
6. Eksperimentinis sprendimo tyrimas

Eksperimentinis sprendimo tyrimas atliekamas trimis etapais. Iš pradžių pagal sukurtos *D2RQ* praktinio taikymo metodikos I-ąją dalį sugeneruotos ir modifikuotos virtualizavimo taisyklės ištestuojamos *D2RQ* platformos teikiamais įrankiais. Tokiu būdu įsitikinama, ar metodikos žingsniai yra teisingi, ir, ar gaunamas siekiamas rezultatas.

Antrajame etape eksperimentiškai įvertinamas pagal *D2RQ* praktinio taikymo metodikos II-ąją dalį sukurto programinio IS prototipo panaudos atvejų išpildymas. Tokiu būdu įsitikinama, ar teisingai realizuoti IS kelti reikalavimai bei koks tokios sistemos pranašumas kitų sistemų atžvilgiu.

Trečiajame etape sukuriama papildoma *JAVA* web aplikacija, parodanti galimą sprendimo panaudojimą išorinėse semantinės paieškos IS.

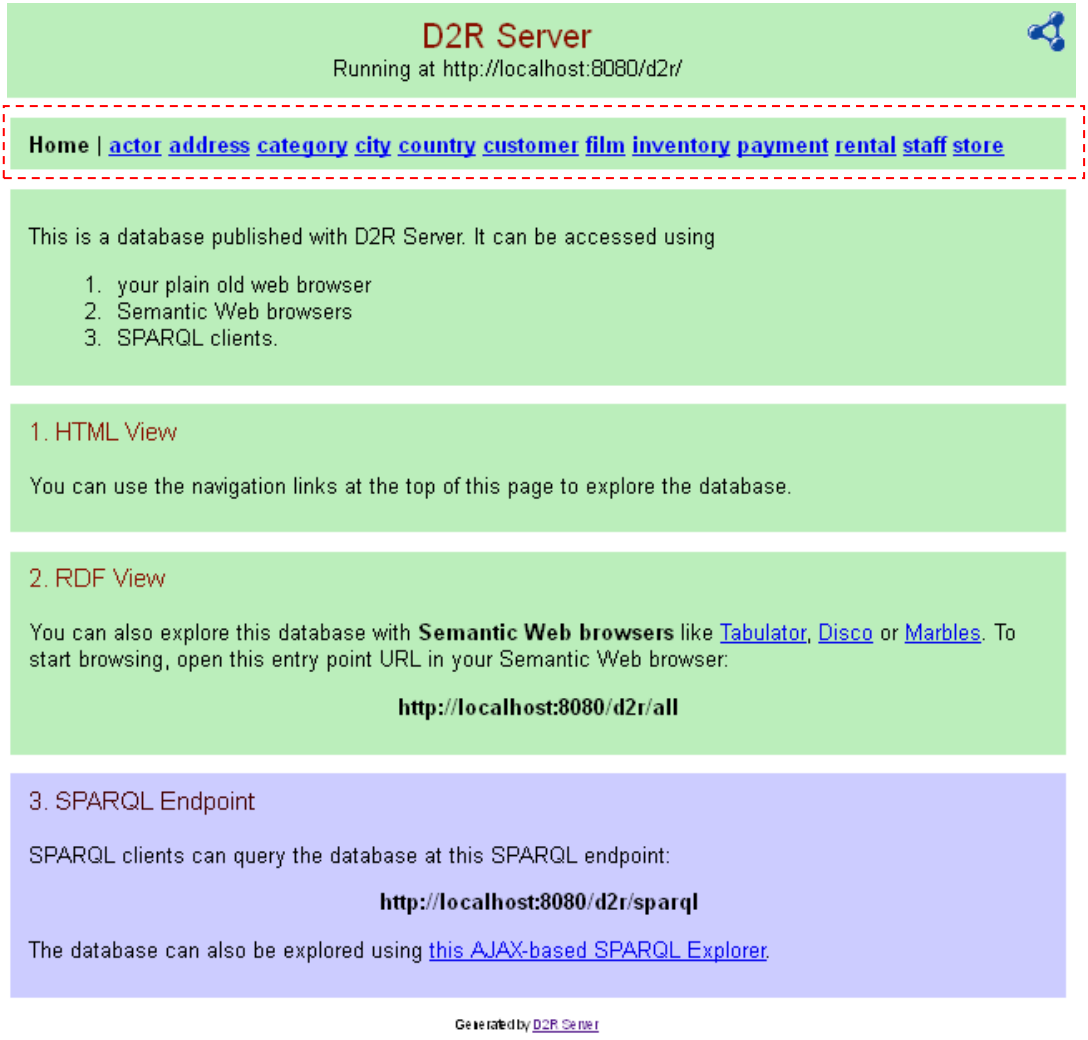
Eksperimento metu sukuriami bei naudojami artefaktai matomi žemiau pateiktoje kontekstinėje schemoje (*žr. 37 pav*.).



