Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Damir Balija

Obogatitev podatkov v relacijski podatkovni bazi z Linked Data slovarji

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2015

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Damir Balija sem avtor magistrskega dela z naslovom:

Obogatitev podatkov v relacijski podatkovni bazi z Linked Data slovarji

S svojim podpisom zagotavljam, da:

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

V Ljubljani, 26. junija 2015 Podpis avtorja:

Najprej se zahvaljujem mentorju doc. dr. Dejanu Lavbiču za vsestransko podporo, nasvete ter strokovno pomoč pri izdelavi magistrske naloge. Prav tako se zahvaljujem staršem, ki so mi študij omogočili in me v času študija podpirali. Na koncu bi se rad zahvalil še bratu, punci, prijateljem in vsem za izkazano pomoč, podporo, potrpežljivost in razumevanje med časom študija ter nastankom te magistrske naloge.

Najlepša hvala vsem!

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
BFS	Breadth - First Search	Algoritem iskanje v širino.
DARPA	Defense Advanced Research Projects Agency	Agencija za napredne obrambne analize.
DM	Direct mapping	Jezik za izražanje neposredne preslikave podatkov iz relacijskih baz v format RDF.
GS	Gold Standard	Referenčni rezultat, ki predstavlja najbolj pravilno rešitev določenega problema.
HTTP	HyperText Transfer Protocol	Protokol za izmenjavo podatkov na spletu.
IR	Information Retrieval	Področje iskanja in izbiranja informacij.
LOD	Linked Open Data	Prosto dostopni viri, združeni v oblak - oblak množic povezanih podatkov.
NLP	Natural Language Processing	Tehnika procesiranja naravnega jezika.
OWL	Web Ontology Language	Standardni jezik za zapis ontologije z visoko izrazno močjo.

kratica	angleško	slovensko
R2RML	Relational to RDF mapping language,	Jezik za izražanje prilagojenih preslikav iz relacijskih baz v format RDF.
RDB2RDF	Relational Databases to RDF	Zbirka priporočil za preslikavo vsebine relacijskih podatkovnih baz v RDF.
RDF	Resource Description Framework	Standardiziran podatkovni model za izmenjavo informacij na spletu.
SKOS	Simple Knowledge Organization System Reference	Sistem za organizacijo znanja.
SPARQL	Simple Protocol and RDF Query Language	Poizvedovalni jezik za delo s podatki v obliki RDF.
SQL	Structured Query Language	Strukturirani poizvedovalni jezik za delo z relacijskimi podatkovnimi bazami.
URI	Uniform resource identifier	Niz znakov namenjen identifikaciji spletnega vira.
URL	Uniform resource locator	Niz, ki enolično določa spletni naslov.
W3C	World Wide Web Consortium	Konzorcij svetovnega spleta, kjer razvijajo standarde za svetovni splet.
XML	Extensible Markup Language	Razširljiv označevalni jezik za shranjevanje strukturiranih podatkov.

Povzetek

Namen magistrske naloge je opredeliti metodo, ki bo omogočala povezavo obstoječe relacijske sheme in strukturiranih podatkov semantičnega spleta. Temeljili bomo na različnih načinih integracije podatkov, kjer se bomo osredotočili predvsem na tehnologije semantičnega spleta (RDF, RDF shema, R2RML in OWL). V okviru magistrskega dela smo razvili metodo, ki omogoča povezovanje podatkov iz relacijske podatkovne baze (sheme in primerkov) z obstoječimi strukturiranimi podatki v RDF formatu oz. Linked Data viri (LOD). Rezultat magistrske naloge sta pristop in metoda, ki z obogatitvijo shem in podatkov poljubne relacijske podatkovne baze omogočata izvedbo porazdeljenih poizvedb. Rešitev nam prav tako omogoča pripravo in objavo podatkov v skladu s priporočili za objavo odprtih podatkov na svetovnem spletu (pet-zvezdna Linked Data oblika). V okviru praktične implementacije smo razvili prototip v obliki orodja, s katerim smo razvito metodo verificirali in evalvirali. Obstajajo različne odprtokodne rešitve, ki omogočajo usklajevanje in bogatenje podatkov, vendar tovrstni pristopi delujejo zgolj s semantično manj bogatimi podatki (npr. XML in CSV oblika). Naš prototip, LogMapFRI, predstavlja dopolnitev in izboljšavo orodja LogMap. Omogoča delo z relacijskimi podatkovnimi bazami in uporablja metapodatke iz teh baz za ugotavljanje konteksta, s čimer uporabnika razbremenimo ročnega povezovanja shem. Prav bolj natančen zajem metapodatkov in razširitev sintaktičnih pristopov sta se izkazali kot ključni izboljšavi našega pristopa.

Ključne besede:

bogatenje podatkov, ontologije, usklajevanje ontologij, semantični splet, RDB2RDF, morfološka normalizacija nizov, shema relacijske podatkovne baze kot usmerjen graf, iskanje v širino, pristopi na področju usklajevanja ontologij, Linked Data viri

Abstract

The goal of the master's thesis is to introduce a method that will define how to link relational schemas with the existing structured data sources on the Semantic Web. Our study is based on different methods of data integration, where we will focus on the technologies of the Semantic Web. More precisely, we developed a method that enables integration of data from a relational database with existing structured data in RDF format - so called Linked Data (LOD) sources. The result of the master's thesis are therefore approach and method which determines the method of implementation of distributed queries; enrichment of schema and data from any relational database. The solution also enables us to produce and publish information in accordance with the recommendations published by Tim Berners-Lee in 2010. The second part of master's thesis presents a prototype in the form of a tool. With this tool we have verified and evaluated method from the first part. There are variety of open source solutions that enable matching and enrichment of data, but such approaches work only with semantically less rich information. Our prototype is actually a complement and enhancement of LogMap tool - we named it LogMapFRI. It allows user to work with relational databases, which implicitly contain useful metadata. These metadata can be used to identify the context of the data in the database and thus relieve the user from integrating schemas manually. Being able to do just that and extension of syntactic approaches are in fact key improvements of LogMap and main contribution of our solution.

Key words:

data enrichment, ontologies, ontology matching, semantic web, RDB2RDF, R2RML, morphological string normalization, relational database schema as directed graph, Breadth-first search, ontology matching techniques, Linked Open Data

Poglavje 1

Uvod

Podjetja v relacijskih podatkovnih bazah hranijo podatke, ki se običajno uporabljajo zgolj v okviru točno določenih aplikacij. Za vsako novo podatkovno bazo se tipično izdela nova shema, ki je prilagojena tipu in načinu delovanja posamezne aplikacije [1]. Na drugi strani na spletu, natančneje v LOD oblaku (ang. Linked Open Data Cloud) [2], najdemo številne strukturirane podatke, ki so izraženi z jeziki in tehnologijami semantičnega spleta. Ti podatki podrobneje predstavljajo določeno problemsko področje in so širše uporabni. Zato bi jih lahko uporabili pri bogatitvi podatkov v obstoječih relacijskih podatkovnih bazah. Natančneje, skrbnikom podatkov bi omogočili pridobitev dodatnih informacij, dopolnitev manjkajočih podatkov ter preverjanje točnosti in zanesljivosti obstoječih podatkov. Naš cilj je najti metodo, ki bo določala način povezave poljubne relacijske sheme in podatkov s strukturiranimi viri na semantičnem spletu.

Podatke iz relacijskih baz je seveda mogoče povezovati tudi z drugimi podatki, vendar se naše magistrsko delo osredotoča na podatke na semantičnem spletu, ki so strukturirani, poleg tega pa vse bolj številni. Ti podatki so pripravljeni z orodji in tehnologijami semantičnega spleta, ki jih podrobneje predstavimo v poglavju 2.3. Opišemo tudi različne standarde in pristope s področja semantičnega spleta: podatkovni model Resource Description Framework (RDF) [3, 4]; RDF shemo (RDFS) [5] kot osnovni slovar, potreben za podatkovno modeliranje; R2RML [6] standard za preslikovanje podatkov v relacijski podatkovni bazi v RDF obliko in napredne gradnike z višjo semantično izrazno močjo jezika OWL [7].

Iniciativa, ki promovira dobre prakse objave in povezovanja podatkov na se-

mantičnem spletu se imenuje Linked Data [2] in je podrobno predstavljena v poglavju 2.4. Viri prosto dostopnih podatkov, ki so opisani s tehnologijami semantičnega spleta, pa tvorijo Linked Open Data oblak. Na ta način podatke povežemo in omogočimo dostop do njih z objavo v t.i. triplestore podatkovni bazi, uporabnik pa do njih dostopa preko protokola HTTP in poizvedovalnega jezika SPARQL [8]. Ti strukturiranimi podatki v RDF formatu oz. Linked Data viri so tisti, s katerimi bi želeli obogatiti podatke v relacijski bazi. Po podatkih iz leta 2011 [9] zajemajo več kot 256 podatkovnih množic in več kot 30 milijard trojic iz področja medijev, geografije, naravoslovnih ved ipd. Na ta način lahko s pomočjo omenjenega poizvedovalnega jezika SPARQL izvajamo porazdeljene poizvedbe, ki jih v izolirani podatkovni bazi seveda ne moremo. Npr. uporabnik lahko izvede poizvedbo, kjer z določitvijo pogojev poizvedbe hkrati preiskujemo podatke iz več LOD virov, kot je npr. DBPedia [10], Freebase [11] in iz uporabnikove interne podatkovne baze.

Magistrska naloga je sestavljena iz dveh sklopov. Rezultat prvega sklopa, ki je predstavljen v poglavju 4, je razvit pristop in metoda, ki omogoča izvoz in hkratno obogatitev sheme in podatkov iz relacijske baze v ontologijo semantičnega spleta. Drugi sklop magistrske naloge je zajet v poglavju 4.3 in predstavlja prototip v obliki orodja, s katerim razvito metodo verificiramo na različnih problemskih domenah.

Predlagana metoda (poglavje 4), ki je zajeta v prvem sklopu, je sestavljena iz dveh glavnih korakov.

Prvi korak metode zajema preslikavo sheme vhodne relacijske podatkovne baze v RDF obliko, pri čemer se upoštevajo priporočila W3C delovne skupine RDB2RD (problematika zajeta v poglavjih 2.3.5 in 4.1.4). Naša izvožena shema oz. ontologija poleg razredov, ki predstavljajo tabele v relacijski podatkovni bazi, vsebuje tudi konkretne primerke.

V drugem koraku metode se preslikava izvede. Shema je po prvem koraku pretvorjena v RDF obliko, in zato lahko podatke povežemo in obogatimo s podatki iz obstoječih prosto dostopnih repozitorijev iz LOD oblaka. Ti prav tako hranijo podatke o shemi, seveda pa tudi konkretne primerke. V drugem koraku metode smo problematiko bogatenja podatkov iz relacijske podatkovne baze aplicirali na področje usklajevanja ontologij. V ta namen smo preučili klasifikacije pristopov in orodij s področja usklajevanja ontologij (poglavje 3) in skušali identificirati tiste,

ki najbolje ustrezajo zahtevam našega problema.

Pregled metod smo uporabili tudi v drugem sklopu naloge, torej pri izdelavi prototipa. Na podlagi pregleda pristopov (poglavje 3) in pregleda orodji (poglavje 3.2), ki implementirajo te tehnike, smo se odločili za uporabo orodja LogMap (poglavje 4.2.1). V poglavju 4.2.1 predstavimo njegovo delovanje in skušamo identificirati dele, kjer so možne izboljšave. Razširitev, s podrobnostmi korakov predlaganega pristopa, je predstavljena v poglavju 4.

Prototip smo razvili, da bi lahko našo metodo verificirali, hkrati pa smo želeli implementirati informacijsko rešitev, namenjeno predstavljenemu problemu. Obstajajo namreč različne odprtokodne rešitve, ki omogočajo usklajevanje in bogatenje podatkov, vendar so te rešitve omejene zgolj na točno določene vire in ne omogočajo povezave z relacijskimi podatkovnimi bazami. Obstoječe rešitve oz. pristopi znajo obdelovati semantično manj bogate podatke (npr. XML in CSV oblika), kjer je zelo malo metapodatkov in zaradi tega so takšne rešitve večinoma ročne, saj so v celoti odvisne od uporabniškega vnosa potrebnega domenskega znanja.

Verifikacija oz. evalvacija je zadnji korak drugega sklopa. Postopek je predstavljen v poglavju 4.3. Pri evalvaciji drugega dela metode bomo v poglavju 4.3.4, kot testno množico podatkov, uporabili "OAEI systematic benchmark suite 2014", iz področja usklajevanja velikih ontologij (FMA [12] in NCI [13]) in področja usklajevanja primerkov.

Poglavje 2

Tehnologije in orodja semantičnega spleta

Naše magistrsko delo sodi na področje integracije podatkov s podatki iz semantičnega spleta, zato se bomo v tem poglavju osredotočili na tehnologije semantičnega spleta, ki smo jih uporabili. V nadaljevanju opišemo različne standarde in pristope iz omenjenega področja, v katere vključujemo: podatkovni model Resource Description Framework (RDF); RDF shemo (RDFS), kot osnovni slovar, potreben za podatkovno modeliranje; napredne gradnike z višjo semantično izrazno močjo v obliki jezika OWL ter R2RML standard za preslikovanje podatkov iz relacijske podatkovne baze v RDF obliko.

2.1 Zakaj sploh semantični splet?

Svetovni splet danes poznamo kot zbirko (največ tekstovnih) dokumentov, pa tudi aplikacij. Iskalnika kot sta Google[1] in Yahoo, te tekstovne dokumente zbirata, besedila smiselno shranita in obdelata, ter na osnovi poindeksiranega besedila omogočita iskanje. Z informacijami zakodiranimi v naravnem jeziku iskalniki težko manipulirajo, in težko omogočajo kaj več kot iskanje ujemanja med uporabniško poizvedbo in besedami v dokumentu. Ideja semantičnega spleta je

[1] `https://www.google.com/search/about/insidesearch/howsearchworks/crawling-indexing.html`

nekoliko drugačna. Informacije zastopane v naravnem jeziku bi zamenjale ali dopolnile informacije z eksplicitno semantiko, ki lahko služi tudi strojni obdelavi. Na semantičnem spletu uporabljamo tehnologije, ki nam omogočajo, da informacije predstavimo v obliki podatkovnega modela, ki temelji na teoriji grafov. Na ta način, se nam odprejo možnosti strojnega (in tudi avtomatskega) dopolnjevanja in integracije podatkov ter tudi distribuiranega poizvedovanja ter sklepanja.

Z iskanjem informacij v tekstovnih dokumentih smo kot uporabnik precej omejeni. Tekstovne dokumente iščemo s pomočjo ključnih besed, še bolj pa smo omejeni pri iskanju slik in drugih multimedijskih vsebin. Tehnologije semantičnega spleta nam omogočajo, da vsak vir na spletu opišemo oz. označimo na standarden in strukturiran način, ki računalniku omogoča, da izvaja poizvedbe nad dejstvi.

2.2 Razvoj spleta

Semantični splet v prvi vrsti seveda temelji na internetu. Internet je razširitev tehnologije računalniških omrežij. V letih 1960 in 1970 je postalo običajno, da se računalnike v organizacijah (npr. univerza, vlada, družba), skupaj poveže v omrežje. Že takrat pa so se začeli prvi poskusi povezovanja celotne mreže, npr. omrežje ARPANET v Združenih državah Amerike. Internet se je v 80-ih najprej razširil v Evropo in Avstralijo, v devetdesetih letih pa še drugam.

Ključen korak pri nastanku interneta je bila standardizacija. S pomočjo protokola IP (Internet Protocol) je bilo moč na standarden način poimenovati in posledično tudi nasloviti vsak računalnik v omrežju.

Velja omeniti tudi koncept hiperteksta, ki ima svoje zametke v članku Busha in Wanga, objavljenega leta 1945: "As we may think" [14]. Ideja povezovanja oddaljenih, vendar tako ali drugače vsebinsko povezanih podatkov, izhaja prav od tu.

Razvoj interneta je v devetdesetih letih prejšnjega stoletja prinesel njegovo najbolj znano storitev, in sicer svetovni splet ("World Wide Web" oz. "WWW"). Svetovni splet je prav z uporabo hiperteksta in internetnih protokolov omogočil mreženje dokumentov po vsem svetu.

Idejni avtor svetovnega spleta Tim Berners-Lee je specifikacijo in prototip programske opreme razvil leta 1990 in ju razvijal še nekaj let pozneje [15].

Leto 1993 je bilo prelomno za svetovni splet, saj se je pojavil spletni brskalnik Mosaic, s katerim je bilo možno poleg besedila prikazati tudi grafične elemente. Od takrat je uporaba spleta hitro rasla. V tej zgodnji fazi razvoja spleta, ki ga danes imenujemo splet 1.0 (ang. Web 1.0), so bile spletne strani večinoma statične in niso omogočale prilagoditev vsebine na podlagi potreb posameznih uporabnikov.

Okoli leta 2000 se je začela druga faza razvoja spleta, in sicer z uporabo tehnologij, ki uporabniku brskalnika omogočajo interakcijo s spletno stranjo in tako tudi prilagajanje vsebine glede na njegove potrebe.

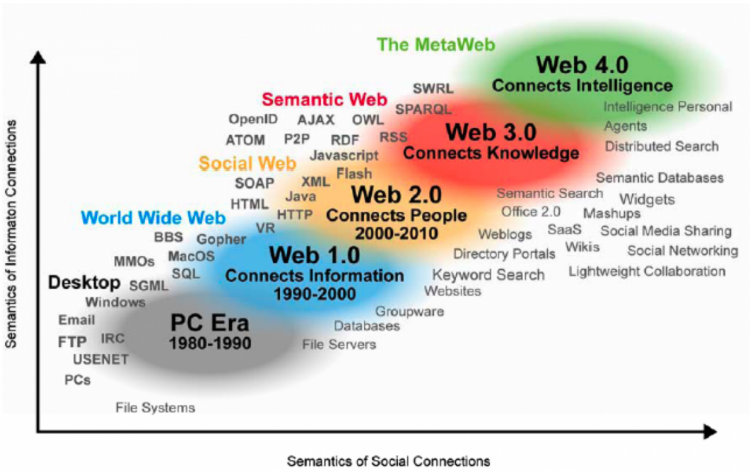
Tehnologije spleta 2.0 so omogočile razrast socialnih omrežij, vključno s klepetalnicami, blogi, wikiji, spletnimi aplikacijami, multimedija ipd. Spletni uporabnik s spletom 2.0 ni več zgolj uporabnik spletnih vsebin, temveč lahko tudi sam prispeva k urejanju in nastanku vsebine, in tako posredno komunicira tudi z drugimi uporabniki.

2.2.1 Splet 3.0 (semantični splet)

Prav tako so že v devetdesetih letih prejšnjega stoletja Tim Berners-Lee in sodelavci razvili predloge za naslednjo stopnjo razvoja spleta, znanega kot semantični splet ali splet 3.0. Ta koncept je bil prvič objavljen v članku v [16], kjer je Berners-Lee, skupaj s soavtorji, poudaril, da so obstoječe spletne vsebine oblikovane tako, da so uporabne oz. razumljive samo ljudem, ne pa računalniškim aplikacijam. Splet 3.0 delno uresničuje takratno vizijo. Raziskovalci pa tudi že razmišljajo, kakšne bi lahko bile značilnosti naslednje verzije spleta, imenovane splet 4.0.

W3C definira semantični splet takole: "Semantični splet nam ponuja ogrodje, ki uporabnikom spleta omogoča objavo in ponovno uporabo podatkov med različnimi aplikacijami, podjetji in drugimi skupnostmi" [18].

Prvi korak k uresničitvi semantičnega spleta je ustvariti povezano omrežje podatkov. Obstaja namreč veliko število podatkov, ki jih uporabniki uporabljajo v svojem vsakdanjem življenju, pa niso objavljeni na spletu. Če bi bili podatki najprej objavljeni v strukturirani obliki, nato pa še povezani, bi uporabnik lahko na spletu npr. v različnih spletnih aplikacijah, v vsakem trenutku pregledal svoj izpisek bančnega računa, poljubno multimedijsko vsebino, kontakte, sestanke v koledarju ipd. Da bi lahko v eni aplikaciji imeli skupen pregled nad vsemi prej naštetimi primeri informacij, potrebujemo omrežje podatkov oz. semantični splet



Slika 2.1: Razvoj spleta [17].

v katerem so tovrstni podatki povezani [18].

Na semantični splet lahko gledamo kot na razširjeno različico spleta, kjer se podatkom definira pomen. Semantika je eksplicitna: podatke formalno opišemo in definiramo povezave med njimi. To nam omogoča lažjo strojno integracijo, procesiranje, objavo, poizvedovanje ipd.

Tehnično gledano gre pri semantičnem spletu v osnovi za dve stvari. Semantični splet zajema vse skupne formate za integracijo in združevanje podatkov, ki prihajajo iz različnih virov, medtem ko se splet v osnovi ukvarja zgolj s področjem izmenjave dokumentov. Kot drugo pa: semantični splet zajema tudi jezike, s katerimi definiramo relacijo med podatki in objekti iz realnega sveta. To nam omogoča, da uporabnik ali računalnik, začne iskanje v enem vozlišču oz. podatkovni bazi in se v korakih pomika po različnih podatkovnih virih s katerim je začetno vozlišče (podatkovna baza) povezano. [18].

Temelj semantičnega spleta, med drugim, predstavljajo standardi: RDF (Resource Description Framework) – za opis podatkov, OWL (Web Ontology Language) – definicija formalnega pomena iz razov, SPARQL (SPARQL Protocol and RDF Query Language) – poizvedovalni jezik in protokol. Osnova za kreiranje podatkov z semantično izraznost je jezik OWL – to je jezik za kreiranje ontologij. Za definicijo ontologije, v kontekstu računalništva, je največkrat uporabljena Gruberjeva definicija: "Ontologija je eksplicitna specifikacija konceptualizacije" [19]. Natančneje,

ontologija je opis (kot npr. formalna specifikacija programske opreme) konceptov in relacij; ta opis pa uporablja določen agent ali skupina agentov. Z ontologijo eksplicitno izrazimo tudi koncepte in relacije, ki so nam ljudem samoumevne in razumljive brez dodatnih pojasnil. Vse te tehnologije semantičnega spleta (ki so hkrati tudi W3C standardi) smo uporabljali tudi v naši implementaciji in jih bomo zato podrobneje predstavili.

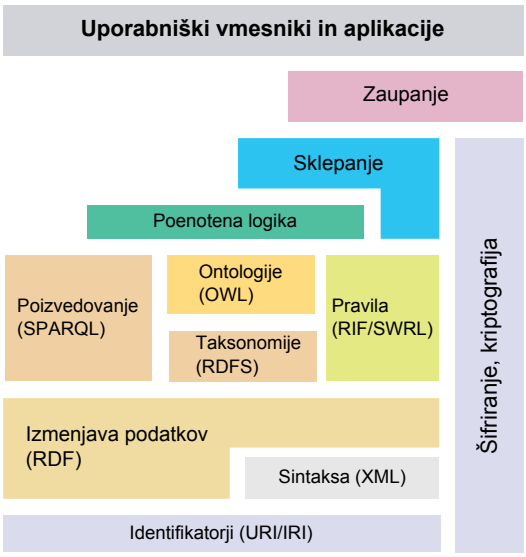
2.3 Standardi in tehnologije semantičnega spleta

Semantični splet sloni na standardih in tehnologijah spleta, kot sta protokol HTTP ter imenski standard URI [20]. HTTP (Hypertext Transfer Protocol) je protokol aplikacijske plasti in najbolj prepoznavni protokol svetovnega spleta (ang. World Wide Web). Gre za sklop konvencij, ki urejajo načine komunikacije med odjemalcem in strežnikom. URI (Uniform resource identifier) je kompaktno zaporedje znakov, ki označujejo abstrakten ali fizičen vir. URI je sestavljen iz imena sheme oz. protokola, imena strežnika, in poti do podstrani (specifičnega vira) na tem strežniku. URI, ki je povezan z dostopnim virom imenujemo tudi "Uniform Resource Locator" ali URL.

Za serializacijo podatkov na semantičnem spletu se je najprej uporabljal standard XML, skupaj z ogrodjem/jezikom RDF [4] ter jezikoma za gradnjo slovarjev oz. ontologij RDFS [5] ter OWL [7]. Kombinacijo podatkovnega modela izraženega v RDF, ki ga predstavimo najprej, zakodiranega v sintakso XML imenujemo kar RDF/XML [21]. Osnovni koncept označevalnega jezika XML je, da so oznake (elementi) postavljene okoli dela besedila, v skladu s temi oznakami pa se potem formatira tekst.

2.3.1 RDF

RDF (ang. Resource Description Framework) [3, 4] ogrodje je bilo narejeno z namenom modeliranja metapodatkov, torej opisovanja lastnosti virov (dokument, slika ali program), kot so npr. avtor, lokacija ali kodirni standard. Leta 1999 je bil RDF standard prvič objavljen kot W3C predlog in od takrat je bilo ogrodje



Slika 2.2: Sklad jezikov semantičnega spleta.

razširjeno tako, da je z njim moč opisati ne več samo metapodatkov, temveč tudi znanje oz. osnovne zakonitosti določenega domenskega prostora.

V naši nalogi, RDF ogrodje, v kombinaciji z jezikoma za gradnjo slovarjev oz. ontologij RDFS [5] ter OWL, uporabljamo za predstavitev sheme in konkretnih primerkov iz relacijske podatkovne baze, ki je vključena v proces bogatenja z Linked Data viri.

Slika 2.3: Osnovni RDF graf, ki prikazuje da je Alfred Hitchcock (osebek) ustvaril (predikat) film Psycho (predmet).

Osnovna ideja RDF je preprosta: trditve, ki predstavljajo znanje, so predstavljeni kot trojice, in sicer v obliki osebek-predikat-predmet (subject-predicate-

object), kot je prikazano na sliki 2.3. Vsaka trojica predstavlja relacijo (predstavljeno z virom predikata) med osebkom in predmetom vira. Formalno je osebek izražen z URI ali praznim vozliščem, predikat z URI in predmet z URI, s številkami ali nizom.

Kot že omenjeno, prvotno priporočilo W3C za izpostavljanje podatkov RDF je bilo, da mora biti kodiran v XML obliki. To je razlog, da pri skladu semantičnega spleta, prikazanega na sliki 2.2, jezik RDF temelji na XML-u. Na sliki 2.4 je prikazan primer trojice izražene v XML-u povzet iz DBpedije (primer je prav tako prikazan na sliki 2.3).

```
1 <rdf:Description rdf:about="http://dbpedia.org/resource/Psycho_(1960_film)">
2   <dbpedia-owl:producer rdf:resource="http://dbpedia.org/resource/Alfred_Hitchcock" />
3 </rdf:Description>
```

Slika 2.4: Primer trojice izražene v RDF/XML obliki.

XML danes ni več edina uporabljena sintaksa za zapis RDF trojic. Ena od lažje berljivih in precej bolj kompaktnih alternativ je turtle, v katerem je izjava preprosto oblikovana z naštevanjem osebka, predikata in predmeta, brez vmesnih ločil, le na koncu sledi ločilo pika "." (primer zapisa je prikazan na sliki 2.5).

```
1 dbpedia:Alfred_Hitchcock ex:ustvaril dbpedia:Psycho_(1960_film) .
2 dbpedia:Psycho_(1960_film) dbpedia-owl:director dbpedia:Alfred_Hitchcock ;
3   dbpedia-owl:producer dbpedia:Alfred_Hitchcock ;
4   dbpprop:director dbpedia:Alfred_Hitchcock .
```

Slika 2.5: Primer trojice izražene v obliki Turtle.

RDF je pravzaprav slovar za izdelovanje slovarjev, elementi slovarja pa so dodatno razdeljeni v razrede in lastnosti. RDF slovar nudi osnovne elemente s katerim lahko opišemo določen vir. Za opis domene na način podoben objektno orientiranemu ali E-R modeliranju, moramo RDF (kot osnovno ogrodje) kombinirati z RDFS slovarjem.

2.3.2 RDFS

RDFS ali RDF Schema oz. shema je razširitev RDF-ja. Je jezik, s pomočjo katerega gradimo slovarje oz. (zelo ohlapno definirane) ontologije. RDFS nam omogoča, da na določeni domeni identificiramo in izrazimo konstrukte, ki jih izrecno opredelimo kot razred, primerek razreda ali lastnost. Razred razumemo kot skupino, ki vsebuje posameznike. Z RDFS-jem, še bolj pa z OWL-jem (kot bomo videli kasneje) razred opišemo z uporabo formalnih (matematičnih) opisov, ki natančno opredeljujejo zahteve za članstvo v posameznem razredu. Primerek je posamezen predstavnik razreda. RDFS podpira tudi nekatere dodatne konstrukte, s katerimi izrazimo odnose med razredi, kot je npr. odnos razred-podrazred. Tako lahko že s pomočjo RDFS-ja gradimo razredno hierarhijo – taksonomijo razredov. RDFS (in tudi OWL) poznata dve glavni vrsti lastnosti [22]:

- "rdfs:ObjectProperty": Z lastnostjo objekta lahko definiramo povezave med posamezniki (osebki). Je lastnost s katero lahko med seboj povežemo dva objekta (primerka). Na primer z lastnostjo "povezan z" lahko povežemo osebo "Alfred Hitchcock" s filmom "Psycho". Tovrstne lastnosti so lahko tranzitivne ali simetrične. Pri definiciji lastnosti objekta je potrebno definirati domeno (rdfs:domain), katera določa kateri primerki lahko zasedajo to lastnost in obseg (rdfs:range), ki definira za katere objekte se definirana lastnost nanaša.

