UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Vidmar

Preslikava in obogatitev podatkov iz relacijskih podatkovnih baz v RDF obliko

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01980 / 2014
Datum: 30.1.2014

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JERNEJ VIDMAR**

Naslov: **PRESLIKAVA IN OBOGATITEV PODATKOV IZ RELACIJSKIH PODATKOVNIH BAZ V RDF OBLIKO**
MAPPING AND ENRICHMENT OF DATA FROM RELATIONAL DATABASES INTO RDF FORMAT

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Na področju preslikave podatkov iz relacijskih podatkovnih baz v RDF obliko je s strani organizacije W3C na voljo več priporočil, ki čakajo predvsem na praktične implementacije. S pomočje omenjene preslikave se nam odprejo številne možnosti semantične integracije obstoječih podatkov z ostalimi spletnimi viri in na takšen način obogatitev vsebine lokalnega repozitorija. V okviru diplomske naloge naj študent pripravi prototip in praktično uporabo konceptov, ki jih predlaga W3C v okviru svojih priporočil. Potrebno je predstaviti obe smeri preslikave in prednosti, ki jih s tem pridobimo. Študent naj na izbrani problemski domeni prikaže scenarij, kjer lokalna podatkovna baza predstavlja interne transakcijske podatke podjetja, nato pa se pojavi potreba po semantični integraciji. Najprej želimo omenjene podatke osamodejno obogatiti in po drugi strani ponuditi dostop do naših internih podatkov v strukturirani RDF obliki, kjer so le-ti opisani s shemo, ki je povezana v Linked Data in omogoča nadaljnja sklepanja.

Mentor:
doc. dr. Dejan Lavbič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Jernej Vidmar,

z vpisno številko 63070135,

sem avtor diplomskega dela z naslovom:

Preslikava in obogatitev podatkov iz relacijskih podatkovnih baz v RDF obliko

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  doc. dr. Dejana Lavbiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 31. 1. 2014 Podpis avtorja/-ice:

Rad bi se zahvalil svojemu mentorju doc. dr. Dejanu Lavbiču za strokovno svetovanje, potrpežljivost in spodbudo pri nastajanju diplomskega dela. Iskrena hvala tudi moji družini in prijateljem, ki ste me podpirali in mi stali ob strani skozi vsa leta študija.

Kazalo

Slike

Povzetek

Svetovni splet je del našega vsakdanjika, uporabljamo ga za iskanje informacij, nakupovanje, komuniciranje in še bi lahko naštevali. Njegov poudarek je na predstavitvi in ne interpretaciji podatkov, ki jo mora opraviti uporabnik sam. Zato je postala samodejna interpretacija s strani programa oziroma računalnika pomembna zahteva. Z razvojem semantičnega spleta, ki bo nadgradil obstoječi splet, bomo tako pridobili novo dimenzijo in to je razumevanje podatkov s strani naprav, kar bo uporabnikom poenostavilo samo uporabo spleta. V ta namen že obstaja podatkovni model RDF, ki omogoča izmenjavo informacij v obliki, ki je razumljiva napravam. Vseeno obstaja še kar nekaj preprek, med katerimi je tudi preslikava podatkov iz relacijske baze v RDF. Tako je bil cilj diplomske naloge predstaviti enega izmed možnih pristopov, s katerim lahko dosežemo želeno preslikavo. Razvili smo prototip, ki je izpostavil podatke relacijske baze v obliki RDF ter jih obogatil s podatki iz drugih prosto dostopnih zbirk. Implementiran prototip predstavlja osebno spletno knjižnico, s katero lahko uporabnik ureja svojo knjižno zbirko. V okviru diplomske naloge smo tudi preučili obstoječe rešitve na tem področju ter predstavili uporabljene tehnologije pri implementaciji prototipa.

Ključne besede:

semantičen splet, RDF, SPARQL, D2RQ, Fuseki, ontologija, Protégé, preslikava podatkov, relacijska baza, prosto dostopne zbirke

Abstract

The use of world wide web is an important part of our lives. We use it to communicate, search for informations, online shopping and more. Its focus is on presentation and not interpretation of data, which has to be carried out by the user. Consequently, the automatic interpretation by a program or computer has become a very important requirement. With development of semantic web, which will only upgrade the world wide web, the requirement will be fulfilled and we will gain a new dimension of functionality and simplicity for users. For this purpose there already exists a standard model called RDF, which enables data interchange in a form that is understandable to machines. Nonetheless there are still many obstacles, one of them is data mapping from relational databases to RDF. Therefore the purpose of this thesis was to present one of the possible approaches with which we can achive RDB2RDF data mapping. We have developed prototype that exposes data in relational database to RDF. Also, we have enriched the exposed data with other information from open source datasets. The implemented prototype represents personal web library, with which a user can edit his book collection. In the context of research for the thesis, we have looked at some of the existing solutions and presented the technologies used in prototype development.

Key words:

semantic web, RDF, SPARQL, D2RQ, Fuseki, ontology, Protégé, relational database, data mapping, open source datasets

Slovar kratic

- `RDF` - Resource Description Framework, standardiziran podatkovni model za izmenjavo informacij na spletu.
- `OWL` - Web Ontology Language, družina jezikov, namenjenih urejanju ontologij in zbirk znanja.
- `SPARQL` - Simple Protocol and RDF Query Language, poizvedovalni jezik, sposoben pridobiti in manipulirati podatke v podatkovnih zbirkah, zapisanih v RDF formatu.
- `URI` - Uniform resource identifier, niz znakov namenjen identifikaciji spletnega vira.
- `HTTP` - HyperText Transfer Protocol, protokol za izmenjavo podatkov na spletu.
- `RDB2RDF` - Relational Databases to RDF, zbirka priporočil za preslikavo vsebine relacijskih podatkovnih baz v RDF.
- `R2RML` - Relational to RDF mapping language, jezik za izražanje prilagojenih preslikav iz relacijskih baz v RDF.
- `DM` - Direct mapping of relational data to RDF, jezik za izražanje neposredne preslikave podatkov iz relacijskih baz v RDF.
- `ISBN` - International Standard Book Number, mednarodna standardna knjižna številka

Poglavje 1

Uvod

Naša življenja so vedno bolj prepletena z uporaba spleta. Uporabljamo ga praktično na vsakem koraku. Iščemo informacije z uporabo našega najljubšega iskalnika, ostajamo v stiku z našimi prijatelji, plačujemo položnice, nakupujemo in še bi lahko naštevali. Nesporno dejstvo je, da je razvoj spleta poenostavil naša življenja in tudi nadaljnji razvoj bo usmerjen k temu.

Vzemimo za primer proces organizacije počitnic v lastni režiji. Splet, kot ga poznamo danes, nam to seveda že omogoča, vendar še zdaleč ne na enostaven način. Vložiti moramo veliko energije, da poiščemo najboljše možnosti nastanitev ter prevozov, ki ustrezajo našim zastavljenim okvirjem. Veliko bolj enostavno bi bilo v spletno aplikacijo vpisati našo želeno destinacijo, podati finačni okvir ter druge kriterije. Aplikacija bi nato na celotnem spletu poiskala najboljše možnosti glede na naše želje. Na žalost takšna aplikacija še ne obstaja, vendar razvoj semantičnega spleta nakazuje, da prihodnost ni tako daleč.

Semantičen splet ne bo nadomestil današnjega spleta, ampak ga bo le nadgradil oziroma razširil. Pravzaprav uporabniška izkušnja ne bo bistveno drugačna, spremembe se bodo namreč odražale v procesih, ki so nam nevidni. Bistvena prednost semantičnega spleta bo povezljivost informacij ter njihovo razumevanje s strani naprav in prav to bo omogočalo obstoj omenjene aplikacije.

Številne raziskave na področju semantičnega spleta so že pripeljale do

nekaterih pozitivnih učinkov [1]. Vendar ima to področje pred seboj še kar nekaj preprek. Ena izmed njih je zagotoviti dostop do podatkov relacijskih zbirk, ki predstavljajo velik vir informacij. Tu imamo v mislih predvsem dostop na način, ki ga predvidevajo priporočila ter standardi organizacije W3C [2], torej uporaba poizvedovalnega jezika SPARQL ter informacije v obliki RDF. Nikar pa ne pozabimo še na eno pomembno prednost, ki jo pridobimo z dostopom do podatkov v relacijskih podatkovnih bazah. Tako bo namreč možno te podatke dodatno obogatiti z informacijami iz drugih virov in obratno.

Cilj diplomske naloge je predstaviti enega izmed možnih pristopov, s katerim lahko omogočimo dostop do relacijskih podatkovnih zbirk. V ta namen bomo razvili prototip, ki bo podatke v relacijski podatkovni bazi izpostavil semantičnemu spletu. Hkrati bomo izpostavljene podatke obogatili z informacijami iz prosto dostopnih virov. Za prikaz prednosti takšnega pristopa bomo razvili uporabniški vmesnik, s katerim bo uporabnik lahko enostavno upravljal.

Tako bomo v prvem poglavju opisali konceptualno idejo prototipa, ki ga bomo razvijali. Ta bo orisala osnovne smernice prototipa ter umestila idejo v neko realno okolje, ki je vsem dobro poznano. Obstoječe rešitve ter implementacije problema preslikave podatkov iz relacijske baze v obliko RDF si bomo pogledali v tretjem poglavju. Obenem bomo predstavili dva pristopa za rešitev omenjenega problema, ki imata status uradnega priporočila s strani organizacije W3C. V četrtem poglavju sledi opis uporabljenih tehnologij. Na kratko bomo opisali posamezne tehnologije in orodja, ki jih bomo uporabili v diplomski nalogi. Podrobnosti, ki so pomembne za izdelavo našega prototipa, bomo opisali v drugih poglavjih. Razvoj prototipa bomo opisali v petem poglavju. Implementacijo bomo razdelili na več korakov. Vsak korak bomo podrobno predstavili ter, za boljše razumevanje, opisano podprli s primeri. V zadnjem, šestem, poglavju bomo povzeli cilje ter rešitve diplomske naloge. Analizirali bomo implementiran prototip ter podali možnosti za njegovo izboljšavo. Ob koncu poglavja bomo podali še zaključne ugotovitve ter sklepno misel.

Poglavje 2

Predstavitev prototipa osebne spletne knjižnice

Spletne aplikacije pogosto uporabljajo relacijske baze za hranjenje podatkov, ki jih nato prikažejo uporabniku. Vzemimo za primer Virtualno knjižnico Slovenije (COBISS/OPAC) [3], po kateri lahko iščemo knjige in njihovo razpoložljivost. COBISS nam za posamezno knjigo prikaže informacije, katerih količina je zelo omejena. Uporabnik lahko izve le tisto, kar o knjigi hrani podatkovna baza. Kljub temu bi lahko iz različnih virov na spletu pridobili še marsikatero uporabno informacijo o knjigi in njenem avtorju.

Konceptualna ideja, ki jo želimo v tej diplomski nalogi implementirati je obogatitev podatkov iz relacijske baze s podatki iz različnih prosto dostopnih spletnih virov in tako naši spletni strani ustvariti dodano vrednost. Za dosego tega cilja bomo zasnovali prototip osebne spletne knjižnice.

Knjižnica bo imela svojo podatkovno bazo, v kateri bomo hranili le osnovne podatke o vsaki knjigi, kot na primer naslov knjige, mednarodno standardno knjižno številko, ime in priimek avtorja ter naziv založbe. Te podatke bomo nato preslikali v RDF obliko, s pomočjo katere bomo lahko pridobili vrsto drugih podatkov o knjigi iz različnih spletnih virov. Slika 2.1 prikazuje diagram, ki predstavlja našo konceptualno idejo in njen poenostavljen način realizacije.

Slika 2.1: Diagram konceptualne ideje osebne spletne knjižnice.

Poglavje 3

Pregled obstoječih rešitev

Želja po rešitvi problema preslikave podatkov iz relacijske baze v obliko RDF obstaja že vse odkar je semantičen splet pridobil na popularnosti ter predvsem uporabnosti. Dejstvo je, da podatki v relacijskih zbirkah predstavljajo velik vir informacij. Obstaja domneva [4], da je količina podatkov, nedostopnih preko spleta, kar petstokrat večja od količine na spletu. Še več, večina teh t.i. "skritih" podatkov se hrani prav v relacijskih bazah. Z vključitvijo teh virov bi semantičen splet izpolnil obljubo o popolni podatkovni integraciji na področju spleta.

V ta namen sta se izoblikovala dva t.i. mapirna jezika, s katerima opišemo postopek preslikave podatkov. Oba jezika, tako R2RML [5] kot tudi DM [6], sta septembra 2012 s strani organizacije W3C postala priporočili pri preslikavi iz relacijske baze v RDF. Bistvena razlika med njima je predvsem v sami RDF obliki preslikanega podatka [7]. Pri Direct mapping, kot že ime samo pove, gre za neposredno preslikavo. To pomeni, da struktura nastalega mapirnega dokumenta, s katerim preslikamo podatke, neposredno odraža strukturo relacijske baze in tudi samo poimenovanje elementov.

Za razliko od neposredne preslikave R2RML omogoča prilagoditev mapiranja po uporabnikovih željah. R2RML mapirini dokument opisuje oziroma se nanaša na eno ali več logičnih tabel. Logična tabela lahko predstavlja:

- Tabelo relacijske baze
- Pogled SQL (*ang. View*)
- Veljavno poizvedbo SQL oziroma t.i. "pogled R2RML"

Tako lahko z uporabo jezika R2RML dosežemo enak učinek kot pri jeziku DM. Poleg tega pa lahko mapiranje prilagodimo specifičnemu primeru. Toda ne glede na to, kateri jezik izberemo, je rezultat mapiranja podatkovna zbirka v obliki RDF, ki odraža mapirni dokument.

Sam opis preslikave seveda ni dovolj, potrebujemo orodje, ki bo izvršilo to preslikavo. Potrebujemo sistem, ki mu kot vhod podamo mapirni dokument ter relacijsko bazo, ta pa nam nato omogoči dostop do podatkov relacijske baze v obliki RDF. Nekaj implementacij takšnega sistema je že na voljo [8], pogledali si bomo tri izmed njih.

Orodje Ultrawrap [9] je rezultat 4-letnega raziskovalnega dela podjetja Capsenta [10] iz Austin Teksasa v Združenih državah Amerike. Podpira uporabo obeh mapirinih jezikov in dostop do podatkov omogoča preko dostopne točke SPARQL (*ang. SPARQL endpoint*). Podjetje kot pomembne prednosti produkta navaja grafično urejanje mapirnega dokumenta, izredno hitro izvajanje poizvedb SPARQL ter skalabilnost. Ultrawrap je sestavljen iz dveh delov, strežnika ter prevajalnika. *Ultrawrap Compile* skrbi za procesiranje mapirnega dokumenta ter generiranje RDF ter OWL podatkov. Medtem ko *Ultrawrap Server* skrbi za izvajanje poizvedb SPARQL.

Naslednje orodje je platforma D2RQ [11], ki ima za razliko od predhodnika implementirano le uporabo jezika Direct mapping. D2RQ je odprto kodna programska oprema, izdana pod licenco Apache [12]. Glavne tri dele sistema predstavljajo D2RQ strežnik, D2RQ pogon ter D2RQ mapirni jezik. Platforma podpira uporabo relacijskih baz *Oracle*, *MySQL*, *PostgreSQL*, *SQL Server*, *HSQLDB* ter *Interbase/Firebird*. Prednost D2RQ predstavlja možnost vpeljave platforme v zbirko orodij JENA [13]. S tem razlogom bomo pri implementaciji našega prototipa uporabili prav platformo D2RQ. Poleg tega je izvorna koda

prosto dostopna na spletu. Tako bo naša naloga uporabiti D2RQ znotraj ogrodja JENA ter nadgraditi uporabnost preslikave podatkov iz relacijske baze v RDF.

Zadnje orodje, ki ga bomo na kratko predstavili, je Virtuoso [14] podjetja Openlink Software [15]. Virtuoso je pravzaprav sistem, ki omogoča simultan dostop do različnih podatkovnih zbirk različnih proizvajalcev. Del funkcionalnosti tega sistema je tudi RDF dostop do podatkov. Pri tem za mapiranje podatkov iz relacijske baze v RDF uporabljajo mapirni jezik Meta Schema Mapping Language [16], ki so ga razvili sami. Podpirajo tudi uporabo R2RML jezika. Tako se mapirni dokument, napisan v jeziku R2RML, pred uporabo prevede v njihov jezik s pomočjo translatorja.

Tako Ultrawrap kot tudi Virtuoso sta tržna produkta, kar kaže na zanimanje trga. Nedvomno se bo potreba po takšnih orodjih povečevala in tako bo posledično število različnih implementacij še večje. Obenem lahko tudi dodamo, da so obstoječe rešitve dobre ter uporabne. Z nadaljnjim razvojem in avtomatizacijo postopka preslikave pa bodo postale še bolj enostavne za vpeljavo v aplikacije, kar bo dodatno povečalo zanimanje za to področje.

Poglavje 4

Uporabljene tehnologije in orodja

4.1 Semantičen splet

"Semantičen splet ni ločen splet, temveč razširitev obstoječega, kjer imajo informacije dobro opredeljen pomen, kar omogoča boljše sodelovanje med računalniki in ljudmi." [17]. Bistvo semantičnega spleta je torej omogočiti delitev vsebine preko meja posameznih spletnih aplikacij ter strani. Problem obstoječega spleta je v tem, da naprave, ki ga poganjajo, ne razumejo vsebine, ki jo prikažejo uporabniku. Naloga teh naprav je trenutno omejena le na dostavo informacij uporabniku, ne pa tudi na razumevanje le-teh. Tako bo semantičen splet resnično zaživel takrat, ko bodo računalniki razumeli, kaj uporabnik želi in na podlagi tega našli ustrezne informacije na celotnem spletu.