37 pav. Eksperimento kontekstinė schema

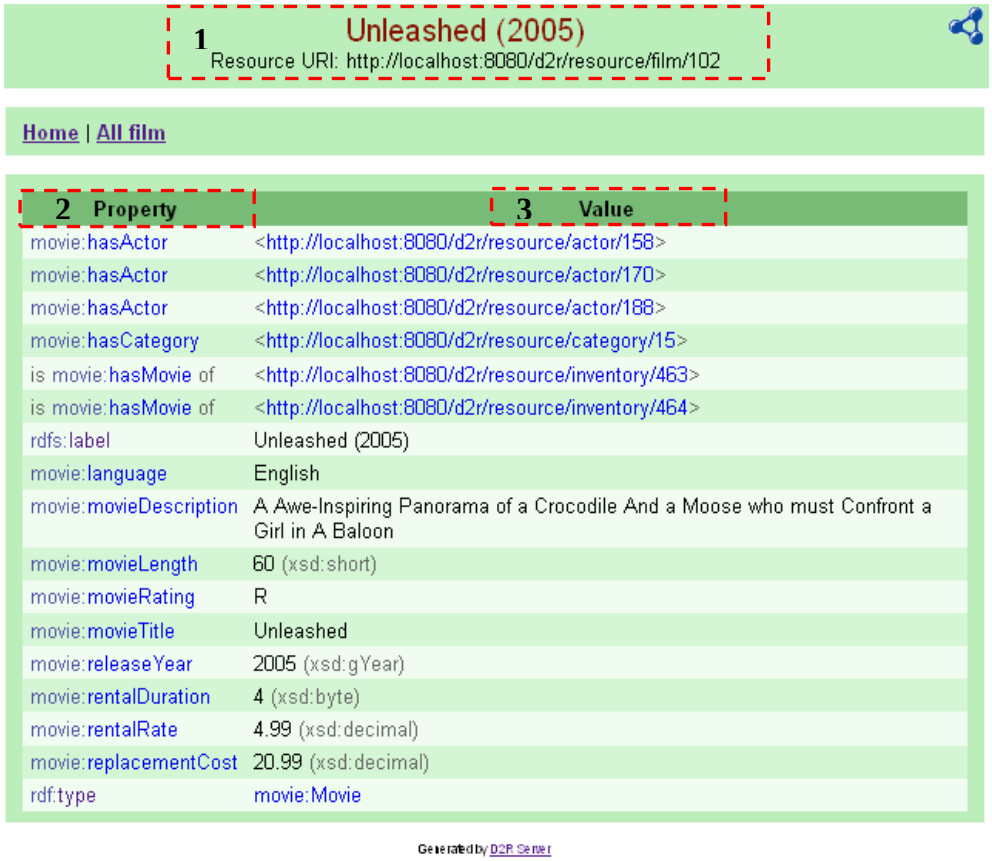
6.1. I Eksperimento etapas

Šiame etape demonstruojamas *D2RQ* platformos sudedamosios dalies, *D2R* serverio, panaudojimas testuojant RDB virtualizavimo taisykles. *D2R* serveris pasiekiamas adresu `http://localhost:2020/` (nutylėtasis adresas) arba nurodytu konfigūravimo bloke (jeigu serveris jau perkeltas į Tomcat aplinką). Atsidarius šį adresą naršyklėje, matomas pradinis *D2R* serverio puslapis (žr. 38 pav.).



38 pav. Pradinis D2R serverio puslapis

Testuojant RDB virtualizavimo taisykles, aktualiausia šiuo atveju yra sugeneruoti *RDF* trejetai, kuriuos *HTML* formatu galima peržiūrėti pasirinkus vieną norimą resursą iš pateiktųjų sąrašo viršutinėje lango eilutėje (*žr. 39 pav. rėmelis*).



39 pav. RDF resursas aprašytas HTML formatu

Pavyzdinis *RDF* resurso aprašas matomas 39 pav. Aprašą sudaro:

1. *RDF* subjektas – resurso *URI* adresas.
2. *RDF* predikatai – ontologijų/*RDFS* žodynų savybės.
3. *RDF* objektai – RDB lentelės stulpelių įrašai bei *URI* nuorodos į kitus *RDF* resursus.

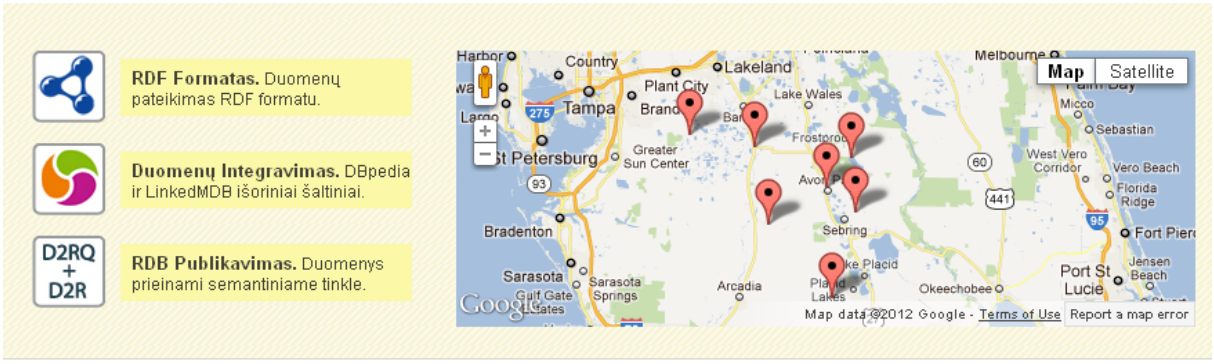
Analizuojant *RDF* resurso *HTML* aprašą, galima aiškiai matyti, ar virtualizavimo taisyklės veikia taip, kaip numatyta, t.y. ar teisingai sukuriamas resurso *URI*, ar teisingai RDB lentelės atributams priskiriamos ontologijų/*RDF* žodynų savybės, ar gaunami visi taisyklėse numatyti *RDF* trejetai ir pan. Šiame pavyzdyje matomas ir tarpinės RDB M:N ryšio lentelės panaikinimo rezultatas – predikatas `movie:hasActor` susietas tiesiogiai su aktoriaus resurso *URI* (*RDF* objektais).

Išanalizavus kitų *RDF* resursų *HTML* aprašus, galima daryti išvadą, kad pagal sudarytą *D2RQ* praktinio taikymo metodiką RDB duomenys virtualizuojami korektiškai.

6.2. II Eksperimento etapas

Įsitikinus, kad RDB virtualizavimo taisyklės veikia taip, kaip numatyta, antrajame eksperimento etape buvo pereita prie sukurtos IS savybių tyrimo. Siekiant pademonstruoti duomenų integravimo su išoriniais *SPARQL* šaltiniais galimybes, RDB „Sakila“ atliktas dalinis duomenų migravimas iš *LinkedMDB* šaltinio, užpildant lentelę „Movie“ realių filmų duomenimis (*žr. 9.3 priedas*). Šis žingsnis buvo reikalingas todėl, kad mokomoji RDB „Sakila“ paremta tik fiktyviais dalykinės srities duomenimis, kurių tolimesnio panaudojimo galimybės yra itin ribotos.

Pagal *D2RQ* praktinio taikymo metodikos II dalį realizuota IS paleidžiama *Tomcat* serveryje kartu su *D2R* serveriu. Naršyklės lange užkraunamas pradinis sistemos puslapis, kuriame pateikiama žemėlapyje pasirinkto nuomos punkto filmų pasiūla.



Kaip naudotis IS?

- Žemėlapyje pasirinkite nuomos punktą
- Iš sąrašo pasirinkite dominantį filmą
- Peržiūrėkite informaciją apie filmą HTML/RDF formatais
- Gaukite papildomą informaciją apie filmą, paspaudę "Noriu žinoti daugiau!"