- "rdfs:DatatypeProperty" ali lastnost spremenljivke, s katero lahko opredelimo relacijo med primerkom in posamezno podatkovno vrednostjo (npr. Paris Hilton ima barvo oči "modro"). Pri definiciji lastnosti spremenljivk ji je potrebno določiti domeno, s katero definiramo razred, katerega primeri bodo lahko vsebovali to lastnost. Poleg razreda, je potrebno definirati tudi tip podatkovne vrednosti.

Nekateri pomembni elementi v RDFS slovarja so torej [5]:

- "rdfs:Class", z uporabo tega elementa izrazimo, kateremu razredu pripada določen vir.

- "rdfs:subClassOf", gre za lastnost (predikat) iz RDFS slovarja, ki nam omogoča, da v obliki trojice izrazimo, da je predmet podrazred objekta.

- "rdfs:subPropertyOf", lastnost, s katero izrazimo, da je predmet pod-lastnost objekta.
- "rdfs:domain", lastnost, s pomočjo katere v obliki trojice izrazimo, da je domena določene lastnosti (ki v trojici nastopa kot predmet), specifičen razred (ki v trojici nastopa kot objekt).
- "rdfs:range", na podoben način kot domeno, lahko določimo tudi obseg. Definiramo lahko namreč, da je določena lastnost za objekt predstavnik le določenega razreda ali razredov.

Z naslednjimi trditvami v sintaksi turtle želimo ponazoriti konstrukte slovarja RDFS. RDFS nam omogoča, da definiramo razrede, ter možne podrazrede tega razreda, kar prikazuje slika 2.6.

```
1 lavbic-owl:OurSakilaClass rdf:type rdfs:Class
2 lavbic-owl:Country rdfs:subClassOf lavbic-owl:OurSakilaClass .
3 lavbic-owl:City rdfs:subClassOf lavbic-owl:OurSakilaClass .
```

Slika 2.6: Primer uporabe konstrukta slovarja rdfs:subClassOf.

Ko smo definirali razred, lahko potem s pomočjo predikata "rdf:type" definiramo posamezne primerke tega razreda (primer definicije primerkov je prikazan na sliki 2.7).

```
1 lavbic-owl:Paris_Hilton rdf:type lavbic-owl:Staff .
2 lavbic-owl:Country/France rdf:type lavbic-owl:Country .
3 lavbic-owl:City/Paris rdf:type lavbic-owl:City
4 lavbic-owl:Paris rdf:type lavbic-owl:City .
```

Slika 2.7: Primer definicije primerkov s pomočjo predikata "rdf:type".

Primer na sliki 2.8 kaže, kako smo s pomočjo jezikov RDF in RDFS oz. z njunim kombiniranjem definirali svojo lastnost, poimenovano "relatedTo", v imenskem prostoru "lavbic-owl". Tej lastnosti smo določili tip (za razliko od prejšnjega primera tip ni rdfs:Class, temveč rdfs:Property), ji določili domeno in obseg (rang),

ter ji pripeli berljivo oznako (poimenovanje) v treh jezikih (v slovenščini, nemščini in angleščini).

```
1 lavbic-owl:relatedTo rdf:type rdfs:Property.
2 lavbic-owl:relatedTo rdfs:domain owl:Class .
3 lavbic-owl:relatedTo rdfs:range owl:Class .
4 lavbic-owl:relatedTo rdfs:label "povezanZ"@sl .
5 lavbic-owl:relatedTo rdfs:label "verbunden"@de .
6 lavbic-owl:relatedTo rdfs:label "relatedTo"@en .
```

Slika 2.8: Primer definicije lastnosti s pomočjo jezikov RDF in RDFS.

RDFS vsebuje tudi nekaj lastnosti, s katerimi povežemo vir z opisnimi podatki. Te lastnosti niso namenjene sklepanju, temveč nam pomagajo pri predstavitvi vira in za navigacijo. Pri prejšnjem primeru smo že srečali lastnost "rdfs:label". Ostale lastnosti, ki sodijo v to kategorijo so:

- "rdfs:comment", viru dodamo berljiv opis.
- "rdfs:seeAlso", iz enega vira nas napoti na drugi vir, ki bi lahko ponudil dodatne informacije glede prvega vira.
- "rdfs:isDefinedBy", podlastnost lastnosti "rdfs:seeAlso", ki nas napoti na vir, ki vsebuje definicijo prvega vira (osebka).

V pristopu, ki je predlagan v poglavju 4, RDFS elemente v kombinaciji z elementi jezika OWL (predstavljen v poglavju 2.3.3) uporabljamo za zapis ontologije, ki predstavlja shemo izbrane relacijske podatkovne baze. Natančneje, te tehnologije uporabljamo za definicijo razredov, konkretnih primerkov in lastnosti objektov s katerimi določimo relacije med razredi in konkretnimi primerki.

Poleg definicije razredov z RDFS elementi prav tako določimo podrazrede in tako skupaj z vsemi definiranimi lastnostmi zgradimo strukturo iz katere se s pomočjo različnih tehnik, ki delujejo nad shemami (predstavljeni v poglavju 3.1.2), v fazi bogatitve, ugotovi kontekst oz. podobnost med primerjanimi razredi. To nam v naslednji fazi omogoča, da razredi dobijo nove trditve, ki predstavljajo povezavo z drugimi viri.

2.3.3 OWL

Web Ontology Language (OWL) [7] je razširitev jezika RDFS. S kombinacijo ogrodij RDF in RDFS smo lahko določen vir definirali kot razred ali lastnost, ter definirali osnovne odnose med njimi in drugimi razredi in lastnostmi. Jezik OWL pa je izrazno še močnejši. Teoretično ozadje jezika OWL je opisna logika, ki nam omogoča izražanje bolj kompleksnih izjav o primerkih, razredih in lastnostih. Tako lahko domeno še bolj natančno definiramo in na ta način tvorimo ontologijo posamezne domene.

V primeru, da ontologijo zakodiramo v katero izmed vej matematične logike, tudi računalnikom omogočimo, da nad podatki izvaja (avtomatično) sklepanje.

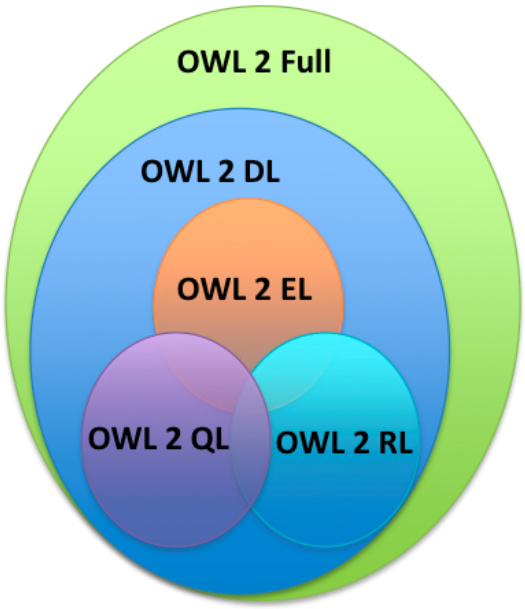
Obstaja več vej formalne logike, ki se med seboj razlikujejo v stopnji izrazne moči in kako učinkovito lahko nad njimi izvajamo sklepanje. Da postane ontologija praktično uporabna, ponavadi žrtvujemo nekaj izrazne moči, da pridobimo na hitrosti sklepanja. OWL temelji na opisni logiki, in to prav zaradi tega razloga. Kljub kompromisom glede izrazne moči in kljub temu, da so za opisno logiko že prej obstajali učinkoviti algoritmi sklepanja, OWL še vedno ni zares uporabljen za sklepanje nad velikimi množicami podatkov. To bi aplikacije, ki slonijo na povezanih podatkih naredilo zares uporabne [23]. Tudi zato se sklepanje nad povezanimi podatki ponavadi izvaja nad podatki zakodiranimi v RDFS, z redkimi OWL elementi, če sploh.

OWL je bil razvit na začetku tega stoletja in je postal W3C standard (skupaj z RDFS) leta 2004. Od leta 2008 je na voljo že druga verzija OWL-ja, imenovana OWL 2. Opisna logika je zapleten formalizem, sestavljen iz več podsistemov in to se izraža tudi v samem jeziku OWL 2. Tako imamo na voljo različne profile jezika OWL 2 – uporabnik izbere tistega, ki mu najbolj ustreza glede izrazne moči ter hitrosti sklepanja.

OWL 2 Full je uporaben v scenarijih, kjer se zahteva maksimalna izrazna moč in sintaktična svoboda RDF ogrodja.

OWL 2 DL je primeren za scenarije, kjer se zahteva visok nivo izrazne moči, izračunljivost in odvisnost pa nista tako pomembni. Nad podatkovnimi množicami opisanimi z ontologijo OWL DL je moč izvajati avtomatično sklepanje.

OWL 2 EL, profil, ki omogoča osnovno klasifikacijo in sklepanje v polinomskem času.



Slika 2.9: Komponente jezika OWL 2.

OWL 2 QL je profil, narejen z namenom prenosljivosti in poizvedovanja v relacijskih podatkovnih bazah.

OWL 2 RL je profil, namenjen implementacijam, kjer je pomembno učinkovito sklepanje na podlagi pravil (ang. rule-based reasoning).

Za delo s povezanimi podatki, podrobno razumevanje jezika OWL niti ni potrebno. Kot že rečeno, ko imamo opravka z veliko količino podatkov, so tako ali tako računsko učinkoviti le najenostavnejši procesi sklepanja, ki jih je moč večinoma izvajati nad viri opisanimi zgolj z RDFS slovnico. Z elementi iz OWL slovarja pa lahko natančneje izrazimo te lastnosti domene:

- Konstrukcija razredov: formiranje novih razredov s pomočjo že obstoječih razredov, lastnosti in primerkov (npr. ObjectIntersectionOf);
- Konstrukcija lastnosti: razlikovanje med lastnostmi objektov (ang. resources as values) in lastnostmi podatkov (ang. literals as values);
- Aksiomi razredov: trditve o razredih, opisovanje podrazredov in izražanje trditev o razredih, izražanje ekvivalence in neujemanja med razredi;

- Aksiomi lastnosti: trditve o lastnostih, vključujoč relacije kot so ekvivalenca in pod-lastnost, pa tudi definiranje atributov lastnosti, kot so funkcijska lastnost, tranzitivna lastnost, itd.;

- Aksiomi o primerkih: trditve o primerkih, npr., da določen primerek pripada določenemu razredu, in da dva primerka predstavljata istega ali pa da gre za dva različna.

Sklepanje

Ontologije, ki so opisane z uporabo jezika OWL-DL, je možno s pomočjo stroja za sklepanje procesirati, kar nam omogoča izvajati logične zaključke na podlagi dejstev in aksiomov. Stroji za sklepanje nam omogočajo gradnjo logičnih hierarhij razredov, na podlagi katerih lahko ugotovimo strukturo razredov znotraj določene hierarhije (npr. ali je razred "Pes"podrazred razreda "Žival"). Omenjena funkcionalnost je v našem pristopu koristna pri gradnji razredne hierarhije ontologij, ki so vključene v proces bogatenja podatkov, s pomočjo katerih na podlagi lokalnosti iščemo nove povezave oz. že povezanim preverimo kontekst. Hierarhija razredov se zgradi samo za razrede, v našem pristopu pa jo prav tako uporabimo za ugotavljanje konteksta primerkov, saj morajo biti definirane kot tip nekega razreda. Poleg gradnje logične hierarhije razredov nam orodje za sklepanje omogoča preverjanje konsistentnosti. Preveri se lahko medsebojno konsistentnost vseh definicij in izjav znotraj modela poljubne ontologije. Možno je tudi preverjanje definicij razredov. V primeru, da stroj za sklepanje ugotovi tovrstno nekonsistenco se razred označi kot nekonsistenten. Orodja za sklepanje ravno tako pomagajo pri vzdrževanju oz. pri nadgradnji logične hierarhije razredov [22]. V našem primeru je to uporabno v koraku, ko iz dveh ontologij iščemo nove povezave in tako širimo strukturo razredov.

V okviru našega dela sposobnost sklepanja jezika OWL uporabljamo v fazi bogatenja podatkov oz. na področju usklajevanja ontologij, kar bo podrobno predstavljeno v nadaljevanju.

2.3.4 SPARQL

SPARQL (ang. SPARQL Protocol and RDF Query Language) je jezik za izvajanje poizvedb nad podatki oblikovanimi s pomočjo ogrodja RDF. SPARQL spominja na poizvedovalni jezik SQL (ang. Structured Query Language), ki je standardni jezik za poizvedovanje po relacijskih bazah podatkov že od leta 1980. SPARQL je postal del sklada semantičnega spleta nedavno, kot W3C priporočilo je bil sprejet šele leta 2008. Njegova nadgradnja, SPARQL 1.1 pa leta 2013.

Poizvedovalni jezik SPARQL nam omogoča izvajanje porazdeljenih poizvedb iz različnih virov, ki hranijo podatke v RDF obliki.

Poizvedbe so oblikovane v obliki seznama trojic, v katerem se podatkovne vrednosti zamenjajo z spremenljivkami, ki so prav tako definirane znotraj poizvedbe.

SPARQL poizvedba je sestavljena iz [24]:

- deklaracije predpon, s pomočjo katerih okrajšamo URI-je v poizvedbi,
- definicije podatkovnih množic, ki povedo po katerem RDF grafu oz. RDF grafih poizvedujemo,
- določilo, s katerim povemo, kaj naj bo rezultat poizvedbe
- vzorec poizvedbe, s katerim povemo, kaj želimo izvedeti iz podatkovne množice,
- modifikatorji poizvedbe: z njimi sortiramo ali kako drugače uredimo rezultat poizvedbe.

Ogrodje poizvedbe v jeziku SPARQL, s prej omenjenimi sestavnimi deli, je prikazano na sliki 2.10.

Primer SPARQL poizvedbe iz naše implementacije (brez modifikatorjev poizvedbe) je prikazan na sliki 2.11.

Primer na sliki 2.11 prikazuje preprosto vrsto poizvedbe. V SELECT stavku definiramo kateri podatki naj bodo vrnjeni, z modifikatorjem DISTINCT pa odstranimo ponavljajoče se vrstice iz seznama rezultatov. V WHERE stavku se nato sprehajamo po RDF grafu. Rezultat poizvedbe je prikazan v tabeli 2.1.

SPARQL zagotavlja tudi nekatere bolj zapletene konstrukte: FILTER, s katerim določimo pogoje za vrednosti spremenljivk (npr. da mora biti vrednost

```
# deklaracija predpon
PREFIX foo: <http://example.com/resources/>
...
# definicije podatkovnih množic
FROM ...
# določilo, ki definira rezultat (vrsto) poizvedbe
SELECT ...
# vzorec poizvedbe
WHERE {
...
}
# modifikatorji poizvedbe
ORDER BY ...
```

Slika 2.10: Ogrodje poizvedbe v jeziku SPARQL.

```
PREFIX owl: <http://www.w3.org/2002/07/>
PREFIX rdf: <http://www.w3.org/1999/02/22/>
Prefix rdfs: <http://www.w3.org/2000/01/>
PREFIX lavbic-sakila: <http://www.lavbic.net/onto/>

SELECT DISTINCT ?sakila_element ?oznaka
WHERE{
  lavbic-sakila:Paris rdf:type ?k .
  ?k rdfs:subClassOf lavbic-sakila:ourSakilaClass .
  ?sakila_element rdf:type ?k .
  ?sakila_element rdfs:label ?oznaka .
}
```

Slika 2.11: Primer SPARQL poizvedbe.

številske spremenljivke med letoma 1990 in 2000); OPTIONAL, ki določa kateri podatki naj bodo pridobljeni, če so na voljo, hkrati pa omogoča da se poizvedba uspešno izvede, tudi če podatki niso na voljo.

SPARQL verzija 1.0 je jezik namenjen zgolj za branje (ang. "read-only") iz podatkovne baze, z verzijo 1.1 pa lahko podatke iz RDF grafa tudi dodajamo, posodabljamo in brišemo. Oblika takšnih poizvedb se seveda razlikuje od prej podanega ogrodja.

	sakila_element	oznaka
1	lavbic-sakila:Staff/Paris_Hilton	"Paris Hilton"@en
2	lavbic-sakila:Country/France	"France"@en
3	lavbic-sakila:City/Paris	"Paris"@en
4	lavbic-sakila:Staff/Hilton	"Hilton"@en
5	lavbic-sakila:Address/48_Boulevard_J	"48 Boulevard Jourdan"@en
6	lavbic-sakila:Staff/Paris	"Paris"@en

Tabela 2.1: Rezultat SPQRQL poizvedbe.

Da lahko sploh izvedemo poizvedbo v jeziku SPARQL, moramo uporabiti program ali spletno stran, ki služi kot SPARQL poizvedovalna točka (ang. SPARQL endpoint). V prototipu našega pristopa za hranjenje rezultatov v RDF obliki uporabljamo strežnik Fuseki (podrobneje je obravnavan v točki 2.5.2). Seznam poizvedovalnih točk je na voljo na spletnem mestu [25].

Vmesnik, pri poizvedovanju preko poizvedovalne točke, tipično izgleda tako, da uporabnik v tekstovno polje vnese URI podatkovne množice in poizvedbo samo. Ko uporabnik zažene poizvedbo, se mu prikaže seznam rezultatov poizvedbe. Poizvedovalne točke ponavadi nudijo uporabniku tudi možnost, da rezultate poizvedbe shrani na računalnik, v katerem izmed formatov, npr. RDF/XML ali JSON-LD. S pomočjo ustreznih knjižnic lahko uporabnik poizvedbe poganja tudi dinamično – iz svojega programa – primer take knjižnice za programski jezik Java je Jena [26].

2.3.5 R2RML

Obstajajo različni pristopi, ki omogočajo realizacijo objave podatkov, v skladu s principi povezanih podatkov [27] (poglavje 2.4), vendar so takšni pristopi večkrat prilagojeni za specifičen namen in niso namenjeni splošni uporabi [28]. Da bi se držali principov povezanih podatkov za podatke, ki jih hranimo v tabelah znotraj relacijske podatkovne baze, potrebujemo način za definicijo preslikav tovrstnih podatkov v RDF obliko.

Kot rešitev omenjene problematike sta na voljo dva usklajevalna jezika, s katerima lahko opišemo preslikavo objektov iz relacijske podatkovne baze v RDF obliko, in sicer R2RML (ang. Relational Database to RDF Mapping Language)

[6] in DM (ang. Direct Mapping) [29]. R2RML ima za razliko od DM-ja fleksibilnejši pristop, saj uporabniku omogoča različne prilagoditve usklajevalnega dokumenta, vključitev definicije več logičnih tabel hkrati ter tudi predstavitev logične tabele kot SQL poizvedbo ipd [30]. Ravno zaradi naštetih prednosti smo se v našem pristopu odločili za uporabo usklajevalnega jezika R2RML, saj, kot bomo videli v nadaljevanju (poglavje 4.1.4), potrebujemo pri zajemu tako podatkov kot metapodatkov iz relacijske podatkovne baze dosti fleksibilnosti.

R2RML se lahko uporablja za objavo podatkov iz relacijske podatkovne baze v RDF obliko na dva načina:

1. Prvi način je takšen, da se podatki v celoti pretvorijo v RDF izpis, ki se ga lahko naloži v poljuben strežnik, ki zna hraniti podatke v RDF obliki (ang. triplestore server; v našem primeru je to Jena Fuseki [31]). Ta nam s pomočjo poizvedovalnega jezika SPARQL, omogoča poizvedovanje po nastali zbirki podatkov.

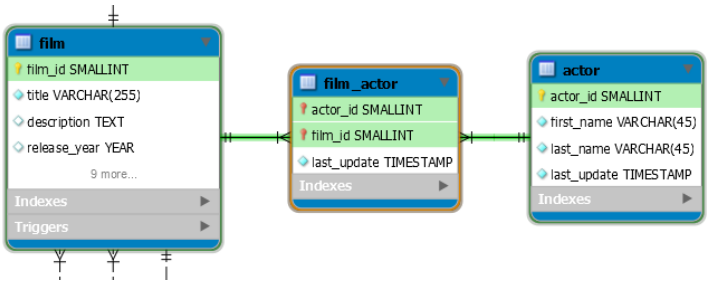
2. V drugem načinu lahko uporabimo poseben R2RML mehanizem, ki nam sproti prevaja SPARQL poizvedbe v SQL poizvedbe, s katerim lahko poizvedujemo po relacijski podatkovni bazi in tako pridobimo željene podatke.

Preslikovalni dokument R2RML (tudi preslikovalni graf R2RML) vsebuje RDF trojice, ki so zapisane v obliki Turtle [6]. V nadaljevanju bomo s pomočjo konkretnega primera, ki je prikazan na sliki 2.12, predstavili glavne komponente preslikovalnega jezika R2RML (slika 2.13; več primerov uporabe R2RML je na voljo [32]). Za primer smo si znotraj podatkovne baze Sakila, izbrali tri tabele in sicer:

- Tabela "film", znotraj katere smo si izbrali stolpec "id" (primarni ključ) in stolpec "title",

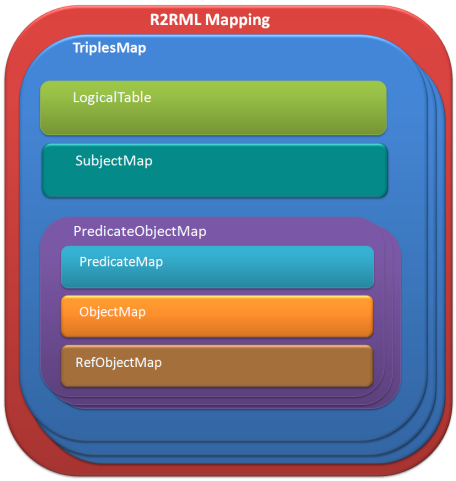
- Tabela "film_actor", ki vsebuje dva stolpca: "actor_id" (tuji ključ, ki označuje številko igralca) in "film_id" (tuji ključ, ki označuje identifikacijsko številko filma).

- Tabela "actor", katera vsebuje stolpec "actor_id" (primarni ključ) in ime in priimek igralca (stolpca "first_name" in "last_name").



Slika 2.12: Prikaz tabel znotraj relacijske podatkovne baze Sakila.

Preslikovalni dokument sestavlja več "TriplesMaps" elementov, ki določajo kako preslikati relacije iz elementa "logicalTable" v RDF obliko. Element "logicalTable" je lahko definiran kot posamezna tabela poljubne podatkovne baze, SQL pogled (ang. view) ali pa kot veljavna SQL poizvedba. V našem primeru ga uporabljamo za definicijo podsheme izbrane relacijske podatkovne baze, ki jo želimo vključiti v proces bogatenja (predstavljeno v poglavju 4.1.4).



Slika 2.13: Komponente jezika R2RML.

Posamezen element "TriplesMap" lahko vsebuje natanko en element "SubjectMap" in poljubno mnogo "PredicateObjectMap" elementov. Za vsako vrstico nabora vrednosti, ki se definira v elementu "logicalTable", se znotraj elementa "SubjectMap" za vsak osebek generira URI, ki pa mora biti enoličen. Osebke se

običajno loči na podlagi primarnega ključa znotraj tabele relacijske podatkovne baze, ki je definirana v elementu "logicalTable". Element "SubjectMap" znotraj definicije "TriplesMap", ki smo jo v primeru, ki je prikazan na sliki 2.14, poimenovali "Film", definira URI, kot združitev imenskega prostora "http://www.lavbic.-net/onto/sakila/film/" in vrednosti polja id, ki predstavlja primarni ključ izbrane tabele "film". Element "PredicateObjectMap" sestavljajo elementi "PredicateMap" in "ObjectMap", s pomočjo katerih se v kombinaciji z osebkom (definiran znotraj elementa "SubjectMap") zgradijo RDF trojice. Torej znotraj "TriplesMap" elementa "Film", se posamezen osebek označi z enoličnim URI-jem, ki bo vseboval predikat "lavbic-owl:title", ta pa bo kazal na vrednost objekta oz. na vrednost iz stolpca "title" znotraj tabele "film".

Znotraj "TriplesMap" elementa "FilmActor", s pomočjo lastnosti objekta "rr:-subjectMap" definiramo URI za osebek "Actor", medtem ko z elementom "rr:predicateObjectMap" definiramo trditev, ki je ekvivalent INNER JOIN SQL stavku med tabelo "film" in tabelo "film_actor". Združevanje je označeno z vrednostjo elementa "rr:objectMap", ki zajema dve lastnosti, in sicer: lastnost rr:parentTriplesMap, ki se nanaša na "TriplesMap" tabele "film", ki je predstavljena kot starš znotraj stavka za združevanje; in lastnost "rr:joinCondition" – z njo definiramo katere stolpce združujemo (v zgornjem primeru združujemo film_actor.film_id in film.film_id).

Ob predpostavki, da tabela Film vsebuje film z vrednostjo ključa 1 in film z naslovom (stolpec "title") "The Shawshank Redemption" in da tabela "film_actor" vsebuje enega igralca s ključem 2, ki igra v filmu s ključem 1. Torej, na podlagi predpostavke in zgoraj predstavljenega preslikovalnega dokumenta R2RML, bi s pomočjo orodja za preslikovanje podatkov v relacijski podatkovni bazi v RDF trojice, dobili rezultat, ki je predstavljen na sliki 2.15.

Privzeto se vse RDF trojice shranijo v privzeti graf (ang. "Default graph"). Graf kamor želimo shraniti rezultate, nastale s pomočjo preslikovalnega dokumenta R2RML, lahko definiramo s pomočjo lastnosti "GraphMap", s katerim se določi v kateri graf želimo zapisati rezultate preslikovalnega dokumenta R2RML.

V elementu "logicalTable" je definiran vir podatkov (primer tabela v relacijski podatkovni bazi) iz katerega se ustvarijo trojice. V primeru na sliki 2.16 je prikazano, da je lahko element "logicalTable" predstavljen kot SQL poizvedba, ki

```
@prefix rr:<http://www.w3.org/ns/r2rml#> .
@prefix lavbic-owl: <http://www.lavbic.net/onto#>
@base <http://www.lavbic.net/onto#> .
<Film>
  rr:logicalTable [ rr:tableName "film"] ;
  rr:subjectMap [
    rr:template http://www.lavbic.net/onto/film/{film_id} ;
    rr:class lavbic-owl:Film ;
  ] ;
  rr:predicateObjectMap [
    rr:predicate lavbic-owl:title;
    rr:objectMap [ rr: column "title"] ;
  ] .
<FilmActor>
  rr:logicalTable [rr:tableName "film_actor"];
  rr:subjectMap [
    rr:template
      "http://www.lavbic.net/onto/actor/{actor_id}";
    rr:class lavbic-owl:Actor;
  ];
  rr:predicateObjectMap [
    rr:predicate lavbic-owl:actsIn;
    rr:objectMap[
      rr:parentTriplesMap <Film>;
      rr:joinCondition
        [rr:child "film_id" ; rr:parent "film_id".];
    ];
  ].
```

Slika 2.14: Primer preslikovalnega dokumenta R2RML.

poizveduje iz več tabel hkrati.

Na podlagi preslikovalnega dokumenta, ki je prikazan na sliki 2.16, bi nam R2RML procesor (seznam R2RML procesorjev je na voljo na povezavi [33]), kot rezultat preslikave, ponudil naslednje trojice, ki so prikazane na sliki 2.17.