4.2 Resource Description Framework

RDF [18] je standardiziran podatkovni model za izmenjavo informacij na spletu, tu imamo v mislih predvsem semantičen splet. Informacije so zapisane v obliki subjekt – predikat – objekt, ki jim pravimo tudi trojke (*ang. triples*). Subjekt nam pove, na koga se izjava nanaša. Iz predikata izvemo, kakšna je povezava med subjektom in objektom, ki vsebuje vrednost informacije. Prednost RDF

modela se kaže predvsem v možnosti izmenjave informacij med spletnimi aplikacijami ter dejstvu, da so razumljive napravam. Organizacija W3C je leta 1999 izdala specifikacijo RDF modela pod oznako "Priporočilo". V nekaj letih se je izoblikovala nova verzja RDF modela, ki je bila v letu 2004 tudi sprejeta s strani W3C.

Poznamo več različnih formatov, s katerimi lahko zapišemo RDF model. Tako Primer 4.1 prikazuje format N3 oziroma Notation3, ki je bil leta 2008 predstavljen s strani T. Berners-Lee in drugih [19]. Gre za enostavno berljiv ter razumljiv format za razliko od formata XML [20], ki je bolj prilagojen napravam. Čeprav je bil format XML predstavljen v sklopu modela RDF, je pomembno razlikovati med njima. Primer 4.2 prikazuje podatek RDF zapisan v formatu XML. Na tem mestu dodajmo, da bomo v diplomski nalogi uporabljali izključno format N3.

```
:Knjiga rdfs:comment "Razred Knjiga predstavlja knjigo."@sl
```

Primer trojke 4.1: Notation3.

```
<rdf:RDF>
      <rdf:Description rdf:about=":Knjiga">
            <rdfs:comment xml:lang="sl">
                  Razred Knjiga predstavlja knjigo.
            </rdfs:comment>
      </rdf:Description>
</rdf:RDF>
```

Primer trojke 4.2: RDF/XML.

4.3 Poizvedovalni jezik SPARQL

Izumitelj svetovnega spleta Tim Berners-Lee je nekoč dejal, da je uporaba semantičnega spleta brez poizvedovalnega jezika SPARQL enaka kot uporaba relacijske baze brez jezika SQL. Prav SPARQL je tisti vmesni člen, ki omogoča dostop do podatkov.

Leta 2008 je SPARQL postal uradno priporočilo organizacije W3C [2] na področju poizvedovanja nad podatkovnim modelom RDF. V tem času se je jezik razvijal in tako se je pojavila nova verzija t.i. SPARQL 1.1, ki je v marcu 2013 postala tudi uradno sprejeta s strani W3C.

Pravzaprav je sintaksa poizvedovalnega jezika SPARQL na prvi pogled zelo podobna jeziku SQL. Tako poznamo rezervirane besede, kot so `SELECT`, `WHERE`, `ORDER BY`, `UNION` in druge. Seveda je tu še vrsta drugih funkcionalnosti, ki jih omogoča jezik. Morda je ena izmed najbolj uporabnih poizvedovanje po t.i. dostopnih točkah SPARQL. Gre za storitve, ki omogočajo poizvedovanje nad podatki v lokalnem repozitoriju. Tako lahko z eno poizvedbo poiščemo podatke na različnih mestih, kar kaže na izjemno moč poizvedovalnega jezika SPARQL. Primer enostavne poizvedbe, kjer ne podamo nobenih pogojev in le vrnemo podatke v obliki trojk, prikazuje Poizvedba 4.1.

```
SELECT ?subjekt ?predikat ?objekt
WHERE
{
      ?subjekt ?predikat ?objekt.
}
```

Poizvedba 4.1: Primer enostavne poizvedbe.

4.4 Platforma D2RQ

Platforma D2RQ [11] ponuja dostop do podatkov relacijske baze v obliki podatkovnega modela RDF, kar nam omogoča izvajanje SPARQL poizvedb nad vsebino relacijske baze. Postopek za dostop se začne s kreiranjem mapirnega dokumenta, v katerem so zapisani podatki za dostop do relacijske baze ter kako se posamezni atributi relacijske sheme preslikajo v obliko RDF. Del platforme D2RQ predstavlja strežnik, ki mu ob zagonu podamo mapirni dokument in tako ustvarimo dostop do vsebine relacijske zbirke.

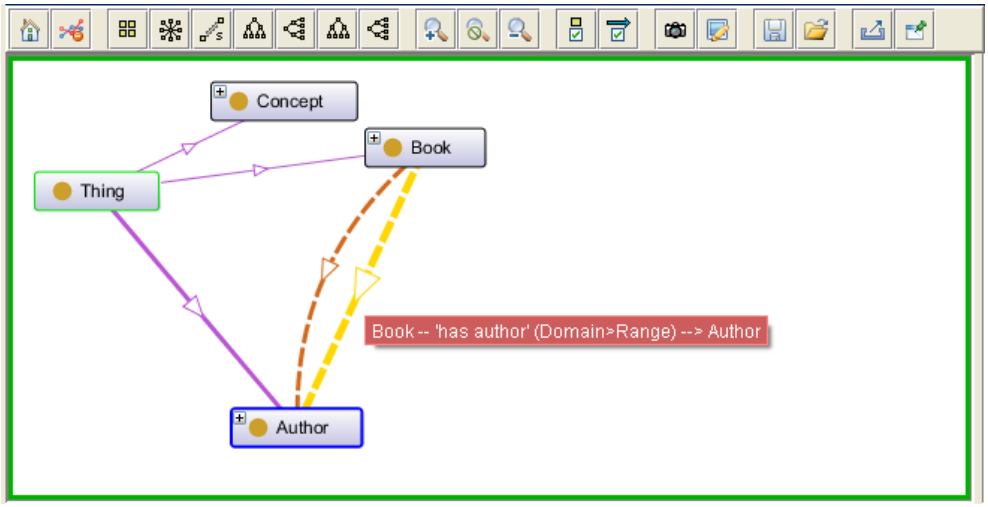
4.5 Strežniška platforma Fuseki

Strežniška platforma Fuseki [21] omogoča prikaz RDF podatkov ter izvajanje poizvedb z uporabo poizvedovalnega jezika SPARQL preko zahtev HTTP, ki temeljijo na REST arhitekturnem stilu. Čeprav je del ogrodja Jena, ga lahko uporabljamo kot samostojno orodje.

Jena [13] je odprto kodno ogrodje, zasnovano na programskem jeziku Java. Namenjeno je uporabi na področju semantičnega spleta. Omogoča pisanje in branje podatkov iz RDF grafov ter vrsto drugih funkcionalnosti, namenjenih obdelavi RDF informacij.

4.6 Programsko orodje Protégé

Protégé [22] je odprto kodno orodje za ustvarjanje ontologij, slovarjev ter domenskih modelov. Ponuja bogato paleto funkcionalnosti, med katere spadajo tudi kreiranje, manipuliranje ter vizualizacija ontologij. Orodje se je razvilo na univerzi Standford v sodelovanju z univerzo v Manchestru in ima trenutno že preko dvesto tisoč registriranih uporabnikov. Slika 4.1 prikazuje Protégéjevo orodje za vizualizacijo ustvarjenih ontologij.



Slika 4.1: Orodje za vizualizacijo ontologij (Protégé).

4.7 Programski vzorec Model - View - Controler

Model – View – Controler [23] je razvijalski programski vzorec, ki ga poznamo že iz sedemdesetih let prejšnjega stoletja, vendar je šele v zadnjih nekaj letih pridobil na popularnosti, predvsem na področju izgradnje spletnih aplikacij. Vzorec je zasnovan na način, pri katerem je predstavitev podatkov oziroma informacij ločena od uporabniške interakcije. Tako je model sestavljen iz podatkov aplikacije, njene poslovne logike in pravil. Pogled (angl. View) skrbi za predstavitev podatkov uporabniku. Z njegovo pomočjo uporabnik sproži akcijo, ki se pošlje v izvedbo kontrolerju. Kontroler (*ang. Controller*) pa je tisti, ki skrbi za pravilno delovanje aplikacije ter manipulira in spreminja podatke.

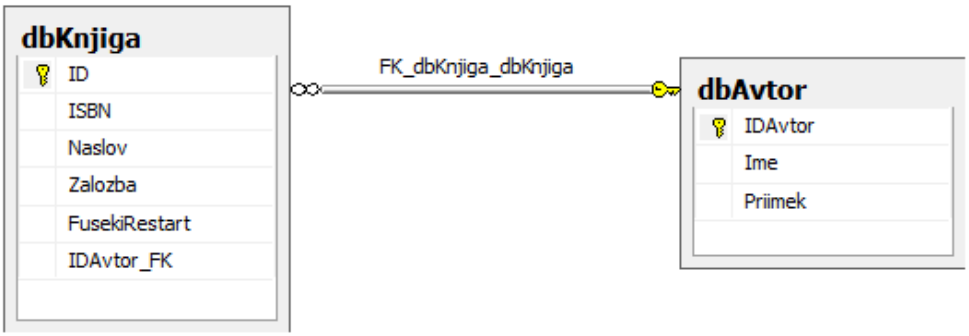
Poglavje 5

Implementacija prototipa osebne spletne knjižnice

Postopek implementacije osebne spletne knjižnice bomo razdelili na več korakov. V prvem koraku bomo opisali postavitev relacijske baze, ki nam bo služila za hranjenje osnovnih informacij o knjigi. Nato bomo v drugem koraku prikazali postopek kreiranja mapirnega dokumenta, s pomočjo katerega bomo omogočili dostop do relacijske baze preko poizvedovalnega jezika SPARQL. V tretjem koraku sledi izdelava slovarja, s katerim bomo definirali razrede ter njihove lastnosti in določili njihovo povezavo z informacijami iz drugih virov. Proces postavitve in konfiguracije strežnika Fuseki, ki nam bo služil za izvajanje SPARQL poizvedb in iskanje informacij iz drugih spletnih virov, bomo opisali v četrtem koraku. Na koncu sledi še opis izdelave spletne aplikacije, s katero bo uporabnik upravljal osebno spletno knjižnico in ki bo v zaokroženo celoto povezala vse ostale dele našega prototipa.

5.1 Postavitev relacijske baze

Za hranjenje podatkov bomo uporabili brezplačno Microsoftovo relacijsko bazo SQL Express 2008 R2 [24]. Diagram podatkovne baze prikazuje Slika 5.1.



Slika 5.1: Diagram podatkovne relacijske baze.

Kot vidimo, je baza sestavljena le iz dveh tabel, ki sta namenjeni hranjenju najosnovnejših informacij o knjigi.

Tabela `dbKnjiga` je namenjena hranjenju informacij o posameznih knjigah in je sestavljena iz naslednjih atributov:

- `ID` – predstavlja primarni ključ knjige
- `ISBN` – predstavlja mednarodno standardno knjižno številko knjige
- `Naslov` – predstavlja naslov knjige
- `Zalozba` – predstavlja ime založbe, pri kateri je bila knjiga izdana
- `FusekiRestart` – predstavlja vrednost, ki označuje resetiranje strežnika Fuseki
- `IDAvtor_FK` – predstavlja tuj ključ, ki se navezuje na avtorja knjige

Tabela `dbAvtor` je namenjena hranjenju informacij o posameznih avtorjih knjig in je sestavljena iz naslednjih atributov:

- `IDAvtor` – predstavlja primarni ključ avtorja
- `Ime` – predstavlja ime avtorja
- `Priimek` – predstavlja priimek avtorja

V nasprotju z drugimi spletnimi aplikacijami, ki imajo obsežno podatkovno bazo, je naša zelo preprosta in ne vsebuje veliko podatkov. Hranimo le tiste informacije, s pomočjo katerih bomo uspeli iz drugih spletnih virov pridobiti dodatne informacije. Tako se že tu kažejo prednosti takšne zasnove, saj prihranimo veliko časa z načrtovanjem baze in posledično tudi z vzdrževanjem le-te.

5.2 Kreiranje mapirnega dokumenta

Mapirni dokument je najpomembnejši člen pri preslikavi podatkov iz relacijske podatkovne baze v RDF obliko. Njegov koncept je podoben "pogledom" (*ang. Views*) pri relacijskih podatkovnih bazah, le da gre tu za virtualno strukturo podatkov v obliki RDF in ne virtualno relacijsko tabelo. Celotna vsebina dokumenta predstavlja virtualen RDF graf z informacijami o podatkovni bazi.

Platforma D2RQ vsebuje svoj mapirini jezik [25] za opisovanje povezav med shemo relacijske baze in slovarji, ki opisujejo RDF obliko. S pomočjo tega jezika lahko sami napišemo mapirni dokument, ob tem moramo upoštevati njegova pravila. Kreiranje lahko prepustimo tudi platformi sami, pri čemer bo mapirni dokument predstavljal neposredno preslikavo ciljne relacijske baze.

Generiranje mapirnega dokumenta, za relacijsko bazo SQL Express, sprožimo z ukazom:

```
generate-mapping -u [] -p [] -d com.microsoft.sqlserver.jdbc.
SQLServerDriver -o mapping.ttl jdbc:"sqlserver:[];databaseName=[]"
```

Pojasnilo vhodnih parametrov:

- `-u`, uporabniško ime za dostop do relacijske baze
- `-p`, geslo za dostop do relacijske baze
- `-d`, polno ime Java razreda za krmilnik izbrane relacijske baze

- `-o`, ime datoteke, v katero se bo zapisal mapirni dokument
- `jdbc`, url povezava do relacijske baze, oblika le-te je odvisna od uporabljene baze

Mapirni dokument, ki ga zgenerira platforma D2RQ, je sestavljen iz večih delov. Pogledali si bomo le najpomembnejše gradnike na našem primeru relacijske baze. Celoten dokument je na ogled v Dodatku A.

Na začetku dokumenta so nanizane predpone, ki poenostavijo izgled dokumenta in jih lahko razumemo kot bližnjice. Nato sledi prvi del, ki je vsem skupen in definira povezavo do relacijske baze. Tako se vrednosti podanih vhodnih parametrov zapišejo v dokument in na podlagi le-teh se vrši dostop do baze podatkov. Zapis 5.1 predstavlja prvi del mapirnega dokumenta za naš primer relacijske baze.

```
map:database a d2rq:Database;
      d2rq:jdbcDriver "com.microsoft.sqlserver.jdbc.
         SQLServerDriver";
      d2rq:jdbcDSN "jdbc:sqlserver://NYMERON-PC\\
         SQLEXPRESS_FRI;databaseName=D2RQ_Test";
      d2rq:username "sa";
      d2rq:password "sasa";
      .
```

Zapis 5.1: Prvi del mapirnega dokumenta.

Na tem mestu omenimo, da lahko definiramo dostop do več kot samo ene relacijske baze. To pomeni, da v primeru, ko imamo podatke razdrobljene v dveh ali več relacijskih bazah, lahko s pravilno definiranim mapirnim dokumentom ustvarimo virtualni RDF graf, ki predstavlja te podatke. Prednost se prikaže predvsem pri poizvedovanju nad podatki, saj jih imamo na voljo na enem mestu, čeprav se dostop do njih vrši preko več relacijskih baz.

Drugi del dokumenta je sestavljen iz opisa tabel, ki se nahajajo v relacijski shemi in njihovih atributov. Zapis 5.2 predstavlja preslikavo tabel `dbKnjiga` ter `dbAvtor`.

```
# Tabela dbo.dbAvtor
map:dbo_dbAvtor a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "dbo/dbAvtor/@@dbo.dbAvtor.
         IDAvtor@@";
      d2rq:class vocab:dbo_dbAvtor;
      d2rq:classDefinitionLabel "dbo.dbAvtor";
      .
# Tabela dbo.dbKnjiga
map:dbo_dbKnjiga a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "dbo/dbKnjiga/@@dbo.dbKnjiga.
         ID@@";
      d2rq:class vocab:dbo_dbKnjiga;
      d2rq:classDefinitionLabel "dbo.dbKnjiga";
      .
```

Zapis 5.2: Drugi del mapirnega dokumenta.

Ob generiranju dokumenta se vsaka tabela preslika v svoj razred, ki ga predstavlja `d2rq:ClassMap`. Razredu je potrebno podati izvor tabele, torej v kateri relacijski bazi se tabela nahaja. To naredimo tako, da objekt predikata `d2rq:dataStorage` kaže na del dokumenta, kjer smo definirali dostop do baze.

Objekt, ki sledi predikatu `d2rq:uriPattern`, določa enolični identifikator vsake vrstice v naši tabeli. Naslov lahko poljubno spremenimo, vendar moramo zagotoviti enoličnost, kar ponavadi dosežemo z uporabo primarnega ključa v naslovu.

Atributi tabel se preslikajo v lastnosti, ki jih predstavlja `d2rq:PropertyBridge`. Vsaka definirana lastnost mora pripadati nekemu razredu, kar določimo z objek-

tom, ki sledi predikatu `d2rq:belongsToClassMap`. Poleg tega je potrebno določiti, s katerim stolpcem oziroma atributom v tabeli se lastnost poveže, kar naredimo z objektom, ki sledi predikatu `d2rq:column`. V primeru, da tega ne podamo, moramo določiti vzorec, ki bo določal vrednost lastnosti, kar naredimo z objektom, ki sledi `d2rq:pattern`.