Pavadinimas	Leidimo metai	Kalba	Nuomos kaina
City of God	2002	English	0.99
Mad Max 4: Fury Road	2009	English	2.99
Halo	2008	English	2.99
Elizabeth: The Golden Age	2007	English	1.24
Resident Evil: Extinction	2007	English	0.99
Transformers	2007	English	2.99
Next	2007	English	0.99
Ghost Rider	2007	English	0.99
Alice	2007	English	0.99
Delgo	2007	English	4.99

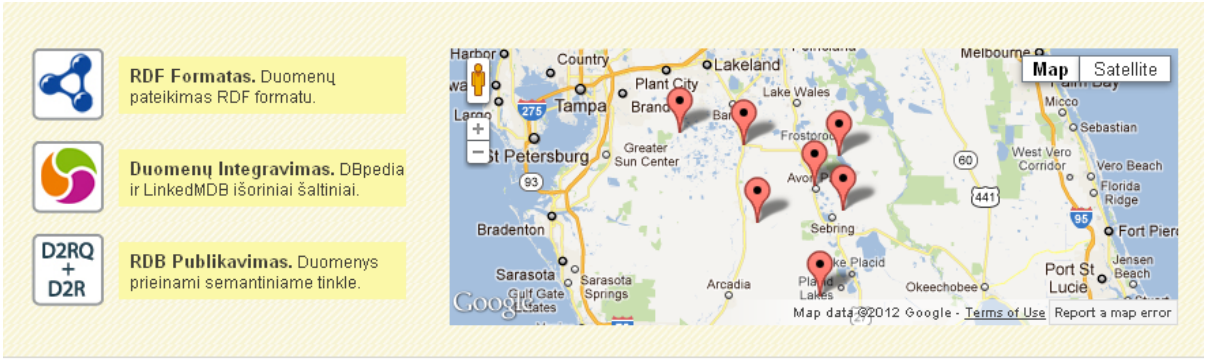
1 2 3 4 5 6 7 8 9 10 >

40 pav. Pradinis IS puslapis

Iš sąrašo pasirinkus dominantį filmą, užkraunamas puslapis su detaliu to filmo aprašu (žr. 41 pav.). Puslapyje suteikiama galimybė peržiūrėti aprašymą *HTML* bei *RDF* formatais.

Paieškos voriukams aprašas *RDF* formatu pateikiamas kaip alternatyvi puslapio versija[24] *HTML* `<head>` elemente:

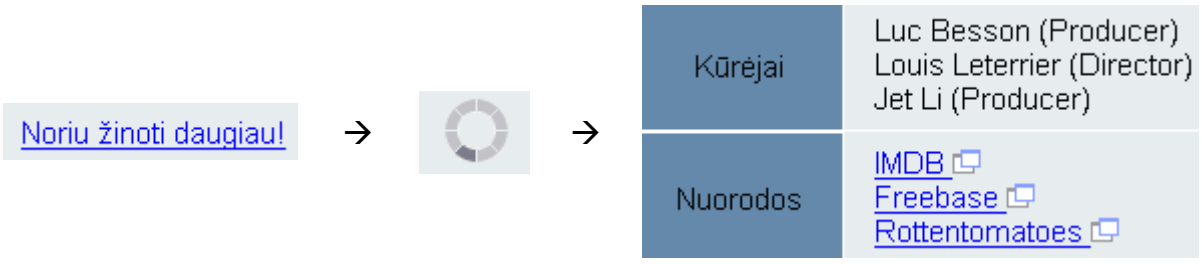
```
<link title="RDF version" type="application/rdf+xml" rel="alternate"
href="./MovieServlet?movieID=film/102&amp;RDF=true"/>
```



Pavadinimas	Unleashed
Aprašymas	A Awe-Inspiring Panorama of a Crocodile And a Moose who must Confront a Girl in A Baloon
Žanras	Sports
Leidimo metai	2005
Kalba	English
Aktoriai	VIVIEN BASINGER MENA HOPPER ROCK DUKAKIS
Noriu žinoti daugiau!	

41 pav. Konkretaus filmo aprašo puslapis

Duomenų integravimo proceso principinė schema pateikta 42 pav. Iš išorinio *LinkedMDB* šaltinio „ištraukiami“ duomenys apie filmo kūrybinę grupę bei nuorodos su papildoma informacija apie filmą.



42 pav. Duomenų integravimo procesas kliento pusėje

[24] Magistrinio darbo rašymo metu nėra priimto bendro standarto RDF duomenų pateikimui HTML puslapiuose. Metodas suformuotas apjungiant HTML (http://www.w3.org/TR/html401/struct/links.html#h-12.3.3) ir RDF (http://www.w3.org/TR/REC-rdf-syntax/#section-MIME-Type) specifikacijas.

IS duomenų integravimo funkcionalumas atspindi semantinio tinklo naudą eilinio interneto vartotojo atžvilgiu. Viešai prieinamų *SPARQL* prieigos taškų dėka, papildomos informacijos išgavimas supaprastėja iki vienintelio žingsnio – mygtuko/nuorodos paspaudimo. Įprastiniais atvejais, vartotojui tektų pačiam ieškoti papildomos informacijos paieškos sistemose ar kituose tinklo resursuose, kas neretai užtrunka nemažai laiko.