Način kako podatke, ki so shranjeni v relacijskih podatkovnih bazah, narediti dostopne na semantičnemu spletu, je v zadnjih desetih letih zelo aktivno raziskovalno področje. Pretvorbo podatkov iz relacijskih podatkovnih baz v RDF obliko imenujemo RDB-to-RDF proces. Konzorcij W3C je septembra 2012 podal predlog za standarden jezik R2RML, s katerim lahko opišemo pravila preslikave med rela-

```
<http://www.lavbic.net/onto/sakila/film/1>
  a lavbic-owl:Film;
  lavbic-owl:title "The Shawshank Redemption" .

<http://www.lavbic.net/onto/actor/2>
  a lavbic-owl:Actor;
  lavbic-owl:actsIn
  <http://www.lavbic.net/onto/film/1> .
```

Slika 2.15: Trojice, ki jih dobimo kot rezultat na podlagi R2RML preslikave iz slike 2.14.

cijsko podatkovno bazo in ekvivalentno podatkovno množico v RDF obliki. Ta korak je pomembno prispeval k udejanjanju semantičnega spleta. R2RML omogoča načrtovalcem "RDB-to-RDF" orodij, da uporabljajo enoten in standardni jezik za preslikovanje relacijskih podatkovnih baz v RDF obliko. S standardizacijo tega področja so veliko pridobili tudi ponudniki podatkov, saj lahko integracijo relacijskih podatkov naredijo neodvisno od specifičnih orodji ali pristopov, ki prevladujejo na tem področju in na ta način zagotovijo trajnost svojih podatkov [34].

2.4 Semantični splet in povezani podatki (Linked Open Data)

Leta 2006 je Tim Berners-Lee objavil štiri predloge [2], ki bi lahko predstavljali prvi korak k realizaciji semantičnega spleta. Namen teh predlogov je doseganje boljše razumljivosti podatkov za računalnike in boljše avtomatsko povezovanje podatkov. Predlogi niso mišljeni kot stroga pravila, temveč bolj kot priporočila dobrih praks: več ponudnikov podatkov kot jim bo sledilo bolj uporabni in povezljivi bodo ti podatki. Ti štirje principi so naslednji [2]:

- uporaba URI-jev za identifikacijo stvari,
- uporaba HTTP URI-jev, z namenom izkoriščanja razrešitvenih možnosti HTTP protokola,

```
@prefix rr:<http://www.w3.org/ns/r2rml#> .
@prefix lavbic-owl: <http://www.lavbic.net/onto#>
@base <http://www.lavbic.net/onto#> .
<FilmActor> a rr:TriplesMap ;
  rr:logicalTable [rr:sqlQuery
    """SELECT CONCAT(film.film_id, '-', film_actor.actor_id) AS id,
              film.title, actor.first_name, actor.last_name
    FROM sakila.film
     INNER JOIN sakila.film_actor on film.film_id = film_actor.film_id
       INNER JOIN sakila.actor on film_actor.actor_id = actor.actor_id"""];
  rr:subjectMap [
    rr:template "http://www.lavbic.net/onto/sakila/FilmActor/{id}";
    rr:class lavbic-owl:Staff ;
  ];
  rr:predicateObjectMap [
    rr:predicate lavbic-owl:title ;
    rr:objectMap [ rr: column "title"] ;
  ] ;
  rr:predicateObjectMap [
    rr:predicate lavbic-owl:actor_first_name ;
    rr:objectMap [ rr:column "first_name"] ;
  ] ;
  rr:predicateObjectMap [
    rr:predicate lavbic-owl:actor_last_name ;
    rr:objectMap [ rr:column "last_name"] ;
  ] .
```

Slika 2.16: Primer R2RML preslikave, z definicijo elementa "logicalTable" kot SQL poizvedbo.

- uporabniku povedati kaj se skriva za določenim URI-jem, in to na standarden način, t.j. z uporabo standardov (RDF/RDFS, SPARQL),
- vključiti povezave do drugih (smiselno povezanih) URI-jev, da lahko uporabnik odkrije še več stvari.

Priporočila so jasna in imajo jasen namen. Uporaba URi-jev za identifikacijo posameznikov, razredov in lastnosti nam omogoči dve stvari. Temu posamezniku oz. razredu oz. lastnosti dodelimo (enoličen) identifikator, poleg tega pa, če je ta URI tudi HTTP URI, s klikom na njega lahko izvemo več informacij o tem, kaj se skriva za tem identifikatorjem oz. kaj ta identifikator opisuje. Za izboljšanje

```
<http://www.lavbic.net/onto/sakila/film/1>
  a lavbic-owl:Film;
  lavbic-owl:title "The Shawshank Redemption" .

<http://www.lavbic.net/onto/sakila/film/1>
  a lavbic-owl:Film;
  lavbic-owl:actor_first_name "Morgan" .

<http://www.lavbic.net/onto/sakila/film/1>
  a lavbic-owl:Film;
  lavbic-owl:actor_last_name "Freeman" .
```

Slika 2.17: Trojice, ki jih dobimo kot rezultat na podlagi R2RML preslikave iz slike 2.16.

povezljivosti podatkov je zaželeno, da objavljeni podatki vsebujejo tudi URI-je, ki kažejo na druge stvari in hkrati tudi lokacije na spletu.

Leta 2010 je Tim Berners-Lee priporočilom dodal še lestvico s petimi stopnjami/zvezdicami [27, 35], s katerimi merimo kolikšen nivo uporabnosti je bil dosežen pri objavi podatkov na spletu. Natančneje:

- * podatki so objavljeni na spletu v kakršnemkoli (nestrukturiranem) formatu, npr. optično zajeta tabela v PDF formatu,
- ** podatki so objavljeni v strukturirani, strojno berljivi obliki, npr. prej omenjeno tabelo objavimo v obliki dokumenta narejenega z Excelom,
- *** podatki so zapisani v nelastniških formatih, npr. CSV namesto Excela, tako, da lahko uporabniki dostopajo do podatkov v surovi obliki,
- **** podatki so opisani z uporabo razširjenih in standardnih formatov za predstavitev podatkov, tako kot npr. RDF, ki zajema tako sintakso kot semantiko,
- ***** podatki so povezani z ostalimi podatkovnimi množicami na spletu in so tako umeščeni v kontekst .

Vsaka naslednja stopnja na lestvici nujno zajema tudi priporočila iz prejšnjih stopenj. Prve tri stopnje je relativno lahko doseči in že spoštovanje teh priporočil

omogoča širšo uporabnost podatkov. Kljub vsemu moramo ljudje semantične probleme pri integraciji podatkov v takšni obliki še vedno reševati ročno. Da lahko ponudimo podatke, v takšni obliki, da bomo na semantičnem spletu po njih najlažje poizvedovali, je potrebno na lestvici doseči vsaj četrto ali peto stopnjo oz. štiri ali pet zvezdic.

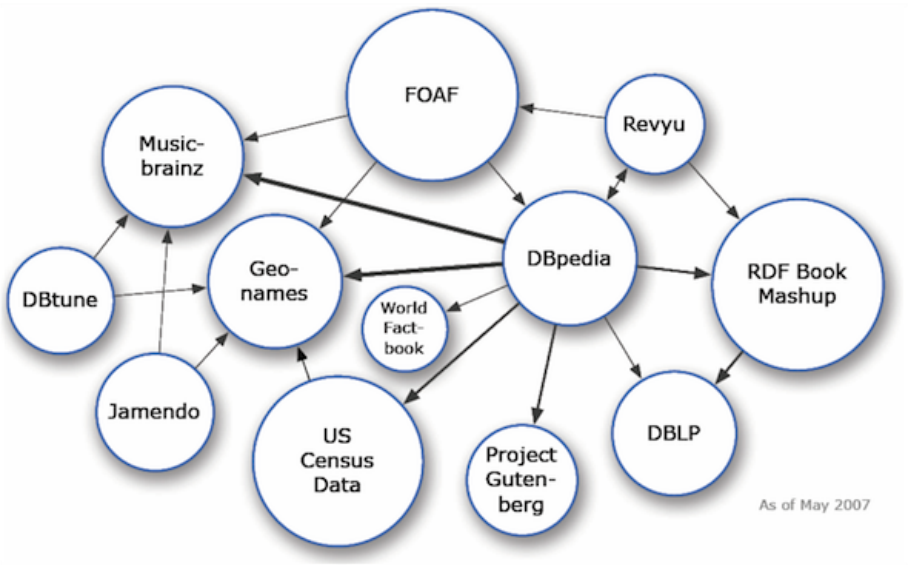
Na semantičnem spletu pet zvezdic dosežemo: z objavo podatkov v RDF obliki; z uporabo RDFS/OWL jezikov za opis slovarja/ontologije; z nudenjem SPARQL dostopne točke in z ustvarjanjem povezav med koncepti (npr. s pomočjo predikata "owl:sameAs").

2.4.1 Rast povezanih podatkov

Kako razširjeno je objavljanje podatkov na semantičnem spletu, v skladu s principi povezanih podatkov, nam grafično ponazarja LOD oblak oz. oblak množic povezanih podatkov. Na sliki 2.18 vidimo kako je oblak izgledal eno leto po objavi priporočil Tim Berners-Leeja, se pravi leta 2007. Za primerjavo, slika 2.20 prikazuje LOD oblak leta 2014. S primerjavo si lahko ustvarimo vtis o rasti v teh sedmih letih. Še en podatek, ki ga imamo na voljo, pravi, da je število RDF trojic zraslo iz okrog 2 milijardi v letu 2007 na okrog 30 milijard leta 2011 [9]. Te podatke med seboj povezuje preko 500 milijonov RDF povezav, katerih glavni namen je povezovanje URI-jev, ki se nanašajo na isto stvar. Na ta način se torej ustvarja povezana celota objavljenih množic podatkov – LOD oblak.

Tako slika 2.18, kot slika 2.20 torej prikazujeta obstoječe množice podatkov in povezave med njimi. Vsako vozlišče v tem oblaku predstavlja ločeno množico podatkov objavljenih v skladu s principi povezljivosti podatkov. Puščice označujejo, da med elementi v povezanih množicah podatkov obstajajo RDF povezave. Odebeljene puščice ponazarjajo večje število povezav med množicami podatkov, medtem ko dvosmerne povezave označujejo, da RDF povezave izhajajo iz obeh množic podatkov.

Če vzamemo za primer sliko 2.18; povezava iz DBpedije do oblačka MusicBrainz pomeni, da DBpedia ne vsebuje samo RDF trojic, ki podajajo informacije o stvareh, temveč tudi trojice, ki povezujejo imena iz DBpedije z njihovimi sinonimi iz MusicBrainz. Primer na sliki 2.19 podaja RDF trojico, ki povezuje dve različni imeni oz. identifikatorja za glasbeno skupino Beatles.



Slika 2.18: Linked data oblak, leta 2007 [9].

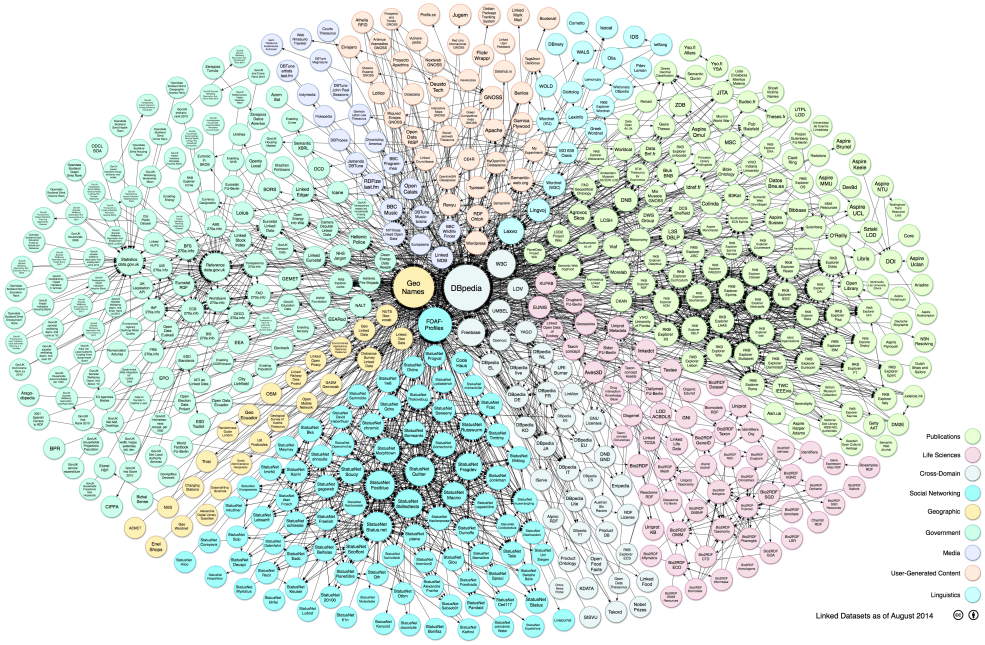
```
1 <http://musicbrainz.org/artist/b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d>
2 <http://www.w3.org/2002/07/owl#sameAs>
3 <http://dbpedia.org/resource/The_Beatles>.
```

Slika 2.19: Trojica, ki povezuje dva različna identifikatorja.

Ker je relacija "sameAs" tranzitivna in komutativna, lahko iz dveh trditev, npr. "X sameAs Y" in "Y sameAs Z" (oz. lahko tudi "Z sameAs Y") izpeljemo sklep "X sameAs Z"; na ta način lahko, s pomikanjem po relacijah (tudi po različnih množicah podatkov), iz oblaka izluščimo seznam vseh sinonimov.

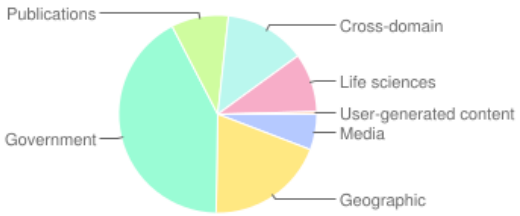
Novejši diagrami so, z namenom večje preglednosti, tudi obarvani. S pomočjo barv, množice podatkov na grobo kategoriziramo po področjih. Legenda k sliki 2.20 razloži pomen posameznih barv.

Nekoliko drugačno kategorizacijo množic podatkov glede na področja, prikazuje spodnja tabela [9], skupaj z številom podatkovnih množic na tem področju, številom ter deležem trojic in številom ter deležem RDF povezav - kategorija zunanje povezave. Število povezav se nanaša na vse odhodne povezave, ki iz podatkovnih množic na določenem področju kažejo na druge množice podatkov.



Slika 2.20: Linked data oblak, leta 2014 [9].

V diagramih na sliki 2.21 in 2.22 so deleži trojic in deleži povezav po področjih še grafično predstavljeni [9].



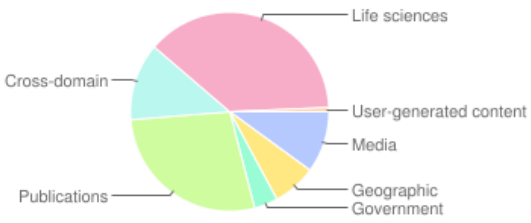
Slika 2.21: Diagram, ki prikazuje delež trojic po področjih [9].

2.5 Razvojno okolje

V prejšnjem poglavju smo opisali preslikovalni jezik R2RML. Da pa bi preslikovanje izvedli potrebujemo R2RML procesor. Torej potrebujemo orodje, ki s pomočjo

Področje	Št. podatkovnih množic	Število trojic	%	Zunanje povezave	%
Mediji	25	1,841,852,061	5.82 %	50,440,705	10.01 %
Geografija	31	6,145,532,484	19.43 %	35,812,328	7.11 %
Javna uprava	49	13,315,009,400	42.09 %	19,343,519	3.84 %
Objave in založništva	87	2,950,720,693	9.33 %	139,925,218	27.76 %
Iz večih področij	41	4,184,635,715	13.23 %	63,183,065	12.54 %
Biološke vede	41	3,036,336,004	9.60 %	191,844,090	38.06 %
Uporabniške vsebine	20	134,127,413	0.42 %	3,449,143	0.68 %
	295	31,634,213,770		503,998,829	

Tabela 2.2: Kategorizacija množic glede na področja v LOD [9].



Slika 2.22: Diagam, ki prikazuje razporeditev izhodnih RDF povezav glede na področje [9].

R2RML preslikovalnega dokumenta in s povezavo na relacijsko podatkovno bazo preslika podatke v RDF obliko. V tem poglavju bo predstavljen R2RML procesor DB2Triples, ki smo ga v našem prototipu uporabili za tovrstno preslikavo.

Po izvedeni preslikavi imamo v RDF obliki definirano strukturo izbrane relacijske podatkovne baze, v kateri so razredi in primerki, s pomočjo različnih lastnosti, med seboj povezani in skupaj tvorijo ontologijo. Za lažjo definicijo ontologije smo si pomagali z odprtokodnim orodjem za urejanje ontologij Protege, ki je predstavljeno v poglavju 2.5.3.

Vse trojice, s katerimi je definirana naša ontologija, hranimo v triplestore strežniku Apache Jena Fuseki, ki nam s pomočjo poizvedovalnega jezika SPARQL omogoča tudi izvajanje poizvedb. Fuseki strežnik je predstavljen v poglavju 2.5.2.

V okviru magistrske naloge smo uporabili več orodij, vsa pa bodo predstavljena kasneje in sicer v okviru prestavitve samega pristopa in delovanja prototipa, ki je nastal kot verifikacija zastavljenega pristopa.

2.5.1 DB2Triples

DB2Triples [34] je orodje za pridobivanje podatkov iz relacijskih podatkovnih baz, njihovo pretvorbo v RDF in vnos le teh v triplestore podatkovno bazo (v našem primeru je to Fuseki Triplestore, ki je predstavljen v poglavju 2.5.2). Orodje podpira standarda R2RML [6] in DM [29] (ang. Direct mapping) za preslikavo podatkov iz relacijske podatkovne baze v RDF format. Oba standarda sta definirana znotraj delovne skupine RDB2RDF, ki skrbi za standardizacijo jezikov za preslikavo relacijskih podatkov in sheme podatkovne baze v RDF in OWL format. Tovrstnim orodjem pravimo tudi R2RML ali DM procesorji. Orodje DB2Triples je bilo razvito s strani družbe Antidot[2], kot del večjega programskega paketa. DB2Triples je odprtokodno orodje, ki je na voljo kot javanska knjižnica[3] in objavljeno pod licenco LGPL. Orodje zna brati iz MySQL, MSSQL in PostgreeSQL podatkovnih baz. Graf, ki je rezultat preslikave s pomočjo preslikovalnih dokumentov (R2RML ali DM), lahko predstavimo v RDF/XML, N3, N-Triples ali obliki Turtle.

Orodje v našem pristopu uporabljamo pri preslikavi podatkov iz izbrane podatkovne baze v RDF obliko, na podlagi preslikovalnega dokumenta R2RML. Nato dobljene podatke vključimo v proces bogatenja oz. usklajevanja z drugimi podatki.

2.5.2 Fuseki

Fuseki je SPARQL strežnik, ki nam omogoča izvajanje SPARQL poizvedb nad podatki v RDF obliki. S pomočjo REST storitve nam omogoča izvajanje poizvedb kar preko protokola HTTP [31].

Strežnik Fuseki je del javanskega odprtokodnega ogrodja Jena [26], ki je namenjeno gradnji aplikacij na področju semantičnega spleta. Strežnik lahko uporabljamo neodvisno od omenjenega ogrodja (slika 2.23). Poganjamo ga lahko kot storitev operacijskega sistema, kot javansko spletno aplikacijo znotraj npr. Tomcat strežnika ali kot samostojni strežnik. Strežniška platforma Fuseki nam nudi implementacijo varnostnih nastavitev s pomočjo ogrodja Apache Shiro[4] in ponuja prijazen uporabniški vmesnik za spremljanje in administracijo sistema.

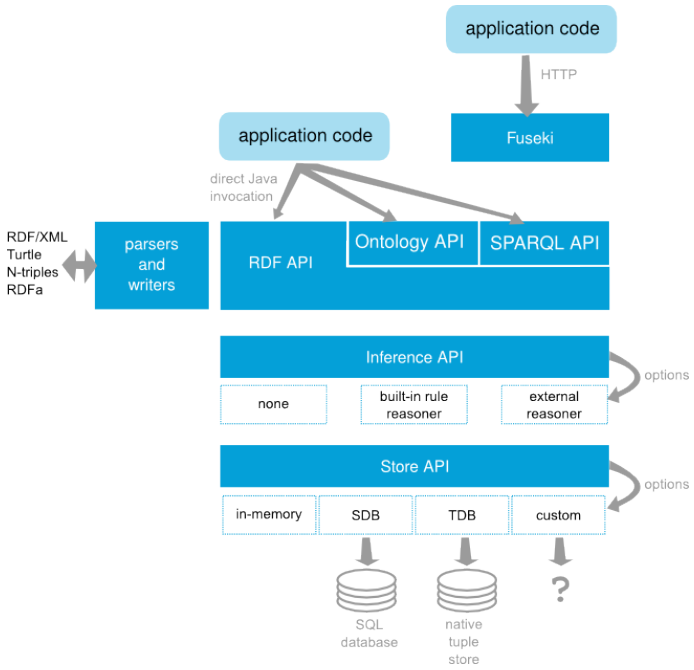
[2] `http://www.antidot.net/`

[3] `https://github.com/antidot/db2triples/`

[4] `https://shiro.apache.org/`

Za nas je najpomembneje, da podpira protokol SPARQL verzijo 1.1 za izvajanje poizvedb in posodabljanje trojic (SPARQL update funkcija).

Fuseki je tesno integrirana s TDB podatkovno bazo za shranjevanje trojic, s katero zagotavlja nivo za trajno shranjevanje. Strežnik uporablja tudi tehnologiji "Jena Text Query"in "Jena Spatial Query". Prva omogoča kombinacijo SPARQL jezika s funkcijo iskanja s pomočjo nizov, druga tehnologija pa omogoča iskanje po geografskih podatkih. Fuseki torej lahko uporabljamo tudi kot protokol za druge RDF sisteme, bodisi pri poizvedbah, bodisi pri shranjevanju.



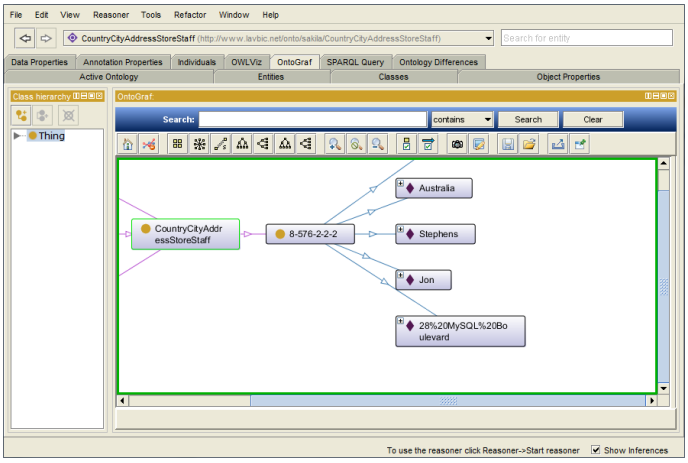
Slika 2.23: Umestitev Triplestore strežnika Fuskeki v arhitekturo programskega paketa Jena [26].

2.5.3 Protege

Odprtokodno orodje Protégé[5] je eden najbolj priljubljenih urejevalnikov ontologij in ogrodje za implementacijo baz znanj. Orodje je bilo razvito na univerzi Standford v sodelovanju z univerzo v Manchestru in ima trenutno že preko dvesto tisoč

[5] `http://protege.stanford.edu/overview/`

registriranih uporabnikov. Znotraj orodja lahko ustvarimo, manipuliramo ter vizualiziramo poljubne ontologije. Na sliki 2.24 je prikazano delovno okolje orodja Protégé.



Slika 2.24: Delovno okolje orodja Protégé.

Arhitektura orodja Protégé je zgrajena modularno, implementiran pa je v programskem jeziku Java. Modularnost omogoča razvijalcem razširljivost ter dodajanje novih funkcionalnosti. Poleg tega ga je možno prilagajati glede na različne zahteve uporabnikov in jim nuditi prijaznejšo podporo pri kreiranju modelov znanja in vnašanju podatkov. Protégé lahko uporabimo za ustvarjanje Protégé-Frames ontologij in OWL ontologij [36].

Pri naši nalogi smo uporabljali verzijo orodja za namizje (obstaja tudi spletna verzija), ki v celoti podpira OWL 2 in ima direktno povezavo s stroji za sklepanje, kot sta HermiT in Pellet.

Poglavje 3

Klasifikacija tehnik in pristopov pri usklajevanju ontologij

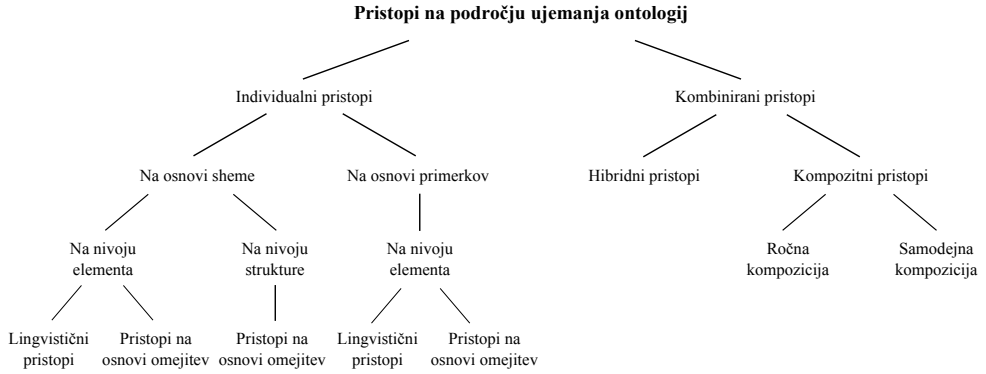
V prejšnjem poglavju so bile predstavljene tehnologije in standardi, s katerimi se srečujemo na področju semantičnega spleta. Ker smo problematiko bogatenja podatkov preslikali na področje usklajevanja ontologij, bodo v ta namen v nadaljevanju prestavljene aktualne tehnike in pristopi iz omenjenega področja.

Po tem, ko v našem pristopu podatke iz relacijske baze pretvorimo v ontologijo, bi želeli te podatke še obogatiti. Pri tem bomo uporabljali tehnike za usklajevanje ontologij (ang. Onotology Matching). Usklajevanje ontologij je proces iskanja povezav med entitetami različnih ontologij [37]. Izboljšana različica orodja za usklajevanje LogMap v naši implementaciji (prototip, ki temelji na zastavljenem pristopu) počne prav to, in sicer išče ujemanje med ontologijo, ki je nastala iz relacijskih podatkov in drugimi ontologijami in tudi primerki opisanimi s to ontologijo. Usklajevanje ontologij je pomembna operacija v kontekstu izgradnje ontologij, saj je okolje v katerem so ontologije načrtovane, razvite in v katerem delujejo, zelo heterogeno [37]. Iskanje ujemanja med ontologijami je uporabno in smiselno, saj nam omogoča:

- da razvijamo manjše in samozadostne module namesto obsežnih ontologij,
- da izrazimo povezavo med dvema različnima verzijama iste ontologije in nadgradimo starejšo verzijo z novejšo,

- lažjo izdelavo visokonivojske ontologije, s katero povežemo druge (bolj specifične) ontologije na nekem domenskem prostoru [37].

Uporabnost ujemanja ontologij je zelo široka in pravilen izbor tehnik ujemanja je ključen, da lahko najdemo najprimernejšo kombinacijo tehnik za naš specifičen problem. V ta namen bomo uporabili že uveljavljeno klasifikacijo [37], ki jo bomo v nadaljevanju uporabili tudi za pregled orodij, ki so na tem področju najbolj aktualna in hkrati tudi najuspešnejša.

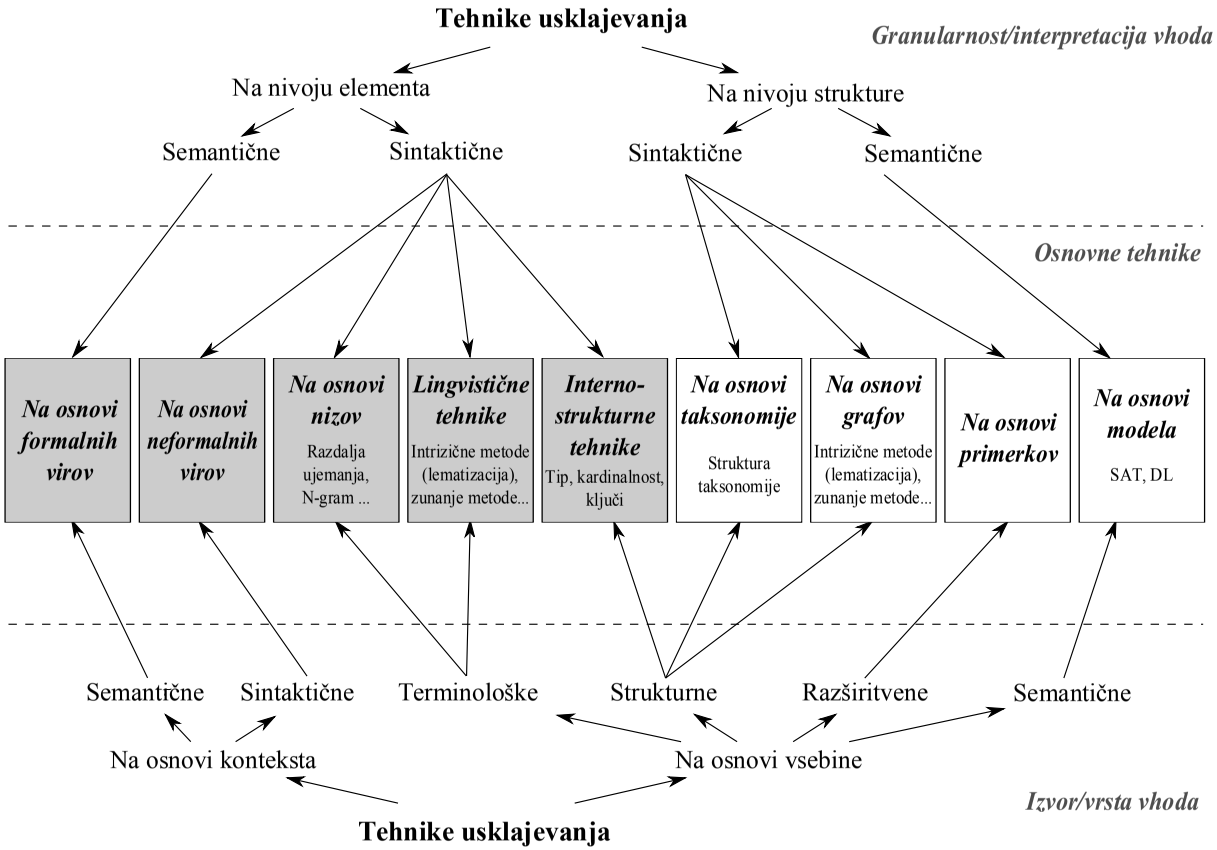


Slika 3.1: Klasifikacija pristopov po [38].