Lastnostim lahko določimo, za kakšen tip podatka gre, kar naredimo z objektom, ki sledi predikatu `d2rq:datatype`. Vrednost objekta, ki sledi predikatu `d2rq:property`, pa nam pove, kako definirano lastnost navajamo pri pisanju poizvedb. Zapis 5.3 prikazuje preslikavo atributa `IDAvtor` iz tabele `dbAvtor` ter novo ustvarjeno lastnost, ki je bila dodana ob generiranju mapirnega dokumenta.

```
map:dbo_dbAvtor__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbAvtor;
      d2rq:property rdfs:label;
      d2rq:pattern "dbAvtor #@@dbo.dbAvtor.IDAvtor@@";
      .
map:dbo_dbAvtor_IDAvtor a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbAvtor;
      d2rq:property vocab:dbo_dbAvtor_IDAvtor;
      d2rq:propertyDefinitionLabel "dbAvtor IDAvtor";
      d2rq:column "dbo.dbAvtor.IDAvtor";
      d2rq:datatype xsd:integer;
      .
```

Zapis 5.3: Preslikava atributov iz RDB v RDF.

Tabele v relacijskih bazah so pogosto med seboj povezane. Povezave so lahko realizirane na več načinov, eden izmed njih je tudi s pomočjo tujega ključa. V slednjem primeru je tuj ključ definiran kot atribut tabele in le-to povezuje z drugo tabelo.

Tudi naši tabeli `dbKnjiga` ter `dbAvtor` sta med seboj povezani s tujim ključem. Tako atribut `IDAvtor_FK` v tabeli `dbKnjiga` predstavlja tuj ključ in

kaže na atribut `IDAvtor` v tabeli `dbAvtor`, kar v Zapisu 5.4 opisuje objekt, ki sledi predikatu `d2rq:join`.

```
map:dbo_dbKnjiga_IDAvtor_FK__ref a d2rq:PropertyBridge
    ;
       d2rq:belongsToClassMap map:dbo_dbKnjiga;
       d2rq:property vocab:dbo_dbKnjiga_IDAvtor_FK;
       d2rq:refersToClassMap map:dbo_dbAvtor;
       d2rq:join "dbo.dbKnjiga.IDAvtor_FK => dbo.
          dbAvtor.IDAvtor";
       .
```

Zapis 5.4: Preslikava tujega ključa iz RDB v RDF.

Poleg tega, da moramo podati, kateremu razredu pripada lastnost, ki predstavlja tuj ključ, moramo povedati tudi, na kateri razred se tuj ključ navezuje. Slednje naredimo z objektom, ki sledi predikatu `d2rq:refersToClassMap`.

S takim načinom povezovanja tabel lahko priredimo mapirni dokument povsem po naših željah. Tako lahko ustvarimo povezave med tabelami, ki sicer v relacijski bazi niso povezane. Še več, med seboj lahko povežemo tabeli, ki sta vsaka v svoji relacijski bazi.

S tem zaključujemo opis kreiranja mapirnega dokumenta s pomočjo platforme D2RQ. Ugotovimo lahko, da avtomatsko kreiranje dokumenta zadošča za osnovne primere relacijskih shem. Za bolj kompleksne primere pa maprini jezik omogoča, da sami ustvarimo dokument, ki pokrije naše zahteve.

5.2.1 Uporaba mapirnega dokumenta s strežnikom D2RQ

Platforma D2RQ je opremljena s strežnikom [26], ki omogoča dostop do podatkov relacijske baze. V povezavi s spletnim brskalnikom lahko pregledujemo vsebino baze in izvajamo poizvedbe s poizvedovalnim jezikom SPARQL. Zahteva za dostop oziroma poizvedba se s pomočjo mapirnega dokumenta prepiše v poizvedbo SQL ter nato izvede. Prav tako se podatki, ki jih vrne relacijska baza,

preslikajo v RDF obliko na podlagi mapirnega dokumenta in nato prikažejo v brskalniku. S tem t.i. "*on-the-fly*" načinom preslikave izključimo potrebo po repliciranju podatkov v začasno skladišče RDF trojk (*ang. triplestore*), kar se izkaže kot prednost predvsem pri večjih relacijskih bazah. Slika 5.2 prikazuje diagram poteka pregledovanja podatkov s spletnim brskalnikom ter proces preslikave podatkov s pomočjo mapirnega dokumenta.

Slika 5.2: Diagram preslikave podatkov.

Strežnik D2RQ zaženemo v ukazni lupini z ukazom, ki mu kot vhodni parameter podamo številko vrat ter ime mapirnega dokumenta. Slika 5.3 prikazuje uspešen zagon strežnika z uporabo mapirnega dokumenta `mapping.ttl` ter številko vrat `2020`.

Slika 5.3: Zagon strežnika D2RQ.

Za pregled in izvajanje poizvedb odpremo spletni brskalnik ter vpišemo naslov strežnika D2RQ, v našem primeru `http://localhost:2020/`. Prikaže se enostaven spletni vmesnik, ki ga vidimo na Sliki 5.4, s pomočjo katerega lahko navigiramo po vsebini relacijske baze.

Slika 5.4: Spletni vmesnik strežnika D2RQ.

Za primer demonstracije vpišimo Poizvedbo 5.1 v urejevalnik poizvedb na naslovu `http://localhost:2020/sparql` ter jo podajmo strežniku D2RQ. Poizvedba za posamezno knjigo vrne njen naslov ter ime in priimek avtorja. Slika 5.5 prikazuje rezultate poizvedbe v uporabniku prijazni obliki.

```
PREFIX vocab: <http://localhost:2020/resource/vocab/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?knjiga ?naslov ?imeAvtorja ?priimekAvtorja
WHERE
{
      ?knjiga  rdf:type vocab:dbo_dbKnjiga;
      vocab:dbo_dbKnjiga_Naslov ?naslov;
      vocab:dbo_dbKnjiga_IDAvtor_FK ?avtor.
```

```
        ?avtor  vocab:dbo_dbAvtor_Ime ?imeAvtorja;
        vocab:dbo_dbAvtor_Priimek ?priimekAvtorja.
}
```

Poizvedba 5.1: Osnovni podatki o knjigi.

SPARQL results:

knjiga	naslov	imeAvtorja	priimekAvtorja
db:dbo/dbKnjiga/1	"A game of thrones"	"George"	"R.R.Martin"
db:dbo/dbKnjiga/2	"Odiseja človeštva"	"Spencer"	"Wells"
db:dbo/dbKnjiga/3	"The curious incident of the dog in the night time"	"Mark"	"Haddon"
db:dbo/dbKnjiga/4	"Korak Zadaj"	"Henning"	"Manken"
db:dbo/dbKnjiga/26	"Alamut"	"Judith"	"Tarr"

Slika 5.5: Rezultat poizvedbe 5.1.

Preslikava podatkov iz relacijske baze v RDF obliko je na ta način dosežena. Uspešno smo izvedli SPARQL poizvedbo in pridobili podatke iz relacijske baze. Tako se izkaže, da je platforma D2RQ zares dobro orodje, ki nam na enostaven način pomaga izpostaviti podatke semantičnemu spletu.

5.3 Opis domenskega slovarja

Na začetku razložimo sam pomen pojma slovar. Vzemimo za primer Slovar slovenskega knjižnega jezika. V tem slovarju so opisane besede, ki jih poznamo v slovenskem jeziku, njihov izvor, izgovorjava, primeri uporabe ter druge informacije. Gre torej za dokument, v katerem so podrobno opisani izrazi za neko področje, za naš primer je to slovenski jezik.

Tudi v semantičnem spletu poznamo pojem slovar in ga uporabljamo za doseg iste naloge. Torej definirati pojme, izraze ter povezave, s katerimi opišemo neko področje našega delovanja. Tako lahko za posamezno aplikacijo določimo dovoljen nabor izrazov, njihove omejitve ter tudi, kako so med seboj povezani.

V diplomski nalogi smo omenili tudi pojem ontologija, ki je pravzaprav neke vrste ekvivalent slovarju. Splošno sprejeta definicija pravi, da je ontologija

formalna specifikacija skupne konceptualizacije [27], kar na kratko povzame naš opis slovarja. Ne obstaja stroga ločnica med slovarjem in ontologijo. Obstaja le nenapisano pravilo, ki pravi, da se ontologija uporablja za bolj kompleksne ter formalne opise. Medtem ko se slovar uporablja za enostavne ter neformalne opise izrazov.

Ne glede na to, ali gre za ontologijo ali za slovar, je namen uporabe enak. Poleg tega, da opišemo neko področje, je bistvena prednost ta, da z njuno uporabo lahko dosežemo integracijo podatkov. V semantičnem spletu se pogosto zgodi, da je isti tip podatka v različnih podatkovnih zbirkah opisan z različnimi izrazi. Z uporabo ontologije in slovarja lahko te razlike poenotimo in s tem dosežemo integracijo z različnimi viri podatkov. Prav zaradi te prednosti predstavljajo ontologije in slovarji enega izmed glavnih stebrov semantičnega spleta.

5.3.1 Izdelava slovarja

Za prototip osebne spletne knjižnjice bomo ustvarili slovar, s katerim bomo opisali vse izraze, ki jih bomo uporabili v aplikaciji. Določili bomo tudi povezave ter integrirali naše izraze z drugimi viri. Slovar bomo oblikovali s pomočjo orodja Protégé, nekatere gradnike pa bomo dodali kar z navadnim tekstovnim urejevalnikom. Slovar bomo zapisali v obliki RDF z uporabo formata Notation3.

Na tem mestu naštejmo še nekaj glavnih predpon, ki jih bomo uporabljali v opisu slovarja. Predpona, ki naslavlja naš slovar in vse izraze v njem, je definirana kot:

- `@prefix :  <http://www.test.org/VidmarJernej/diploma#>.`

Predpona s katero naslavljamo mapirni dokument, ki smo ga ustvarili, je definirana kot:

- `@prefix vocab:`
  `<file:///C%3A/Users/FRI/Documents/Fuseki/fuseki-server/vocab/>.`

Za razliko od drugih predpon, ki naslavljajo razrešljive spletne naslove, ta predpona kaže na fizično mesto v računalniku, kjer se nahaja naš mapirni dokument. Zadnji del predpone, torej `vocab/`, pa naslavlja predpono, ki je definirana v samem mapirnem dokumentu. Druge uporabljene predpone so navedene v Dodatku B, ki prikazuje celoten slovar za naš primer.

Uporabili bomo dva prosto dostopna vira, s katerima bomo integrirali naše podatke. Prvi vir je Britanska narodna knjižnica [28], pri kateri bomo za lažjo integracijo uporabili njihovo shemo podatkovnega modela [29]. Drugi vir je DBpedia [30], trenutno najobširnejši vir podatkov v semantičnem spletu. Integracija bo potekala v obliki trojke subjekt – predikat – objekt. Vrednost predikata bo `rdfs:subClassOf`, če bo izraz predstavljal razred, sicer bo vedno `rdfs:subPropertyOf`. Vrednost subjekta bo predstavljal izraz iz drugega vira, ki ga želimo integrirati z našim izrazom, ki bo vedno v vlogi objekta. Prav tako bomo morali integrirati tudi podatke iz naše relacijske baze, kar bomo storili na enak način.

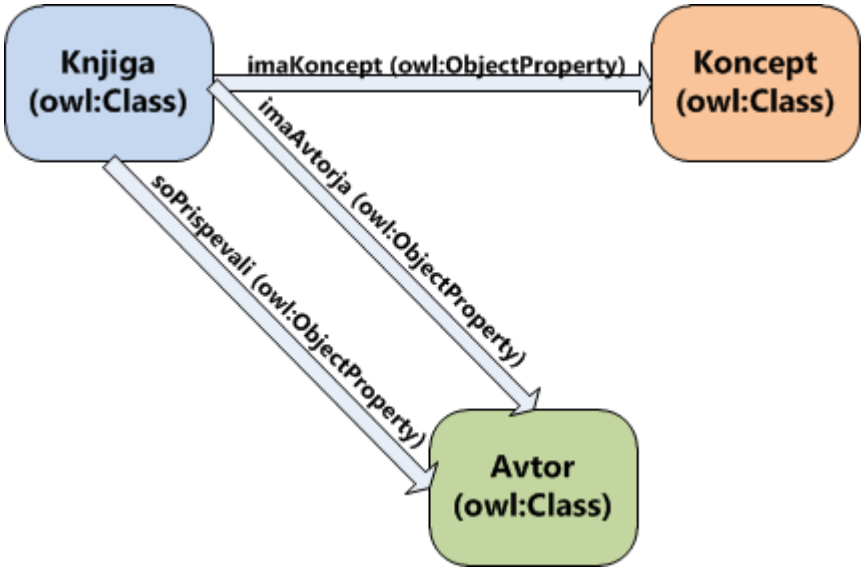
Slika 5.6 prikazuje tri razrede, ki bodo vsebovani v našem slovarju ter povezave med njimi. Vsi razredi so tipa `owl:Class`, povezave med njimi pa tipa `owl:ObjectProperty`, ki predstavlja povezavo med objekti. Sledi podrobnejši opis razredov ter relacij med njimi.

Opis relacij

Relacija `imaKoncept` med seboj povezuje razreda `Knjiga` in `Koncept`. Izvor povezave je razred `Knjiga`, kar določimo z objektom, ki sledi predikatu `rdfs:domain`. Ponor povezave je razred `Koncept`, kar določimo z objektom, ki sledi predikatu `rdfs:range`. Integracija povezave `imaKoncept` z drugimi viri je dosežena z naslednjo trojko:

- `dct:subject rdfs:subPropertyOf :imaKoncept.`

Relacija `imaAvtorja` med seboj povezuje razreda `Knjiga` in `Avtor`. Prav tako je pri tej povezavi izvor razred `Knjiga`, ponor pa predstavlja razred `Avtor`.



Slika 5.6: Diagram razredov slovarja.

Integracija povezave `imaAvtorja` z drugimi viri je dosežena s trojkami:

- `dct:creator rdfs:subPropertyOf :imaAvtorja.`
- `blt:hasCreated rdfs:subPropertyOf :imaAvtorja.`
- `dbpedia-owl:author rdfs:subPropertyOf :imaAvtorja.`
- `dbpprop:author rdfs:subPropertyOf :imaAvtorja..`
- `vocab:dbo_dbKnjiga_IDAvtor_FK rdfs:subPropertyOf :imaAvtorja.`

Zadnja trojka predstavlja integriranje s podatkom v naši relacijski bazi, ki predstavlja tuj ključ v tabeli `dbKnjiga` in kaže na tabelo `dbAvtor`.

Relacija `soPrispevali` med seboj povezuje razreda `Knjiga` in `Avtor`. Izvor ter ponor povezave sta enaka kot pri povezavi `imaAvtorja`. Razlika med slednjo ter povezavo `soPrispevali` je naslednja: povezava `imaAvtorja` določa

avtorja knjige, medtem ko povezava `soPrispevali` določa osebe, ki so prispevali pri ustvarjanju knjige. Integracija z drugimi viri je dosežena z naslednjima trojkama:

- `dct:contributor rdfs:subPropertyOf :soPrispevali.`
- `blt:hasContributedTo rdfs:subPropertyOf :soPrispevali.`

Razred Knjiga

Slika 5.7 predstavlja razred `Knjiga` in njegove atribute oziroma lastnosti. S tem razredom bomo v naši aplikaciji predstavili informacije o posamezni knjigi. Integracijo razreda `Knjiga` z drugimi viri bomo dosegli s trojkami:

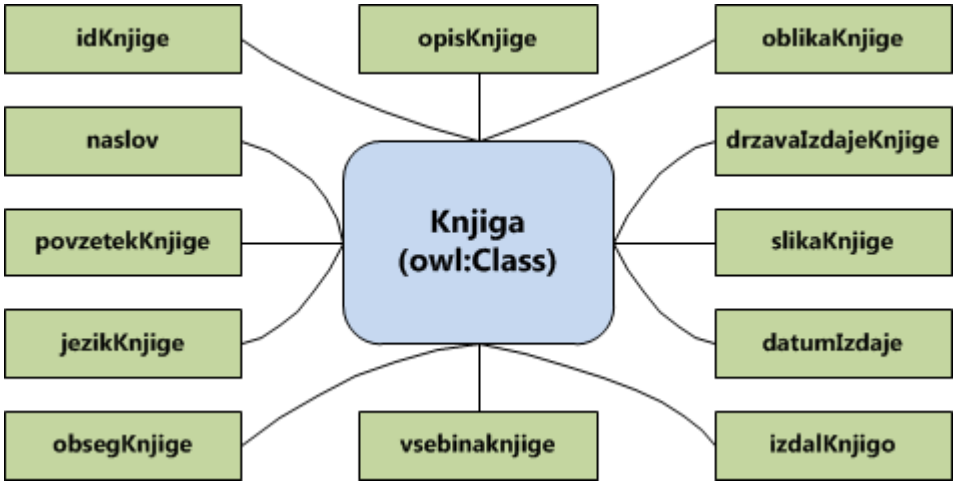
- `bibo:Book rdfs:subClassOf :Knjiga.`
- `dbpedia-owl:Book rdfs:subClassOf :Knjiga.`
- `vocab:dbo_dbKnjiga rdfs:subClassOf :Knjiga.`

Zadnja trojka predstavlja integriranje z našo relacijsko bazo in sicer z razredom `dbo_dbKnjiga` tabele `dbKnjiga`.

Vsi atributi razreda Knjiga so tipa `owl:DatatypeProperty`, kar predstavlja lastnost, ki nosi informacijo. Sledi podrobnejši opis teh atributov.