Paspaudus 41 pav. matomą *RDF* ikoną, atveriama dinamiškai suformuota nuoroda, kurioje pateikiamas filmo aprašas *RDF* formatu (*žr. 43 pav.*).

```
− <rdf:RDF>
  − <rdf:Description rdf:about="http://localhost:8080/d2r/resource/film/102">
      <movie:language>English</movie:language>
      <movie:movieLength rdf:datatype="http://www.w3.org/2001/XMLSchema#short">60</movie:movieLength>
      <movie:hasActor rdf:resource="http://localhost:8080/d2r/resource/actor/170"/>
      <movie:hasCategory rdf:resource="http://localhost:8080/d2r/resource/category/15"/>
      <movie:releaseYear rdf:datatype="http://www.w3.org/2001/XMLSchema#gYear">2005</movie:releaseYear>
      <movie:hasActor rdf:resource="http://localhost:8080/d2r/resource/actor/188"/>
      <movie:rentalRate rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">4.99</movie:rentalRate>
      <movie:hasActor rdf:resource="http://localhost:8080/d2r/resource/actor/158"/>
      <rdfs:label>Unleashed (2005)</rdfs:label>
      <movie:movieTitle>Unleashed</movie:movieTitle>
      <movie:movieRating>R</movie:movieRating>
    − <movie:movieDescription>
        A Awe-Inspiring Panorama of a Crocodile And a Moose who must Confront a Girl in A Baloon
      </movie:movieDescription>
      <movie:replacementCost rdf:datatype="http://www.w3.org
      /2001/XMLSchema#decimal">20.99</movie:replacementCost>
      <rdf:type rdf:resource="http://localhost:8080/d2r/movieRental.owl#Movie"/>
      <movie:rentalDuration rdf:datatype="http://www.w3.org/2001/XMLSchema#byte">4</movie:rentalDuration>
    </rdf:Description>
</rdf:RDF>
```

43 pav. Filmo aprašas RDF formatu

RDF trejetų objektai, išreikšti kaip *RDF* resursai, *D2R* serverio dėka yra pasiekiami tinkle *URI* adresais, lygiai taip pat kaip ir *RDF* trejetų subjektai.

Kitų IS posistemių veikimo principas analogiškas apžvelgtajam. Vienintelis ryškesnis skirtumas yra duomenų integravimas klientų valdymo posistemyje (*žr. 44 pav.*).

44 pav. Duomenų integravimo procesas klientų posistemyje

Šiuo atveju iš išorinio *DBpedia* šaltinio „ištraukiamos“ miesto, kuriame gyvena klientas koordinatės, taip dar kartą pademonstruojant galimą *SPARQL* prieigos taškų panaudojimą semantinėse IS.

Išbandžius visų IS posistemių funkcionalumą, galima daryti išvadą, kad pagal antrąją *D2RQ* praktinio taikymo metodikos dalį sukurta sistema pilnai tenkina jai keltus reikalavimus. Semantinių technologijų dėka, IS suteikia vartotojui reikiamą informaciją vienoje vietoje, tokiu būdu išvengiant papildomo informacijos ieškojimo. IS taip pat prisideda prie semantinio tinklo plėtros, programiniams agentams/paieškos voriukams pateikdama duomenis *RDF* formatu.

6.3. III Eksperimento etapas

Pirmuosiuose dviejuose eksperimento etapuose apžvelgiamos *D2RQ* platformos praktinio taikymo metodikos panaudojimo galimybės kuriant IS, kuomet turime tiesioginę prieigą prie virtualizuojamos RDB. Tačiau semantinio tinklo idėja yra atverti RDB turinį išoriniams šaltiniams, neturintiems tiesioginės prieigos prie jos. Šioje eksperimento dalyje demonstruojamas galimas *D2R* serverio sukurto *SPARQL* prieigos taško panaudojimas semantinėse informacijos paieškos sistemose. Tam realizuota demonstracinė Java web aplikacija[25], skirta nurodyto filmo nuomos statistikai pateikti.

45 pav. Eksperimentinės IS pagrindinis puslapis

Pagrindinis IS puslapis atveriamas naršyklės lange surinkus adresą `http://localhost:8080/NuomaMVC/eksperimentas.jsp`. Jame pateikiama forma filmo pavadinimui bei leidimo metams įvesti. Pagal šiuos užduotus kriterijus, *SPARQL*

[25] Web aplikacija sukurta filmų nuomos IS pagrindu, išskiriant papildomus MVC komponentus.

prieigos taško (`http://localhost:8080/d2r/sparql`) atžvilgiu programiškai įvykdoma *SPARQL* užklausa:

```
SELECT ?movie ?store ?copyCount ?rentalCount ?rentalRate WHERE {
  {
    SELECT ?store ?movie (COUNT(?movie) AS ?rentalCount)
    WHERE {
  ?rental movie:fromInventory ?invURI.
  ?invURI movie:hasMovie ?movieURI.
  ?invURI movie:belongsToStore ?storeURI.
  ?storeURI rdfs:label ?store.
  ?movieURI rdfs:label ?movie.
  ?movieURI movie:movieTitle ?title;
            movie:releaseYear ?year.
  FILTER ((?title="Unleashed") && (?year="2005"))
    }
  GROUP BY ?movie ?rentalCount ?store
  ORDER BY DESC(?rentalCount)
  }
  { SELECT (COUNT(?movie) AS ?copyCount) ?store ?rentalRate
    WHERE {
  ?invURI movie:hasMovie ?movieURI.
  ?invURI movie:belongsToStore ?storeURI.
  ?storeURI rdfs:label ?store.
  ?movieURI rdfs:label ?movie.
  ?movieURI movie:movieTitle ?title;
            movie:releaseYear ?year;
            movie:rentalRate ?rentalRate.
  FILTER ((?title="Unleashed") && (?year="2005"))
  }
   GROUP BY ?copyCount ?store ?rentalRate
  }
 }
```

Užklausos vykdymo rezultatai pateikiami 46 pav.

46 pav. Semantinės paieškos rezultatai

Šiuo atveju *SPARQL* užklausa įvykdoma vieno šaltinio, virtualizuotos RDB „Sakila“ atžvilgiu. Rezultate pateikiama informacija, nusakanti kur galima nurodytą filmą išsinuomoti, kiek kartų jis buvo išnuomotas konkrečiame nuomos punkte, turimų filmo kopijų kiekis bei nuomos kaina.