Rahm in Bernstein sta leta 2001 objavila klasifikacijo pristopov za usklajevanje shem/ontologij, ki je prikazana na sliki 3.1, in služi kot osnova za nekatere novejše klasifikacije.

V omenjeni klasifikaciji so tehnike/pristopi za usklajevanje ontologij razdeljeni med individualne in kombinirane pristope. Individualni pristopi lahko za iskanje preslikav uporabljajo podatke iz sheme kot so: imena elementov, podatkovni tipi, strukture shem in relacije med shemami in podshemami. Elementarne pristope delimo na tiste, ki delujejo na nivoju enega elementa, in tiste, ki delujejo na nivoju strukture/sheme elementov. Med elementarne pristope sodijo tudi lingvistične tehnike in pristopi, ki temeljijo na osnovi omejitev.

Kombinirani pristopi pa so definirani kot kombinacija več metod oz. pristopov hkrati. Ti se, po podatkih iz [38], ravno zaradi uporabe rezultatov iz različnih metod, ki jih hkrati uporabljajo, izkažejo kot najuspešnejši.



Slika 3.2: Osrednji sloj predstavlja razrede konkretnih tehnik usklajevanja.

Na sliki 3.2 je prikazana dopolnjena Euzenat in Shvaiko-va klasifikacija iz [37], ki je za naš namen dovolj natančna in ki nam je v prvi fazi omogočala spoznavanje in izbiro tehnik, ki smo jih uporabili v koraku usklajevanja naše sheme z prosto dostopnimi ontologijami.

Klasifikacijo sestavljata dve drevesi, ki si med seboj delita liste. Listi dreves predstavljajo osrednji nivo klasifikacije in sicer osnovne razrede tehnik usklajevanja in njihove konkretne primere.

Zgornja klasifikacija, ki razlikuje tehnike glede na nivo granularnosti in način interpretacije vhoda, se najprej loči na tehnike oz. pristope, ki delujejo na nivoju elementa in tiste, ki delujejo na nivoju strukture. Hkrati (v drugem nivoju) dela razliko med tehnikami glede na interpretacijo vhodnih podatkov, torej, na kakšen način različne tehnike interpretirajo vhodne podatke (na semantičnem ali sintaktičnem nivoju).

Spodnja klasifikacija oz. spodnje drevo, v prvem nivoju, razlikuje tehnike oz. pristope glede na izvor podatkov nad katerimi operirajo, tako podatki lahko izvirajo neposredno iz vsebine vhodnih ontologij (na sliki 3.2: na osnovi vsebine) ali pa iz relacij (na sliki 3.2: na osnovi konteksta) med ontologijami in različnimi zunanjimi viri.

Osrednji nivo, imenovan osnovne tehnike, predstavlja razrede osnovnih tehnik za usklajevanje. Naša predstavitev se bo orientirala na te osnovne tehnike, in vsako od njih bomo na kratko opisali.

3.1 Razredi osnovnih tehnik

V osrednjem nivoju klasifikacije, ki je prikazana na sliki 3.2, so razvrščeni razredi osnovnih tehnik za usklajevanje. Razredi so razdeljeni na podlagi načina interpretacije vhodnih podatkov oz. informacij. Na primer, različne oznake lahko interpretiramo kot niz (zaporedje znakov iz abecede), kot besedo ali frazo v nekem naravnem jeziku. Hierarhijo, na drugi strani, lahko interpretiramo kot graf (množica med seboj odvisnih vozlišč) ali kot taksonomijo (hierarhična razvrstitev po določenem kriteriju).

V nadaljevanju so predstavljeni glavni razredi osrednjega nivoja osnovnih tehnik za usklajevanje na podlagi prej predstavljenih klasifikacij.

3.1.1 Tehnike na nivoju elementa

Tehnike na nivoju elementov, entitete ali primerke iz ontologij obravnavajo neodvisno od njihove umestitve v strukturo sheme oz. neodvisno od njihove strukturne relacije z drugimi entitetami ali njihovimi primerki. Zato med tehnike na nivoju elementa uvrščamo:

- tehnike na nivoju nizov,
- lingvistične tehnike,
- interno-strukturne tehnike in
- tehnike na osnovi formalnih in neformalnih virov.

Tehnike na nivoju nizov

Tehnike na nivoju nizov so tehnike, ki temeljijo na nizih. Najpogosteje se uporabljajo pri usklajevanju imen in opisov entitet v ontologijah. Bolj kot so si nizi med seboj podobni, večja je verjetnost, da opisujejo enak koncept. Uporabljajo se funkcije oz. metrike, ki uskladijo par nizov in kot rezultat podajo realno število, ki nam pove razliko med podanimi nizi. Primeri široko uporabljenih tehnik, ki delujejo na osnovni nizov so:

- Ugotavljanje/iskanje predpon (ang. prefix): preverjajo ali je v začetku niza vsebovan podniz.
- Iskanje pripon oz. končnic (ang. suffix): preverjajo ali je poljuben niz vsebovan na koncu nekega niza.
- Levenshteinova razdalja urejanja [39]: gre za izračun razdalje urejanja med dvema nizoma. Rezultat ujemanja je podan kot število, ki predstavlja število potrebnih operacij nad črkami v nizu (dodajanje, brisanje, zamenjava), da dosežemo popolno ujemanje med besedama.
- N-gram (podniz dolžine n): nizi se pretvorijo v zaporedje nizov, kjer vsak element predstavlja zaporedni podniz dolžine n (npr. za niz ”damir” nam algoritem vrne 3-grame: ”dam”, ”ami”, ”mir”).

- Stoilos-jeva metrika podobnosti nizov oz. SMOA (ang. String Metric for Ontology Alignment) [40] je tehnika na nivoju nizov, ki nam izračuna mero podobnosti med poljubnimi vhodnimi nizi. Tehniko SMOA uporabljamo tudi v našem pristopu in je podrobneje predstavljena v poglavju 4.2.1.

Lingvistične tehnike

Za razliko od tehnik iz predhodnega poglavja, ki obravnavajo niz kot zaporedje znakov, tovrstne tehnike uporabljajo zaporedje nizov oz. tekst. Torej nizi se smatrajo kot besede v nekem nevtralnem jeziku (npr. angleščina). Lingvistične tehnike temeljijo na tehnikah procesiranja naravnega jezika (NLP - Natural Language Processing), ki izkoriščajo lastnosti oblikoslovja (morfologija) besed in pomagajo pri ugotavljanju pomena izrazov iz teksta. S primerjavo izrazov in njihovih relacij lahko lažje ocenimo podobnost entitet v ontologijah. Pri tem lahko uporabljajo tudi zunanje vire kot so slovarji ipd.

Z namenom izboljšanja rezultata ujemanja s pomočjo ugotavljanja konteksta, se jezikovne tehnike uporabijo pred izvedbo leksikografskih tehnik in tehnik na nivoju nizov. Med lingvističnimi tehnikami bomo v nadaljevanju izpostavili naslednje pristope:

- lingvistična normalizacija,
- zunanje lingvistične metode in
- večjezične metode.

Z **lingvistično normalizacijo** spremenimo vsak izraz oz. besedo v standardno obliko, iz katere lahko dobljen niz hitreje prepoznamo in lažje primerjamo z drugimi. Po [41] glede na variacijo izrazov razlikujemo med:

- morfološkimi oz. oblikoslovni (morfologija preučuje razmerja med koreni besed in obrazili, ki se priključujejo korenom),
- sintaktičnimi (variacija glede na strukturo izraza),
- semantični (ang. semantic variation): variacija na podlagi konteksta niza, običajno z uporabo nadpomenk, podpomenk ipd.,

- večjezičnimi variacijami (ang. Multilingual variation).

Nekatera orodja za pomoč pri iskanju podobnosti med nizi uporabljajo **zunanje lingvistične metode** oz. zunanje vire, ki so lahko:

- leksikon ali slovar (ang. Lexicon): je zbirka besed s pripadajočimi definicijami (pomen, sinonimi, nadpomenkami, podpomenkami – primer vnosa iz Wordneta, kot primera tovrstnega leksikona, je prikazan na sliki 3.3).
- semantično sintaktični leksikoni: slovarji, ki poleg definicij besed hranijo tudi njihove kategorije in prislovna določila, ki opisujejo okoliščine, v katerih poteka dejanje izraženo z glagolom. Tovrstni slovarji so redki, zato jih na področju usklajevanja ontologij redko uporabljamo.
- tezavri: zbirka sopomenk (sinonimov) in asociativnih izrazov, v kateri lahko najdemo besede s podobnim ali istim pomenom, včasih pa tudi protipomenke. Od slovarjev se razlikujejo po tem, da ne vsebujejo definicij in podatkov o izgovorjavi - primer: WordNet.
- terminološki tezavri: v njih so izrazi definirani kot fraza in ne kot samostojne besede. Ker z veliko verjetnostjo hranijo podatek o domeni, jih lahko pri usklajevanju uporabimo za odpravo dvoumnosti.

```
author1 noun: Someone who originates or causes or initiates something; Example
''he was the generator of several complaints''. Synonym generator, source.
Hypernym maker. Hyponym coiner.

author2 noun: Writes (books or stories or articles or the like) professionally
(for pay). Synonym writer2. Hypernym communicator. Hyponym abstractor, alliterator,
authoress, biographer, coauthor, commentator, contributor, cyberpunk, drafter,
dramatist, encyclopedist, essayist, folk writer, framer, gagman, ghostwriter,
Gothic romancer, hack, journalist, libretist, lyricist, novelist, pamphleter,
paragrapher, poet, polemist, rhymer, scriptwriter, space writer, speechwriter...

author3 verb.: Be the author of; Example "She authored this play". Hypernym write.
Hyponym co-author, ghost.
```

Slika 3.3: Primer vnosa iz leksikona WordNet.

Omenjali smo uporabo zunanjih virov, od katerih bomo pri naši rešitvi uporabljali WordNet [42]. WordNet je semantični leksikon angleškega jezika, razvit na Univerzi Princeton v ZDA. Podatkovno bazo, ki vsebuje že preko 150.000 besed, lahko uporabniki brezplačno namestijo na računalnik ali pa uporabljajo kar internetno različico. Slovar vsebuje angleške sopomenke, kratke definicije in različne relacije med njimi. Podatkovna baza aktualne različice vsebuje 155.287 besed, ki so združene v 117.659 sklopov sopomenk. Sklop sopomenk je množica vseh besed, ki predstavljajo isti koncept. Med sklopi sopomenk obstajajo tudi relacije, ki definirajo sopomenke, nadpomenke, protipomenke itd. Poleg angleške različice Wordnet slovarja obstajajo tudi različice v drugih jezikih (tudi slovenski [43], BalkaNet [44]), vendar so dostopne samo v namenskih spletnih aplikacijah.

Predstavljeni so bili različni lingvistični viri, ki nam omogočajo ugotavljanje konteksta elementov, ki jih usklajujemo. Z izboljšanjem razlage besed oz. z ugotavljanjem njihovega pomena, povečujemo možnost za uspešnejše iskanje pravilnih ujemanj (ang. true positives, podrobneje v poglavju 4.3.1) med elementi ontologij, ki jih usklajujemo. Hkrati pa se poveča možnost za odkritje napačnih ujemanj - težavo z napačnim ujemanjem nam lahko predstavljajo enakozvočnice [45]. Z omenjeno problematiko, ki je znana tudi kot dvoumen pomen besed (ang. word sense disambiguation) so se ukvarjali v [46, 47], kjer so poskušali določiti pomen elementov iz konteksta in ga primerjali med entitetami za usklajevanje.

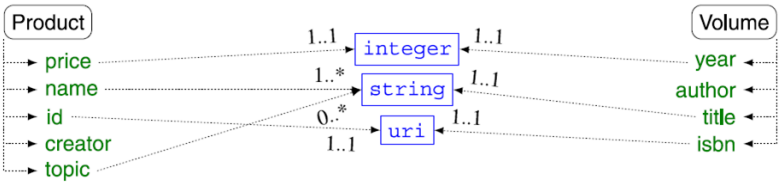
Pri **večjezičnih metodah** imamo opravka z ontologijami, ki so lahko enojezične ali večjezične. Z jeziki ontologij OWL ali SKOS [48] je možno za (imenske) značke opredeliti v katerem jeziku so zapisane (npr. "Article"@en). Večjezično ujemanje ontologij lahko poteka na dva načina: s primerjavo z vmesnim jezikom (ang. Pivot language - obe ontologiji se prevedeta v vmesni jezik) ali s tehniko "cross-translation" (prvo ontologijo se v celoti prevede v jezik druge) [49].

V pristopu, ki ga predlagamo, uporabljamo tovrstne tehnike (konkretno imenske značke @) za ugotavljanje jezika, v katerem je določen element (niz razreda ali njegovega primerka) zapisan in rezultate uporabimo pri lematizaciji.

Interno-strukturne tehnike

Interno-strukturne tehnike [38] so algoritmi, ki izkoriščajo informacije o notranji strukturi entitet oz. o njihovi definiciji, kot so na primer: tip, kardinalnost

atributov in ključi.



Slika 3.4: Na sliki je prikazano usklajevanje razredov: "Product"in "Volume"na podlagi njihovih tipov.

Tehnike, ki izkoriščajo tovrstne podatke so zelo učinkovite in enostavne za implementacijo. Vendar notranja struktura za primerjanje entitet ne zagotavlja dovolj informacij, da bi se lahko popolnoma zanesli na njih, zato jih pogosto uporabimo v kombinaciji z drugimi tehnikami.

Tehnike na osnovi formalnih virov

Tovrstne tehnike v postopku uskladitve ontologij, izbrane ontologije najprej povežejo z enim ali več formalnimi oz. zunanjimi ontologijami. Dobljene povezave pripomorejo pri iskanju ujemanj med ontologijami, ki jih usklajujemo.

3.1.2 Tehnike na nivoju strukture

V nasprotju s tehnikami na nivoju elementov, tehnike na nivoju strukture obravnavajo relacije med različnimi elementi oz. primerki znotraj iste ontologije. Pridobljene relacije se izkoristijo pri iskanju ujemanj oz. za primerjavo relacij z elementi oz. primerki druge ontologije. Pri usklajevanju elementov obeh ontologij se upoštevajo relacije z elementi znotraj posameznih ontologij. V nadaljevanju bodo predstavljene različne tehnike tega tipa.

Tehnike na osnovi grafov

V razred tehnik, ki temeljijo na grafih spadajo algoritmi, ki operirajo nad grafi. Tako morajo biti tudi vhodne ontologije, ki jih primerjamo, podane v obliki

označenih grafov. Običajno je podobnost med parom vozlišč oz. entitet iz dveh ontologij, ki jih primerjamo, definirana na podlagi njihovih položajev znotraj grafa. Ideja teh tehnik je ta, da če sta dve vozlišči iz primerjanih ontologij med seboj podobni, morajo biti podobni oz. morajo biti v relaciji tudi njihovi sosedje. Poleg tehnik na osnovi grafov, obstajajo tudi druge specifične tehnike, ki temeljijo na strukturi, kot na primer, tehnike, ki vključujejo drevesno strukturo oz. drevesa.

V našem pristopu je struktura oz. hierarhija razredov znotraj vhodnih ontologij indeksira s pomočjo sheme za označevanje intervalov (ang. interval labelling schema). To je podatkovna struktura za hranjenje usmerjenih acikličnih grafov ali dreves, in omogoča enostavno poizvedovanje po relacijah med razredi vhodnih ontologij.

Tehnike na osnovi modelov

Glavna značilnost teh semantičnih tehnik je da uporabljajo semantiko na osnovi teoretičnega modela za dosego rezultata.

Algoritmi, ki temeljijo na teh modelih, upravljajo z vhodnimi ontologijami na podlagi njihove semantične interpretacije. Osnovna predpostavka je, da če sta dve entiteti enaki, potem si delita enako interpretacijo. Torej gre za deduktivne metode. Primeri tovrstnih tehnik so pogojna zadovoljivost (ang. "propositional satisfiability" (SAT)) in sklepanje na osnovi opisne logike (ang. "description logics (DL) reasoning").

Tovrstnim metodam je potrebno zagotoviti vezi (ang. anchors). To so entitete oz. ujemanja, ki smo jih, na podlagi drugih tehnik za usklajevanje ontologij, ocenili kot ekvivalentne. Vezi predstavljajo začetna ujemanja s pomočjo katerih deduktivne metode lahko delujejo. Semantične metode delujejo kot ojačevalci tehnik oz. orodij za usklajevanje ontologij. Za sklepanje na osnovi opisne logike v našem primeru uporabljamo stroj za sklepanje Hermit DL.

Tehnike na osnovi primerkov

Pri tehnikah na nivoju primerkov primerjamo primerke razredov iz različnih ontologij za ugotavljanje podobnosti med razredi. Tovrstne tehnike lahko temeljijo na sklepanju nad teorijo množic ali podrobnejši analizi podatkov in statističnih tehnikah. Tehnike pomagajo pri združevanju primerkov in/ali preračunavanju

razdalje med njimi. Pri tehnikah, ki temeljijo na primerkih na podlagi tehnik analize podatkov poznamo klasifikacijo, ki išče podobnost na podlagi razdalje (ang. "distance-based classification"), formalno analizo konceptov (ang. "formal concept analysis") in analizo korespondence (ang. "correspondence analysis"). Pri metodah, ki slonijo na statistični analizi pa poznamo distribucijo pojavnosti (ang. "frequency distributions").

V našem pristopu uporabljamo nekoliko enostavnejši pristop za preverjanje konteksta primerkov na osnovi njihovih lastnosti (podrobneje je predstavljen v poglavju 4.2.2).

3.2 Predlog rešitve in izbor orodja za usklajevanje ontologij

Cilj magistrske naloge je razviti pristop in metodo, ki določa način obogatitev shem in podatkov (primerkov) iz poljubne relacijske podatkovne baze. Da bi lahko podatke iz naše relacijske baze obogatili, jih je treba na kvaliteten način povezati z drugimi podatki.

V prvem koraku je treba torej izdelati čimbolj kvalitetno predstavitev podatkov v relacijski bazi, poiskati vire s katerimi bi naše podatke lahko obogatili in jih nato povezati. V relacijski bazi namreč posredno najdemo veliko metapodatkov, ki jih pri ugotavljanju konteksta podatkov lahko tudi uporabimo in tako uporabnika razbremenimo ročnega povezovanja shem. Podatke iz relacijske baze, skupaj z koristnimi metapodatki, tako najprej izvozimo v ustreznem formatu za semantični splet in jih v naslednjem koraku skušamo povezati z drugimi podatki na semantičnem spletu. Izziv bogatenja podatkov smo tako aplicirali na problem ujemanja ontologij, tudi zato smo v prejšnjem poglavju izvedli podroben pregled tehnik ujemanja.

V drugem sklopu magistrske naloge pa predstavljamo prototip, s katerim skušamo razvito metodo verificirati in evalvirali. Obstajajo različne odprtokodne rešitve, ki omogočajo usklajevanje in bogatenje podatkov, vendar tovrstni pristopi delujejo zgolj s semantično manj bogatimi podatki (npr. XML in CSV oblika). Naš prototip v prvi vrsti omogoča izvoz podatkov iz relacijske podatkovne baze skupaj s kontekstom. Natančen zajem teh metapodatkov je ključen in strukturne

tehnike pri ujemanju ontologij so tu v veliko pomoč.

Našega prototipa nismo razvili od začetka, ampak smo nadgradili obstoječo rešitev. Izbor smo izvedli s primerjavo sistemov iz področja usklajevanja ontologij in najprej zmanjšali število kandidatov. Orodij oz. sistemov za usklajevanje ontologij je namreč veliko. V grobem jih razdelimo na tri skupine: sistemi, ki temeljijo na shemah; sistemi, ki temeljijo na primerkih in kombinirani sistemi oz. sistemi, ki temeljijo na obeh vrstah informacij, torej shemah in primerkih, ki jih najdemo znotraj ontologij (seveda če so primerki v ontologijah na voljo). Tako razdelitev in orodja, ki spadajo v posamezno kategorijo, prikazuje slika 3.5.

Najprej smo v grobem definirali kriterije predizbora. Prvi kriterij je, da je orodje sposobno povezati oz. uskladiti tako razrede kot primerke poljubnih ontologij (kombiniran pristop).

Drugi kriterij po katerem smo izbirali orodje je, da nam omogoča možnost uskladitve naše ontologije s poljubno prosto dostopno ontologijo, kot je na primer DBPedia, Feebase ipd. Orodje mora torej biti sposobno dela z velikimi ontologijami, ki poleg podatkov o shemi vsebujejo tudi primerke razredov ontologij.

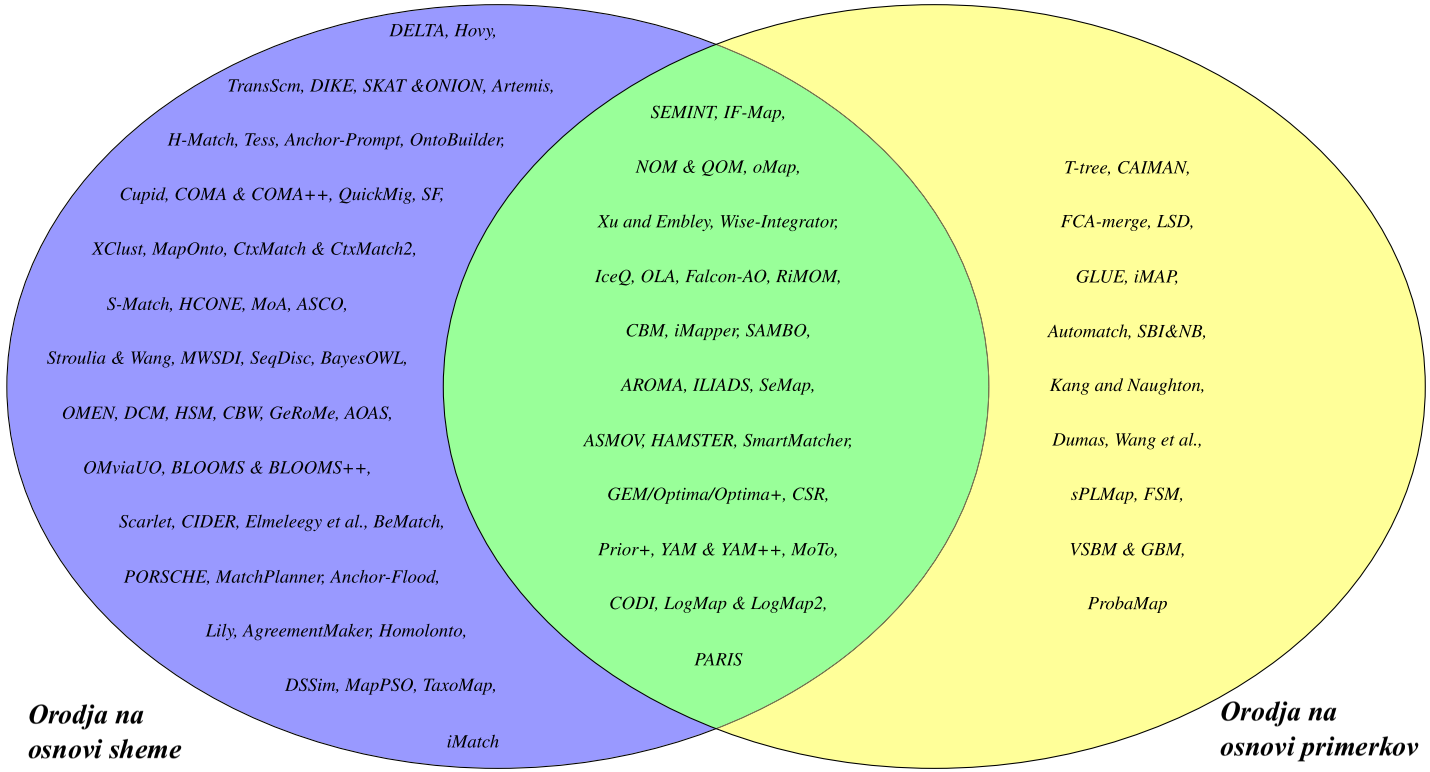
Tretji kriterij glede na katerega smo izbirali je, da mora biti orodje sposobno prepoznavanja konteksta entitet, ki jih primerjamo (bodisi razredov bodisi primerkov znotraj vhodnih ontologij, ki jih usklajujejo). To v praksi pomeni, da orodje ponuja napredne tehnike na nivoju strukture.

Naša rešitev je narejena za semantični splet in četrti kriterij je bila zato tudi zahteva, da je orodje sposobno dela z vhodnimi podatki v formatu RDF in v skladu s slovnico OWL. Pri naši rešitvi namreč podatke iz relacijske podatkovne baze s pomočjo preslikovalnega dokumenta R2RML in sistema za pretvarjanje podatkov v relacijski podatkovni bazi pretvorimo v RDF trojice.

Peti kriterij določa, da smo se omejili na orodja, ki so bodisi samodejna bodisi delno samodejna. V naslednji fazi smo potem dajali prednost delno samodejnim orodjem, kar v večini primerov pomeni, da je v proces usklajevanja udeležen tudi uporabnik in tako pripomore k višji natančnosti pri definiciji ujemanj. Na takšen način se zajamejo uskladitve, ki jih sistemi, npr. zaradi visoke dvoumnosti, samodejno ne zajamejo.

Ker smo vedeli, da bomo izbrano orodje, vsaj v manjši meri, tudi dopolnili, je bil nujen predpogoj odprtokodnost orodja. Posreden kriterij kvalitete orodja pri

izbiri je tudi aktiven razvoj. V ožji izbor so tako prišla le orodja, ki so na tem področju v zadnjih petih letih najbolj aktivna.



Slika 3.5: Pregled orodij za usklajevanje: V modrem delu (levo) je prikazanih 47 orodij, ki temeljijo na usklajevanju na osnovi shem (Delta, iMatch); v rumenem delu (desno) 15 orodij, ki temeljijo na usklajevanju na osnovi primerkov (T-tree, ProbaMap), in v zelenem polju (sredina) je prikazanih 27 mešanih sistemov (LogMap, Paris, RiMOM. . .).