Atribut `idKnjige` predstavlja ISBN številko, s katero enolično definiramo knjigo. Informacija je zapisana v obliki tekstovnega niza. Atribut je z drugimi viri integriran z naslednjimi trojkami:

- `bibo:isbn10 rdfs:subPropertyOf :idKnjige.`
- `bibo:isbn13 rdfs:subPropertyOf :idKnjige.`
- `bibo:isbn rdfs:subPropertyOf :idKnjige.`
- `dbpedia-owl:isbn rdfs:subPropertyOf :idKnjige.`



Slika 5.7: Diagram razreda Knjiga.

- `dbpprop:isbn rdfs:subPropertyOf :idKnjige.`
- `vocab:dbo_dbKnjiga_ISBN rdfs:subPropertyOf :idKnjige.`

Zadnja trojka predstavlja integriranje z našo relacijsko bazo, in sicer z atributom `ISBN` tabele `dbKnjiga`.

Atribut `naslov` predstavlja naslov knjige in je z drugimi viri integriran z naslednjimi trojkami:

- `dct:title rdfs:subPropertyOf :naslov.`
- `dct:alternative rdfs:subPropertyOf :naslov.`
- `dbpedia-owl:name rdfs:subPropertyOf :naslov.`
- `dbpprop:name rdfs:subPropertyOf :naslov.`
- `vocab:dbo_dbKnjiga_Naslov rdfs:subPropertyOf :naslov.`

Zadnja trojka predstavlja integriranje z našo relacijsko bazo, in sicer z atributom `Naslov` tabele `dbKnjiga`.

Atribut `opisKnjige` predstavlja kratek opis o knjigi in je z drugimi viri integriran z naslednjimi trojkami:

- `dct:description rdfs:subPropertyOf :opisKnjige.`
- `rdfs:comment rdfs:subPropertyOf :opisKnjige.`

Atribut `povzetekKnjige` predstavlja povzetek knjige in je z drugimi viri integriran z naslednjimi trojkami:

- `dbpedia-owl:abstract rdfs:subPropertyOf :povzetekKnjige.`
- `dct:abstract rdfs:subPropertyOf :povzetekKnjige.`

Atribut `jezikKnjige` predstavlja jezik, v katerem je knjiga napisana in je z drugimi viri integriran z naslednjimi trojkami:

- `isbd:P1073 rdfs:subPropertyOf :jezikKnjige.`
- `dbpprop:language rdfs:subPropertyOf :jezikKnjige.`

Atribut `obsegKnjige` predstavlja opis o obsegu oziroma številu strani knjige in je z drugimi viri integriran z naslednjimi trojkami:

- `dbpprop:pages rdfs:subPropertyOf :obsegKnjige.`
- `isbd:P1053 rdfs:subPropertyOf :obsegKnjige.`

Atribut `vsebinaKnjige` predstavlja opis vsebine knjige oziroma seznam poglavij, ki jih najdemo v knjigi. Atribut je z drugimi viri integriran s trojko:

- `dct:tableOfContents rdfs:subPropertyOf :vsebinaKnjige.`

Atribut `izdalKnjigo` predstavlja ime založbe, ki je knjigo izdala in je z drugimi viri integriran s trojko:

- `dbpprop:publisher rdfs:subPropertyOf :izdalKnjigo.`

Atribut `datumIzdaje` predstavlja datum izdaje knjige in je z drugimi viri integriran z naslednjo trojko:

- `dbpprop:pubDate rdfs:subPropertyOf :datumIzdaje.`

Atribut `oblikaKnjige` predstavlja obliko, v kateri je knjiga izdana, torej ali gre za tiskano ali digitalno obliko ali pa morda za katero drugo. Atribut je z drugimi viri integiran s trojko:

- `dbpprop:mediaType rdfs:subPropertyOf :oblikaKnjige.`

Atribut `drzavaIzdajeKnjige` predstavlja državo, v kateri je bila knjiga izdana in je z drugimi viri integrirana z naslednjo trojko:

- `dbpprop:country rdfs:subPropertyOf :drzavaIzdajeKnjige.`

Atribut `slikaKnjige` predstavlja spletno povezavo do mesta, kjer se nahaja slika knjige in je z drugimi viri integirana s trojko:

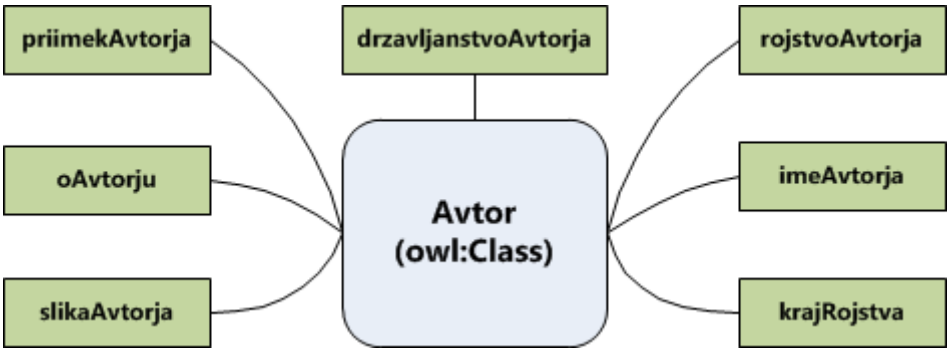
- `dbpedia-owl:thumbnail rdfs:subPropertyOf :slikaKnjige.`

Razred Avtor

Razred `Avtor` in njegove atribute predstavlja Slika 5.8. S pomočjo tega razreda bomo v aplikaciji predstavili informacije o avtorju knjige. Z drugimi viri bomo razred Avtor integrirali z naslednjimi trojkami:

- `dct:Agent rdfs:subClassOf :Avtor.`
- `foaf:Person rdfs:subClassOf :Avtor.`
- `foaf:Agent rdfs:subClassOf :Avtor.`
- `dbpedia-owl:author rdfs:subClassOf :Avtor.`
- `dbpprop:author rdfs:subClassOf :Avtor.`
- `vocab:dbo_dbAvtor rdfs:subClassOf :Avtor.`

Zadnja trojka predstavlja integriranje z našo relacijsko bazo, in sicer z razredom `dbo_dbAvtor` tabele `dbAvtor`.



Slika 5.8: Diagram razreda Avtor.

Tudi atributi razreda Avtor so tipa `owl:DataProperty`, kar pomeni, da hranijo informacijo. Sledi njihov podrobnejši opis.

Atribut `imeAvtorja` predstavlja ime avtorja in je z drugimi viri integriran z naslednjimi trojkami:

- `foaf:givenName rdfs:subPropertyOf :imeAvtorja.`
- `foaf:name rdfs:subPropertyOf :imeAvtorja.`
- `vocab:dbo_dbAvtor_Ime rdfs:subPropertyOf :imeAvtorja.`

Zadnja trojka predstavlja integriranje z našo relacijsko bazo, in sicer z atributom `Ime` tabele `dbAvtor`.

Atribut `priimekAvtorja` predstavlja priimek avtorja in je z drugimi viri integriran z naslednjimi trojkami:

- `foaf:familyName rdfs:subPropertyOf :priimekAvtorja.`
- `foaf:surname rdfs:subPropertyOf :priimekAvtorja.`
- `vocab:dbo_dbAvtor_Priimek rdfs:subPropertyOf :priimekAvtorja.`

Zadnja trojka predstavlja integriranje z našo relacijsko bazo, in sicer z atributom `Priimek` tabele `dbAvtor`.

Atribut `oAvtorju` predstavlja kratek opis o avtorju in njegovem delu. Z drugimi viri je atribut integriran s trojko:

- `dbpedia-owl:abstract rdfs:subPropertyOf :oAvtorju.`

Atribut `slikaAvtorja` predstavlja spletno povezavo do mesta, kjer se nahaja slika avtorja in je z drugimi viri integriran s trojko:

- `dbpedia-owl:thumbnail rdfs:subPropertyOf :slikaAvtorja.`

Atribut `drzavljanstvoAvtorja` predstavlja informacijo o avtorjevem državljanstvu. Z drugimi viri je atribut integriran s trojkama:

- `dbpprop:citizenship rdfs:subPropertyOf :drzavljanstvoAvtorja.`
- `dbpprop:nationality rdfs:subPropertyOf :drzavljanstvoAvtorja.`

Atribut `rojstvoAvtorja` predstavlja datum rojstva avtorja in je z drugimi viri integriran s trojkama:

- `dbpedia-owl:birthDate rdfs:subPropertyOf :rojstvoAvtorja.`
- `dbpprop:birthDate rdfs:subPropertyOf :rojstvoAvtorja.`

Atribut `krajRojstva` predstavlja kraj, v katerem se je avtor rodil, in je z drugimi viri integriran z naslednjo trojko:

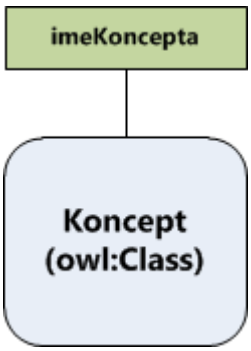
- `dbpprop:placeOfBirth rdfs:subPropertyOf :krajRojstva.`

Razred Koncept

Slika 5.9 predstavlja razred `Koncept`, s katerim bomo podali koncepte, ki se pojavijo v posamezni knjigi. Razred bomo z drugimi viri integrirali z naslednjimi trojkami:

- `skos:Concept rdfs:subClassOf :Koncept.`
- `blt:TopicDDC t rdfs:subClassOf :Koncept.`
- `blt:TopicLCSH rdfs:subClassOf :Koncept.`
- `blt:PersonConcept rdfs:subClassOf :Koncept.`

Edini atribut razreda je `imeKoncepta`, ki predstavlja ime oziroma oznako koncepta. Atributa ne bomo integrirali z drugimi viri, ampak le vezali na oznako koncepta.



Slika 5.9: Diagram razreda Koncept.

Dostop do drugih virov

Poleg izrazov, ki smo jih definirali z orodjem Protégé [22], bomo v slovarju hranili tudi informacije za dostop do drugih virov. V ta namen bomo v slovarju definirali poseben izraz, ki ga bomo poimenovali `endpoint`. Z njim bomo identificirali podatke za dostop.

Vsak vir bo imel v slovarju trojko v obliki subjekt – predikat – objekt. Subjekt bo naš novo definirani izraz, predikat bo `rdfs:seeAlso`, medtem ko bo objekt hranil informacijo o dostopu do vira. Tako bomo v aplikaciji slovar povprašali po izrazu `endpoint` in pridobili podatke za dostop do virov.

Informacija o dostopu je pravzaprav spletni naslov, na katerem ima vir izpostavljeno storitev, ki deluje po specifikaciji protokola SPARQL [31]. Takšnemu

spletnemu naslovu pravimo dostopna točka SPARQL. Naloga storitve je tako sprejeti poizvedbo SPARQL, jo izvesti ter vrniti rezultate. Z njegovo pomočjo lahko uporabniki in naprave poizvedujejo po vsebini podatkovne zbirke.

Spodnji trojki predstavljata informacijo za dostop do dostopne točke SPARQL virov DBpedia ter Britanske narodne knjižnjice.

- `:endpoint rdfs:seeAlso <http://bnb.data.bl.uk/sparql>.`
- `:endpoint rdfs:seeAlso <http://dbpedia.org/sparql>.`

5.3.2 Dodajanje novih virov

Britanska narodna knjižnica [28] in DBpedia [30] nista edina prosto dostopna vira, kjer najdemo informacije o knjigah. Obstaja še vrsta drugih, ki jih lahko vključimo v našo aplikacijo. Edini pogoj za vključitev je, da ima vir izpostavljeno mesto, kamor lahko aplikacija pošlje poizvedbo SPARQL in pridobi rezultate le-te. Spletni naslov dostopne točke SPARQL moramo v slovar zapisati v obliki trojke:

- `:endpoint rdfs:seeAlso <SPARQLendpoint>.`

Prav tako moramo izraze novega vira integrirati z izrazi definiranimi v našem slovarju. To storimo na enak način kot pri integraciji virov DBpedia ter Britanska narodna knjižnica, torej v obliki ustreznih trojk. Tako lahko vsebino naše spletne knjižnice še dodatno obogatimo s tem enostavnim načinom dodajanja novih virov.

5.4 Postavitev strežnika Fuseki

Jedro naše aplikacije bo predstavljal strežnik Fuseki, saj bo skrbel za izvajanje poizvedb SPARQL in pravilno uporabo našega slovarja. Čeprav Fuseki ne omogoča dostopa do podatkov relacijskih baz, bomo v ta namen vpeljali platformo D2RQ ter mapirni dokument v sam strežnik Fuseki. Iz tega razloga bomo

uporabljali Fuseki 0.2.4, saj je platforma D2RQ nekompatibilna z novejšimi verzijami strežnika.

Za uporabo funkcionalnosti platforme D2RQ moramo strežniku Fuseki podati referenco na platformo D2RQ ter na krmilnik za dostop do relacijske baze Microsoft SQLExpress. Tako strežnik kot tudi platforma in krmilnik so zapisani v programskem jeziku Java. Zato je za pravilno referenciranje potrebno posodobiti t.i. "manifest" datoteke .jar, v kateri je zapakirana programska koda strežnika Fuseki. V ta namen bomo ustvarili datoteko z naslednjo vsebino:

```
Class-Path:  d2rq-0.8.1.jar sqljdbc4.jar
```

Nato bomo v ukazni lupini izvedli spodnji ukaz. Na ta način bomo v manifest strežnika Fuseki dodali referenci, ki ju potrebujemo za dostop do relacijske baze.

```
jar umf manifest.txt fuseki-server.jar
```

5.4.1 Konfiguracija strežnika Fuseki

Strežniku Fuseki lahko ob zagonu podamo vrsto parametrov, s katerimi določimo njegov način delovanja. Eden izmed teh parametrov omogoča, da ob zagonu podamo konfiguracijsko datoteko, zapisano v obliki RDF, s katero strežnik nastavimo glede na naše zahteve. Prav zaradi specifičnosti našega primera, ki zahteva uporabo slovarja, mapirnega dokumenta ter funkcionalnosti platforme D2RQ, se bomo tudi sami poslužili uporabe te možnosti. Vsebino konfiguracije bomo opisali po delih in zraven podali ustrezno kodo, medtem ko se celoten dokument nahaja v Dodatku C.

Najprej definiramo storitev z imenom `MODEL`, ki bo omogočala izvajanje poizvedb SPARQL ter dodajanje podatkov v podatkovno zbirko. To bomo dosegli s Konfiguracijo 5.1.

```
<#service> rdf:type fuseki:service;
      fuseki:name     "MODEL";
      fuseki:serviceQuery  "query";
      fuseki:serviceUpdate "update" ;
      fuseki:dataset   <#dataset>;
      .
```

Fuseki konfiguracija 5.1: Definiranje Fuseki storitve.

Podatkovna zbirka hrani informacije v obliki RDF in jo navajamo z `<#dataset>`. Sestavljena je iz oznake ter osnovnega grafa, ki ga definira referenca `<#model>`. Opisano predstavlja Konfiguracija 5.2.

```
<#dataset> rdf:type ja:RDFDataset;
      rdfs:label   "dataset for aplication";
      ja:defaultGraph <#model>;
      .
```

Fuseki konfiguracija 5.2: Definiranje podatkovne zbirke.

Gradnik `<#model>` je določen s posebnim tipom, ki mu pravimo ontologijski model. Ta skrbi za upoštevanje pravil in izrazov, definiranih v slovarjih oziroma ontologijah. To doseže skupaj z orodjem za sklepanje t.i. "reasoner", ki smo ga podali kot objekt po predikatu `ja:reasoner`. Sklepanje se bo izvajalo na podlagi specifikacije `RDF_MEM_RDFS_INF`, ki smo jo podali kot objekt, ki sledi predikatu `ja:ontModelSpec`. Omenjena specifikacija se uporablja za ontologijske modele RDFS, ki so shranjeni v pomnilniku in je nad njimi potrebno izvajati RDFS sklepanje. Uporabnost specifikacije `RDF_MEM_RDFS_INF` sovpada z našim načinom integracije tujih virov v slovar ter konfiguracijo strežnika Fuseki. Prav zaradi teh razlogov smo se tudi odločili za uporabo te specifikacije. Podatki, nad katerimi se bo sklepanje izvajalo, se nahajajo v osnovnem modelu, ki smo ga definirali v `<#rdfs_model>`. Konfiguracija 5.3 predstavlja definiranje ontologijskega modela za naš primer.

```
<#model> rdf:type ja:OntModel;
      ja:ontModelSpec ja:RDFS_MEM_RDFS_INF;
      rdfs:label  "Model for integration";
      ja:baseModel <#rdfs_model> ;
      ja:reasoner [ja:reasonerURL <http://jena.hpl.hp.
         com/2003/RDFSExptRuleReasoner>];
      .
```

Fuseki konfiguracija 5.3: Definiranje ontologijskega modela.

Model **<#rdfs_model>** je pravzaprav unija modelov, sestavljenih iz glavnega modela ter enega podmodela. V glavnem modelu se nahaja naš slovar, ki smo ga uvozili kot zunanjo datoteko. Z definicijo podmodela pa vpeljujemo funkcionalnost platforme D2RQ v strežnik Fuseki. V gradniku **<#relationalDatabase>** namreč podamo mapirni dokument, s katerim omogočimo dostop do podatkov relacijske baze. Poleg tega moramo podati še predpono za unikaten naslov podatkov. Vse to dosežemo s Konfiguracijo 5.4.

```
<#rdfs_model> rdf:type ja:UnionModel;
      ja:rootModel
      [
            a ja:MemoryModel;
            ja:content [ja:externalContent <file:
               DiplomaOntologyModel.n3>];
      ];
      ja:subModel <#relationalDatabase>
      .
<#relationalDatabase> a d2rq:D2RQModel;
   d2rq:mappingFile <mapping.ttl>;
   d2rq:resourceBaseURI <http://localhost:2020/
      resource/>;
   .
```

Fuseki konfiguracija 5.4: Vpeljava slovarja ter platforme D2RQ.