Viešai preinamas RDB turinys gali būti naudojamas daugelio taikomųjų uždavinių formavimui bei sprendimui. Žemiau pateikti pora *SPARQL* užklausų pavyzdžių, įvykdomų *D2R* serverio užklausų vykdymo lange (`http://localhost:8080/d2r/snorql/`):

Floridos valstijoje nuomotų filmų TOP 5

```
SELECT ?movie (COUNT(?movie) AS ?rentalCount) WHERE {
  ?rental movie:rentedBy ?clientURI.
  ?clientURI movie:hasAddress ?addressURI.
  ?addressURI vcard:Locality "Florida".
  ?rental movie:rentedBy ?clientURI.
  ?rental movie:fromInventory ?invURI.
  ?invURI movie:hasMovie ?movieURI.
  ?movieURI rdfs:label ?movie.
}
GROUP BY ?rentalCount ?movie
ORDER BY DESC(?rentalCount)
LIMIT 5
```

Užklausos vykdymo rezultatai:

SPARQL results:

movie	rentalCount
"LEGO Star Wars: Revenge of the Brick (2005)"	33
"The X Files (1998)"	33
"54 (1998)"	32
"Harry Potter and the Chamber of Secrets (2002)"	32
"Darkness Falls (2003)"	31

47 pav. *SPARQL* užklausos rezultatai (I)

Filmų TOP 5 pagal nurodytą nuomos punktą

```
SELECT ?movie (COUNT(?movie) AS ?rentalCount) WHERE {
 ?inventory movie:belongsToStore
            <http://localhost:8080/d2r/resource/store/4>.
 ?rental movie:fromInventory ?inventory.
 ?inventory movie:hasMovie ?movieURI.
 ?movieURI rdfs:label ?movie.
}
GROUP BY ?rentalCount ?movie
ORDER BY DESC(?rentalCount)
LIMIT 5
```

Užklausos vykdymo rezultatai:

SPARQL results:

movie	rentalCount
"The Treatment (2000)"	19
"The Funeral (1996)"	17
"Booty Call (1997)"	15
"Ginger Snaps (2000)"	15
"Master and Commander: The Far Side of the World (2003)"	15

48 pav. SPARQL užklausos rezultatai (II)

Plečiantis semantiniam tinklui bei didėjant *SPARQL* prieigos taškų skaičiui, semantinė paieška gali būti praplečiama panaudojant *Federated* užklausas kelių išorinių šaltinių atžvilgiu. Pavyzdžiui., šioje eksperimento dalyje sukurtos IS funkcionalumas gali būti praplečiamas iki skirtingų filmų nuomos punktų duomenų agregavimo variklio. Tam tereiktų virtualizuoti skirtingų punktų RDB duomenis ir parašyti *SPARQL* užklausas, kurios būtų vykdomos visų jų atžvilgiu.

6.4. Eksperimento išvados

Atliktas eksperimentas parodė, kad sudaryta praktinė *D2RQ* įrankio taikymo metodika yra tinkama tiek RDB duomenų virtualizavimui, tiek semantinėms IS kurti. Iš virtualizuotos RDB gauti tokie *RDF* trejetai, kokių tikėtasi rašant virtualizavimo taisykles. Sukurta IS pilnai atitinka sistemai keltus reikalavimus, kurie išryškina semantinio tinklo naudą eiliniam interneto vartotojui – integruotos informacijos pateikimą vienoje vietoje.

Metodiniais tikslais buvo sėkmingai virtualizuoti visi RDB esantys duomenys, tačiau, kaip parodė eksperimentas, prieš publikuojant RDB duomenis semantiniame tinkle, būtina įvertinti jų jautrumą privatumo bei konfidencialumo klausimais. Net jei ir pavieniai duomenys iš pirmo žvilgsnio atrodo saugūs bei tinkami publikavimui pasauliniame semantiniame tinkle, vertėtų nepamiršti, kad *SPARQL* užklausos suteikia galimybę juos analizuoti įvairiais pjūviais. O tai gali sąlygoti ne tik privatumo pažeidimo, bet ir komercinių paslapčių nutekėjimo atvejus.

7. Išvados

1. Išanalizavus semantinio tinklo ir RDB turinio publikavimo semantiniame tinkle technologijas, nustatyta, kad daugelio *RDB2RDF* įrankių praktinio taikymo principai yra migloti, o galimybės iki galo neištirtos. Lyginamosios įrankių analizės metu nustatyta, kad plačiausias teorines panaudojimo galimybes turi *D2RQ* platforma. Atlikus išsamų šio įrankio galimybių tyrimą, sukurta *D2RQ* platformos praktinio taikymo metodika. Metodika sudaryta iš dviejų sudedamųjų dalių: pirmoji nusako RDB turinio virtualizavimo taisyklių kūrimo žingsnius, o antroji - *D2RQ* platformos panaudojimo taikomuosiuose IS sprendimuose principus. Sukūrus įrankio praktinio taikymo metodiką, pasiektas pagrindinis darbo tikslas - RDB turinio publikavimo pasauliniame semantiniame tinkle procesui suteikta daugiau aiškumo ir nuoseklumo.
2. Atliktas eksperimentas parodė, kad sukurtoji metodika yra veiksminga, nes pavyko sėkmingai virtualizuoti ir išpublikuoti mokomosios reliacinės duomenų bazės duomenis *RDF* formatu.
3. Pasiūlytas architektūrinis sprendimas *D2RQ* platformos panaudojimui semantinėse IS leidžia užtikrinti efektyvų sistemos palaikymą ir lankstumą sparčiai keičiantis funkciniams reikalavimams, nes jame panaudota *MVC* paradigma. Sprendimas buvo patikrintas sukuriant IS prototipą bei sėkmingai išpildant visus jam keltus reikalavimus.
4. Realizuojant demonstracinę IS, praktiškai iliustruota semantinio tinklo teikiama nauda eilinio interneto vartotojo atžvilgiu – ženkliai sumažėjęs žingsnių kiekis, reikalingas norimai papildomai informacijai pasiekti.
5. Sukurtoji metodika prisideda prie pagrindinių vartotojų aibės problemų sprendimo: RDB savininkams palengvinamas duomenų išpublikavimo uždavinys, semantinio tinklo technologijų kūrėjams suteikiama prieiga prie didesnio *RDF* duomenų kiekio, o pastarųjų naudojimas semantinėse IS sąlygoja greitesnį ir efektyvesnį struktūrizuotos informacijos pateikimą interneto vartotojams.

8. Literatūra

[1] Resource Description Framework (RDF):
Concepts and Abstract Syntax. [žiūrėta 2010.11.06] Prieiga per internetą <http://www.w3.org/TR/rdf-concepts/>.