Orodje	Vhod	Rezultat	Samodejno	Na nivoju elementa		Na nivoju strukture	
				Sintaktični	Semantični	Sintaktični	Semantični
SAMBO	OWL	Uskladitve	Da	Tehnike nad nizi, Naivni Bayes, WordNet	UMLS	Iteraktivna strukturna podobnost, ki bazira na relacijah is-a, part-of (tehnike na osnovi grafov)	-
Falcon-AO	RDFS, OWL	Uskladitve	Da	Tehnike nad nizi (n-gram, razdalja urejanja), UMLS, WordNet	-	Strukturna afiniteta	-
DSsim	SKOS, OWL	Uskladitve	Da	Tehnike nad nizi (Tokenizacija, Monger-Elkan, Jaccard), lingvistične tehnike, WordNet	-	Podobnost med grafi na podlagi listov	Rule-based fuzzy inference
RiMOM	OWL	Uskladitve	Da	Tehnike nad nizi (Razdalja urejanja, vektorska razdalja), Naivni Bayes, WordNet	-	Taksonomska struktura, Similarity propagation	-
ASMOV	OWL	Uskladitve	Da	Tehnike nad nizi (Tokenizacija, enakost med nizi, mera podobnosti nizov Winkler), lingvistične tehnike, WordNet	UMLS	Iterative fix point computation, hierarhije, restriction similarities	Rule-based inference
Anchor-Flood	RDFS, OWL	Uskladitve	Da	Tehnike nad nizi (Tokenizacija, enakost med nizi, Levenshtein distance), lingvistične tehnike, WordNet	UMLS	Internal, external similarities; iterative anchor-based similarity propagation	-
COMA & COMA++	XSD, XML, OWL, relational schemas	Uskladitve	Da	Tehnike nad nizi, lingvistične tehnike, Interno-strukturne tehnike (podatkovni tipi), tezavri	Ponovna uporaba uskladitev	Tehnike na osnovi grafov, tehnike na osnovi formalnih virov (repozitorijev struktur)	-
YAM++	XML, OWL	Uskladitve	Da (semi)	Metode strojnega učenja, SecondString, SimMetrics, WordNet	-	Structure profiles, Tehnike na osnovi grafov (Similarity flooding)	-
LogMap	OWL API (RDF/XML, OWL...)	Uskladitve, prekrivanja	Da (semi)	Tehnike nad nizi (UMLS, SMOA, stemming), lingvistične tehnike	UMLS	Primerjava struktur, Tehnike na osnovi grafov (shema za intervalno označevanje)	Hornovi stavki, Dowling-Gallier algoritem

Tabela 3.1: Značilnosti orodij na področju usklajevanja, ki delujejo nad shemami in primerki - mešani pristopi

Na podlagi prej omenjenih kriterijev so v ožji izbor prišla orodja, ki so prikazana v tabeli 3.1. Značilnosti orodji smo prikazali v skladu s klasifikacijo prikazano na zgornjem drevesu na sliki 3.2. Poskušali smo sprejeti enotno predstavitev pristopov primerjanih orodij, vendar smo zaradi podrobnejšega opisa namesto terminologije pristopov pri usklajevanju ontologij iz poglavja 3.1 uporabili enake opise, kot so jih definirali razvijalci posameznih sistemov.

Prva polovica tabele ponuja splošen pregled nad izbranimi sistemi. Stolpec "Vhod" predstavlja vhodni tip podatka, nad katerim operirajo posamezna orodja. V stolpcu "Rezultat" je zavedeno, v kakšni obliki oz. kaj nam orodje vrne kot rezultat. Druga polovica tabele uvršča pristope usklajevanja glede na uporabljeno klasifikacijsko tehniko po kriteriju nivo granularnosti oz. interpretacije vhoda oz. v skladu z zgornjim drevesom slike 3.2.

Na podlagi primerjave orodij, ki so predstavljena v tabeli 3.1 lahko opazimo, da vsa orodja v svoj proces usklajevanja ontologij vključujejo več različnih pristopov. Kot smo že v prejšnjem poglavju omenili, se ravno takšna orodja (kombinirani pristopi) po podatkih iz [38] v praksi izkažejo kot najuspešnejša.

Vsi sistemi v tabeli so samodejni. Takšni sistemi nam omogočajo samodejno usklajevanje vhodnih ontologij na podlagi različnih tehnik, ki so opisane v drugem delu tabele 3.1.

Vsi izmed sistemov uporabljajo različne tehnike na nivoju nizov (n-gram, tokenizacija, Jaccardov algoritem, Winkler, Levenshteinova razdalja), s katerimi lahko ugotovimo podobnost med primerjanimi nizi elementov. Na nivoju elementov, različna orodja (SAMBO, ASMOV, Anchor-Flood, LogMap) za semantični opis elementov uporabljajo namenski tezaver UMLS (Unified Medical Language System) s področja biomedicine.

Na nivoju strukture se uporabljajo različne tehnike, s katerimi se informacije, ki jih pridobimo iz strukture vhodnih ontologij (razredi, primerki in relacije med njimi) predstavijo v različnih urejenih strukturah (usmerjen graf, drevo, hierarhija itd.). S pomočjo teh struktur se ugotavlja ali primerjani objekti sodijo v isti kontekst, kar veliko pripomore k odkrivanju pravilnih uskladitev. Redka orodja uporabljajo semantične tehnike na nivoju strukture. Eno izmed orodij, ki uporablja tovrstne tehnike je LogMap, ki z uporabo Hornovih stavkov in Dowling-Gallier algoritmom [50, 51] preverja ustreznost ujemanj, ki so bile najdene s pomočjo teh-

nik, ki delujejo na nivoju nizov. Delovanje LogMap sistema podrobneje opisujemo v poglavju 4.2.

Na koncu smo se odločili prav za orodje LogMap. Orodje delno samodejno - pri prikazu rezultata nam prikaže tudi uskladitve za katere smatra da niso pravilne. Če uporabnik presodi, da je bilo katero izmed ujemanj, ki ga je orodje označilo kot nepravilno v resnici pravilno, lahko to ujemanje prenese v nabor pravilnih. Orodje je odprtokodno v pravem pomenu besede. Izvorno kodo smo z lahkoto našli in pridobili, kar ne moremo reči za vsa orodja, ki se sicer razglašajo za odprtokodna – npr. DSSim in ASMOV.

LogMap je zelo napredno orodje in ima implementirane algoritme/tehnike, ki so sposobne izrabljati informacije na vseh nivojih (strukture in elementa). Uporablja paleto strukturnih tehnik na osnovi grafov in modelov, vendar pa ne uporablja tehnik na osnovi primerkov. To za nas ni bilo moteče, saj smo že na začetku imeli idejo kako ta korak izboljšati. Nasprotno, orodja kot so SAMBO, Falcon-AO, Anchor-Flood, Rimom, COMA in YAM++, nimajo implementiranih naprednih tehnik na semantičnem nivoju.

Zaradi lastnosti slovenskega jezika, smo predvidevali da krnjenje besed, kot ga v osnovi ponuja orodje, verjetno ne bo dovolj in da bo treba poseči po lematizaciji. Kot omejitev se je pokazalo tudi, da LogMap za bogatenju uporablja zbirko UMLS, torej zbirko iz področja biomedicine. Vendar smo v fazi pregleda ugotovili, da te ni problem zamenjati s kakšno drugo zbirko (v naši implementaciji je to WordNet).

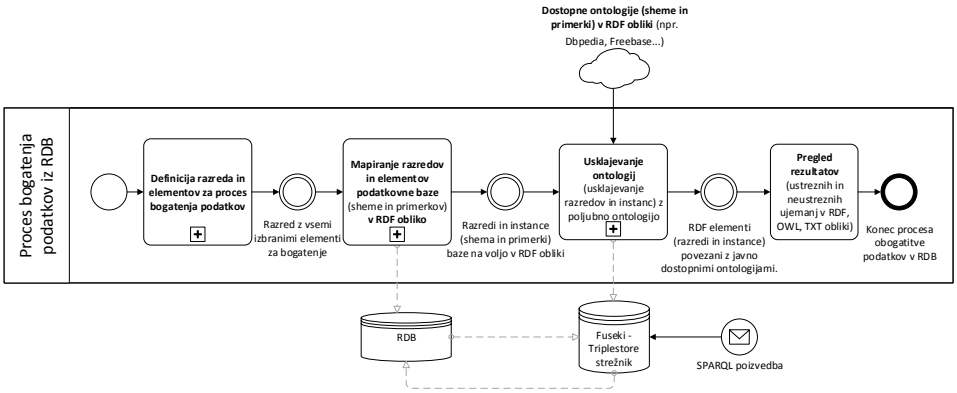
Poglavje 4

Metoda obogatitve podatkov v relacijski podatkovni bazi z Linked Data slovarji

Kot smo že v prejšnjem poglavju predstavili, je naš pristop sestavljen iz dveh glavnih korakov. Prvi korak zajema preslikavo sheme vhodne relacijske podatkovne baze v RDF obliko. V drugem koraku sledi preslikava. To je povezovanje sheme vhodne podatkovne baze, pretvorjene v ontologijo (skupaj s konkretnimi primerki), z obstoječimi prosto dostopnimi repozitoriji iz LOD oblaka (DBpedia, Freebase, Geonames, Doublin Core, SKOS ipd.). Tudi ti poleg podatkov o shemi hranijo še konkretne primerke. Vsakega izmed omenjenih dveh korakov smo razdelili še na dva koraka - prikazani so na sliki 4.1. Torej razširjen krovni proces procesa sestavljajo štiri koraki, ki jih bomo v nadaljevanju tudi podrobneje predstavili.

V prvem koraku procesa, ki je prikazana na sliki 4.1, se uporabnik poveže na poljubno podatkovno bazo in definira katere podatke bo uporabil za bogatenje. Pri tem se samodejno identificirajo tudi podporni razredi, ki pripomorejo k ugotavljanju konteksta pri procesu usklajevanja oz. bogatenja podatkov.

Drugi korak, ki je podrobneje predstavljen v poglavju 4.2, je korak preslikave podatkov iz relacijske podatkovne baze v RDF obliko. Na podlagi izbranih razredov oz. tabel za bogatenje iz prvega koraka, ki jih izbere uporabnik, se definira oz. ustvari preslikovalni dokument R2RML. Na podlagi tega dokumenta, se nato



Slika 4.1: Krovni proces pristopa bogatenja podatkov iz relacijske podatkovne baze.

s pomočjo orodja DB2triples [52] opravi pretvorba podatkov iz relacijske podatkovne baze v RDF obliko. Rezultat drugega koraka je, da je izbrana shema iz relacijske podatkovne baze z vsemi pripadajočimi primerki na voljo v RDF obliki. Podatke v taki obliki shranimo v triplestore strežnik Fuseki in po njih lahko že poizvedujemo s pomočjo jezika SPARQL.

V tretjem koraku smo problematiko bogatenja podatkov preslikali na področje usklajevanja ontologij. V ta namen smo v poglavju 3 opravili pregled najprej pristopov, nato pa še orodij iz področja usklajevanja ontologij. Namreč, ko smo se seznanili s sodobnimi tehnikami oz. pristopi na tem področju, smo lahko izbrali primerno odprtokodno orodje, ki smo ga nadgradili za potrebe našega problema. Na podlagi pregleda orodij na omenjenem področju smo si v poglavju 3.2 izbrali orodje LogMap, čigar delovanje je podrobneje predstavljeno v poglavju 4.2.1. Glede na identificirane izboljšave iz omenjenega poglavja, so predstavljene rešitve, ki jih kasneje v poglavju 4.3, evalviramo s testnimi podatki iz OAEI 2014. Nato jih še primerjamo z orodji, ki so se na prej omenjenem izzivu najbolje izkazali. Rezultat četrtega koraka so trojice oz. podatki v RDF formatu, ki so povezani z izbrano javno dostopno ontologijo.

V zadnjem, četrtem koraku, si lahko uporabnik ogleda rezultat uskladitve bodisi v RDF, CSV ali OWL obliki. Uporabniku so prikazane tudi uskladitve oz.

povezave, ki niso bile označene kot ustrezne. Tovrstne povezave lahko uporabnik sam odobri in jih ročno umesti med pravilne. Vsi podatki so povezani z javno dostopno ontologijo in so na voljo za poizvedovanje. S pomočjo poizvedovalnega jezika SPARQL lahko poizvedujemo po več podatkovnih zbirkah hkrati in tako našim "skritim" podatkom dodajamo nove dimenzije.

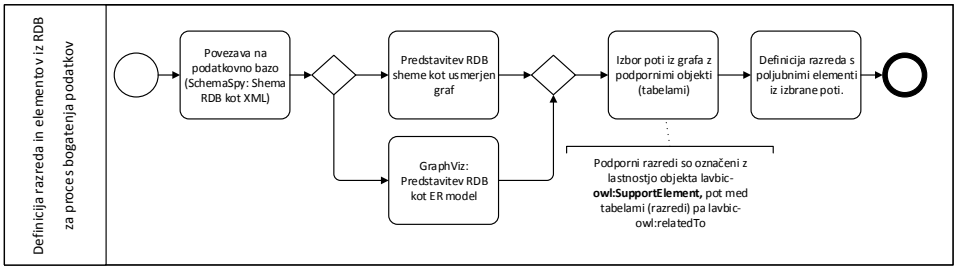
4.1 Predstavitev sheme relacijske podatkovne baze in izbira objektov za bogatenje

Prvi korak krovnega procesa obogatitve podatkov v relacijski podatkovni bazi je predstavitev sheme relacijske podatkovne baze in izbira objektov za bogatenje, ki je tudi prikazana na sliki 4.2. Da bi uporabnik lahko izbral želene podatke za proces obogatitve in pri tem sistemu pomagal izbrati primerne podporne razrede, ki bi pripomogli pri ugotavljanju konteksta izbranih elementov, shemo poljubne podatkovne baze prikažemo kot usmerjen graf ali kot ER model (poglavje 4.1). Način oz. pravila, ki smo jih upoštevali pri predstavitvi relacijske podatkovne baze v obliki usmerjenega grafa so opisana v poglavju 4.1.2. Uporabniku se na podlagi izbrane začetne in končne točke znotraj usmerjenega grafa ponudi seznam vseh možnih poti med izbranima točkama. Vse možne poti se preiščejo s pomočjo algoritma za iskanje v širino (BFS), ki je podrobneje predstavljen v poglavju 4.1.3. Sledi izbor želene poti iz katere se ustvari en razred, ki vsebuje atribute tabel. Tudi atribute tabel uporabnik izbere ročno. Ustvarjen razred se vpne v graf podpornih razredov, ki nam služijo za ugotavljanje konteksta in učinkovitejše bogatenje podatkov.

4.1.1 Predstavitev sheme podatkovne baze

V našem pristopu s pomočjo orodja SchemaSpy [53] opravimo analizo sheme izbrane podatkovne baze. Vse podatke o izbrani podatkovni bazi (relacije, ključi, pogledi idr.) zapišemo v XML dokument, ki se uporabi za kasnejšo predstavitev sheme, bodisi kot usmerjen graf bodisi kot E-R diagram.

SchemaSpy je odprtokodno javansko orodje za analizo metapodatkov podatkovne baze. Orodje nam omogoča prikaz entitetno-relacijskega (E-R) diagrama poljubne podatkovne baze. Deluje skoraj z vsako podatkovno bazo, ki podpira



Slika 4.2: Proces definicije podsheme in primerkov (razredov in elementov) iz relacijske podatkovne baze za proces bogatenja podatkov.

JDBC povezavo (Oracle, MySQL, DB2, MSSQL, PostgreSQL, Sybase ipd.). Je brezplačno orodje ki je objavljeno pod licenco GNU Public License 2.1.

Orodje SchemaSpy uporablja knjižnico Graphviz [54] za prikaz vseh objektov znotraj izbrane podatkovne baze. Pri tem se ustvari tudi XML datoteka v kateri je opisana celotna shema podatkovne baze, vključno z vsemi relacijami med tabelami in pogledi znotraj izbrane podatkovne baze.

4.1.2 Shema relacijske podatkovne baze kot usmerjen graf

Za lažji zajem vseh morebitnih podpornih razredov, izbrano podatkovno bazo predstavimo kot usmerjen graf v katerem so tabele podatkovne baze predstavljene kot vozlišča, ključi pa kot (usmerjene) povezave. V viru [55], je definiranih več pravil, ki jih je potrebno upoštevati pri gradnji usmerjenega grafa. Ta navodila opredelijo način, kako opisati relacije med tabelami v poljubni relacijski podatkovni bazi.

V našem pristopu smo pri gradnji grafa upoštevali pravila, ki so definirana v [55].

Vsaka relacijska podatkovna baza se lahko prikaže kot usmerjen graf $G = (T, F)$ v katerem: $T = \{T_1, T_2, ..., T_n\}$ predstavlja množico vseh tabel v podatkovni bazi, primer: vsaka tabela se obravnava kot vozlišče znotraj grafa. $F =$

$\{F_1, F_2, ..., F_m\}$ je množica vseh tujih ključev, ki so definirani znotraj podatkovne baze: vsak tuji ključ se obravnava kot usmerjena povezava $F_x = (T_a, T_b)$, kjer je $F_x \in F$ in $T_a, T_b \in T$.

Pri gradnji usmerjenega grafa na podlagi relacij med tabelami v podatkovni bazi smo upoštevali različna pravila s katerimi predstavimo relacije med tabelami:

- $T_x \in T$ je v odvisnosti oz. v relaciji z $T_y \in T$, če in samo če, obstaja pot med T_x in T_y.
- $T_x \in T$ je v neposredni odvisnosti oz. v relaciji z $T_y \in T$, če in samo če, obstaja $F_x \in F$, kjer je $F_x \in (T_x, T_y)$ oz. je F_x vsebovan v obeh tabelah.
- $T_x \in T$ ni odvisen od tabele $T_y \in T$, če in samo če, ne obstaja pot med T_x in T_y.
- $T_x \in T$ je odvisna sama s seboj natanko tedaj, ko obstaja neposredni cikel, ki vsebuje vozlišče T_x.

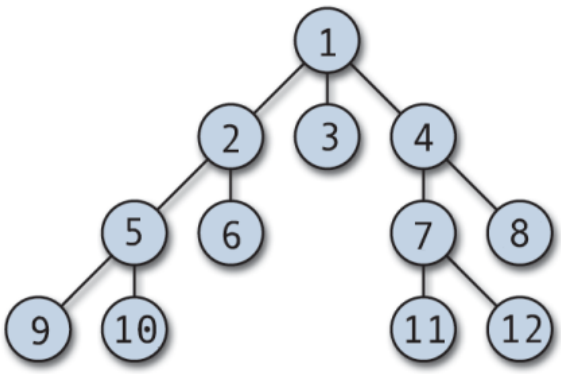
4.1.3 Iskanje vseh možnih poti v usmerjenem grafu

Z algoritmom za preiskovanje grafov, poiščemo vse možne poti znotraj usmerjenega grafa, ki predstavlja poljubno izbrano relacijsko podatkovno bazo. Natančneje, algoritem preišče vse možne poti med dvema poljubno izbranima vozliščema znotraj usmerjenega grafa. Naše orodje pa nato uporabniku omogoči izbiro poti, s pomočjo katere bo lažje odkril ali upošteval kontekst izbranih elementov za bogatenje.

Iskanje v širino oz. BFS (ang. Breadth-first search) je algoritem za preiskovanje grafov, dreves ali različnih struktur, ki so predstavljene v obliki grafa [56].

Osnovni algoritem preiskovanje (možne) poti začne v začetnem vozlišču in preišče vsa neposredno povezana oz. sosednja vozlišča, nato pa se pomakne za nivo globlje. Postopek ponovi dokler ne preišče vseh vozlišč oz. dokler ne najde ciljnega vozlišča [57, 58].

V našem pristopu smo uporabili nekoliko prilagojen algoritem iskanja v širino, ki nam za razliko od osnovnega vrne vse možne poti med začetno in končno točko. Psevdokoda algoritma je prikazana na sliki 4.4. Uporabnik v procesu bogatenja podatkov definira začetno in končno vozlišče znotraj usmerjenega grafa $G = (V, E)$, ki predstavlja izbrano podatkovno bazo. Algoritem v prvem koraku vzame začetno



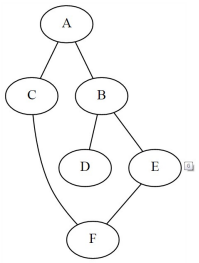
Slika 4.3: Zaporedje preiskovanja drevesa.

Vhod: Graf G, vozlišče $v_{zacetek} \in G$ in vozlišče $v_{cilj} \in G$
Izhod: Množica vseh poti iz vozlišča $v_{zacetek}$ do v_{cilj}

```
1  function BFS(G; v_zacetek; v_cilj)
2    Množica R //množica najdenih poti
3    Množica Q //množica poti za obdelavo
4    dodaj {v_zacetek} pod ključem v_zacetek v Q //dodamo začetno pot v Q
5
6    while Q ni prazna do
7      vzemi par {v_zacasni, P} iz množice Q
8        for each n in {G[v_zacasni] - množica P} // brez že obiskanih vozlišč iz množice P
9          if n enak v_cilj then // pot najdena
10           dodaj n na konec množice P
11           dodaj P v množico najdenih poti R
12         else
13           dodaj n na konec množice P
14             dodaj P pod ključem n v Q
15         end if
16   end while
17   return R
18 end function
```

Slika 4.4: Psevdokoda algoritma za iskanje vseh poti – iskanje v širino.

vozlišče in ta se kot začetna pot (pot, ki je sestavljena iz enega elementa) doda v vrsto Q. Nato se zaporedoma izloči po en element P (pot) iz vrste Q. Za zadnji element iz poti P, algoritem preišče vsa vozlišča, ki so v njegovi neposredni bližini (sosede). V primeru, da se končni element iz poti P ne ujema s ciljnim v_{cilj} vozliščem in če predhodno nismo še bili v tem vozlišču, se vozlišče doda na konec trenutne poti, pot pa se doda v vrsto Q. Algoritem teče dokler ne preišče vseh vozlišč znotraj grafa G oz. dokler se vrsta Q ne izprazni. Algoritem deluje tako na usmerjenih kot na neusmerjenih grafih. Primer delovanja je prikazan spodaj [59].



Slika 4.5: Testni graf.

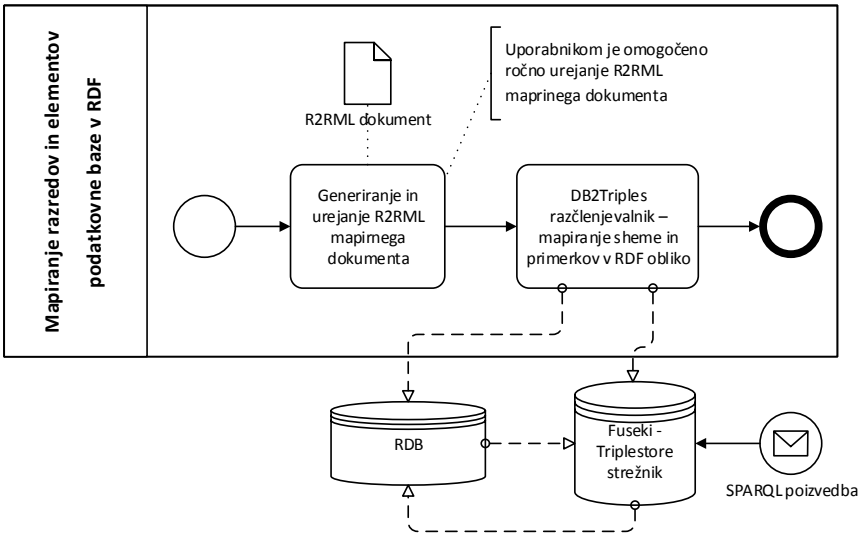
```
dfs_paths(graph, 'A', 'F'))
rezultat: # [['A', 'C', 'F'], ['A', 'B', 'E', 'F']]
```

Dobra lastnost algoritma iskanja v širino je, da nam kot prvi rezultat poda najkrajšo pot. Ker algoritem preiskuje vozlišča v širino (na prvem nivoju eno vozlišče, na drugem b vozlišč, na tretjem b^2) in vse možne poti shrani na sklad, je časovna zahtevnost algoritma eksponentna: Torej $O\left(b^d\right)$, kjer b predstavlja faktor vejitve oz. največje število naslednikov kateregakoli vozlišča in d predstavlja globino iskanega oz. ciljnega vozlišča [60]. Ker algoritem na skladu hrani vse možne poti, ki jih je v danem trenutku preiskal, je prostorska zahtevnost algoritma eksponentna, torej $O\left(b^d\right)$.

4.1.4 Preslikava RDB2RDF

Na podlagi ustvarjenega razreda s poljubnimi atributi izbranih tabel, se v tem koraku ustvari preslikovalni dokument R2RML [6] (poglavje 2.3.5). S pomočjo tega

dokumenta, se z orodjem db2triples opravi preslikava izbrane podsheme in njenih pripadajočih podatkov v RDF trojice. Omenjeno orodje nam omogoča izvoz trojic v različne formate in tudi neposredno v triplestore strežnik. V našem primeru je to strežnik Fuseki - predstavljen je v poglavju 2.5.2, preko katerega lahko s pomočjo poizvedovalnega jezika SPARQL (predstavljen v poglavju 2.3.4) poizvedujemo po prevedenem viru. V poglavju evalvacije metode je tudi podrobneje predstavljen sam proces orodja, ki je nastalo za ta namen.



Slika 4.6: Proces koraka preslikave razredov in elementov iz relacijske podatkovne baze v RDF obliko.

Na sliki 4.6 je prikazan proces preslikave izbrane podsheme iz poljubne relacijske podatkovne baze z njenimi pripadajočimi podatki (razredov in elementov) v RDF obliko. V prvem koraku procesa se na podlagi izbranih tabel oz. poti v usmerjenem grafu ustvari preslikovalni dokument R2RML, katerega lahko uporabnik po svoji želji poljubno spremeni (dodatni razredi, urejanje logičnih tabel ipd.). Naslednji korak v procesu zajema preslikavo podsheme in njenih podatkov iz relacijske podatkovne baze v RDF obliko. V fazi evalvacije metode se v tem koraku uporablja orodje za preslikovanje podatkov iz relacijske podatkovne baze v trojice

DB2Triples. Sprva je bila v tem koraku predvidena uporaba platforme Virtuoso[6], ki prav tako podpira tovrstne preslikave podatkov v relacijskih podatkovnih bazah, vendar smo v fazi implementacije okolja za evalvacijo zastavljenega pristopa, prišli do omejitve. Omenjena platforma namreč ne podpira definicijo logičnih tabel z SQL poizvedbami (element "rr:sqlQuery"). Ta funkcionalnost je ključnega pomena v našem pristopu, saj uporabniku omogoča ročno prilagoditev preslikovalnega dokumenta in tako v proces bogatenja podatkov iz relacijske podatkovne baze lahko vključi različne podatke. Ti so ključni pri ugotavljanju konteksta podatkov, ki jih bogatimo oz. usklajujemo z javno dostopnimi ontologijami (poljubnim virom iz LOD oblaka). V okviru iskanja primernega orodja smo uporabili naslednjo raziskavo [33]. Znotraj nabora orodij, ki so na voljo, smo se odločili za odprtokodno orodje DB2Triples, ki se je dobro izkazalo na preizkušnji, ki je predstavljena v [33].

Prejšnji korak predlaganega pristopa je definiral izbor poti iz usmerjenega grafa, ki prestavlja shemo poljubno izbrane podatkovne baze. Poleg tabel iz poti se izberejo tudi njim ustrezni podatki. Izbor pa temelji na SQL poizvedbi oz. na pogledu R2RML, ki je definiran znotraj logične tabele (rr:logicalTable) preslikovalnega dokumenta R2RML. V SQL poizvedbi poleg podatkov zgradimo tudi enoličen identifikacijski element oz. podatek, na podlagi katerega lahko povežemo podatke v isti kontekst (npr. ime in priimek, ki sta običajno ločena podatka – več o izboljšavi v poglavju 4.2.2).

Kot smo že v predhodnih poglavjih omenili, v relacijski podatkovni bazi hranimo veliko metapodatkov, ki bi jih lahko uporabili v procesu bogatenja podatkov znotraj le teh. oz. v procesu njihovega usklajevanja z drugimi ontologijami. Tako smo dosegli pravilnejši rezultat bogatenja oz. višje število pravilnih ujemanj z javnimi ontologijami. V sam proces vključujemo različne metapodatke, ki jih tudi ustrezno označimo (natančen opis podpornih elementov je v 4.2.2). Na tem mestu zgolj izpostavimo, da so podporni razredi označeni z lastnostjo "lavbic-owl:supportClass". Izbrano pot v grafu in vse relacije med izbranimi razredi označimo z lastnostjo "lavbic-owl:relatedTo" (več o lastnostih, ki smo jih vpeljali kot izboljšava orodja LogMap so predstavljene v poglavju 4.2.2). Na sliki 4.7 je prikazan primer podpornega razreda v preslikovalnem dokumentu, ki vsebuje obe prej omenjeni lastnosti in logično tabelo definirano z SQL poizvedbo.

[6] `http://www.w3.org/wiki/VirtuosoUniversalServer`

```
<#TriplesMapCountryCityAddressStoreStaff> a rr:TriplesMapClass;
  rr:logicalTable [ rr:sqlQuery
    """SELECT CONCAT(country.country_id, '-', city.city_id, '-',
               address.address_id, '-', store.store_id, '-', staff.staff_id) AS id,
               country.country, city.city, address.address, staff.first_name,
               staff.last_name, staff.username
    FROM country, city, address, store, staff
        WHERE country.country_id = city.country_id AND
              city.city_id = address.city_id AND
              address.address_id = store.address_id AND
              store.store_id = staff.store_id""" ] ;
rr:subjectMap [ rr:constant "lavbic-sakila:CountryCityAddressStoreStaff" ; ];

rr:predicateObjectMap [
  rr:predicateMap [ rr:constant owl:Class ] ;
  rr:objectMap [ rr:constant "lavbic-sakila:CountryCityAddressStoreStaff" ] ;
] ;

rr:predicateObjectMap [
       rr:predicateMap [ rr:constant lavbic-owl:SupportClass ] ;
       rr:objectMap [ rr:constant "TRUE" ] ;
] ;

rr:predicateObjectMap [
       rr:predicateMap [ rr:constant lavbic-owl:relatedTo ] ;
       rr:objectMap [ rr:constant "lavbic-owl:Country" ] ;
] ;
...
rr:predicateObjectMap [
  rr:predicate rdfs:label ;
  rr:objectMap [ rr:constant "CountryCityAddressStoreStaff" ] ;
] .
```

Slika 4.7: Izsek R2RML preslikovalnega dokumenta, kjer so definirani podporni razredi.