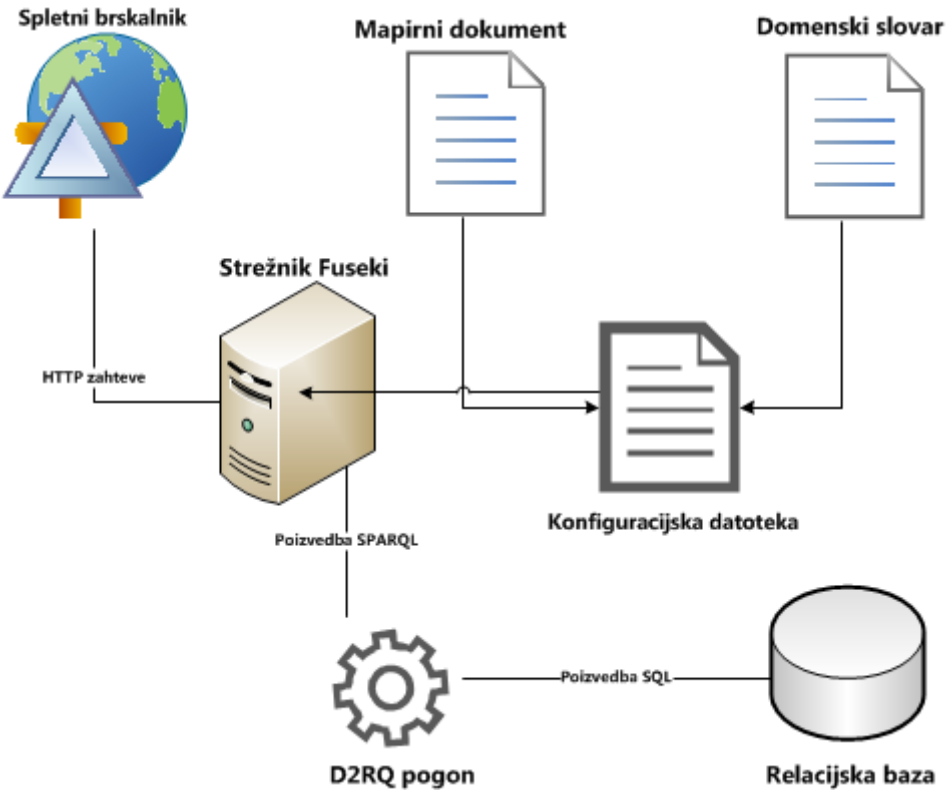
Definicije za gradnike `D2RQModel`, `mappingFile` ter `resourceBaseURI` se nahajajo na spletnem naslovu, ki ga določa predpona `d2rq:`. Vendar se je tekom procesa izdelave aplikacije izkazalo, da je bolje, če definicije za te gradnike vključimo v samo konfiguracijsko datoteko. Zato moramo, za uspešen dostop do podatkov relacijske baze preko platforme D2RQ, dodati Konfiguracijo 5.5.

```
d2rq:D2RQModel a rdfs:Class;
      rdfs:subClassOf ja:Model;
      rdfs:label "D2RQ model";
      rdfs:comment "Jena Assemler specification for
         RDB, mapped to RDF using the D2RQ tool.";
      ja:assembler "de.fuberlin.wiwiss.d2rq.assembler.
         D2RQAssembler";
      .

d2rq:mappingFile a rdf:Property;
      rdfs:label "Mapping file";
      rdfs:comment "URL of a D2RQ mapping file.";
      rdfs:domain d2rq:D2RQModel;
      .

d2rq:resourceBaseURI a rdf:Property;
      rdfs:label "Resource base URI";
      rdfs:comment "Base URI for resources generated
         by relative URI patterns.";
      rdfs:domain d2rq:D2RQModel;
      .
```

Fuseki konfiguracija 5.5: Definicija D2RQ gradnikov.

Postopek kreiranja konfiguracijske datoteke je tako končan in Slika 5.10 prikazuje celotno arhitekturo strežnika. Tako bo Fuseki s tem dokumentom, ki vključuje funkcionalnost platforme D2RQ, omogočal dostop do podatkov naše relacijske baze. Obenem pa bo tudi upošteval pravila integracije ter izraze zapisane v slovarju.

Poizvedbe SPARQL moramo strežniku Fuseki podati v obliki HTTP zahtev po REST arhitekturnem stilu. V ta namen bomo uporabili spletni brskalnik ter z njim dostopali do urejevalnika poizved, s katerim je opremljen Fuseki. Rezultate poizvedb bo strežnik vrnil brskalniku, ki jih bo nato prikazal na uporabniku prijazen način.

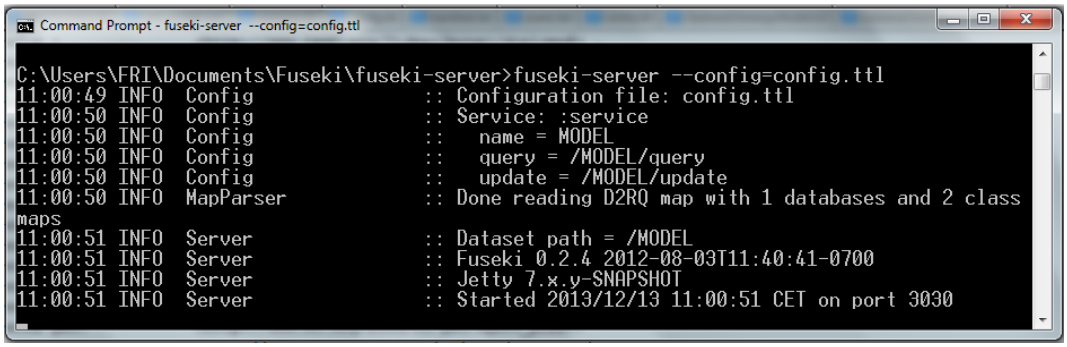


Slika 5.10: Diagram strežnika Fuseki.

5.4.2 Delovanje strežnika Fuseki

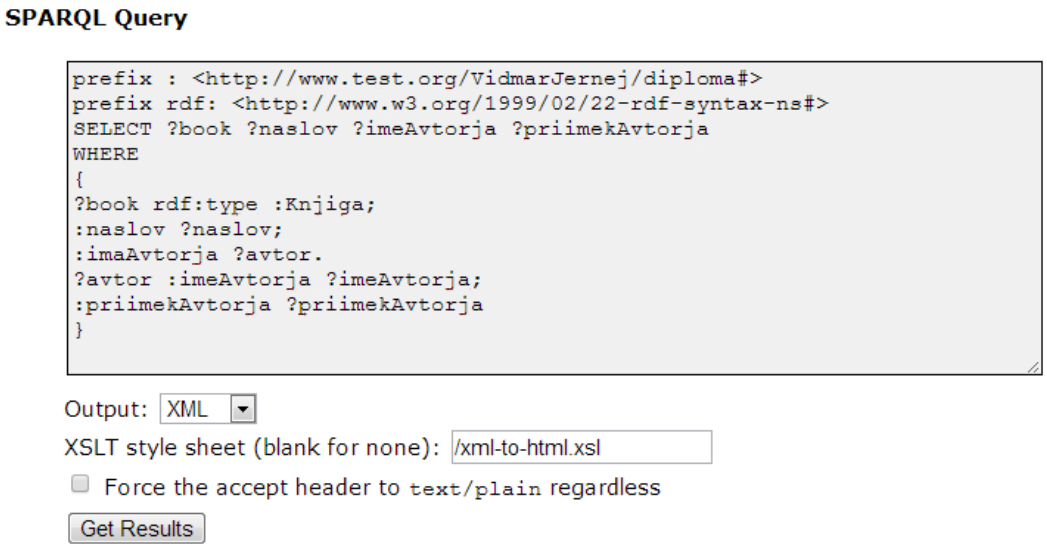
Strežnik Fuseki zaženemo v ukazni lupini z ukazom, ki mu kot vhodni parameter podamo konfiguracijsko datoteko. Primer zagona prikazuje Slika 5.11, iz katere je razvidno tudi branje mapirnega dokumenta. Prav tako vidimo, da se je strežnik inicializiral na privzetem naslovu `localhost:3030`, saj v konfiguraciji nismo specificirali drugače.

Poizvedbe SPARQL lahko izvajamo s pomočjo urejevalnika, ki ga strežnik izpostavi na naslovu `localhost:3030/sparql.tpl`. Fuseki lahko rezultate poizvedb vrne v različnih oblikah. Tako lahko izbiramo med formati `XML`, `JSON`, `CSV`, `TSV` ter tekstovno datoteko. Ob zahtevi za izvedbo spletni brskalnik pošlje strežniku HTTP zahtevo, v kateri je zakodirana naša poizvedba.



Slika 5.11: Zagon strežnika Fuseki.

Slika 5.13 prikazuje rezultate poizvedbe SPARQL na Sliki 5.12. Poizvedba uporablja le izraze, ki smo jih definirali v slovarju. Vendar smo kljub temu pridobili podatke iz relacijske baze, kar dokazuje pravilno delovanje strežnika Fuseki.



Slika 5.12: Pisanje poizvedb v urejevalniku strežnika Fuseki.

SPARQLer Query Results

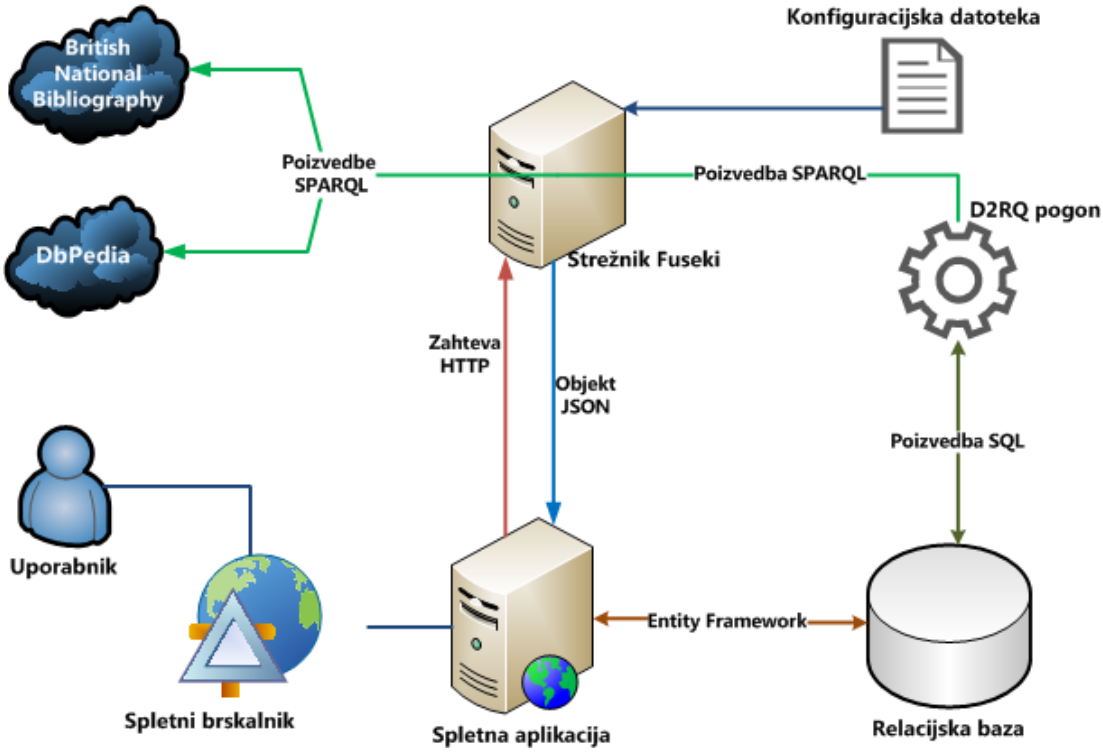
naslov	imeAvtorja	priimekAvtorja
"A game of thrones"	"George"	"R.R.Martin"
"Odiseja človeštva"	"Spencer"	"Wells"
"The curious incident of the dog in the night time"	"Mark"	"Haddon"
"Korak Zadaj"	"Henning"	"Manken"
"Alamut"	"Judith"	"Tarr"

Slika 5.13: Rezultat poizvedbe iz Slike 5.12.

S tem primerom uporabe smo prikazali pravilno konfiguracijo strežnika, s čimer smo naredili pomemben korak pri implementaciji našega prototipa osebne spletne knjižnice. Fuseki nam zdaj omogoča, da podatke iz relacijske baze obogatimo s podatki iz drugih virov in ob tem uporabljamo le izraze, definirane v našem slovarju. To pomeni pomembno prednost in predvsem generičnost za naš prototip.

5.5 Izdelava spletne aplikacije

Spletna aplikacija prototipa bo predstavljala vmesni člen med uporabnikom in strežnikom Fuseki. Skrbela bo za pošiljanje poizvedb strežniku in prikaz vrnjenih podatkov na uporabniku prijazen način. Z njo bo uporabnik urejal svojo osebno spletno knjižnico. Slika 5.14 predstavlja umestitev spletne aplikacije v relaciji z drugimi gradniki prototipa. Obenem Slika 5.14 prikazuje celotno sestavo osebne spletne knjižnjice ter tudi način komunikacije med posameznimi deli.



Slika 5.14: Diagram prototipa osebne spletne knjižnice.

Aplikacijo bomo izdelali v programskem okolju Visual Studio 2010 in pri tem upoštevali programski vzorec Model – View - Controller. Na strani spletnega strežnika bomo uporabili programski jezik C#, medtem ko bomo na strani uporabnika uporabili označevalni jezik HTML, skriptni jezik JavaScript ter jezik

CSS. Na tem mestu naj dodamo, da smo za obliko in izgled spletne aplikacije uporabili javno knjižnico Bootstrap [32].

Čeprav lahko aplikacija dostopa do relacijske baze s pomočjo strežnika Fuseki, bomo implementirali tudi neposreden dostop do baze. V ta namen bomo uporabili ogrodje Entity Framework [33], ki je priporočena tehnologija za podatkovni dostop do relacijskih baz s strani Microsofta.

Komunikacija s strežnikom Fuseki bo potekala z uporabo spletnih zahtev oziroma zahtev HTTP. Za namen generiranja teh zahtev bomo ustvarili metodi `QueryFuseki` ter `UpdateFuseki`. Metoda `QueryFuseki` kot vhodni parameter sprejme poizvedbo SPARQL, s katero nato kreira zahtevo. Fuseki vrne rezultat poizvedbe v obliki objekta JSON [34], ki ga metoda sprejme in vrne aplikaciji. Za razliko od `QueryFuseki`, metoda `UpdateFuseki` ne poizveduje po podatkih, ampak jih pošilja. Z njo bomo v strežnik Fuseki shranili nove podatke, ki jih bomo pridobili iz drugih virov. Podatki bodo zapisani v obliki trojk in jih bomo podali kot vhodni parameter metodi. Strežnik Fuseki bo na poslano zahtevo odgovoril z opisom o uspešni ali neuspešni izvedbi. Implementacijo metode `QueryFuseki` predstavlja Koda 5.1, medtem ko Koda 5.2 predstavlja implementacijo metode `UpdateFuseki` .

```
public static JObject QueryFuseki(string query)
{
      JObject Jsondata = new JObject();
      try
      {
            var request =
            (HttpWebRequest)WebRequest.Create(query);
            request.Method = WebRequestMethods.Http.Get;
            request.Accept = "application/json";
            var response =
            (HttpWebResponse)request.GetResponse();
            using (var streamreader = new
                StreamReader(response.GetResponseStream()))
            {
```

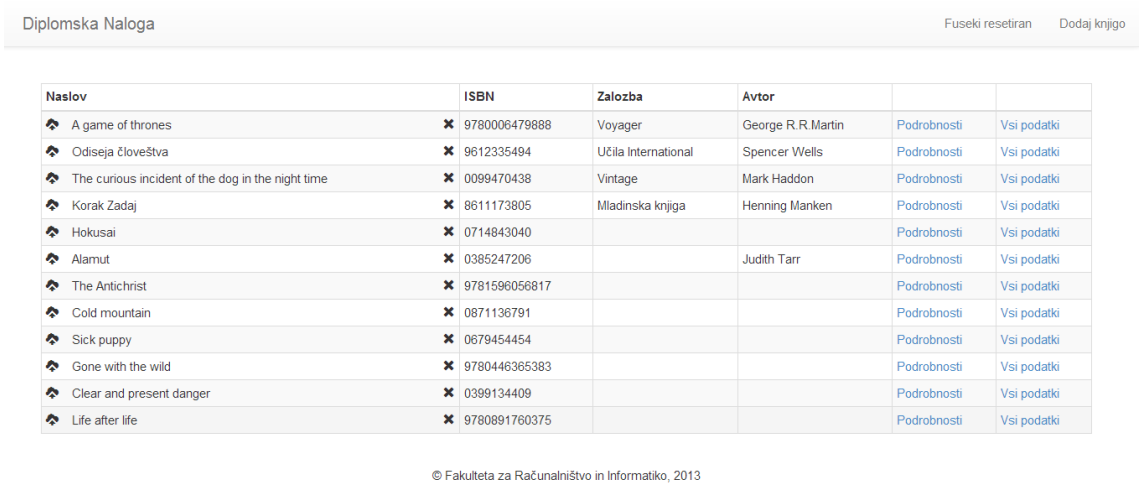
```
                    var responseText =
                        streamreader.ReadToEnd();
                    Jsondata =
                    (JObject)JsonConvert.DeserializeObject(responseText);
                }
                return Jsondata;
            }
            catch { return Jsondata; }
        }
```

C# koda 5.1: Implementacija metode `QueryFuseki`.

```
        public static string UpdateFuseki(string query)
        {
            try
            {
            ASCIIEncoding encoding = new ASCIIEncoding();
            byte[] data = encoding.GetBytes(query);
            var request =
            (HttpWebRequest)WebRequest.Create("http://localhost:3030/MODEL/update");
            request.Method = WebRequestMethods.Http.Post;
            request.ContentLength = data.Length;
            using (var stream = request.GetRequestStream())
            { stream.Write(data, 0, data.Length); }
            var response =
                (HttpWebResponse)request.GetResponse();

            return
                !String.IsNullOrEmpty(response.StatusDescription)
                ? response.StatusDescription : String.Empty;
            }
        catch (WebException e)
        {
            return String.Empty;
        }
        }
```

C# koda 5.2: Implementacija metode `UpdateFuseki`.