[2] Cool URIs for the Semantic Web. W3C Interest Group Note 03 December 08. [žiūrėta 2010.11.28] Prieiga per internetą <http://www.w3.org/TR/cooluris/>;

[3] OWL Web Ontology Language. W3C Recommendation 10 February 2004. [žiūrėta 2010.09.28] Prieiga per internetą <http://www.w3.org/TR/owl-features/>;

[4] SPARQL Query Language for RDF. W3C Recommendation 15 January 2008. [žiūrėta 2010.09.28] Prieiga per internetą <http://www.w3.org/TR/rdf-sparql-query/>;

[5] Relational Database and the SemanticWeb. [žiūrėta 2010.10.20] Prieiga per internetą <http://semanticweb.com/relational-database-and-the-semantic-web_b16083>;

[6] The D2RQ Platform v0.7 - *Treating Non-RDF Relational Databases as Virtual RDF Graphs. User Manual and Language Specification*. Prieiga per internetą <http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/>;

[7] D2R Server - *Publishing Relational Databases on the Semantic Web*. [žiūrėta 2010.10.01] Prieiga per internetą <http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/>;

[8] SquirrelRDF. [žiūrėta 2010.10.02] Prieiga per internetą <http://jena.sourceforge.net/SquirrelRDF/>;

[9] Mapping Relational Data to RDF in Virtuoso. [žiūrėta 2010.10.02] Prieiga per internetą <http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQLRDF>;

[10] Triplify. Expose Semantics. [žiūrėta 2010.10.02] Prieiga per internetą <http://triplify.org/Overview>;

[11] Triplify – Light-Weight Linked Data Publication from Relational Databases. [žiūrėta 2010.11.20] Prieiga per internetą <http://www.informatik.uni-leipzig.de/~auer/publication/triplify.pdf>

[12] W3C Incubator Group. A Survey of Current Approaches for Mapping of Relational Databases to RDF. [žiūrėta 2010.10.15] Prieiga per internetą <http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf>.