4.2 Usklajevanje ontologij

Linked open data oblak vsebuje veliko število med seboj povezanih primerkov, iz katerih lahko dobimo veliko novih informacij oz. lahko podatkom, ki jih povežemo z njimi, dodamo nov pomen in različna dodatna znanja. V okviru postopka obogatitve podatkov v relacijski podatkovni bazi z Linked data slovarji, se v tem koraku problematika aplicira na področje usklajevanja ontologij.

Na podlagi klasifikacije pristopov, tehnik in orodij na področju usklajevanja ontologij, ki je bila predstavljena v poglavju 3, smo v poglavju 3.2 predlagali rešitev na področju usklajevanja ontologij in se odločili, da bomo v tem koraku procesa bogatenja podatkov iz relacijske podatkovne baze vključili odprtokodno orodje in algoritem LogMap. Z namenom identifikacije izboljšav je v naslednjem poglavju podrobno predstavljeno delovanje tovrstnega orodja. V poglavju, ki mu sledi, pa so predstavljene aplicirane izboljšave.

4.2.1 Identifikacija izboljšav in delovanje LogMap

LogMap (Logic-based and Scalable Ontology Matching) je odprtokodno orodje za usklajevanje velikih ontologij, ki je razvito v programskem jeziku Java. Velika prednost orodja je, da lahko operira s semantično bogatimi ontologijami, ki vsebujejo veliko število entitet, razredov oz. več milijonov trojic (npr. NCI[7], FMA[8] ipd.). Orodje ima vgrajeno funkcionalnost sklepanja (ang. reasoning), kar nam omogoča uspešno zaznavanje neskladji znotraj usklajevanih ontologij.

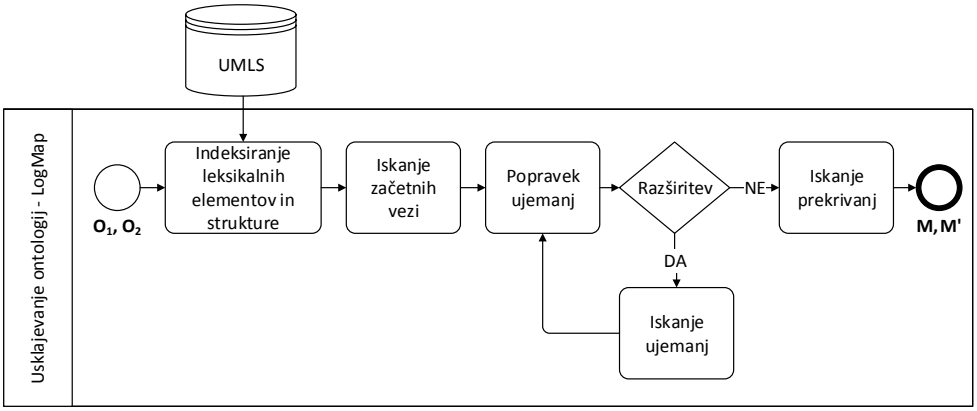
Kot smo že omenili v poglavju 3.2, orodje spada v kategorijo mešanih sistemov, saj, poleg usklajevanja razredov znotraj ontologij, podpira usklajevanje lastnosti in primerkov.

LogMap uporabnikom omogoča vključitev interakcije med procesom usklajevanja, kjer lahko uporabnik neposredno vpliva na odločitve orodja. Funkcionalnost je namenjena detekciji povezav, ki jih sistem ni sposoben poiskati. V poglavju 3 smo pokazali, da nam orodje ponuja različne sodobne tehnike na področju usklajevanja ontologij, ki delujejo bodisi na nivoju elementa bodisi na nivoju strukture vhodnih ontologij.

[7] `http://bioportal.bioontology.org/ontologies/NCIT`

[8] `http://sig.biostr.washington.edu/projects/fm/AboutFM.html`

Orodje pri branju ontologij uporablja programsko orodje OWL API in tako omogoča podporo različnim formatom zapisom ontologij (RDF/XML, OWL/XML, OWL Functional, OBO, KRSS, in Turtle - N3). OWL API je odprtokodni vmesnik, ki je razvit v programskem jeziku Java in nam poenostavi rokovanje z jezikom "Web Ontology Language" (OWL), s katerimi so predstavljene ontologije.



Slika 4.8: Proces usklajevanja ontologij znotraj orodja LogMap.

Na sliki 4.8 je prikazan proces usklajevanja ontologij znotraj orodja LogMap, ki na vhodu sprejme ontologiji O_1 in O_2. Kot rezultat, poleg vseh pozitivnih ujemanj (ujemanja, ki jih je označil kot pravilna) M med vhodnima ontologijama, vrne tudi ujemanja, ki jih je v fazi usklajevanja označil kot nepravilna - ta so označena s M'. V nadaljevanju bodo podrobneje opisani posamezni koraki.

Prvi korak procesa usklajevanja z orodjem LogMap je **indeksiranje leksikalnih elementov** in strukture vhodnih ontologij. LogMap najprej razčleni vhodne ontologije tako, da za vsako zgradi t.i. "invertirane leksikalne indekse" (primer invertiranih leksikalnih indeksov z razredi ontologije je prikazan v tabeli 4.1). Indeksiranje poteka tako nad razredi kot nad primerki obeh vhodnih ontologij. V indeksiranje se zajame tudi variacija nizov, kot je krnenje (ang. stemming). Prav tako se upoštevajo tudi alternativni nizi (sinonimi, podpomenke, sopomenke), ki jih orodje pridobi iz zbirke slovarjev iz področja biomedicine, UMLS [61] (Unified Medical Language System). Ker je operacija leksikalnega indeksiranja časovno zelo zahtevna, se izvede za vsako vhodno ontologijo le enkrat.

Ker se v našem pristopu nismo želeli omejiti na neko specifično področje (medi-

Invertirani indeksi nizov znotraj ontologije NCI		**ID elementov ontologije NCI**	
Niz	**ID**	**ID**	**URI**
secretion	49901	49901	NCI:CellularSecretion
cellular,secretion	49901	37975	NCI:Trapezoid
cellular,secrete	49901	62999	NCI:TrapezoidBone
trapezoid	37975,62999	60791	NCI:Smegma
trapezoid,bone	62999		
smegma	60791		
Invertirani indeksi nizov znotraj ontologije FMA		**ID elementov ontologije FMA**	
Niz	**ID**	**ID**	**URI**
secretion	36792	36792	FMA:Secretion
bone,trapezoid	20948,47996	47996	FMA:Bone_of_Trapezoid
trapezoid	20948	20948	FMA:Trapezoid
smegma	60947	60947	FMA:Smegma

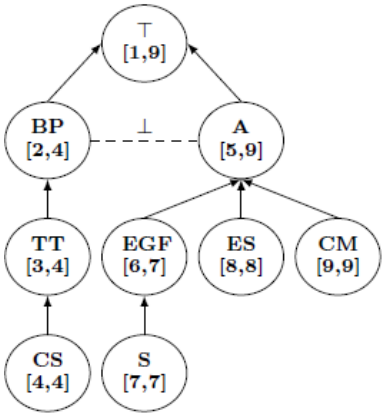
Tabela 4.1: Del leksikografskih indeksov ontologij NCI in FMA.

cina, poslovna znanost ipd.) oz. ker smo želeli zajeti neko splošno domeno znanja, smo zbirko slovarjev UMSL nadomestili s slovarjem Wordnet [42]. Znotraj prototipa zastavljenega pristopa, uporabniku omogočamo tudi urejanje in dodajanje trditev v slovar, ki temelji na prej omenjenemu WordNetu. Več o izboljšavi si lahko preberete v poglavju 4.2.2. Prav tako smo v tem koraku krnjenje zamenjali z lematizacijo, več o zamenjavi si lahko preberete v poglavju 4.2.2. V našem pristopu indeksiranje izvajamo nad nizi iz katerih smo odstranili mašila. Uporabniku poleg urejanja slovarja zato omogočamo tudi urejanje in dodajanje mašil.

V koraku **indeksiranja strukture**, orodje LogMap, s pomočjo orodij za sklepanje (npr. HermiT [9]), za vsako vhodno ontologijo zgradi hierarhijo razredov (umeščeni samo razredi vhodnih ontologij). Hierarhije razredov se indeksirajo s pomočjo sheme za označevanje intervalov (ang. interval labelling schema) [62, 63, 64]. To je podatkovna struktura za hranjenje usmerjenih acikličnih grafov ali dreves, ki v fazi preiskovanja konteksta elementov oz. v fazi ugotavljanja

[9] `http://hermit-reasoner.com/`

ujemanj med razredi vhodnih ontologij, omogoča enostavno poizvedovanje po relacijah med njimi. To vpliva na mero podobnosti med elementi, ki jih usklajujemo. Poizvedovanje po tovrstni strukturi je računsko zelo nezahtevna operacija, kar je lahko omejujoče pri delu z velikimi ontologijami kot so DBpedia, Freebase ipd. Zato se za vsako od vhodnih ontologij izvede zgolj enkrat.



Slika 4.9: Hierarhija razredov ontologije NCI. Okrajšave: BP=BiologicalProcess, A=Anatomy, TT=TransmembraneTransport, CM=CellularMembrane, EGF=ExocrineGlandFluid, CS=CellularSecretion, ES=ExocrineSystem, S=Smegma.

Na sliki 4.9 je prikazan primer hierarhije razredov, ki jo dobimo v procesu indeksiranja strukture vhodnih ontologij. Za lažje razumevanje poizvedovanja nad tovrstnimi strukturami vzemimo tale primer: če bi radi znotraj hierarhije razreda iz slike 4.9 ugotovili, ali je S podrazred razreda A, preverimo ali je indeks razreda S vsebovan v intervalu razreda S. Torej, ker velja trditev da $7 \in [5, 9]$, velja tudi, daj je S podrazred razreda A.

V drugem koraku pridobimo množico začetnih vezi in sicer tako, da se izvede presek množic invertiranih indeksov obeh vhodnih ontologij, ki jih zgradimo v fazi indeksiranja leksikalnih elementov. Ujemanja, ki jih dobimo, dodatno preverimo in sicer na podlagi primerjave elementov, ki jo izvedemo v kombinaciji s Stoilosjevo mero podobnosti nizov (podrobneje predstavljena v poglavju 4.2.1) in mere zaupanja. Mera zaupanja temelji na principu lokalnosti [65]. Torej, če sta razreda

Element	FMA id	NCI id	Ujemanja
secretion	36792	49901	FMA:Secretion ≡ NCI:CellularSecretion
smegma	60947	60791	FMA:Smegma ≡ NCI:Smegma
trapezoid	20948	37975, 62999	FMA:Trapezoid ≡ NCI:Trapezoid FMA:Trapezoid ≡ NCI:TrapezoidBone
trapezoid,bone	20948, 47996	62999	FMA:Trapezoid ≡ NCI:TrapezoidBone FMA:Bone of Trapezoid ≡ NCI:TrapezoidBone

Tabela 4.2: Del preseka množic vhodnih ontologij (presek množic invertiranih indeksov ontologij FMA in NCI).

C_1 in C_2 označena kot usklajena, potem vsi razredi, ki so v relaciji z razredom C_1 znotraj vhodne ontologije O_1, z veliko verjetnostjo ujemajo z vsemi razredi, ki so v relaciji z razredom C_2 znotraj vhodne ontologije O_2 (primer je prikazan na sliki 4.10). Več o Stoilos-jevi meri podobnosti nizov si lahko preberete v poglavju 4.2.1. Dobljena ujemanja se smatrajo kot točna in se v nadaljevanju procesa usklajevanja zgolj dopolnjujejo. V tabeli 4.2, je prikazan rezultat preseka invertiranih indeksov, ki so prikazani v tabeli 4.1.

Mera zaupanja pa ne deluje na nivoju primerkov. Primerki se usklajujejo oz. primerjajo med seboj zgolj na podlagi leksikografskega ujemanja in sicer s pomočjo Stoilos-jeve mere podobnosti nizov. Iz tega sledi, da osnovna implementacija orodja LogMap pri ujemanju primerkov ne upošteva njihovega konteksta. Zato smo v poglavju 4.2.2 predstavili novo mero, pri kateri kontekst primerkov upoštevamo.

Uskladitve na nivoju primerkov, ki jih dobimo v tem koraku veljajo že kot končne, saj tehnike preverjanj uskladitev v naslednjih korakih delujejo samo nad razredi.

Naslednji korak je predstavljen kot zanka med dvema korakoma: "popravek ujemanj" in "iskanje ujemanj". Omenjena koraka se izvajata v zanki dokler slednji ne uspe najti novih ujemanj razredov na podlagi lokalnosti (dokler se ne preišče lokalnost za vse razrede znotraj vhodnih ontologij).

Tretji korak je **korak popravka ujemanj**, v katerem se s pomočjo tehnik sklepanja in na podlagi do sedaj pridobljenih ujemanj obeh vhodnih ontologij, identificirajo razredi, katerih ujemanja niso zadovoljiva. Podrobneje, začetne vezi

Propositional FMA (P1)		**Propositional NCI (P2)**	
(1)	Smegma → Secretion	(8)	Smegma → ExocrineGlandFluid
(2)	Secretion → PortionBodySusbstance	(9)	ExocrineGlandFluid → Anatomy
(3)	PortionBodySusbstance → AnatomicalEntity	(10)	CellularSecretion → TransmembraneTransport
Computed mappings (PM)		(11)	TransmembraneTransport → TransportProcess
(m_4)	FMA:Secretion → NCI:CellularSecretion	(12)	TransportProcess → BiologicalProcess
(m_5)	NCI:CellularSecretion → FMA:Secretion	(13)	Anatomy ∧ BiologicalProcess → false
(m_6)	FMA:Smegma → NCI:Smegma	(14)	ExocrineGlandFluid ∧ ExfolCells → Smegma
(m_7)	NCI:Smegma → FMA:Smegma		

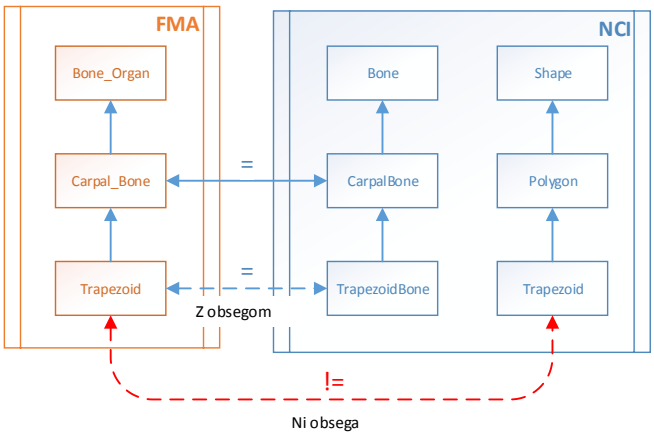
Tabela 4.3: Del preseka množic vhodnih ontologij (presek množic invertiranih indeksov ontologij FMA in NCI).

(ujemanja, ki smo jih označili kot ustrezna) in hierarhije razredov s katerimi so predstavljene vhodne ontologije, se zapišejo v obliki Hornovih stavkov (predikatna logika za zapis hipotez, ki je močnejša od Boolove algebre). Ti tvorijo podmnožico izjav logike prvega reda. Za preverjanje ustreznosti Hornovih stavkov oz. ustvarjenih izjav prvega reda se uporablja algoritem Dowling-Gallier (deduktivna metoda). Algoritem je podrobneje opisan v [50].

V tabeli 4.3 so prikazani Hornovi stavki, pridobljeni iz hierarhij razredov ontologij FMA in NCI (preračunani s pomočjo orodja za sklepanje HermiT) in vezmi (ang. anchors). Kot lahko razberemo iz tabele se zaradi popravka ujemanj vsaka uskladitev preračuna v dva Hornova stavka oz. zapisa.

V naslednjem koraku algoritem LogMap s pomočjo mere zaupanja **išče nova ujemanja** nad hierarhijo razredov, ki je sedaj zgrajena zgolj z začetnimi vezmi. Na podlagi lokalizacije se pridobi nova množica potencialnih ujemanj. Dobljena ujemanja se primerjajo s pomočjo Stoilos-jeve mere podobnosti nizov. Razredi, ki presežejo prag ustreznosti ujemanja se označijo kot ustrezni, ostali pa kot neustrezni. Algoritem LogMap nadaljuje interakcijo popravkov in iskanja novih ujemanj, dokler se v zadnji fazi kontekst potencialno ujemajočih se razredov ne razširi več oz. dokler na podlagi vidika lokalizacije ne dobimo več novih ujemanj.

Zadnji korak usklajevanja z algoritmom LogMap je korak ocene prekrivanja ontologij. Poleg možice ujemanj med vhodnimi ontologijami, ki so končni rezultat usklajevanja, sistem uporabniku ponudi na ogled okrnjene različice vhodnih ontologij. Te nam predstavljajo njihova prekrivanja oz. elemente, za katere je, v procesu usklajevanja, algoritem LogMap njihovo ujemanje ocenil kot pomanjkljivo

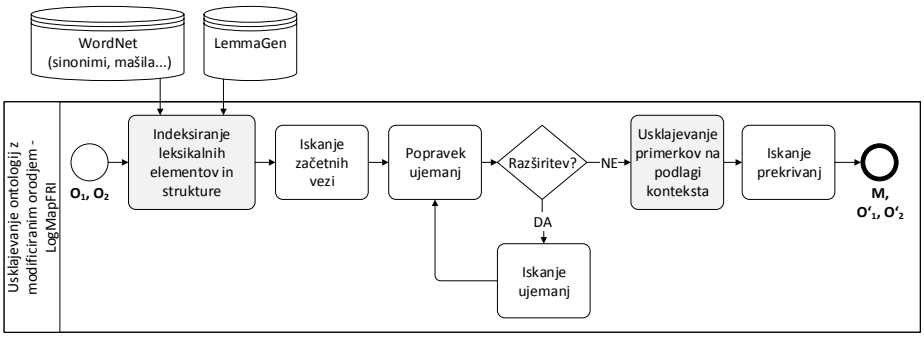


Slika 4.10: Primer mere vrednosti zaupanja, ki deluje po principu lokalnosti in pregleda obseg za vsa zaznana ujemanja med razredi iz ontologij FMA in NCI.

(nizek koeficient ujemanja) in jih zato ni uskladil v množico pravilnih uskladitev. Pri ročnem pregledu usklajevanj, se lahko uporabniki osredotočijo zgolj na tovrstna ujemanja in tako zajamejo izjeme, ki jih je LogMap označil kot pomanjkljive oz. neustrezne.

Uporabnik je tako lahko vključen v sam proces usklajevanja, tekom katerega se mu zastavi omejeno število vprašanj o skladnosti ujemanj, ki jih lahko bodisi zavrne bodisi sprejme.

Pri pregledu delovanja orodja smo zaznali, da orodje sicer usklajuje primerke, ki jih najdemo znotraj ontologij vendar pri tem ne ugotavlja njihovega konteksta oz. ne upošteva lastnosti primerkov v procesu usklajevanja. Zato smo v predzadnjem koraku procesa delovanja algoritma LogMap, vpeljali nov korak, in sicer korak **usklajevanja primerkov na podlagi konteksta**. Modificiran proces algoritma je prikazan na sliki 4.11, njegova nadgradnja pa je predstavljena v poglavju 4.2.2. V naslednjem poglavju pa je podrobneje predstavljena Stoilos-jeva metrika podobnosti nizov.



Slika 4.11: Predlagani proces usklajevanja ontologij.

Stoilos-jeva metrika podobnosti nizov (SMOA)

SMOA [66] je mera podobnosti nizov, ki je bila posebno razvita za uporabo v sistemih za usklajevanje ontologij. Glavna ideja mere je hkrati upoštevati mere razlikovanja med nizi in skupnih značilnosti med primerjanimi nizi. Definirana je s spodnjo formulo:

$$Sim\left(s_1, s_2\right) = Comm\left(s_1, s_2\right) - Diff\left(s_1, s_2\right) + JaroWinkler\left(s_1, s_2\right) \tag{4.1}$$

V koraku $Comm(s_1, s_2)$ se poišče najdaljši skupni podniz, ki se ga v naslednjem koraku iz obeh nizov odstrani, nato se ponovno poišče naslednji najdaljši podniz in tako naprej, dokler nam nič ne ostane. Nato se dolžine podnizov seštejejo in delijo z vsoto dolžin obeh nizov. Korak $Comm$ je definiran s spodnjo formulo:

$$Comm(s_1, s_2) = \frac{2 \times \sum length\left(maxCommonSubString\left(s_1, s_2\right)\right)}{length\left(s_1\right) + length\left(s_2\right)} \tag{4.2}$$

$Diff\left(s_1, s_2\right)$ dobimo s spodnjo formulo:

$$Diff\left(s_1, s_2\right) \quad = \quad \frac{uLen(s_1) \times uLen(s_2)}{p + (1-p) \times (uLen(s_1) + uLen(s_2) - uLen(s_1) \times uLen(s_2))} \tag{4.3}$$

uLen v zgornji formuli predstavlja dolžino neujemajočega se dela niza iz prvega koraka, ki jo delimo z dolžino ustreznega začetnega niza, in faktorja p, s katerim obtežimo pomembnost razlikovanja med nizi.

SMOA zajema področje vrednosti od -1 (kar predstavlja popolno neujemanje) do 1 (kar predstavlja popolno ujemanje oz. identičnost nizov).

Jaro-Winkler-jeva razdalja

Razdalja Jaro–Winkler [67] je mera podobnosti med dvema nizoma. Izhaja iz mere Jaro [67], pri kateri razdalja predstavlja minimalno število operacij, ki jih potrebujemo za preoblikovanje enega niza v drugega. Mera deluje tako, da prešteje ujemanja med znaki v primerjanih nizih. Mera Jaro-Winkler je namenjena ugotavljanju podobnosti med kratkimi nizi, kot so imena oseb ipd. Večja kot je vrednost mere med dvema nizoma, bolj sta si niza podobna. Vrednost 0 pomeni popolno razlikovanje, medtem ko vrednost 1 pomeni popolno ujemanje (identičnost) nizov.

Mera daje prednost nizom, ki si delijo enake predpone. Sloni na predpostavki, da so si podobni tisti nizi, ki imajo skupne korene, različne pa so končnice teh nizov. To so zlasti pridevniki in glagoli. Mera je definirana s spodnjo formulo:

$$JaroWinkler\left(s_1, s_2\right) = Jaro\left(s1, s_2\right) + \left(lp(1 - Jaro\left(s_1, s_2\right)\right) \tag{4.4}$$

l predstavlja dolžino skupne predpone niza, p pa predstavlja faktor, s katerim določamo težo pomembnosti skupne predpone.

$$Jaro\left(s_1, s_2\right) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \tag{4.5}$$

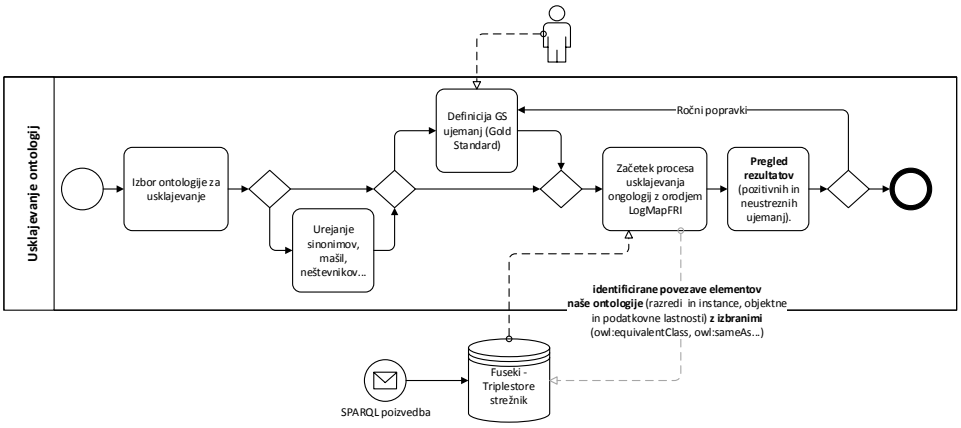
Mera $Jaro$ pa je definirana z formulo, v kateri m predstavlja število črk, ki jih primerjamo; t predstavlja število transpozicij, ki je definirano s številom ujemajočih se znakov deljeno z 2. Dva znaka znotraj nizov se ujemata če nista narazen za več kot $\lfloor\left(max\left(s_1.length, s_2.length\right)/2\right) - 1\rfloor$. Primer: "CRATE" in "TRACE": $m = 3$ in $t = 0$; ujemajo se črke "R", "A" in "E", čeprav se črki "C" in "T" pojavita v obeh nizih, se črke ne ujemajo, zato ker so si narazen za več kot $floor(5/2) - 1 = 1$.

4.2.2 Usklajevanje ontologij s prilagojenim orodjem LogMapFRI

Na podlagi prejšnjega poglavja, kjer je bilo podrobneje predstavljeno delovanje algoritma LogMap in na podlagi že zapisanega v poglavju predloga rešitve in izbora orodja za usklajevanje ontologij, lahko povzamemo naslednje pomanjkljivosti LogMap-a:

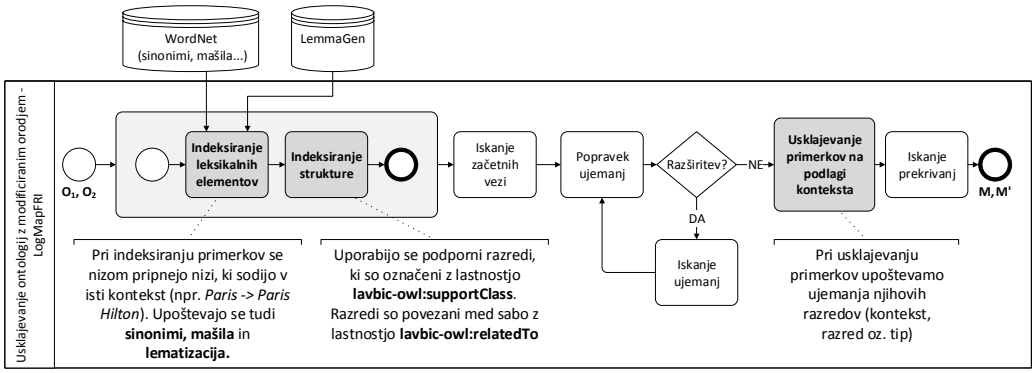
- neupoštevanje konteksta primerkov na podlagi njihovih razredov
- neupoštevanje povezanosti primerkov v isti kontekst
- uporaba namenskih slovarjev za ugotavljanja semantičnega pomena objektov (UMLS)
- uporaba krnenja nizov (ang. stemming), ki je primeren samo za jezike, ki niso morfološko bogati (angleščina).

Naštete omejitve smo v naši različici orodja LogMapFRI odpravili. Izboljšave so opisane v nekaj naslednjih poglavjih.



Slika 4.12: tretji korak pristopa bogatenja podatkov v relacijski podatkovni baze – usklajevanje ontologij.

Na sliki 4.12 je predstavljen krovni proces koraka usklajevanja ontologij. Pri tem se na začetku najprej izbere poljubno javno ontologijo (iz LOD oblaka), s katero se bo naša ontologija oz. shema relacijske podatkovne baze usklajevala. Pred začetkom procesa lahko uporabniki uredijo seznam sinonimov, mašil, neštevnikov, skratka podpornih nizov, ki pripomorejo k ugotavljanju semantičnega pomena usklajevalnih elementov.



Slika 4.13: Izboljšan proces orodja LogMap. Zatemnjena polja predstavljajo področja v katerih so izboljšave aplicirane.

Na sliki 4.13 je predstavljen izboljšan proces orodja LogMapFRI, ki za razliko od osnovnega orodja dopolnjen s korakom usklajevanja primerkov na podlagi konteksta, ki vključuje nekatere zunanje vire (LemmaGen [68], WordNet). Prav tako izrablja lastnosti, ki so definirane v kontekstu naših izboljšav za ugotavljanje konteksta posameznih primerkov.