Spletna aplikacija bo sestavljena iz večih spletnih strani, ki bodo ponujale različne funkcionalnosti. Osnovna oziroma prva stran osebne spletne knjižnice je prikazana na Sliki 5.15. Stran vsebuje prikaz vseh podatkov, ki jih za posamezno knjigo hranimo v relacijski bazi. Uporabnik ima za vsako knjigo na voljo štiri funkcionalnosti. S klikom na križec sprožimo izbris izbrane knjige iz baze. Klik na ikono oblačka s puščico sproži proces pridobitve dodatnih informacij o izbrani knjigi. Urejen prikaz informacij o knjigi se nam prikaže s klikom na povezavo "Podrobnosti", medtem ko povezava "Vsi podatki" prikaže vse informacije o knjigi v obliki trojk. Poleg tega ima uporabnik v vrstici menija na voljo dve dodatni funkciji. Tako lahko s klikom na "Dodaj knjigo" doda novo knjigo v relacijsko podatkovno bazo. S klikom na "Fuseki resetiran" pa sporoči aplikaciji, da je prišlo do ponovnega zagona strežnika Fuseki. Sledi podrobnejši opis posameznih funkcionalnosti.



Slika 5.15: Osnovna stran osebne spletne knjižnice.

5.5.1 Dodajanje nove knjige

Uporabnik lahko doda novo knjigo preko zaslonske maske, ki jo prikazuje Slika 5.16. Vpis podatkov v vnosna polja je neobvezen, z izjemo polja ISBN. S

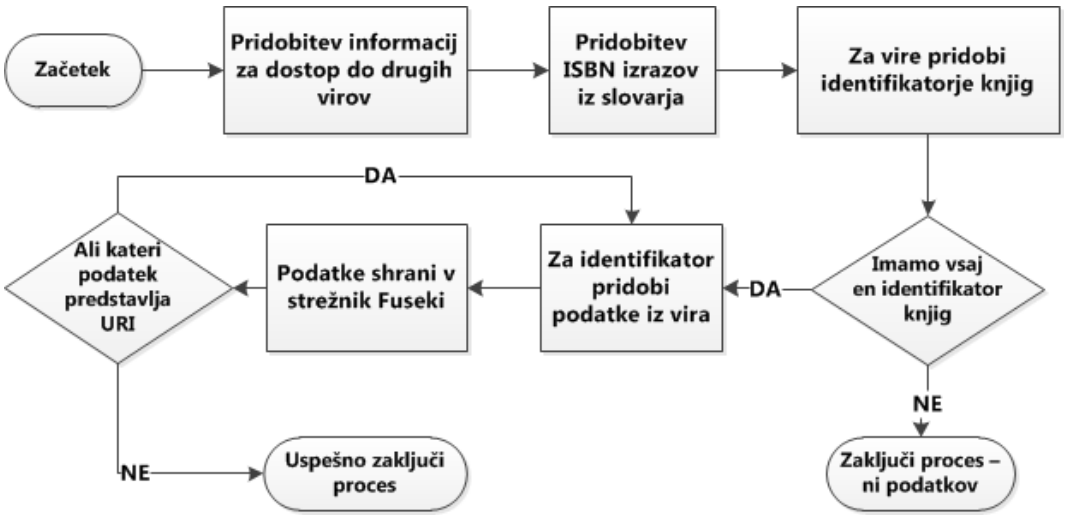
pomočjo številke ISBN aplikacija poišče informacije o knjigi iz drugih virov. Knjigo shranimo s pritiskom na gumb "Shrani". Pri tem aplikacija s pomočjo ogrodja Entity Framework shrani vpisane podatke v relacijsko bazo. Uporaba platforme D2RQ je na tem mestu nemogoča, saj platforma ne podpira pisanja v relacijsko bazo, ampak le branje iz nje. Aplikacija nas po uspešni shranitvi knjige preusmeri na osnovno stran oziroma nas obvesti, če je prišlo do napake.

Strežnik Fuseki je potrebno ponovno zagnati, ko uspešno dodamo novo knjigo v relacijsko bazo. Razlog za to se skriva v načinu vpeljave platforme D2RQ v strežnik Fuseki. Povezava med obema gradnikoma ne omogoča t.i. "žive" poizvedbe po podatkih relacijske baze. Tako se ob izvedbi prve poizvedbe na strežnik Fuseki prenesejo podatki iz relacijske baze. Iz tega razloga je potrebno strežnik ponovno zagnati, če želimo dostopati do novo dodanih podatkov. Prav tako moramo aplikaciji sporočiti, da je prišlo do ponovnega zagona strežnika Fuseki, kar storimo s klikom na gumb "Fuseki resetiran" v vrstici menija.

Slika 5.16: Zaslonska maska "Dodajanje nove knjige".

5.5.2 Pridobitev dodatnih informacij o knjigi

Proces pridobitve dodatnih informacij sprožimo s klikom na ikono oblačka s puščico, ki se nahaja poleg naslova knjige. Cilj procesa je iz drugih virov poiskati dodatne informacije o knjigi in jih nato prenesti v strežnik Fuseki. Knjige iz drugih virov bomo na podlagi številke ISBN povezali s knjigo iz relacijske baze in tako zagotovili avtentičnost podatkov. Slika 5.17 prikazuje potek procesa po korakih, ki jih bomo v nadaljevanju tudi opisali.



Slika 5.17: Diagram procesa pridobitve dodatnih informacij.

V prvem koraku procesa pridobimo spletne naslove dostopnih točk SPARQL, ki omogočajo dostop do drugih virov. Strežniku Fuseki podamo Poizvedbo 5.2 in ta nam vrne naslove, ki so zapisani v slovarju.

```
SELECT ?endpoint
WHERE
{
        :endpoint rdfs:seeAlso ?endpoint
}
```

Poizvedba 5.2: Pridobitev informacij za dostop do virov.

Nato moramo pridobiti izraze, s katerimi so določene številke ISBN v drugih virih. Prav s tem namenom smo integrirali te izraze v slovar kot podlastnost izraza `:idKnjige`. Tako nam ob izvedbi Poizvedbe 5.3 Fuseki vrne vse integirane izraze.

```
SELECT ?isbnEquivalent
WHERE
{
        ?isbnEquivalent rdfs:subPropertyOf :idKnjige.
}
```

Poizvedba 5.3: Pridobitev izrazov za ISBN iz slovarja.

V naslednjem koraku strežniku Fuseki podamo poizvedbo, s katero bo dostopal do posameznih virov. Pri tem bo vire povprašal po knjigi, ki ustreza podanemu pogoju. Primer pogoja, kjer iščemo knjigo, ki ustreza ISBN številki 0399134409, prikazuje Poizvedba 5.4. Razlog za večje število pogojev, ki na prvi pogled izgledajo enaki, je posledica nestandardiziranosti podatkov. Kot smo že omenili, različni viri predstavljajo isti podatek z različnimi izrazi. Prav tako pa znotraj istega vira podatki niso vedno zapisani v enakem formatu. Zato moramo imeti za primer vira DBpedia podatek o številki ISBN zapisan v različnih oblikah.

```
SELECT ?BookUri WHERE {SERVICE <endpoint> {
SELECT ?BookUri WHERE {
        {?BookUri <http://dbpedia.org/property/isbn> "0-399-13440-9"@en}
        UNION
        {?BookUri <http://dbpedia.org/property/isbn> "0-39913-440-9"@en}
        UNION
        {?BookUri <http://dbpedia.org/ontology/isbn> "0-399-13440-9"@en}
        UNION
        {?BookUri <http://dbpedia.org/ontology/isbn> "0-39912-440-9"@en}
        UNION
        {?BookUri <http://purl.org/ontology/bibo/isbn10> "0399134409"}
        UNION
```

```
        {?BookUri <http://purl.org/ontology/bibo/isbn13> "0399134409"}
        UNION
        {?BookUri <http://purl.org/ontology/bibo/isbn> "0399134409"}
    }}}
```

Poizvedba 5.4: Pridobitev identifikatorjev knjig iz drugih virov.

Pravzaprav naša poizvedba povprašuje vir po podatku, ki ustreza kateremu izmed podanih pogojev. Podatek, ki ga vir vrne, predstavlja identifikator knjige, ki ima enak ISBN kot knjiga, za katero iščemo nove podatke. Tako lahko z veliko gotovostjo sklepamo, da gre za enaki knjigi. Številka ISBN zagotavlja, da dve različni knjigi ne moreta imeti enake številke.

V koraku, ki sledi, bomo na podlagi pridobljenih identifikatorjev poiskali dodatne informacije o knjigi. Posameznemu viru bomo podali identifikator, ki ga je vrnil, in zahtevali vse informacije, ki jih hrani za to knjigo. To bomo dosegli s Poizvedbo 5.5, ki jo pošljemo strežniku Fuseki. Rezultat poizvedbe so trojke, katerih subjekt je enak podanemu identifikatorju, vrednosti predikata ter objekta pa sta poljubni.

```
SELECT ?p ?o
WHERE
{SERVICE <endpoint>
        {
                SELECT ?p ?o
                WHERE
                {
                        <Identificator> ?p ?o.
                }
        }
}
```

Poizvedba 5.5: Pridobitev dodatnih informacij o knjigi.

Nato informacije, ki jih je vir vrnil, s pomočjo Poizvedbe 5.6 shranimo na strežnik Fuseki. V primeru, da objekt trojke predstavlja `URI`, ga podamo Poizvedbi 5.5 kot identifikator. Vir nato vrne podatke, ki jih prav tako shranimo

v Fuseki. Na tak način pridobimo, poleg informacij o knjigi, tudi informacije o avtorju, konceptih in drugih stvareh, ki so povezani s knjigo.

```
INSERT DATA { <Identificator> ?p ?o. }
```

Poizvedba 5.6: Dodajanje novih informacij v strežnik Fuseki.

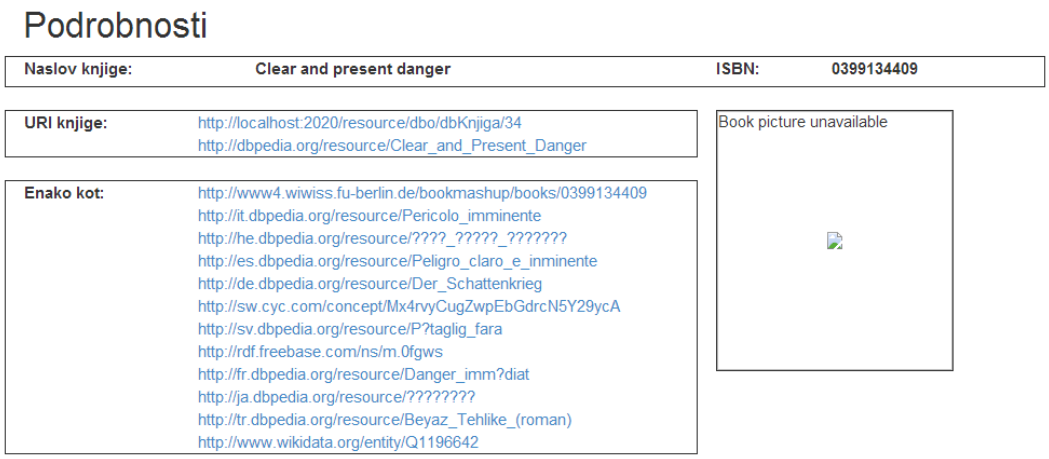
Proces pridobitve podatkov se uspešno zaključi, če se podatki iz virov uspešno shranijo v strežnik Fuseki. V tem primeru aplikacija odstrani ikono za sprožitev in s tem uporabnika opozori o končanju procesa. Posamezna knjiga bo zdaj imela poleg osnovnih informacij na voljo še vrsto drugih, ki jih bo lahko ponudila uporabniku. Na tem mestu naj dodamo, da se ikona ne odstrani v primeru, ko za izbrano knjigo nismo našli nobenih podatkov iz drugih virov.

5.5.3 Podrobnosti o knjigi

Aplikacija omogoča uporabniku urejen prikaz informacij o posamezni knjigi. V ta namen smo definirali pogled, ki bo prikazal le v slovarju določene izraze. Prav tako bomo podatke za pogled pridobili samo z uporabo teh izrazov. Strežnik Fuseki smo nastavili tako, da se nad vsemi podatki, ki jih hrani, izvaja sklepanje z uporabo slovarja. S tem razlogom smo tudi pridobljene informacije iz drugih virov shranili v Fuseki.

Pogled je sestavljen iz prikaza informacij o knjigi ter informacij o njenem avtorju. Podrobnosti o knjigi so razdeljene v dva dela. Prvi del, ki ga prikazuje Slika 5.18, vsebuje naslov knjige, njeno ISBN številko, vrsto spletnih povezav ter sliko knjige, če ta obstaja. Spletni naslovi “URI knjige” predstavljajo identifikatorje knjige iz relacijske baze in virov, ki smo jih vključili v naš slovar. Medtem ko naslovi “Enako kot” prav tako prikazujejo identifikatorje knjige, vendar za vire, ki jih nismo integrirali.

Drugi del predstavlja Slika 5.19 in vsebuje bolj podrobne informacije o knjigi. Večino teh informacij pridobimo iz prosto dostopnih virov. Vendar ni nujno, da ti podatki obstajajo za vse knjige. Tako lahko za slabše dokumentirane knjige



Slika 5.18: Zaslonska maska "Podrobnosti o knjigi" - prvi del.

ti podatki manjkajo. To lahko predstavlja eno izmed manjših slabosti uporabe prosto dostopnih zbirk kot vir informacij.

Slika 5.19: Zaslonska maska "Podrobnosti o knjigi" - drugi del.

Informacije za prikaz podrobnosti o knjigi pridobimo s Poizvedbo 5.7, ki jo podamo strežniku Fuseki. Strežnik na podlagi številke ISBN poišče knjigo in njene podatke. Pri tem uporabljamo samo izraze iz slovarja. Za vsak posamezen podatek uporabimo rezervirano SPARQL besedo `OPTIONAL`, ki v primeru manjkajočega podatka še vedno vrne ostale.

```
SELECT
        ?knjiga ?naslov ?isbn ?avtor ?contributors ?desc
        ?extent ?publishedBy ?lang ?publishDate ?content
        ?abstract ?mediaType ?publishCountry ?thumbnail ?concept
WHERE
{
        ?knjiga :idKnjige "ISBN";
        rdf:type :Knjiga.
        OPTIONAL{?knjiga :naslov ?naslov.}
        OPTIONAL{?knjiga :opisKnjige ?desc.
                FILTER(langMatches(lang(?desc), "EN"))}
        OPTIONAL{?knjiga :povzetekKnjige ?abstract.
                FILTER(langMatches(lang(?abstract), "EN"))}
        OPTIONAL{?knjiga :jezikKnjige ?lang. }
        OPTIONAL{?knjiga :obsegKnjige ?extent.}
        OPTIONAL{?knjiga :vsebinaKnjige ?content.
                FILTER(langMatches(lang(?content), "EN"))}
        OPTIONAL{?knjiga :izdalKnjigo ?publishedBy. }
        OPTIONAL{?knjiga :datumIzdaje ?publishDate. }
        OPTIONAL{?knjiga :oblikaKnjige ?mediaType. }
        OPTIONAL{?knjiga :drzavaIzdajeKnjige ?publishCountry. }
        OPTIONAL{?knjiga :slikaKnjige ?thumbnail.}
        OPTIONAL{?knjiga :imaAvtorja ?avtor.}
        OPTIONAL{?knjiga :soPrispevali ?contributors.}
        OPTIONAL{?knjiga :imaKoncept ?concept}
}
```

Poizvedba 5.7: Pridobitev informacij o knjigi.

Prikaz podrobnosti o avtorju vsebuje nekaj osnovnih informacij, kratek življenjepis, njegovo sliko ter seznam avtorjevih knjig. Tudi tu se večina podatkov pridobi iz prosto dostopnih virov. S pomočjo Poizvedbe 5.8 pridobimo informacije o avtorju, s Poizvedbo 5.9 pa poiščemo druge avtorjeve knjige. Avtorjev identifikator oziroma `URI` smo pridobili iz Poizvedbe 5.7, in sicer iz pogoja `?knjiga :imaAvtorja ?avtor`. V primeru, da ima knjiga druge osebe, ki so prispevali pri ustvarjanju, prikažemo tudi podrobnosti o njih. Slika 5.20

predstavlja zaslonsko masko za prikaz podrobnosti o avtorju.

```
SELECT
        ?ime ?priimek ?about ?slika
        ?drzavljanstvo ?rojstvo ?kraj
WHERE
{
        <AvtorjevURI> rdf:type :Avtor.
        OPTIONAL{<AvtorjevURI> :imeAvtorja ?ime.}
        OPTIONAL{<AvtorjevURI> :priimekAvtorja ?priimek.}
        OPTIONAL{<AvtorjevURI> :oAvtorju ?about.
                FILTER(langMatches(lang(?about), "EN"))}
        OPTIONAL{<AvtorjevURI> :slikaAvtorja ?slika.}
        OPTIONAL{<AvtorjevURI> :drzavljanstvoAvtorja ?
            drzavljanstvo.}
        OPTIONAL{<AvtorjevURI> :rojstvoAvtorja ?rojstvo.}
        OPTIONAL{<AvtorjevURI> :krajRojstva ?kraj.}
}
```

Poizvedba 5.8: Pridobitev informacij o avtorju.

```
SELECT ?knjiga ?naslov ?isbn
WHERE
{
        ?knjiga :imaAvtorja <AVtorjevURI>.
        OPTIONAL{?knjiga :naslov ?naslov.}
        OPTIONAL{?knjiga :idKnjige ?isbn.}
}
```

Poizvedba 5.9: Pridobitev drugih avtorjevih knjig.