9. Priedas

9.1. *D2RQ* virtualaus atvaizdavimo taisyklių failo turinys

```
@prefix map: <file:/C:/d2r/taisykles.n3#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix movie: <http://localhost:8080/d2r/movieRental.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix d2r: <http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-
server/config.rdf#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .

<> a d2r:Server;
    rdfs:label "D2R Server";
    d2r:baseURI <http://localhost:8080/d2r/>;
    d2r:port 8080;
    .

map:database a d2rq:Database;
      d2rq:jdbcDriver "com.mysql.jdbc.Driver";
      d2rq:jdbcDSN "jdbc:mysql://localhost/sakila";
      d2rq:username "root";
      d2rq:password "pass";
      jdbc:autoReconnect "true";
      jdbc:zeroDateTimeBehavior "convertToNull";
      .


# Table actor
map:actor a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "actor/@@actor.actor_id@@";
      d2rq:class movie:Actor;
      d2rq:classDefinitionLabel "actor";
      .
map:actor__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:actor;
      d2rq:property rdfs:label;
      d2rq:pattern "@@actor.first_name@@ @@actor.last_name@@";
      .

map:actor_first_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:actor;
      d2rq:property foaf:firstName;
      d2rq:propertyDefinitionLabel "actor first_name";
      d2rq:column "actor.first_name";
      .
map:actor_last_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:actor;
      d2rq:property foaf:lastName;
      d2rq:propertyDefinitionLabel "actor last_name";
      d2rq:column "actor.last_name";
      .

# Table address
```

```
map:address a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "address/@@address.address_id@@";
      d2rq:class movie:Address;
      d2rq:classDefinitionLabel "address";
      .
map:address__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property rdfs:label;
      d2rq:pattern "@@address.address@@";
      .
map:address_address a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property vcard:ADR;
      d2rq:propertyDefinitionLabel "address address";
      d2rq:column "address.address";
      .
map:address_address2 a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property vcard:ADR;
      d2rq:propertyDefinitionLabel "address address2";
      d2rq:column "address.address2";
      d2rq:condition "address.address2 <> ''";
      .
map:address_district a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property vcard:Locality;
      d2rq:propertyDefinitionLabel "address district";
      d2rq:column "address.district";
      .
map:address_postal_code a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property vcard:Pcode;
      d2rq:propertyDefinitionLabel "address postal_code";
      d2rq:column "address.postal_code";
      d2rq:condition "address.postal_code <> ''";
      .
map:address_phone a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property foaf:phone;
      d2rq:propertyDefinitionLabel "address phone";
      d2rq:column "address.phone";
      d2rq:condition "address.phone <> ''";
      .
map:address_city_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:address;
      d2rq:property movie:inCity;
      d2rq:refersToClassMap map:city;
      d2rq:join "address.city_id => city.city_id";
      .

# Table category
map:category a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "category/@@category.category_id@@";
      d2rq:class movie:Category;
      d2rq:classDefinitionLabel "category";
      .
map:category__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:category;
      d2rq:property rdfs:label;
```

```
      d2rq:pattern "@@category.name@@";
      .
map:category_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:category;
      d2rq:property movie:categoryName;
      d2rq:propertyDefinitionLabel "category name";
      d2rq:column "category.name";
      .

# Table city
map:city a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "city/@@city.city_id@@";
      d2rq:class movie:city;
      d2rq:classDefinitionLabel "city";
      .
map:city__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:city;
      d2rq:property rdfs:label;
      d2rq:pattern "@@city.city@@";
      .
map:city_country_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:city;
      d2rq:property movie:inCountry;
      d2rq:refersToClassMap map:country;
      d2rq:join "city.country_id => country.country_id";
      .

# Table country
map:country a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "country/@@country.country_id@@";
      d2rq:class vcard:Country;
      d2rq:classDefinitionLabel "country";
      .
map:country__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:country;
      d2rq:property rdfs:label;
      d2rq:pattern "@@country.country@@";
      .

# Table customer
map:customer a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "customer/@@customer.customer_id@@";
      d2rq:class movie:Customer;
      d2rq:condition "customer.active = true";
      d2rq:classDefinitionLabel "customer";
      .
map:customer__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:customer;
      d2rq:property rdfs:label;
      d2rq:pattern "@@customer.first_name@@ @@customer.last_name@@";
      .
map:customer_first_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:customer;
      d2rq:property foaf:firstName;
      d2rq:propertyDefinitionLabel "customer first_name";
      d2rq:column "customer.first_name";
      .
map:customer_last_name a d2rq:PropertyBridge;
```

```
      d2rq:belongsToClassMap map:customer;
      d2rq:property foaf:lastName;
      d2rq:propertyDefinitionLabel "customer last_name";
      d2rq:column "customer.last_name";
      .
map:customer_email a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:customer;
      d2rq:property movie:hasEmail;
      d2rq:uriPattern "mailto:@@customer.email@@";
      d2rq:propertyDefinitionLabel "customer email";
      .
map:customer_create_date a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:customer;
      d2rq:property movie:memberSince;
      d2rq:propertyDefinitionLabel "customer create_date";
      d2rq:column "customer.create_date";
      d2rq:datatype xsd:dateTime;
      .
map:customer_store_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:customer;
      d2rq:property movie:custStore;
      d2rq:refersToClassMap map:store;
      d2rq:join "customer.store_id => store.store_id";
      .
map:customer_address_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:customer;
      d2rq:property movie:hasAddress;
      d2rq:refersToClassMap map:address;
      d2rq:join "customer.address_id => address.address_id";
      .



# Table film
map:film a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "film/@@film.film_id@@";
      d2rq:class movie:Movie;
      d2rq:classDefinitionLabel "film";
      .
map:film__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property rdfs:label;
      d2rq:pattern "@@film.title@@ (@@film.release_year@@)";
      .

map:film_title a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:movieTitle;
      d2rq:propertyDefinitionLabel "film title";
      d2rq:column "film.title";
      .
map:film_description a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:movieDescription;
      d2rq:propertyDefinitionLabel "film description";
      d2rq:column "film.description";
      .
map:film_release_year a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:releaseYear;
```

```
      d2rq:propertyDefinitionLabel "film release_year";
      d2rq:column "film.release_year";
      d2rq:datatype xsd:date;
      .
map:film_rental_duration a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:rentalDuration;
      d2rq:propertyDefinitionLabel "film rental_duration";
      d2rq:column "film.rental_duration";
      d2rq:datatype xsd:byte;
      .
map:film_rental_rate a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:rentalRate;
      d2rq:propertyDefinitionLabel "film rental_rate";
      d2rq:column "film.rental_rate";
      d2rq:datatype xsd:decimal;
      .
map:film_length a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:movieLength;
      d2rq:propertyDefinitionLabel "film length";
      d2rq:column "film.length";
      d2rq:datatype xsd:short;
      .
map:film_replacement_cost a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:replacementCost;
      d2rq:propertyDefinitionLabel "film replacement_cost";
      d2rq:column "film.replacement_cost";
      d2rq:datatype xsd:decimal;
      .
map:film_rating a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:movieRating;
      d2rq:propertyDefinitionLabel "film rating";
      d2rq:column "film.rating";
      .

map:film_original_language_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:originalLanguage;
      d2rq:column "language.name";
      d2rq:join "film.original_language_id => language.language_id";
      .
map:film_language_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:language;
      d2rq:column "language.name";
      d2rq:join "film.language_id => language.language_id";
      .
map:aktoriaus_filmai a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:hasActor;
      d2rq:refersToClassMap map:actor;
      d2rq:join "film_actor.film_id <= film.film_id";
      d2rq:join "film_actor.actor_id => actor.actor_id";
      .
map:filmo_kategorija a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:film;
      d2rq:property movie:hasCategory;
```

```
      d2rq:refersToClassMap map:category;
      d2rq:join "film_category.film_id <= film.film_id";
      d2rq:join "film_category.category_id <= category.category_id";
      .

# Table inventory
map:inventory a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "inventory/@@inventory.inventory_id@@";
      d2rq:class movie:Inventory;
      d2rq:classDefinitionLabel "inventory";
      .
map:inventory__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:inventory;
      d2rq:property rdfs:label;
      d2rq:pattern "inventory #@@inventory.inventory_id@@";
      .
map:inventory_store_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:inventory;
      d2rq:property movie:belongsToStore;
      d2rq:refersToClassMap map:store;
      d2rq:join "inventory.store_id => store.store_id";
      .
map:inventory_film_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:inventory;
      d2rq:property movie:hasMovie;
      d2rq:refersToClassMap map:film;
      d2rq:join "inventory.film_id => film.film_id";
      .


# Table payment
map:payment a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "payment/@@payment.payment_id@@";
      d2rq:class movie:Payment;
      d2rq:classDefinitionLabel "payment";
      .
map:payment__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:payment;
      d2rq:property rdfs:label;
      d2rq:pattern "payment #@@payment.payment_id@@";
      .
map:payment_amount a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:payment;
      d2rq:property movie:paymentAmount;
      d2rq:propertyDefinitionLabel "payment amount";
      d2rq:column "payment.amount";
      d2rq:datatype xsd:decimal;
      .
map:payment_payment_date a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:payment;
      d2rq:property movie:paymentDate;
      d2rq:propertyDefinitionLabel "payment payment_date";
      d2rq:column "payment.payment_date";
      d2rq:datatype xsd:dateTime;
      .
map:payment_rental_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:payment;
      d2rq:property movie:paidFor;
      d2rq:refersToClassMap map:rental;
```

```
      d2rq:join "payment.rental_id => rental.rental_id";
      .
map:payment_customer_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:payment;
      d2rq:property movie:paidBy;
      d2rq:refersToClassMap map:customer;
      d2rq:join "payment.customer_id => customer.customer_id";
      .

# Table rental
map:rental a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "rental/@@rental.rental_id@@";
      d2rq:class movie:Rental;
      d2rq:classDefinitionLabel "rental";
      .
map:rental__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:rental;
      d2rq:property rdfs:label;
      d2rq:pattern "rental #@@rental.rental_id@@";
      .
map:rental_rental_date a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:rental;
      d2rq:property movie:rentalDate;
      d2rq:propertyDefinitionLabel "rental rental_date";
      d2rq:column "rental.rental_date";
      d2rq:datatype xsd:dateTime;
      .
map:rental_return_date a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:rental;
      d2rq:property movie:returnDate;
      d2rq:propertyDefinitionLabel "rental return_date";
      d2rq:column "rental.return_date";
      d2rq:datatype xsd:dateTime;
      .
map:rental_staff_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:rental;
      d2rq:property movie:rentalStaff;
      d2rq:refersToClassMap map:staff;
      d2rq:join "rental.staff_id => staff.staff_id";
      .
map:rental_customer_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:rental;
      d2rq:property movie:rentedBy;
      d2rq:refersToClassMap map:customer;
      d2rq:join "rental.customer_id => customer.customer_id";
      .
map:rental_inventory_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:rental;
      d2rq:property movie:fromInventory;
      d2rq:refersToClassMap map:inventory;
      d2rq:join "rental.inventory_id => inventory.inventory_id";
      .

# Table staff
map:staff a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "staff/@@staff.staff_id@@";
      d2rq:class movie:Staff;
      d2rq:classDefinitionLabel "staff";
      .
```

```
map:staff__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:property rdfs:label;
      d2rq:pattern "@@staff.first_name@@ @@staff.last_name@@";
      .
map:staff_first_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:property foaf:firstName;
      d2rq:propertyDefinitionLabel "staff first_name";
      d2rq:column "staff.first_name";
      .
map:staff_last_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:property foaf:lastName;
      d2rq:propertyDefinitionLabel "staff last_name";
      d2rq:column "staff.last_name";
      .
map:staff_email a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:propertyDefinitionLabel "staff email";
      d2rq:property movie:hasEmail;
      d2rq:uriPattern "mailto:@@staff.email@@";
      .

map:staff_username a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:property movie:hasUsername;
      d2rq:propertyDefinitionLabel "staff username";
      d2rq:column "staff.username";
      .

map:staff_address_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:property movie:hasAddress;
      d2rq:refersToClassMap map:address;
      d2rq:join "staff.address_id => address.address_id";
      .
map:staff_store_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:staff;
      d2rq:property movie:worksIn;
      d2rq:refersToClassMap map:store;
      d2rq:join "staff.store_id => store.store_id";
      .

# Table store
map:store a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "store/@@store.store_id@@";
      d2rq:class movie:Store;
      d2rq:classDefinitionLabel "store";
      .
map:store__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:store;
      d2rq:property rdfs:label;
      d2rq:pattern "store #@@store.store_id@@";
      .
map:store_address_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:store;
      d2rq:property movie:hasAddress;
      d2rq:refersToClassMap map:address;
      d2rq:join "store.address_id => address.address_id";
```