Na sliki 4.14 je prikazana psevdokoda izboljšanega orodja LogMapFRI, v kateri funkcije, ki so označene ležeče in obarvane v rdečo barvo, predstavljajo predele, kjer so bile aplicirane izboljšave. Torej v prvem koraku algoritma, kjer se opravi **indeksiranje leksikalnih elementov** (slika 4.14, vrstica 2) oz. kjer se za vsako vhodno ontologijo zgradi struktura invertiranih leksikografskih indeksov, se po naši izboljšavi poleg osnovnih nizov upošteva tudi njihova variacija. In sicer: iz osnovnih nizov se odstranijo mašila; nad nizi se izvede lematizacija, kar nam omogoča pridobitev normalne morfološke oblike niza; osnovnim nizom se dodajo sinonimi, ki jih tako kot mašila pridobimo iz slovarja WordNet. V tej točki se ravno tako na nivoju primerkov pregleda njihove lastnosti (”rdf:type”) in na podlagi razredov, ki

```
Vhod: O1, O2 // vhodne ontologije
Izhod: M, M'
1  function LogMapFRI(O1, O2)
2    <LI1, LI2> := LexicalIndexes(O1, O2) // invertirani leksikalni indeksi
3    M' := CandidateMappings(LI1, LI2) // presek množic invertiranih indeksov
4    <O'1,O'2> := Module(O1, O2, M') // operative podmnožice ontologij
5    H1, H2 := StructuralIndex(O'1,O'2, M) // hierarhija razredov O1 in O2
6    M := ReliableMappings(M', H1, H2) // vezi s pomočjo mere SMOA in mere zaupnosti
7    M' := M' / M // zanesljiva ujemanja odstranimo iz množice M'
8    <P'1, P'2> := PropEncoding(H1, H2, M) // O'1 in O'2 kot Hornovimi stavki
9    M := Diagnosis(P'1, P'2, M) // Dowling-Gallier nad Hornovimi stavki
10   M' := M' \ Discarded(LI1, LI2, H1, H2, M')
11   M:= Diagnosis(P'1, P'2, M U M?, M) // preveri se množica M' zaradi alter. nizov
12   while M?:= DiscoverMappings(M, H1, H2) // iskanje novih ujemanj na nivoju razredov
13     M:= Diagnosis(P'1, P'2, M?, M) // Dowling-Gallier, nad Hornovimi stavki
14   M:= InstanceContextMatching(O'1,O'2, M) // ugotavljamo njihov konteksta primerkov
15   return M, M'
16 end function
```

Slika 4.14: Psevdokoda prilagojenega orodja LogMapFRI (prilagojeni deli algoritma so označeni ležeče in obarvani v rdečo barvo).

so bili ustvarjeni za namen povezovanja primerkov v isti kontekst. Tako primerke, ki opisujejo isti osebek iz relacijske podatkovne baze, povežemo znotraj strukture leksikografskih indeksov.

V drugem koraku (slika 4.14, vrstica 3) se naredi presek dobljenih struktur invertiranih indeksov, tako da dobimo vse možne kandidate, ki jih v nadaljnjih korakih preverjamo.

S pomočjo strojev za sklepanje, na podlagi dobljenih ujemanj M' (iz predhodnega koraka), izberemo samo elemente iz vhodnih ontologij, ki so potrebni za nadaljnje delo. To je velika prednost, če imamo opravka z velikimi ontologijami. Tako v nadaljevanju operiramo zgolj z podmnožicami vhodnih ontologij in sicer O'_1 in O'_2.

V koraku **indeksiranja strukture** (slika 4.14, vrstica 5) se s pomočjo orodij za sklepanje (npr. HermiT [10]), za vsako vhodno ontologijo oz. njeno okrnjeno različico (O'_1 in O'_2), zgradi hierarhijo razredov. V to hierarhijo se, s pomočjo lastnosti "lavbic-owl:relatedTo" oz. s poljubno lastnostjo ki jo v konfiguracijski datoteki definira uporabnik sam, zapiše izbrana pot iz sheme relacijske podatkovne baze, ki jo je uporabnik vključil v proces bogatenja. Razredi, ki so označeni z lastnostjo "lavbic-owl:SupportClass", se ravno tako zapišejo v hierarhijo razredov, ampak ti se ne upoštevajo kot rezultat uskladitve.

S pomočjo hierarhij razredov vhodnih ontologij se v naslednjem koraku, s pomočjo mere podobnosti nizov SMOA in mere zaupnosti pridobijo začetne vezi, ki predstavljajo že veljavna ujemanja. Tukaj so prav tako zajeta ujemanja med primerki, ki so bili določeni s pomočjo mere podobnosti nizov SMOA. Nato v naslednjem koraku iz množice M' odstranimo vsa ujemanja, ki smo jih označili kot zanesljiva.

V naslednjem koraku se, s klicem funkcije "PropEncoding", hierarhije razredov obeh vhodnih ontologij in dobljene vezi zapišejo v obliki Hornovih stavkov, ki v kombinaciji z Dowling Gallier algoritmom (slika 4.14, vrstica 8,9), služijo za detekcijo nepravilnih ujemanj znotraj množice vezi.

Nad uskladitvami znotraj množice M' se nato, na podlagi leksikalnih in strukturnih indeksov (sheme za označevanje intervalov), preveri ali obstajajo konflikti v strukturi invertiranih indeksov oz. v hierarhiji razredov. Pri odkrivanju konfliktov

[10] `http://hermit-reasoner.com/`

na podlagi leksikalnih indeksov se ujemanja ponovno pregledajo in sicer tokrat na podlagi alternativnih nizov. Kritična ujemanja se zavržejo. Na novo pridobljena ujemanja se dodatno preverijo še s pomočjo algoritma Dowling-Gallier (slika 4.14, vrstica 11).

V naslednjem koraku se v zanki najprej na podlagi mere zaupanja (ki temelji na principu lokalnosti) in mere podobnosti nizov nad množico pravilnih ujemanj M poiščejo nova ujemanja. Ta pa se s pomočjo algoritma Dowling-Gallier preverijo. Zanka se zaključi takrat ko, se na podlagi lokalnosti (sosedov pravilnih ujemanj), ne najde več novih ujemanj.

Ker smo po vsem tem postopku dobili ujemanja primerkov zgolj na podlagi mere podobnosti nizov (mera zaupnosti, ne deluje nad primerki in primerki niso vključeni v hierarhije razredov), smo v algoritem dodali nov korak (slika 4.14, vrstica 16). Tu, s pomočjo lastnosti primerkov ugotovimo njihov kontekst, tega pa primerjamo s primerki druge ontologije. V primeru neskladja konteksta takšna ujemanja iz množice M označimo kot neustrezna. V ta namen smo definirali novo mero, ki nam pove v kolikšni meri se dva primerka ujemata tako leksikografsko kot kontekstno.

V nadaljevanju bodo predstavljene naslednje izboljšave, ki so bile tudi omenjene v predstavitvi psevdokode iz slike 4.14:

- uporaba (večjezične) lematizacije - Lemmagen, ki nam omogoča pretvorbo nizov v neko normalno obliko in nam tako omogoča uspešnejše ugotavljanja ujemanj med elementi vhodnih ontologij.

- upoštevanje sinonimov in mašil pri usklajevanju elementov ontologij - v fazi leksikalnega indeksiranja je predlagana uporaba tezavra WordNet, ki hrani različne variacije nizov (sinonimi, protipomenke, itd.). Tako se v strukturi invertiranih leksikalnih indeksov (pod istim indeksom) poleg nizov hranijo tudi njihove variacije.

- implementacija lastnosti podpornih elementov, s katerimi označimo podporne elemente ali pa jih povežemo v relacijo.

- s pomočjo podpornih razredov lahko, na nivoju primerkov, ugotovimo kateri primerki opisujejo isti osebek oz. kateri primerki sodijo v isti kontekst.

- upoštevanje lastnosti primerkov za ugotavljanje konteksta. Vpeljana je nova mera, ki nam pove v kolikšni meri se dva primerka ujemata tako leksikografsko kot kontekstno.

Implementacija lastnosti podpornih elementov

V relacijski podatkovni bazi lahko najdemo številne metapodatke, ki jih lahko uporabimo za uspešnejše bogatenje le teh oz. njihovo uspešnejše usklajevanje. Kot že rečeno, naš pristop določa predstavitev sheme izbrane podatkovne baze v obliki usmerjenega grafa. Graf sestavljajo vozlišča, ki predstavljajo razrede. Povezave znotraj usmerjenega grafa pa predstavljajo relacije med vozlišči oz. tabelami. Ker smo omenjene podatke želeli upoštevati v naši ontologiji, smo v ta namen definirali dve lastnosti objekta, s pomočjo katerih lahko hranimo tovrstne informacije in na podlagi katerih lažje ugotovimo kontekst objektov, ki jih bogatimo oz. usklajujemo. Zato ker so zgolj podporni element, se podpornih razredov v procesu usklajevanja oz. bogatitve ne poda kot rezultat uskladitve. Če pogledamo sliko 4.13, ki predstavlja delovanje orodja LogMapFRI, omenjeni lastnosti v prvi vrsti uporabljamo v koraku indeksiranja strukture. Na takšen način povemo orodju za sklepanje (npr. HermiT), ki je v tem koraku zadolžen za izgradnjo hierarhije razredov, da so si razredi, ki jih obravnavamo, v relaciji. Razredi, ki se označijo zgolj kot podporni, se v zadnjem koraku procesa odstranijo iz množice zaznanih ujemanj.

Z lastnostjo "lavbic-owl:supportClass" označimo razrede kot podporne. Definicija lastnosti je prikazana na sliki 4.15. Iz elementa "rdfs:domain" lahko razberemo, da je lastnost definirana za vse razrede znotraj naše ontologije in prav tako kot vrednost objekta, zasede vse razrede znotraj naše ontologije.

V naši implementaciji, z uporabo lastnosti "lavbic-owl:relatedTo", definiramo relacije med tabelami izbrane poti v usmerjenem grafu, s katerim smo v prvem koraku pristopa predstavili shemo podatkovne baze. Iz definicije lastnosti, ki je prikazana na sliki 4.16, lahko na podlagi elementov, ki definirajo domeno in področje vrednosti, povemo, da lastnost definira relacijo med vsemi razredi, ki nastopajo v ontologiji izbrane podatkovne baze.

Slednjo lastnost, kot definicijo relacije med razredi, prepozna tudi orodje LogMapFRI. V okviru gradnje hierarhije razredov oz. v koraku indeksiranja strukture

```
<rdf:Description rdf:about="http://www.lavbic.net/onto#SupportClass">
  <!-- označimo da gre za lastnost objekta -->
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty" />
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property" />

  <rdfs:label xml:lang="en"> SupportClass </rdfs:label>

  <!- - domena in področje vrednosti -->
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>

  <!-- ostali metapodatki -->
  <rdfs:isDefinedBy rdf:resource="http://www.lavbic.net/onto/" />
</rdf:Description>
```

Slika 4.15: Definicija lastnosti "lavbic-owl:supportClass".

```
<rdf:Description rdf:about="http://www.lavbic.net/onto#relatedTo">
  <!-- označimo da gre za lastnost objekta -->
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty" />
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property" />
  <rdfs:label xml:lang="en">relatedTo</rdfs:label>

  <!- - domena in področje vrednosti -->
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>

  <!-- ostali metapodatki -->
  <rdfs:isDefinedBy rdf:resource="http://www.lavbic.net/onto/" />
</rdf:Description>
```

Slika 4.16: Definicija lastnosti "lavbic-owl:relatedTo".

se na podlagi predstavljenih lastnosti zabeležijo relacije med razredi, ki jih lastnost definira.

Relacije znotraj ontologije, predstavljene z omenjenimi lastnostmi, nam koristijo tudi pri ugotavljanju konteksta elementov (razredov in primerkov - v koraku "usklajevanje primerkov na podlagi konteksta"). Način ugotavljanja konteksta primerkov je prestavljen v naslednjih poglavjih.

Upoštevanje sinonimov in mašil pri usklajevanju elementov ontologij

Pred fazo leksikalnega indeksiranja, v fazi usklajevanja ontologij, se nizi pregledajo in iz seznama mašil, ki jih lahko uporabnik poljubno ureja, se odstranijo vsi njihovi deli (nizi, podnizi itd.). Uporabnik to naredi, če smatra, da bi lahko bila ta mašila moteč člen pri usklajevanju ontologij. Tako kot odstranjevanje neželenih nizov (mašil), se lahko kot alternativni nizi v fazi leksikografskega indeksiranja dodajo nizi, ki predstavljajo nedoločnik, neštevnik, sinonim ipd.

Pri identifikaciji omejitev delovanja LogMap smo ugotovili, da za ugotavljanje semantičnega pomena nizov, ki jih algoritem usklajuje, uporablja namensko zbirko slovarjev iz področja biomedicine UMLS [61] (Unified Medical Language System). Ker pri bogatenju podatkov v našem pristopu nismo omejeni na specifično področje oz. domeno, smo v naš pristop vključili uporabo splošnejšega slovarja WordNet [42] (predstavljen je v poglavju 3.1.1).

Poleg angleške različice Wordnet slovarja obstaja tudi slovenska različica, ki se imenuje sloWTool[11]. Na vir podatkovne baze se nismo uspeli povezati, kljub temu pa uporabniku znotraj prototipa, ki je nastal v okvirju evalvacije zastavljenega pristopa, omogočamo dodajanje in urejanje trditev, ki jih v osnovi pridobimo iz podatkovne baze Wordnet. To uporabniku omogoča, da vključi trditve v poljubnem jeziku.

Pri dodajanju je potrebna previdnost, saj pri upoštevanju velikega števila sinonimov obstaja nevarnost dvoumnosti, kar nam povzroči povišan negativni rezultat oz. nepravilne uskladitve. Za ta namen lahko uporabnik definira vhodni parameter, s katerim nastavi maksimalno število alternativnih nizov.

Morfološka normalizacija nizov - lematizacija

V fazi indeksiranja leksikalnih elementov, se poleg osnovnih nizov elementov zajamejo tudi njihovi alternativni nizi (variacija nizov). Poleg upoštevanja sinonimov in odstranjevanja mašil, ki smo jih predstavili v prejšnjem poglavju, se v izboljšanem procesu (predstavljen na sliki 4.11) vsi nizi, s pomočjo lematizacije, pretvorijo v normalno obliko. V ta namen uporabljamo orodje LemmaGen.

[11] `http://nl.ijs.si/slowtool/`

LemmaGen [68] je projekt, v okviru katerega je nastala standardizirana odprtokodna večjezična platforma za lematizacijo. Razvita je bila v sodelovanju z institutom Jožef Štefan v Ljubljani. Povod za nastanek platforme je pomanjkanja visoko kakovostnih slovenskih lematizatorjev. Platforma, poleg lematizatorja za slovenski jezik, zajema lematizatorje za 11 drugih evropskih jezikov. Platforma se je prav tako zmožna naučiti novih lematizacijskih pravil za nove jezike [69].

Lematizacijo lahko izvajamo za vsako besedo posebej, saj deluje neodvisno od stavka. Za uporabo lematizatorjev so na voljo različni programski vmesniki (ang. API). LemmaGen je podprt v različnih platformah in obstaja v različnih implementacijah (C++, C++.Net, Python, C#.Net). Na voljo je množica spletnih storitev, ki jih lahko uporabljamo iz katerekoli naprave, ki je povezana na medmrežje. Platforma se lahko pohvali tudi z visoko učinkovitim izvajanjem, saj njena hitrost procesiranja doseže 1 milijon besed v sekundi.

V okviru našega procesa smo proces krnenja nizov (ang. stemming) zamenjali z lematizacijo. Sistem LemmaGen uporabljamo s pomočjo knjižnice v programskem jeziku C#.net. S pomočjo LemmaGen zgradimo slovar lematizacij, ki ga kasneje uporabimo pri usklajevanju razredov in primerkov ontologij. Torej, v prvem koraku procesa delovanja orodja LogMapFRI - v koraku leksikalnega indeksiranja, se za vsak niz pod enakim indeksom, doda njegova normalna oblika. S pomočjo značke "@en", ki se nahaja poleg vsakega niza znotraj ontologije, ugotovimo (predstavljeno v poglavju 3.1.1) v katerem jeziku je zapisan trenuten niz. To značko podamo kot parameter orodju LemmaGen, ta pa na podlagi značke vrne lemo v izbranem jeziku.

leksikografski indeks		indeks elementa	
element	id	id	element
saving,private,ryan	1554	1554	lavbic-owl:Saving_Private_Ryan
save,private,ryan	1554	...	...

Tabela 4.4: Primer leksikografskega indeksa z lematizacijo (primer Saving in Save).

Kot smo že omenili v poglavju 3, so lematizatorji uspešnejši pri odkrivanju normalne oblike besed, sploh pa pri morfološko bogatih jezikih kot je slovenščina.

Njihova slabost je, da so zelo redki.

Osnovni niz	**Normalizacija nizov**	
	Krnenje	**Lematizacija**
woodridge	*woodridg*	*woodridge*
australia	australia	australia
country	*countri*	*country*
city	*citi*	*city*
address	address	address
paris hilton	*pari hilton*	*paris hilton*
boljši	*boljši*	*dober*

Tabela 4.5: Primer napačne normalizacije s tehniko krnenja v primerjavi z lematizacijo.

Izvorna vsebina	**Lematizirana vsebina**
To je majhen korak za človeka, a velik za človeštvo.	Ta biti majhen korak za človek, a velik za človeštvo.
Moja mama je vedno govorila: Življenje je kot škatla čokoladnih bonbonov – nikoli ne veš, kaj boš dobil.	Moj mama biti vedno govorilo: Življenje biti kot škatla čokoladen bonbon – nikoli ne vedeti, kaj biti dobiti.
Drevo, mogočno, da ga ne objameš z rokami, vzklije iz drobnega semena.	Drevo, mogočen, dati on ne objeti z roka, vzklija iz droben seme.
Some houses have many mice. Our house only has one mouse.	some house has many mouse. our house only has one mouse.
Our greatest weakness lies in giving up.	Our great weakness lie in give up.

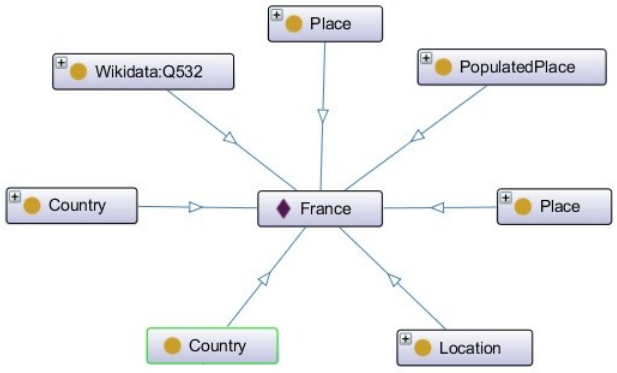
Tabela 4.6: Tabela primer lematizacije primerov v angleškem in slovenskem jeziku

Usklajevanje primerkov na podlagi konteksta

Ontologija pridobljena v drugem koraku pristopa, je definirana z razredi, ki predstavljajo konkretne tabele oz. vozlišča v usmerjenem grafu. Razredi znotraj ontologije so opisani z njihovimi primerki. Da element znotraj ontologije označimo kot primerek razreda, mu je potrebno definirati tip razreda. To storimo s pomočjo lastnosti "rdf:type". Iz primera 4.17 prikažemo, da lahko primerek vsebuje tudi več definicij o njenem tipu.

```
1 dbpedia:France rdf:type dbpedia-owl:Country> .
2 dbpedia:France rdf:type <http://schema.org/Country> .
3 dbpedia:France rdf:type dbpedia-owl:PopulatedPlace> .
4 dbpedia:France rdf:type dbpedia-owl:Wikidata:Q532> .
5 dbpedia:France rdf:type dbpedia-owl:Place> .
6 dbpedia:France rdf:type <http://schema.org/Place> .
7 dbpedia:France rdf:type <http://www.ontologydesignpatterns.org/ont/d0.owl#Location> .
```

Slika 4.17: Primer definicije primerka "dbpedia:France".



Slika 4.18: Primer primerka "France" iz ontologije DBPedia, z vsemi njenimi tipi razredov.

Na podlagi ugotovljenega je sledil naslednji korak. Na sliki 4.13, je predstavljen izboljšan proces orodja LogMapFRI.

Po tem, ko se v procesu usklajevanja ontologij poiščejo preslikave med razredi vhodnih ontologij, se v naslednjem koraku uskladijo primerki na podlagi konteksta. Ta je definiran kot izboljšava obstoječega procesa (prikazano na sliki 4.13), pri katerem se preveri pravilnost vezi (množica začetnih ujemanj) primerkov. V ta namen smo vpeljali novo mero, ki uporablja mero podobnosti trenutne vezi (definirana je Stoilos-jevo mero podobnosti nizov - SMOA). Ta nam pove v kolikšni meri sta si primerka podobna, pri čemer se upošteva tudi mera podobnosti med vsemi razredi. Torej pregleda se koeficient podobnosti razredov obeh primerkov, ki definirajo vez. Ideja mere je sledeča: če so si nizi primerkov, ki definirata vez med seboj podobni (imata visok koeficient podobnosti), razredi teh primerkov pa se med seboj ne ujemajo, se tovrstna ujemanja označijo kot neveljavna. Mera o podobnosti primerkov na podlagi njihovih razredov, ICF (ang. instance confidence factor) je definirana s formulo 4.6. V tej formuli n prestavlja množico razredov primerka i_1, m pa predstavlja množico razredov primerka i_2:

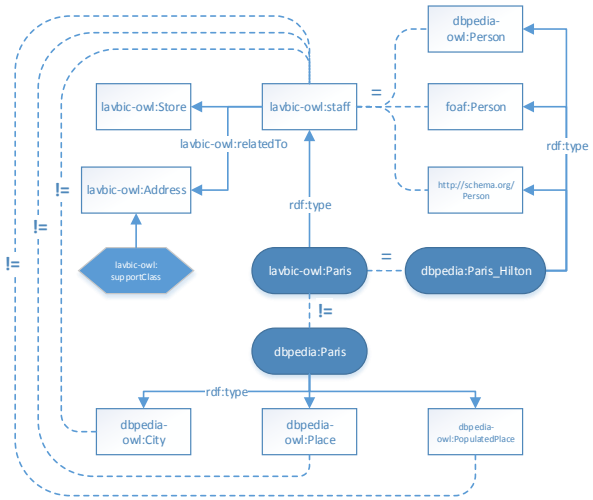
$$ICF\,(n,m) = SMOAAnchors(i_1,i_2) + \frac{\sum_i^n \sum_j^m confidence\,(i,j)}{sizeof\,(n) \times sizeof\,(m)} \tag{4.6}$$

Zaloga vrednosti mere je od 0, kar pomeni, da se razredi obeh primerkov popolnoma razlikujejo, do 2, kar pomeni, se vsi razredi primerkov, ki jih usklajujemo, popolnoma ujemajo. Prvi del mere je definiran z mero podobnosti nizov primerkov, ki jo znotraj procesa delovanja orodja LogMapFRI dobimo v koraku "Iskanje začetnih vezi"(slika 4.13). Drugi del mere definira mera zaupnosti oz. določi se koeficient ujemanja med razredi primerkov, ki so se definirali v prejšnjih korakih. V formuli 4.6 je označena kot $confidence(i,j)$, sama pa je definirana s formulo 4.7.

$$\begin{aligned} confidence\,(i,j) &= (weightSMOA \times SMOAAnchors\,(i,j)) \\ &+ (weightScopeAnchors \times ScopeAnchors\,(i,j)) \end{aligned} \tag{4.7}$$

Mera zaupnosti, ki je definirana v formuli 4.7, je sestavljena iz dveh delov. V prvem delu pregledamo podobnost med nizi razredov (v primeru da obstaja vez se vrne njena mera podobnosti), ki jih primerjamo. V drugem delu pa se pridobi faktor

ujemanja med razredi, ki je bil izračunan na podlagi lokalnosti dveh razredov. Torej, iz drugega dela formule dobimo mero, ki nam pove ali dva razreda sodita v isti kontekst. Uporabnik lahko oba dela formule obteži. Privzete vrednosti uteži so $weightSMOA = 0.5$ in $weightScopeAnchors = 0.5$. Če naj razreda sodita v isti kontekst morata preseči prag vrednosti $good\_confidence = 0.5$.



Slika 4.19: Primer uporabnosti implementirane mere (ang. instance structure similarity factor), kjer se pri usklajevanju primerkov "lavbic-owl:Paris" in "dbpedia:Paris_Hilton" upoštevajo relacije z vsemi njihovimi razredi.

Na sliki 4.19 je na praktičnem primeru prikazano delovanje zgoraj definirane mere. Torej usklajuje se primerek "lavbic-owl:Paris" z primerki iz DBpedije "dbpedia:Paris_Hilton" in "dbpedia:Paris", in sicer tako, da se pregleda definicije njihovih razredov, nato med seboj primerja in poskuša ugotoviti ali primerki spadajo v isti kontekst. Iz primera lahko vidimo da smo uspešno zavrnili ujemanje med "lavbic-owl:Paris" in "dbpedia:Paris", saj slednji definira mesto Paris, prvi pa osebo z imenom Paris.

Kontekst množice primerkov iz izbrane poti grafa relacijske podatkovne baze

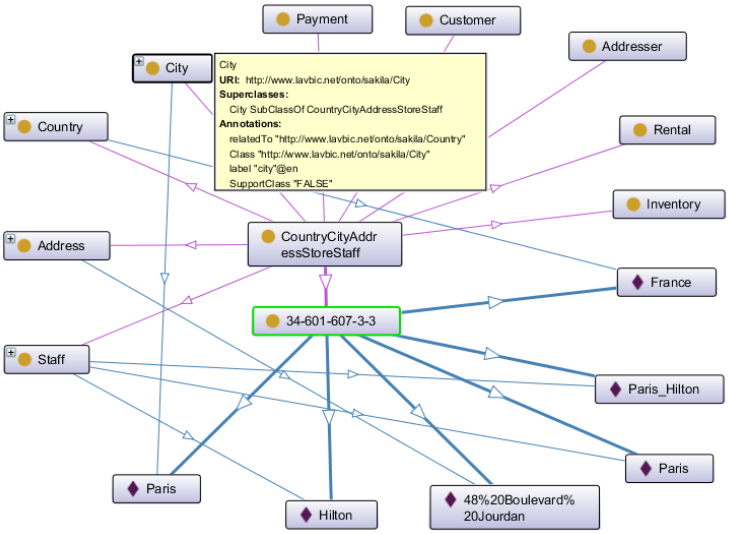
Kot je bilo omenjeno v poglavju 4.2.1, algoritem LogMap usklajuje primerke razredov zgolj na podlagi tehnik na nivoju nizov (mera podobnosti SMOA) in pri tem ne upošteva konteksta v katerega primerek sodi. Orodje namreč ni namenjeno reševanju tovrstnih problemov.

V našem pristopu lahko kontekst primerkov pridobimo na podlagi izbrane poti znotraj usmerjenega grafa, s katerim smo predstavili shemo podatkovne baze, ki jo bogatimo oz. usklajujemo. Tako izbrano pot kot vse podporne razrede beležimo z lastnostjo "lavbic-owl:relatedTo".

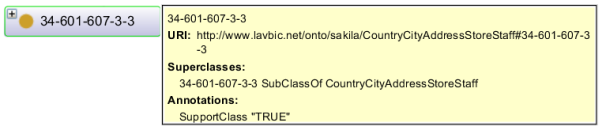
Primerke, ki spadajo v isti kontekst, prepoznamo s pomočjo podpornega razreda, ki je označen z enoličnim identifikacijskim številom. Ta je bil ustvarjen v fazi izdelave preslikovalnega dokumenta R2RML (postopek je predstavljen v poglavju 4.1.4, standard pa v poglavju 2.3.5). Za preslikavo primerkov iz relacijske podatkovne baze v RDF obliko oz. trojice, ki sodijo v razrede iz izbrane poti, se zgradi SQL poizvedba. Znotraj R2RML dokumenta se definira logično tabelo oz. nabor podatkov za preslikovanje v RDF obliko. Rezultat poizvedbe je, da za vsako postavko (vrstico) dobimo unikatno identifikacijsko številko, ki nam pove katera skupina primerkov sodi v skupni kontekst (označijo se primerki, ki opisujejo isti osebek). Torej pri preslikavi se zgradi razred, ki je označen z omenjenim identifikacijskim številom. Ker je omenjen razred označen kot podporni razred, je pri usklajevanju udeležen samo kot interni podatek za zapis konteksta in uporabniku pri predstavitvi rezultata usklajevanja oz. obogatitve ni viden.

Na sliki 4.20 so prikazani primerki s pripadajočimi razredi, ki spadajo v isti kontekst. Iz primera vidimo, da so primerki med seboj povezani z razredom, ki je označen z enoličnim identifikatorjem.

Kontekst se kasneje upošteva znotraj koraka leksikalnega indeksiranja elementov ontologij. Namreč, k vsakem objektu se kot alternativni niz dodajo novi vnosi, ki predstavljajo kombinacijo nizov in sodijo v isti kontekst. Primer: ker primerka "Paris" in "Hilton" sodita v isti kontekst (na podlagi podpornega razreda "lavbic-owl:CoutnryCityAddressStoreStaff#34-601-607-3-3"), se jima doda alternativni niz "ParisHilton", kar pripomore pri identifikaciji povezave "lavbic-owl:Paris ≡ dbpedia:Paris_Hilton", ki je tudi prikazana na sliki 4.20.