Slika 5.20: Zaslonska maska "Podrobnosti o avtorju".

Pregled podrobnosti uporabniku prikaže razliko med začetno količino podatkov o knjigi in količino po pridobitvi podatkov iz drugih virov. Tako spoznamo prednost takšne oblike aplikacije, ki za vir informacij uporablja prosto dostopne zbirke. Prav tako lahko pregled še dodatno razširimo z novimi informacijami, ki smo jim predhodno definirali izraze v slovarju. Na tak način lahko zgradimo urejeno, podrobno in z informacijami bogato stran, ki uporabnikom predstavi knjigo.

5.5.4 Vsi podatki o knjigi

Tekom procesa pridobitve podatkov iz drugih virov prenesemo v strežnik Fuseki veliko več informacij, kot pa jih nato prikažemo s pogledom podrobnosti o knjigi. S tem namenom aplikacija ponuja dodaten pogled, kjer si uporabnik lahko ogleda vse podatke o knjigi in njenem avtorju. Tako s pomočjo tabele prikažemo podatke v enaki obliki, kot so predstavljeni v strežniku Fuseki, torej v obliki trojk. Sliki 5.21 ter 5.22 predstavljata zaslonsko masko za prikaz nerafiniranih podatkov o posamezni knjigi.

O knjigi

Subjekt	Predikat	Objekt
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/property/oclc	"19845912"^^http://www.w3.org/2001/XMLSchema#integer
http://dbpedia.org/resource/Clear_and_Present_Danger	http://www.w3.org/2000/01/rdf-schema#label	"Clear and Present Danger"@en
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/ontology/series	<http://dbpedia.org/resource/Jack_Ryan_(character)>
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/property/mediaType	"Print"@en
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/ontology/oclc	"19845912"@en
http://dbpedia.org/resource/Clear_and_Present_Danger	http://www.w3.org/2002/07/owl#sameAs	<http://de.dbpedia.org/resource/Der_Schattenkrieg>
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/property/isbn	"0399134409"
http://dbpedia.org/resource/Clear_and_Present_Danger	http://xmlns.com/foaf/0.1/isPrimaryTopicOf	<http://en.wikipedia.org/wiki/Clear_and_Present_Danger>
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/ontology/numberOfPages	"656"^^http://www.w3.org/2001/XMLSchema#integer
http://dbpedia.org/resource/Clear_and_Present_Danger	http://dbpedia.org/ontology/author	<http://dbpedia.org/resource/Tom_Clancy>

First Previous 1 2 3 4 5 Next Last

Slika 5.21: Zaslonska maska "Vsi podatki o knjigi".

Podatke o knjigi pridobimo s pomočjo Poizvedbe 5.10. Poizvedba na podlagi številke ISBN poišče knjigo ter nato vrne vse podatke, saj nismo podali nobenih pogojev. Na podoben način deluje Poizvedba 5.11, s katero pridobimo podatke o avtorju. Tu najprej poiščemo knjigo na podlagi številke ISBN, nato povprašamo po njenem avtorju ter na koncu vrnemo vse podatke o njem.

O avtorju

Subjekt	Predikat	Objekt
http://dbpedia.org/resource/Tom_Clancy	http://dbpedia.org/property/occupation	"Novelist"@en
http://dbpedia.org/resource/Tom_Clancy	http://dbpedia.org/property/alternativeNames	"Clancy, Thomas Leo, Jr."@en
http://dbpedia.org/resource/Tom_Clancy	http://dbpedia.org/ontology/wikiPageID	"30265"^^http://www.w3.org/2001/XMLSchema#integer
http://dbpedia.org/resource/Tom_Clancy	http://dbpedia.org/property/dateOfBirth	"1947"^^http://www.w3.org/2001/XMLSchema#integer
http://dbpedia.org/resource/Tom_Clancy	http://purl.org/dc/terms/subject	<http://dbpedia.org/resource/Category:People_from_Calvert_County,_Maryland>
http://dbpedia.org/resource/Tom_Clancy	http://purl.org/dc/terms/subject	<http://dbpedia.org/resource/Category:Living_people>
http://dbpedia.org/resource/Tom_Clancy	http://purl.org/dc/terms/subject	<http://dbpedia.org/resource/Category:Writers_from_Maryland>
http://dbpedia.org/resource/Tom_Clancy	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	<http://umbel.org/umbel/rc/Artist>
http://dbpedia.org/resource/Tom_Clancy	http://dbpedia.org/ontology/birthPlace	<http://dbpedia.org/resource/Maryland>
http://dbpedia.org/resource/Tom_Clancy	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	<http://dbpedia.org/class/yago/LivingPeople>

First Previous 3 4 5 6 7 Next Last

Slika 5.22: Zaslonska maska "Vsi podatki o avtorju".

```
SELECT ?s ?p ?o
WHERE
{
        ?s :idKnjige "ISBN";
                ?p ?o.
}
```

Poizvedba 5.10: Pridobitev vseh podatkov o knjigi.

```
SELECT ?s ?p ?o
WHERE
{
        ?book :idKnjige "ISBN";
                        :imaAvtorja ?s.
        ?s ?p ?o.
}
```

Poizvedba 5.11: Pridobitev vseh podatkov o avtorju.

Poglavje 6

Zaključek

Na začetku diplomske naloge smo si zadali dva glavna cilja. Prvi je bil doseči preslikavo podatkov iz relacijske baze v obliko RDF. Za dosego tega cilja smo uspešno uporabili platformo D2RQ, ki je podatke naše baze izpostavila semantičnemu spletu v RDF obliki. Drugi cilj je bil obogatiti izpostavljene podatke z dostopom do informacij prosto dostopnih virov. V ta namen smo izdelali slovar, s katerim smo integrirali druge vire informacij v naš prototip. Slovar smo nato vpeljali v strežnik Fuseki, ki smo mu dodali tudi funkcionalnost platforme D2RQ. S tem smo dosegli uresničitev drugega cilja, ker pa je uporaba strežnika Fuseki brez uporabniškega vmesnika povsem nepraktična, smo izdelali tudi spletno aplikacijo. Z njo uporabniku prikažemo prednost uporabe prosto dostopnih zbirk kot vir informacij, seveda ob predpostavki izpostavitve podatkov iz relacijske baze. Obenem pa ponudimo aplikacijo, s katero lahko uporabnik ureja osebno spletno knjižnico in pridobi dodatne informacije o knjigi in njenem avtorju.

Zastavljene cilje smo torej dosegli in ob tem uspešno implementirali začetno konceptualno idejo. Vendar prostor za izboljšavo obstaja. Tako bi spletno aplikacijo lahko dodelali in pri tem vpeljali avtentikacijo uporabnika ter ponudili več funkcionalnosti. A kljub temu v trenutni obliki zadošča svojemu namenu, prav tako cilj diplomske naloge ni bil razvoj spletne aplikacije. Glavna

pomankljivost, ki jo vidimo v razvitem prototipu, je predvsem potreba po vnovičnem zagonu strežnika Fuseki ob spreminjanju in dodajanju podatkov v relacijsko bazo. S tem, ko ponovno zaženemo strežnik, izgubimo podatke, ki smo jih predhodno že prenesli iz drugih virov. Tako moramo za vsako knjigo ponovno sprožiti proces pridobivanja dodatnih informacij iz prosto dostopnih zbirk. Shranjevanje pridobljenih informacij iz drugih virov v datoteko ali podatkovno zbirko TDB [35] bi lahko rešilo nastali problem. Kljub temu pa bi ostal problem ponovnega zagona strežnika Fuseki ob spremembi relacijske baze, kar je posledica same konfiguracije strežnika. Če želimo nad podatki uporabljati sklepanje s pomočjo slovarja, moramo podatke hraniti v strežniku. Posledično to pomeni, da podatki v strežniku ter relacijski bazi ne bodo sinhronizirani. Vendar brez uporabe slovarja ne moremo učinkovito pridobiti informacije iz drugih virov. Gre za neke vrste salomonsko odločitev, ki prinaša tako prednosti kot tudi slabosti.

Zanimanje za področje RDB2RDF obstaja, kar dokazuje barvita paleta orodij, ki že omogočajo to funkcionalnost. V razvoju pa se bodo zagotovo odpravile tudi težave, s katerim smo se srečali. Tako lahko rečemo, da imamo na voljo orodja za dosego popolne integracije podatkov na področju spleta. Potrebno je le še prepričati razvijalce v izpostavitev podatkov relacijskih baz semantičnemu spletu v obliki RDF. Na ta način bo semantičen splet zares zaživel in popeljal svetovni splet v prihodnost.

Literatura

[1] Scott Brinker, "SEO + Semantic Web = SEO++"
Dostopno na naslovu: http://chiefmartec.com/2008/03/seo-semantic-we/

[2] World Wide Web Consortium
Dostopno na naslovu: http://www.w3.org/

[3] Virtualna knjižnica Slovenije
Dostopna na naslovu: http://www.cobiss.si/scripts/cobiss?ukaz=GETID

[4] RDB2RDF Full-Day Tutorial
Dostopno na naslovu: http://www.rdb2rdf.org/eswc2013-tutorial/

[5] RDF to RDF Mapping Language
Dostopno na naslovu: http://www.w3.org/TR/r2rml/

[6] A Direct Mapping of relational data to RDF
Dostopno na naslovu: http://www.w3.org/TR/rdb-direct-mapping/

[7] Introduction to R2RML
Dostopno na naslovu: http://www.w3.org/TR/r2rml/#introduction

[8] RDB2RDF implementation report
Dostopno na naslovu: http://www.w3.org/2001/sw/rdb2rdf/wiki/Implementations

[9] Ultrawrap
Dostopno na naslovu: http://www.capsenta.com/product/ultrawrap/

[10] Capsenta
Dostopno na naslovu: http://www.capsenta.com/company/

[11] D2RQ: Accessing Relational Databases as Virtual RDF Graphs
Dostopno na naslovu: http://d2rq.org/

[12] Apache License
Dostopno na naslovu: http://www.apache.org/licenses/LICENSE-2.0.html

[13] Apache Jena: Java framework for building Semantic Web and Linked data applications
Dostopno na naslovu: http://jena.apache.org/

[14] Virtuoso Universal Server
Dostopno na naslovu: http://virtuoso.openlinksw.com/

[15] OpenLink Software
Dostopno na naslovu: http://www.openlinksw.com/company/

[16] Mapping Relational Data to RDF with Virtuoso's RDF Views
Dostopno na naslovu: http://virtuoso.openlinksw.com/whitepapers/relational rdf views mapping.html

[17] T.Berners-Lee, J.Hendler, O. Lassila, "Scientific American: Feature Article: The Semantic Web", May 2001

[18] RDF working group, "Resource Description Framework (RDF)", Februar 2004
Dostopno na naslovu: http://www.w3.org/RDF/

[19] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf and Jim HendlerR (2008). N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming, 8, pp 249-269
Dostopno na naslovu: http://dx.doi.org/10.1017/S1471068407003213

[20] RDF/XML Syntax Specification
Dostopno na naslovu: http://www.w3.org/TR/rdf-syntax-grammar/

[21] Fuseki: serving RDF data over HTTP
Dostopno na naslovu: http://jena.apache.org/documentation/serving_data/

[22] Protégé
Dostopno na naslovu: http://protege.stanford.edu/overview/

[23] Model - View - Controler
Dostopno na naslovu: http://en.wikipedia.org/wiki/Model-view-controller

[24] Microsoft® SQL Server® 2008 R2
Dostopno na naslovu: http://www.microsoft.com/en-us/download/details.aspx?id=30438

[25] The D2RQ Mapping Language
Dostopno na naslovu: http://d2rq.org/d2rq-language

[26] D2R Server: Accessing databases with SPARQL and as Linked Data
Dostopno na naslovu: http://d2rq.org/d2r-server

[27] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications.Knowledge Acquisition, 5(2):199-220, 1993
Dostopno na naslovu: http://tomgruber.org/writing/ontolingua-kaj-1993.htm

[28] The British National Bibliography
Dostopno na naslovu: http://bnb.data.bl.uk/

[29] The British National Bibliography: Book data model
Dostopno na naslovu: http://www.bl.uk/bibliographic/pdfs/bldatamodelbook.pdf

[30] DBpedia
Dostopna na naslovu: http://dbpedia.org/About

[31] SPARQL 1.1 Protocol
Dostopno na naslovu: http://www.w3.org/TR/sparql11-protocol/

[32] Bootstrap
Dostopno na naslovu: http://getbootstrap.com/

[33] Entity Framework
Dostopno na naslovu: http://msdn.microsoft.com/en-us/data/ef.aspx

[34] JSON
Dostopno na naslovu: http://en.wikipedia.org/wiki/JSON

[35] TDB, JENA ccomponent
Dostopno na naslovu: http://jena.apache.org/documentation/tdb/

Dodatek A

Mapirni dokument

```
@prefix map: <#> .
@prefix db: <> .
@prefix vocab: <vocab/> .
@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 d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .

map:database a d2rq:Database;
      d2rq:jdbcDriver "com.microsoft.sqlserver.jdbc.SQLServerDriver";
      d2rq:jdbcDSN "jdbc:sqlserver://NYMERON-PC\\SQLEXPRESS_FRI;
                              databaseName=D2RQ_Test";
      d2rq:username "sa";
      d2rq:password "sasa";
      .

# Table dbo.dbAvtor
map:dbo_dbAvtor a d2rq:ClassMap;
      d2rq:dataStorage map:database;
```

```
      d2rq:uriPattern "dbo/dbAvtor/@@dbo.dbAvtor.IDAvtor@@";
      d2rq:class vocab:dbo_dbAvtor;
      d2rq:classDefinitionLabel "dbo.dbAvtor";
      .
map:dbo_dbAvtor__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbAvtor;
      d2rq:property rdfs:label;
      d2rq:pattern "dbAvtor #@@dbo.dbAvtor.IDAvtor@@";
      .
map:dbo_dbAvtor_IDAvtor a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbAvtor;
      d2rq:property vocab:dbo_dbAvtor_IDAvtor;
      d2rq:propertyDefinitionLabel "dbAvtor IDAvtor";
      d2rq:column "dbo.dbAvtor.IDAvtor";
      d2rq:datatype xsd:integer;
      .
map:dbo_dbAvtor_Ime a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbAvtor;
      d2rq:property vocab:dbo_dbAvtor_Ime;
      d2rq:propertyDefinitionLabel "dbAvtor Ime";
      d2rq:column "dbo.dbAvtor.Ime";
      .
map:dbo_dbAvtor_Priimek a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbAvtor;
      d2rq:property vocab:dbo_dbAvtor_Priimek;
      d2rq:propertyDefinitionLabel "dbAvtor Priimek";
      d2rq:column "dbo.dbAvtor.Priimek";
      .

# Table dbo.dbKnjiga
map:dbo_dbKnjiga a d2rq:ClassMap;
      d2rq:dataStorage map:database;
```

```
      d2rq:uriPattern "dbo/dbKnjiga/@@dbo.dbKnjiga.ID@@";
      d2rq:class vocab:dbo_dbKnjiga;
      d2rq:classDefinitionLabel "dbo.dbKnjiga";
      .
map:dbo_dbKnjiga__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property rdfs:label;
      d2rq:pattern "dbKnjiga #@@dbo.dbKnjiga.ID@@";
      .
map:dbo_dbKnjiga_ID a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property vocab:dbo_dbKnjiga_ID;
      d2rq:propertyDefinitionLabel "dbKnjiga ID";
      d2rq:column "dbo.dbKnjiga.ID";
      d2rq:datatype xsd:integer;
      .
map:dbo_dbKnjiga_ISBN a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property vocab:dbo_dbKnjiga_ISBN;
      d2rq:propertyDefinitionLabel "dbKnjiga ISBN";
      d2rq:column "dbo.dbKnjiga.ISBN";
      .
map:dbo_dbKnjiga_Naslov a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property vocab:dbo_dbKnjiga_Naslov;
      d2rq:propertyDefinitionLabel "dbKnjiga Naslov";
      d2rq:column "dbo.dbKnjiga.Naslov";
      .
map:dbo_dbKnjiga_Zalozba a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property vocab:dbo_dbKnjiga_Zalozba;
      d2rq:propertyDefinitionLabel "dbKnjiga Zalozba";
```

```
      d2rq:column "dbo.dbKnjiga.Zalozba";
      .
map:dbo_dbKnjiga_FusekiRestart a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property vocab:dbo_dbKnjiga_FusekiRestart;
      d2rq:propertyDefinitionLabel "dbKnjiga FusekiRestart";
      d2rq:column "dbo.dbKnjiga.FusekiRestart";
      d2rq:datatype xsd:integer;
      .
map:dbo_dbKnjiga_IDAvtor_FK__ref a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:dbo_dbKnjiga;
      d2rq:property vocab:dbo_dbKnjiga_IDAvtor_FK;
      d2rq:refersToClassMap map:dbo_dbAvtor;
      d2rq:join "dbo.dbKnjiga.IDAvtor_FK => dbo.dbAvtor.IDAvtor";
      .
```