9.2. Dalykinės srities *OWL* ontologijos išeities tekstas

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://localhost/movieRental#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://localhost/movieRental">
  <owl:Ontology rdf:about="">
    <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Created with TopBraid Composer</owl:versionInfo>
  </owl:Ontology>
  <owl:Class rdf:ID="Movie"/>
  <owl:Class rdf:ID="Country"/>
  <owl:Class rdf:ID="ActorInMovie"/>
  <owl:Class rdf:ID="Address"/>
  <owl:Class rdf:ID="Category"/>
  <owl:Class rdf:ID="Store"/>
  <owl:Class rdf:ID="Customer"/>
  <owl:Class rdf:ID="Rental"/>
  <owl:Class rdf:ID="Inventory"/>
  <owl:Class rdf:ID="Actor"/>
  <owl:Class rdf:ID="City"/>
  <owl:Class rdf:ID="Staff"/>
  <owl:Class rdf:ID="Payment"/>
  <owl:ObjectProperty rdf:ID="hasMovie">
    <rdfs:domain rdf:resource="#Inventory"/>
    <rdfs:range rdf:resource="#Movie"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasRoles">
    <rdfs:range rdf:resource="#ActorInMovie"/>
    <rdfs:domain rdf:resource="#Movie"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasActor">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="#Actor"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="isActorIn"/>
    </owl:inverseOf>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="paidBy">
    <rdfs:domain rdf:resource="#Payment"/>
    <rdfs:range rdf:resource="#Customer"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="actedAs">
    <rdfs:range rdf:resource="#ActorInMovie"/>
    <rdfs:domain rdf:resource="#Actor"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="rentalStaff">
    <rdfs:domain rdf:resource="#Rental"/>
    <rdfs:range rdf:resource="#Staff"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="fromInventory">
    <rdfs:domain rdf:resource="#Rental"/>
    <rdfs:range rdf:resource="#Inventory"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="belongsToStore">
    <rdfs:domain rdf:resource="#Inventory"/>
    <rdfs:range rdf:resource="#Store"/>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inCity">
  <rdfs:domain rdf:resource="#Address"/>
  <rdfs:range rdf:resource="#City"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="custStore">
  <rdfs:domain rdf:resource="#Customer"/>
  <rdfs:range rdf:resource="#Store"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inCountry">
  <rdfs:domain rdf:resource="#City"/>
  <rdfs:range rdf:resource="#Country"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="worksIn">
  <rdfs:domain rdf:resource="#Staff"/>
  <rdfs:range rdf:resource="#Store"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="rentedBy">
  <rdfs:domain rdf:resource="#Rental"/>
  <rdfs:range rdf:resource="#Customer"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasCategory">
  <rdfs:domain rdf:resource="#Movie"/>
  <rdfs:range rdf:resource="#Category"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasAddress">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Staff"/>
        <owl:Class rdf:about="#Customer"/>
        <owl:Class rdf:about="#Store"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#Address"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="paidFor">
  <rdfs:domain rdf:resource="#Payment"/>
  <rdfs:range rdf:resource="#Rental"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="categoryName">
  <rdfs:domain rdf:resource="#Category"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="playsRole">
  <rdfs:domain rdf:resource="#ActorInMovie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="paymentAmount">
  <rdfs:domain rdf:resource="#Payment"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="rentalDate">
  <rdfs:domain rdf:resource="#Rental"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="movieLength">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="originalLanguage">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasPassword">
  <rdfs:domain>
```

```
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Staff"/>
        <owl:Class rdf:about="#Customer"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="rentalDuration">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="releaseYear">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasUsername">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Staff"/>
        <owl:Class rdf:about="#Customer"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasEmail">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Staff"/>
        <owl:Class rdf:about="#Customer"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="movieRating">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="movieTitle">
  <rdfs:domain rdf:resource="#Movie"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="memberSince">
  <rdfs:domain rdf:resource="#Customer"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="paymentDate">
  <rdfs:domain rdf:resource="#Payment"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="movieDescription">
  <rdfs:domain rdf:resource="#Movie"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="replacementCost">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="returnDate">
  <rdfs:domain rdf:resource="#Rental"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="rentalRate">
  <rdfs:domain rdf:resource="#Movie"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="isActive">
```

```
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Staff"/>
          <owl:Class rdf:about="#Customer"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="language">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
</rdf:RDF>

<!-- Created with TopBraid -->
```

9.3. Informacinės sistemos išeities tekstai (*CD*)

Naudojimo instrukcija pateikta faile *Instrukcija.txt*.