Dodatek B

Slovar

```
@prefix :   <http://www.test.org/VidmarJernej/diploma#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix bibo:   <http://purl.org/ontology/bibo/> .
@prefix dct:   <http://purl.org/dc/terms/>.
@prefix owl:   <http://www.w3.org/2002/07/owl#> .
@prefix xml:    <http://www.w3.org/XML/1998/namespace> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix foaf:   <http://xmlns.com/foaf/0.1/>.
@prefix blt:    <http://www.bl.uk/schemas/bibliographic/blterms#>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpprop:   <http://dbpedia.org/property/>.
@prefix isbd:    <http://iflastandards.info/ns/isbd/elements/>.
@prefix skos:    <http://www.w3.org/2004/02/skos/core#>.
#--------------------DATABASE PREFIXES-----------------------#
@prefix vocab: <<file:///C%3A/Users/FRI/Documents/Fuseki/fuseki-server/vocab/>.
#------------------------------------------------------------#

<http://www.test.org/VidmarJernej/diploma>
     rdf:type owl:Ontology ;
```

```
     rdfs:comment "Model for integration of different
                                   book data sources."@en ;
     rdfs:comment "Model integracije razlicnih podatkovnih
                                   virov knjig."@sl .

#-----------------------------------------------------------------#
#
# Classes
#
#-----------------------------------------------------------------#

:Knjiga
      rdf:type owl:Class ;
      rdfs:label "Knjiga"@sl ,
                       "Book"@en ;
      rdfs:comment "Razred Knjiga predstavlja posamezno knjigo."@sl,
                       "Class Knjiga represents a book."@en .
### to so podrazredi nasega razreda: Knjiga ###
bibo:Book rdfs:subClassOf :Knjiga.
vocab:dbo_dbKnjiga rdfs:subClassOf :Knjiga.
dbpedia-owl:Book rdfs:subClassOf :Knjiga.

:Avtor
       rdf:type owl:Class ;
       rdfs:label "Avtor"@sl ;
       rdfs:label "Author"@en ;
       rdfs:comment "Razred Avtor predstavlja avtorja knjige."@sl ;
       rdfs:comment "Class Avtor represents the author of a book."@en .
### to so podrazredi nasega razreda: Avtor ###
dct:Agent rdfs:subClassOf :Avtor.
foaf:Person rdfs:subClassOf :Avtor.
foaf:Agent rdfs:subClassOf :Avtor.
```

```
vocab:dbo_dbAvtor rdfs:subClassOf :Avtor.
dbpedia-owl:author rdfs:subClassOf :Avtor.
dbpprop:author rdfs:subClassOf :Avtor.

:Koncept
        rdf:type owl:Class ;
        rdfs:label "Koncept"@sl ;
        rdfs:label "Concept"@en .
        rdfs:comment "Razred Koncept predstavlja koncept knjige."@sl ;
        rdfs:comment "Class Koncept represents a concept of the book."@en .
### to so podrazredi nasega razreda: Koncept ###
skos:Concept rdfs:subClassOf :Koncept.
blt:TopicDDC rdfs:subClassOf :Koncept.
blt:TopicLCSH rdfs:subClassOf :Koncept.
blt:PersonConcept rdfs:subClassOf :Koncept.

#-----------------------------------------------------------------#
#
# Data properties
#
#-----------------------------------------------------------------#

#------------------------KNJIGA-------------------------------#
:idKnjige
        rdf:type owl:DatatypeProperty ;
        rdfs:label "ID knjige"@sl ;
        rdfs:label "book ID"@en ;
        rdfs:comment "ID knjige vsebinsko predstavlja ISBN."@sl ;
        rdfs:comment "BookID is equal to ISBN."@en ;
        rdfs:domain :Knjiga ;
        rdfs:range xsd:string .
### to so podatributi nase lastnosti: Avtor ###
```

```
bibo:isbn10 rdfs:subPropertyOf :idKnjige.
bibo:isbn13 rdfs:subPropertyOf :idKnjige.
bibo:isbn rdfs:subPropertyOf :idKnjige.
vocab:dbo_dbKnjiga_ISBN rdfs:subPropertyOf :idKnjige.
dbpedia-owl:isbn rdfs:subPropertyOf :idKnjige.
dbpprop:isbn rdfs:subPropertyOf :idKnjige.

:naslov
     rdf:type owl:DatatypeProperty ;
     rdfs:label "naslov"@sl ;
     rdfs:label "title"@en ;
     rdfs:comment "Naslov knjige."@sl ;
     rdfs:comment "Title of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: naslov ###
dct:title rdfs:subPropertyOf :naslov.
dct:alternative rdfs:subPropertyOf :naslov.
vocab:dbo_dbKnjiga_Naslov rdfs:subPropertyOf :naslov.
dbpedia-owl:name rdfs:subPropertyOf :naslov.
dbpprop:name rdfs:subPropertyOf :naslov.

:opisKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "opis"@sl ;
     rdfs:label "description"@en ;
     rdfs:comment "Opis knjige."@sl ;
     rdfs:comment "Description of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: opisKnjige ###
dct:description rdfs:subPropertyOf :opisKnjige.
```

```
rdfs:comment rdfs:subPropertyOf :opisKnjige.

:povzetekKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "abstrakt"@sl ;
     rdfs:label "abstract"@en ;
     rdfs:comment "Abstrakt knjige."@sl ;
     rdfs:comment "Abstract of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: povzetekKnjige ###
dbpedia-owl:abstract rdfs:subPropertyOf :povzetekKnjige.
dct:abstract rdfs:subPropertyOf :povzetekKnjige.

:jezikKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "jezik"@sl ;
     rdfs:label "language"@en ;
     rdfs:comment "Jezik knjige."@sl ;
     rdfs:comment "Language of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: jezikKnjige ###
isbd:P1073 rdfs:subPropertyOf :jezikKnjige.
dbpprop:language rdfs:subPropertyOf :jezikKnjige.

:obsegKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "obseg"@sl ;
     rdfs:label "extent"@en ;
     rdfs:comment "Obseg knjige."@sl ;
     rdfs:comment "Extent of the book."@en ;
```

```
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: obsegKnjige ###
dbpprop:pages rdfs:subPropertyOf :obsegKnjige.
isbd:P1053 rdfs:subPropertyOf :obsegKnjige.

:vsebinaKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "vsebina"@sl ;
     rdfs:label "content"@en ;
     rdfs:comment "Vsebina knjige."@sl ;
     rdfs:comment "Tabel of Contents of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: vsebinaKnjige ###
dct:tableOfContents rdfs:subPropertyOf :vsebinaKnjige.

:izdalKnjigo
        rdf:type owl:DatatypeProperty ;
     rdfs:label "zalozba"@sl ;
     rdfs:label "publisher"@en ;
     rdfs:comment "Zalozba knjige."@sl ;
     rdfs:comment "Publisher of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: izdalKnjigo ###
dbpprop:publisher rdfs:subPropertyOf :izdalKnjigo.

:datumIzdaje
        rdf:type owl:DatatypeProperty ;
     rdfs:label "datum izdaje"@sl ;
     rdfs:label "publish date"@en ;
```

```
     rdfs:comment "Datum izdaje knjige."@sl ;
     rdfs:comment "Publish date of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: datumIzdaje ###
dbpprop:pubDate rdfs:subPropertyOf :datumIzdaje.

:oblikaKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "oblika"@sl ;
     rdfs:label "form"@en ;
     rdfs:comment "Oblika izdaje knjige."@sl ;
     rdfs:comment "Publish form of the book."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: oblikaKnjige ###
dbpprop:mediaType rdfs:subPropertyOf :oblikaKnjige.

:drzavaIzdajeKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "drzava"@sl ;
     rdfs:label "country"@en ;
     rdfs:comment "Drzava kjer je bila knjiga izdana."@sl ;
     rdfs:comment "Country where book was published."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: drzavaIzdajeKnjige ###
dbpprop:country rdfs:subPropertyOf :drzavaIzdajeKnjige.

:slikaKnjige
        rdf:type owl:DatatypeProperty ;
     rdfs:label "slika knjiga"@sl ;
```

```
     rdfs:label "thumbnail of the book"@en ;
     rdfs:domain :Knjiga ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: slikaKnjige ###
dbpedia-owl:thumbnail rdfs:subPropertyOf :slikaKnjige.

#-------------------------KONCEPT-------------------------------#
:imeKoncepta
        rdf:type owl:DatatypeProperty ;
     rdfs:label "ime koncepta"@sl ;
     rdfs:label "concept label"@en ;
     rdfs:domain :Koncept ;
     rdfs:range xsd:string .

#-------------------------AVTOR---------------------------------#
:imeAvtorja
     rdf:type owl:DatatypeProperty ;
     rdfs:label "imeAvtorja"@sl ;
     rdfs:label "authorsName"@en ;
     rdfs:comment "Ime avtorja knjige."@sl ;
     rdfs:comment "Book authors name."@en ;
     rdfs:domain :Avtor ;
     rdfs:range xsd:string .
### to so podatributi nase lastnosti: imeAvtorja ###
foaf:givenName rdfs:subPropertyOf :imeAvtorja.
foaf:name rdfs:subPropertyOf :imeAvtorja.
vocab:dbo_dbAvtor_Ime rdfs:subPropertyOf :imeAvtorja.

:priimekAvtorja
     rdf:type owl:DatatypeProperty ;
     rdfs:label "priimekAvtorja"@sl ;
     rdfs:label "authorsSurname"@en ;
```

```
    rdfs:comment "Priimek avtorja knjige."@sl ;
    rdfs:comment "Book authors surname."@en ;
    rdfs:domain :Avtor ;
    rdfs:range xsd:string .
### to so podatributi nase lastnosti: priimekAvtorja ###
foaf:familyName rdfs:subPropertyOf :priimekAvtorja.
foaf:surname rdfs:subPropertyOf :priimekAvtorja.
vocab:dbo_dbAvtor_Priimek rdfs:subPropertyOf :priimekAvtorja.

:oAvtorju
       rdf:type owl:DatatypeProperty ;
    rdfs:label "o avtorju"@sl ;
    rdfs:label "about author"@en ;
    rdfs:domain :Avtor ;
    rdfs:range xsd:string .
### to so podatributi nase lastnosti: oAvtorju ###
dbpedia-owl:abstract rdfs:subPropertyOf :oAvtorju.

:slikaAvtorja
       rdf:type owl:DatatypeProperty ;
    rdfs:label "slika avtorja"@sl ;
    rdfs:label "thumbnail of author"@en ;
    rdfs:domain :Avtor ;
    rdfs:range xsd:string .
### to so podatributi nase lastnosti: slikaAvtorja ###
dbpedia-owl:thumbnail rdfs:subPropertyOf :slikaAvtorja.

:drzavljanstvoAvtorja
       rdf:type owl:DatatypeProperty ;
    rdfs:label "drzavljanstvo avtorja"@sl ;
    rdfs:label "nationality of author"@en ;
    rdfs:domain :Avtor ;
```

```
    rdfs:range xsd:string .
### to so podatributi nase lastnosti: drzavljanstvoAvtorja ###
dbpprop:citizenship rdfs:subPropertyOf :drzavljanstvoAvtorja.
dbpprop:nationality rdfs:subPropertyOf :drzavljanstvoAvtorja.

:rojstvoAvtorja
       rdf:type owl:DatatypeProperty ;
    rdfs:label "rojstvo avtorja"@sl ;
    rdfs:label "birth of author"@en ;
    rdfs:domain :Avtor ;
    rdfs:range xsd:date .
### to so podatributi nase lastnosti: rojstvoAvtorja ###
dbpedia-owl:birthDate rdfs:subPropertyOf :rojstvoAvtorja.
dbpprop:birthDate rdfs:subPropertyOf :rojstvoAvtorja.

:krajRojstva
       rdf:type owl:DatatypeProperty ;
    rdfs:label "kraj rojstva avtorja"@sl ;
    rdfs:label "place of birth of author"@en ;
    rdfs:domain :Avtor ;
    rdfs:range xsd:string .
### to so podatributi nase lastnosti: krajRojstva ###
dbpprop:placeOfBirth rdfs:subPropertyOf :krajRojstva.

#-----------------------------------------------------------------#
:endpoint
       rdf:type owl:DatatypeProperty ;
       rdfs:label "drugi viri za iskanje knjige"@sl ;
       rdfs:label "other resources for books"@en ;
       rdfs:domain :Knjiga ;
       rdfs:range xsd:string .
```

```
#-------------------------------------------------------------#
#
# Object Properties
#
#-------------------------------------------------------------#

:imaAvtorja
     rdf:type owl:ObjectProperty ;
     rdfs:label "ima avtorja"@sl ;
     rdfs:label "has author"@en ;
     rdfs:comment "Knjiga ima avtorja oz. lahko tudi vec avtorjev."@sl ;
     rdfs:comment "Book has an author or multiple authors."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range :Avtor .
### to so podatribut nase lastnosti: imaAvtorja ###
dct:creator rdfs:subPropertyOf :imaAvtorja.
blt:hasCreated rdfs:subPropertyOf :imaAvtorja.
vocab:dbo_dbKnjiga_IDAvtor_FK rdfs:subPropertyOf :imaAvtorja.
dbpedia-owl:author rdfs:subPropertyOf :imaAvtorja.
dbpprop:author rdfs:subPropertyOf :imaAvtorja.

:soPrispevali
     rdf:type owl:ObjectProperty ;
     rdfs:label "so prispevali"@sl ;
     rdfs:label "has contributor"@en ;
     rdfs:comment "H knjigi so prispevali ena ali vec oseb."@sl ;
     rdfs:comment "Book has one or more contributors."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range :Avtor .
### to so podatribut nase lastnosti: soPrispevali ###
dct:contributor rdfs:subPropertyOf :soPrispevali.
```

```
blt:hasContributedTo rdfs:subPropertyOf :soPrispevali.

:imaKoncept
        rdf:type owl:ObjectProperty ;
     rdfs:label "ima koncept"@sl ;
     rdfs:label "has concept or subject"@en ;
     rdfs:comment "Knjiga ima koncept.."@sl ;
     rdfs:comment "Book has one or more subjects or concepts."@en ;
     rdfs:domain :Knjiga ;
     rdfs:range :Koncept .
### to so podatribut nase lastnosti: imaKoncept ###
dct:subject rdfs:subPropertyOf :imaKoncept.

#----------------------------------------------------------------#
#
# SPARQL ENDPOINT definitions
#
#----------------------------------------------------------------#
:endpoint rdfs:seeAlso <http://bnb.data.bl.uk/sparql>.
:endpoint rdfs:seeAlso <http://dbpedia.org/sparql>.
```

Dodatek C

Fuseki konfiguracija

```
@prefix : <#> .
@prefix fuseki: <http://jena.apache.org/fuseki#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix tdb: <http://jena.hpl.hp.com/2008/tdb#> .
@prefix ja: <http://jena.hpl.hp.com/2005/11/Assembler#> .
@prefix d2rq:  <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#>.


#---------------------------------------------------------------#
[] rdf:type fuseki:Server ;

  fuseki:services (
      <#service>
  ) .

#----------------------D2RQ integracija-------------------------#
d2rq:D2RQModel a rdfs:Class;
      rdfs:subClassOf ja:Model;
      rdfs:label "D2RQ model";
```

```
      rdfs:comment "Jena Assemler specification for a RDB,
                              mapped to RDF using the D2RQ tool.";
      ja:assembler "de.fuberlin.wiwiss.d2rq.assembler.D2RQAssembler";
      .
d2rq:mappingFile a rdf:Property;
      rdfs:label "Mapping file";
      rdfs:comment "URL of a D2RQ mapping file.";
      rdfs:domain d2rq:D2RQModel;
      .
d2rq:resourceBaseURI a rdf:Property;
      rdfs:label "Resource base URI";
      rdfs:comment "Base URI for resources generated by relative
                              URI patterns.";
      rdfs:domain d2rq:D2RQModel;
      .

#----------------------------------------------------------------#
<#service> rdf:type fuseki:service;
      fuseki:name      "MODEL";
      fuseki:serviceQuery    "query";
      fuseki:serviceReadGraphStore "get";
      fuseki:serviceUpdate "update" ;
   fuseki:serviceUpload "upload" ;
   fuseki:serviceReadWriteGraphStore "data" ;
      fuseki:dataset     <#dataset>;
      .

<#dataset> rdf:type ja:RDFDataset;
      rdfs:label   "dataset for aplication";
      ja:defaultGraph <#model>;
      .
```

```
<#model> rdf:type ja:OntModel;
      ja:ontModelSpec ja:RDFS_MEM_RDFS_INF;
      rdfs:label  "Model for integration";
      ja:baseModel <#rdfs_model> ;
      ja:reasoner [ja:reasonerURL <http://jena.hpl.hp.com/2003/
         RDFSExptRuleReasoner>];
      .
<#rdfs_model> rdf:type ja:UnionModel;
      ja:rootModel
      [
            a ja:MemoryModel;
            ja:content [ja:externalContent <file:DiplomaOntologyModel.n3>];
      ];
      ja:subModel <#relationalDatabase>
      .

<#relationalDatabase> a d2rq:D2RQModel;
   d2rq:mappingFile <mapping.ttl>;
   d2rq:resourceBaseURI <http://localhost:2020/resource/>;
   .
#-----------------------------------------------------------------#
```