Slika 4.20: Graf v katerem razred 34-601-607-3-3 povezuje primerke, ki spadajo v isti primerek iz relacijske podatkovne baze.



Slika 4.21: Definicija podpornega razreda za ugotavljanje konteksta primerkov.

4.3 Verifikacija metode

Verifikacijo metode smo izvedli v dveh sklopih. V prvem sklopu skušamo obogatiti testno relacijsko bazo Sakila s podatki iz DBpedije na tri načine: brez, z delnim upoštevanjem in s polnim upoštevanjem vseh izboljšav. V drugem sklopu pa orodje testiramo s pomočjo testov, ki jih je leta 2014 pripravila organizacija OAEI [70]. Testirali smo v dveh kategorijah in sicer na področju ujemanja primerkov in na področju usklajevanja velikih ontologij.

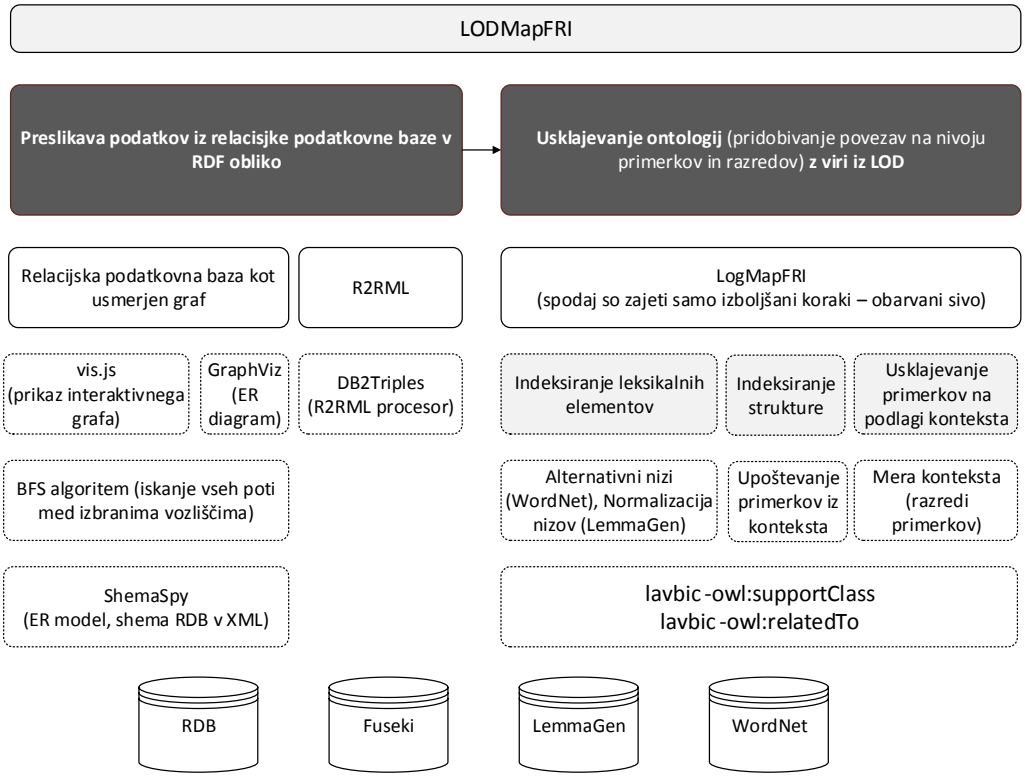
Na sliki 4.22 je prikazan sklad tehnologij in pristopov, ki je nastal kot izboljšava obstoječih rešitev in kot pripomoček za boljši zajem konteksta iz relacijske podatkovne baze, in ki bi pripomogel k lažji uskladitvi tovrstnih podatkov s prosto dostopnimi viri iz LOD oblaka. Vsi elementi predstavljenega sklada so zajeti v prototipu, s katerim smo zastavljen pristop v nadaljevanju evalvirali.

4.3.1 Mere uspešnosti

Da bi lahko ocenili uspešnost zastavljene metode oz. dela metode, pri katerem se soočimo z izzivom usklajevanja ontologij, bomo usklajevanja primerjali z referenčnimi rezultati usklajevanj, ki so nastali na podlagi specifičnih kriterijev. Izvedli bomo nekaj ocenitev pobude OAEI 2014 [70].

V tem razdelku bomo zajeli različne mere za evalvacijo orodij, in sicer tako kvalitativne kot kvantitativne mere. Mere lahko delimo na skladnostne mere (ang. compliance measures) in performančne mere [37]. S pomočjo skladnostnih mer evalviramo stopnjo skladnosti pridobljenih uskladitev tako, da jih primerjamo pričakovanimi rezultati. S performančnimi ukrepi uspešnosti pa izrazimo lastnosti algoritmov kot sta hitrost in poraba pomnilnika. Uporabniške mere se osredotočajo na evalvaciji uporabnikove interakcije s sistemom za usklajevanje. Predstavili bomo nekaj klasičnih evalvacijskih mer, ki smo jih pri ocenjevanju naše metode tudi uporabili.

Poleg skladnostnih mer, ki jih bomo predstavili v nadaljevanju (natančnost, priklic, f-mera, število pravilnih ujemanj), spremljamo tudi čas, v katerem se usklajevanje ontologij izvede. Uporabniško mero pa bomo upoštevali v enem izmed naslednjih primerov evalvacije, pri katerem pa ne bomo upoštevali (uporabniško urejenih) slovarjev (v poglavju 4.3.3 je označen kot primer 2).



Slika 4.22: Na sliki je prikazan sklad vseh izboljšav, ki ga bomo v tej fazi naloge evalvirali (Elementi pod LogMapFRI, predstavljajo zgolj korake pri katerih so bile aplicirane izboljšave, vsi koraki so predstavljeni na sliki 4.13).

Uporabnik lahko namreč sodeluje pri procesu usklajevanja ontologij tako, da se mu kot rezultat poleg pravilnih ujemanj podajo tudi ujemanja, ki jih je orodje označilo za neustrezna. Iz tega nabora ujemanj lahko uporabnik označi tiste, za katere smatra, da so sicer pravilni in jih ročno umesti med pravilne.

Drugi način interakcije uporabnika je, da ob začetku procesa v konfiguraciji orodja označi, da želi interaktivni proces in želeno ujemanje umesti v vnaprej opredeljeno datoteko z pravilnimi ujemanji oz. uskladitvami (ang. Gold Standard (GS), predstavljena v poglavju 4.3.1). V tem primeru ima uporabnik neposreden vpliv na mero priklica, saj se po procesu usklajevanja postavi na vrednost 1, relevantna postane le mera natančnosti, ki upošteva število nepravilno označenih ujemanj. Obe meri sta predstavljeni v naslednjem poglavju.

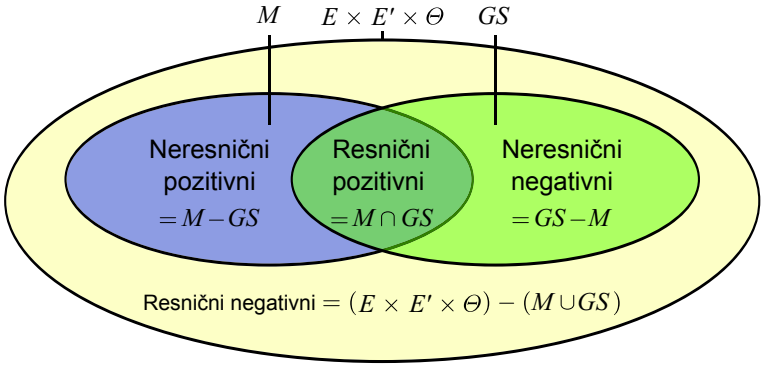
Natančnost, priklic in f-mera

Natančnost, priklic in f-mera so običajne mere, ki jih uporabljamo na področju pridobivanja informacij o virih iz velikih zbirk tekstovnih dokumentov. Na sliki 4.23 je, kot množici ujemanj in njihovih relacij, prikazana uskladitev med dvema ontologijama. E in E' predstavljata množico elementov prve druge ontologije in θ predstavlja množico vseh ujemanj. Mere temeljijo na primerjavi pridobljenega nabora preslikav z vnaprej pripravljenimi preslikavami - t.i. zlati standard (v nadaljevanju GS). To je ročno ustvarjena zbirka ujemanj za katero se smatra, da je najbolj pravilna rešitev določenega problema usklajevanja. Uskladitev M, ki jo dobimo kot rezultat ocenjevalnega sistema, se primerja s pravilnimi ujemanji definiranimi kot zlati standard, ki pa so, kot smo že povedali, definirana vnaprej [71].

Na področju pridobivanja informacij (ang. information retrieval (IR)) je natančnost (ang. precision) definirana kot mera, ki pove razmerje med vsemi pravilno usklajenimi primeri in številom vseh primerov, za katere smo napovedali, da pripadajo pozitivnemu razredu (na sliki 4.23 resnični pozitivni in neresnični pozitivni oz. množica M). S pomočjo omenjene mere lahko preverimo pravilnost metode.

Priklic (ang. recall) ali delež resničnih pozitivnih primerov, meri razmerje med številom vseh pozitivnih primerov, ki jih pravilno uskladimo in med številom vseh pozitivnih primerov. Mera nam poda oceno popolnosti metode.

Definicija natančnosti ujemanja. Na podlagi reference ujemanj GS, je na-



Slika 4.23: Uskladitev med dvema ontologijama prikazana kot možici ujemanj in njihovih relacij.

tančnost ujemanja M podana kot:

$$P(M, GS) = \frac{|M \bigcap GS|}{|M|} \tag{4.8}$$

Definicija priklica. Priklic je podan kot:

$$R(M, GS) = \frac{|M \bigcap GS|}{|GS|} \tag{4.9}$$

Ostale mere, kot so fallout, šum, f-mera ipd., lahko izpeljemo iz prej omenjenih mer [72]. Mera s katero bomo preverili skupno uspešnost našega orodja je f-mera. Definirana je kot harmonična sredina med natančnostjo in priklicem. Mera nam omogoča lažjo predstavitev skupne uspešnosti pri usklajevanju ontologij.

Definicija f-mera (ang. F-measure). Če je natančnost ujemanja podana s P in priklic z R je skupna f-mera (F) definirana kot:

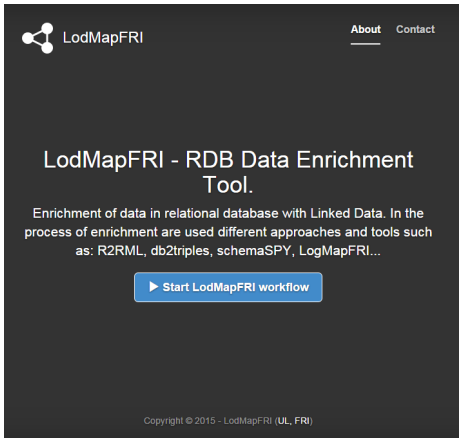
$$F = \frac{2 \times |M \bigcap GS|}{|M| + |GS|} = \frac{2 \times P \times R}{P + R} \tag{4.10}$$

Vse tri naštete mere bomo uporabili pri evalvaciji zastavljene metode, tako, da

bomo z njimi primerjali naš pristop z obstoječimi orodji na področju usklajevanja ontologij.

4.3.2 Verifikacija metode z orodjem LodMapFRI

V okviru verifikacije zastavljenega pristopa je nastalo orodje za bogatenje podatkov v relacijski podatkovni bazi. Orodje je razvito v obliki spletne aplikacije, ki smo jo poimenovali LodMapFRI (slika 4.24). Njena izvorna koda je na voljo v javnem repozitoriju Bitbucket [73]. Spletna aplikacija LodMapFRI, je razvita v programskem jeziku PHP 5 in sicer v odprtokodnem PHP ogrodju Codeigniter[12], ter s pomočjo različnih spletnih tehnologij kot so: Javascript (jQuery), AJAX, CSS itd. Spletna aplikacija se izvaja na spletnem strežniku Apache. Aplikacija vodi uporabnika skozi celoten proces bogatenja podatkov iz relacijske podatkovne baze z linked data viri. Proces skozi katerega nas pelje je sestavljen iz šestih korakov, ki bodo s pomočjo konkretnega primera v nadaljevanju tudi predstavljeni. Znotraj posameznega koraka aplikacije, se s pomočjo različnih vmesnikov kličejo različna orodja (SchemaSpy, DB2Triples in LogMapFRI), ki so zadolžena za posamezen korak. V nadaljevanju je skozi konkreten primer predstavljen celoten proces bogatitve.



Slika 4.24: Vstopna točka v aplikacijo.

V prvem koraku uporabnik izbere vrsto povezave oz. tip podatkovne baze, ki

[12] `https://ellislab.com/codeigniter`

jo želi umestiti v proces bogatitve podatkov. Nato uporabnik vnese vse potrebne avtentikacijske podatke za povezavo na željeno podatkovno bazo. V spodnjem delu obrazca si poleg željene podatkovne baze izbere tudi način prikaza njene sheme, in sicer lahko izbira med:

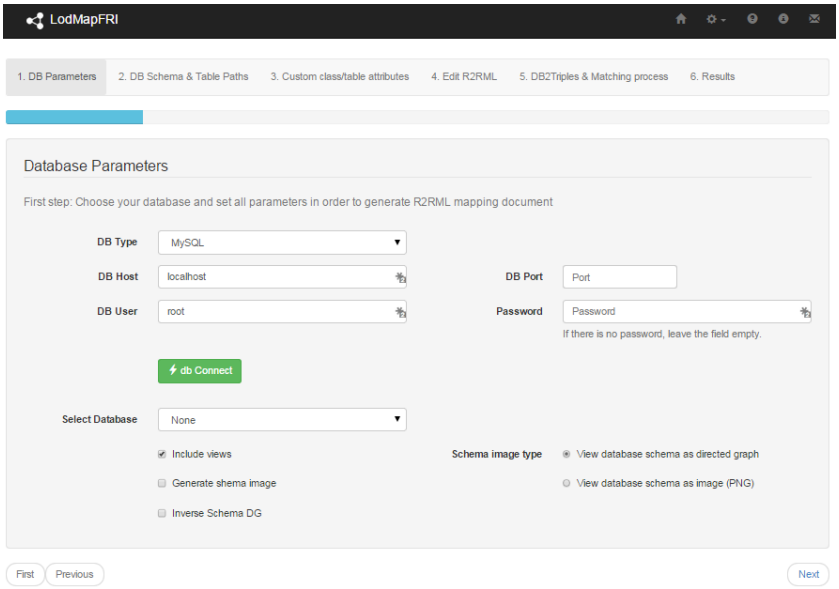
- prikazom sheme statično oz. kot ER diagram, ki ga izrišemo s pomočjo knjižnice Graphviz [54] ali
- prikazom sheme dinamično, ki jo prikažemo s pomočjo Javascript knjižnice vis.js [74].

Pri preiskovanju sheme podatkovne baze se uporablja orodje SchamaSpy [53] (predstavljeno v poglavju 4.1.1), ki za izbrano podatkovno bazo generira XML datoteko, ki vsebuje podatke o njenih elementih, vključno z vsemi njihovimi medsebojnimi relacijami.

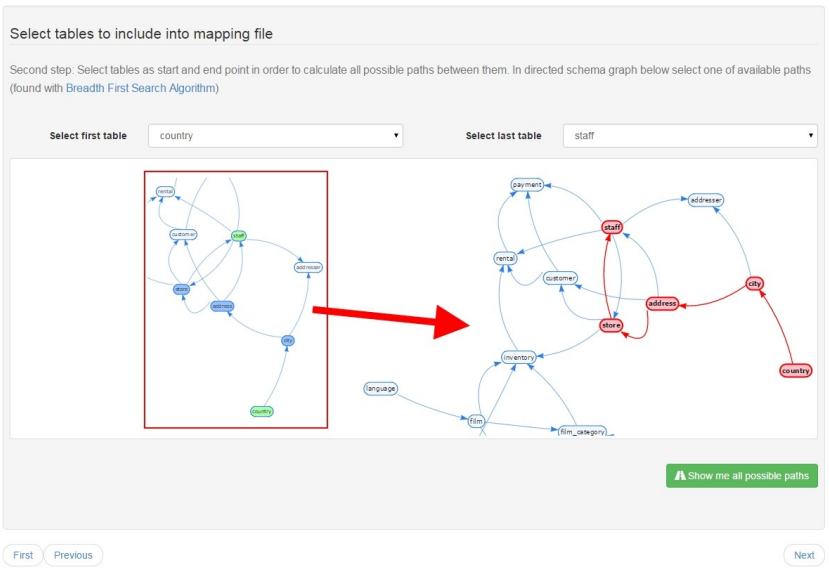
V naslednjem koraku se shema podatkovne baze prikaže kot usmerjen graf, ki je zgrajen na podlagi pravil iz poglavja 4.1.2. Uporabnik ima na začetnem obrazcu prav tako možnost izbire invertiranega grafa s pomočjo katerega lahko pri izbiri poti zajame vse želene tabele.

Na podlagi izbire v prejšnjem koraku, se shema izbrane podatkovne baze prikaže v obliki EER diagrama ali kot usmerjen graf. Uporabnik si izbere želeno pot – označi tabele in podatke, ki jih želi vključiti v proces bogatitve, izbere pa tudi začetno in končno tabelo oz. vozlišče. Možne poti med točkama pa se nato izračunajo s pomočjo algoritma iskanje v širino. Rezultat postopka je usmerjen graf sheme izbrane podatkovne baze, ki se izriše s pomočjo jQuery knjižnice vis.js [74].

Na sliki 4.26 je prikazan usmerjen graf, na katerem so prikazana modro in zeleno obarvana vozlišča. Modro obarvana vozlišča predstavljajo nabor vseh možnih vozlišč, ki jih lahko z izbiro poljubne poti zajamemo. Zelena vozlišča pa predstavljajo izbrani začetni in končni vozlišči. Belo obarvana vozlišča z modro obrobo so preostala vozlišča oz. tabele, ki jih pri določeni izbiri začetnega in končnega vozlišča ne moremo zajeti. S postavitvijo miškinega kazalca nad modro obarvana vozlišča se nam v rdeče obarva možna pot, ki jo lahko izberemo z enojnim klikom. S klikom na gumb "Show me all possible paths" se nam izpišejo oz. izrišejo vse

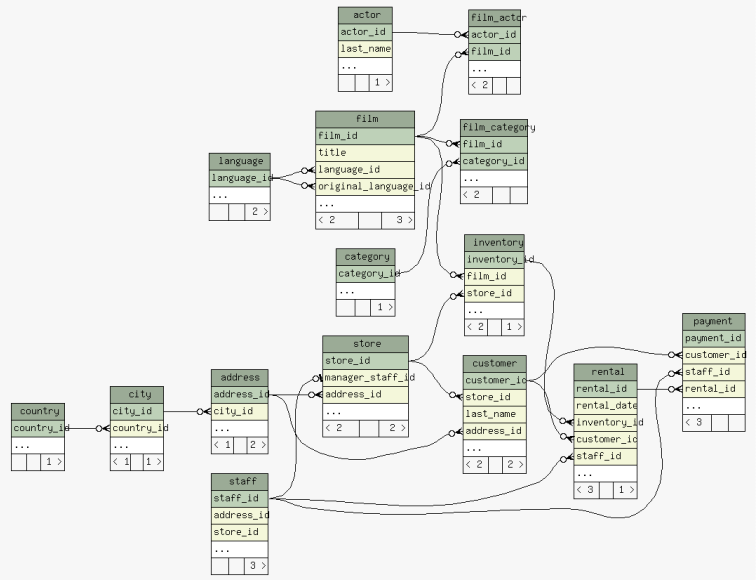


Slika 4.25: Prvi korak bogatitve podatkov v relacijski podatkovni bazi znotraj orodja LodMapFRI, in sicer izbira in povezava na relacijsko podatkovno bazo.



Slika 4.26: Drugi korak procesa orodja LodMapFRI, in sicer prikaz sheme izbrane relacijske podatkovne baze kot usmerjen graf in izbira poti z množico tabel za vključitev v proces bogatenja.

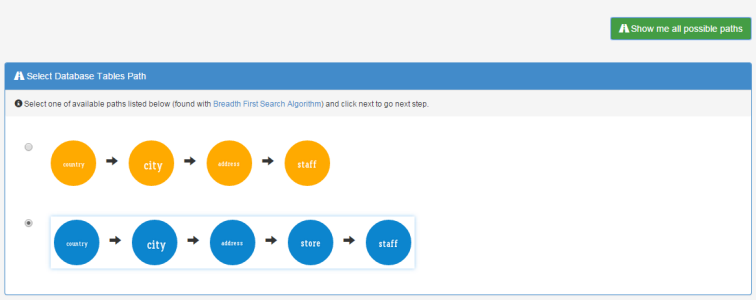
možne poti med izbranima vozliščema (glej sliko 4.28). Z drsnikom miške lahko graf povečamo ali pomanjšamo.



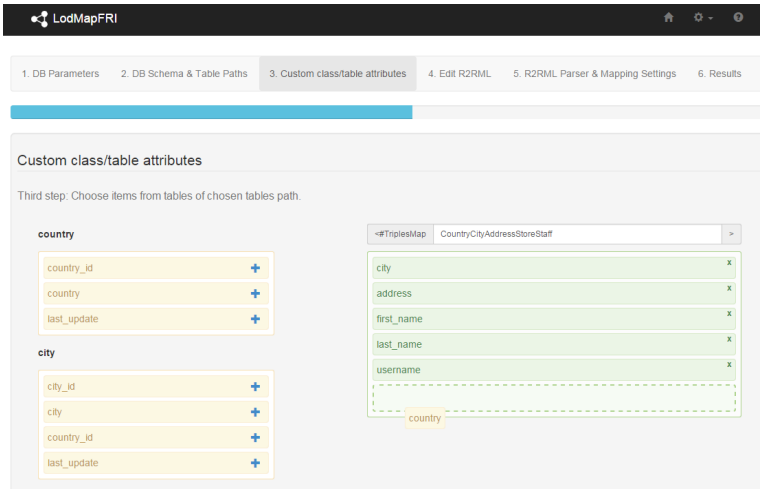
Slika 4.27: Prikaz sheme v obliki E-R diagrama izbrane podatkovne baze Sakila.

V tretjem koraku procesa bogatenja z orodjem LodMapFRI (glej sliko 4.29), se na levi strani okna izpišejo vse tabele z vsemi njihovimi pripadajočimi polji. S pomočjo metode "primi in spusti" ali s klikom na gumb "+", ki se nahaja znotraj polja posamezne tabele, uporabnik izbere polja, ki bodo del nove tabele oz. razreda. Ta se v naslednjem koraku, skupaj z vsemi podpornimi elementi, ki jih najdemo v relacijski podatkovni bazi, preslika v RDF obliko. Tabela oz. razred se lahko z vnosom vrednosti v polje nad tabelo na desni strani poljubno preimenuje. S klikom gumb "x" se izbrani element podatkovne baze odstrani iz izbire.

V četrtem koraku se na podlagi vseh izbranih elementov znotraj tabel, ki so del izbrane poti usmerjenega grafa, ustvari preslikovalni (mapirni) dokument R2RML (slika 4.30), ki je uporabniku nato na voljo za urejanje. Tako lahko uporabnik po lastni presoji vključi poljuben element v proces bogatenja podatkov. V dobljenem preslikovalnem dokumentu so vključene vse tabele in izbrani elementi iz prejšnjih

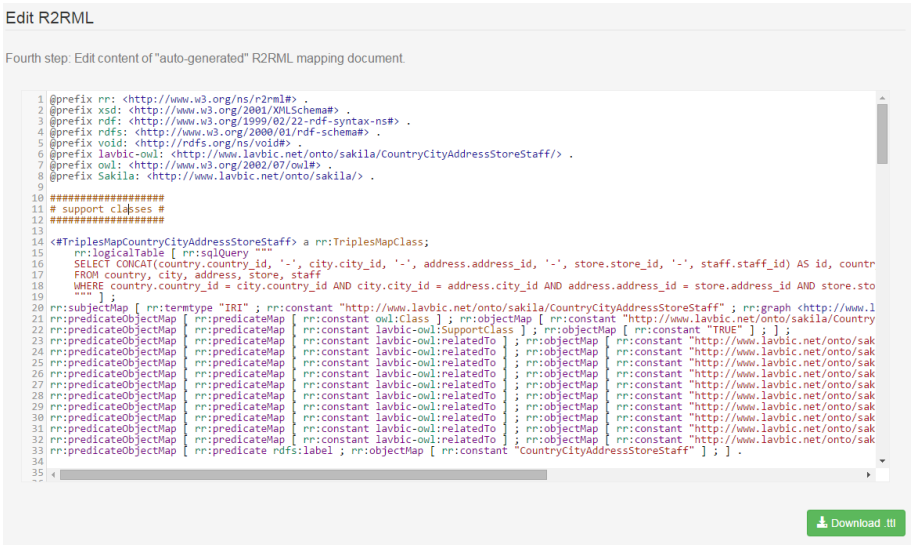


Slika 4.28: Prikaz vseh možnih poti med vozlišči "Country" in "Staff".



Slika 4.29: Kreiranje razreda za preslikavo podatkov iz relacijske podatkovne baze v RDF format in njihovo bogatenje. Razred se gradi s pomočjo elementov iz tabel, ki so vsebovana na izbrani poti iz prejšnjega koraka.

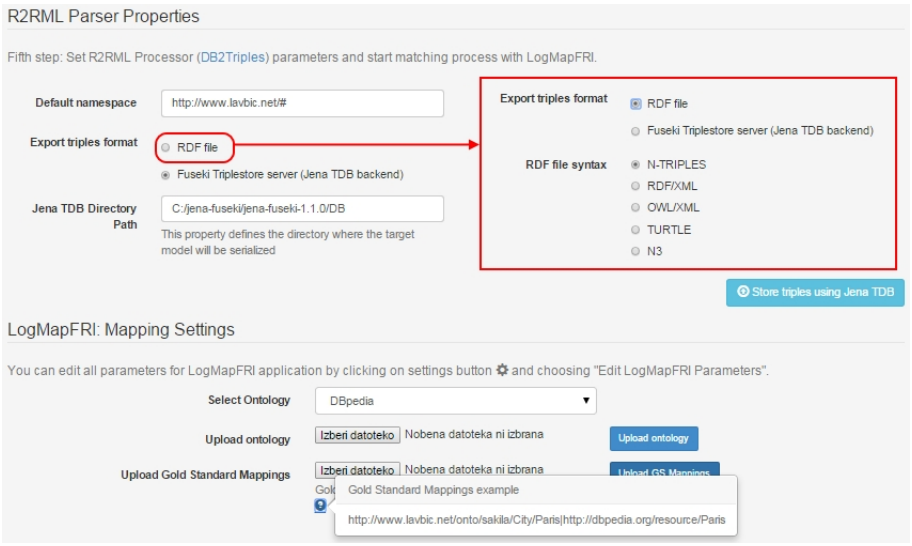
korakov procesa. V preslikovalnem dokumentu je tako definirano, da se tabele preslikajo v razrede. Posamezni primerki, ki so odvisni od izbranih elementov tabel, pa se preslikajo v primerke posameznih razredov oz. tabel. Relacije med razredi označimo oz. povežemo z lastnostjo objekta "lavbic-owl:relatedTo". Za lažje ugotavljanje konteksta v fazi usklajevanja ontologij, se poleg vseh tabel iz izbrane poti vključijo tudi sosedi začetnega in končnega vozlišča. Vozlišča, ki služijo zgolj za ugotavljanje konteksta objektov, ki jih bogatimo, imenujemo podporni razredi. Omenjene razrede označimo z lastnostjo objekta "lavbic-owl:supportClass".



Slika 4.30: Zgrajen R2RML (problematika predstavljena v poglavju 4.1.4) preslikovalni dokument z možnostjo urejanja.

Za preslikovanje podatkov iz relacijske podatkovne baze v RDF format s pomočjo preslikovalnega dokumenta R2RML uporabljamo orodje - R2RML procesor DB2Triples (podrobneje predstavljeno v poglavju 2.5.1). Na zgornjem delu slike 4.31 je prikazan konfiguracijski obrazec orodja DB2Triples, v katerega vnesemo privzeti imenski prostor (ang. namespace) in obliko izvoza. Orodje DB2Triples omogoča izvoz trojic neposredno v datoteko, v kateri so trojice lahko zapisane v različnih formatih (N-Triples, RDF/XML, OWL/XML, Turtle ali N3). Poleg izvoza v lokalno datoteko, orodje omogoča preslikavo trojic neposredno v shrambo Jena TDB nad katero je postavljen triplestore strežnik Apache Jena Fuseki (po-

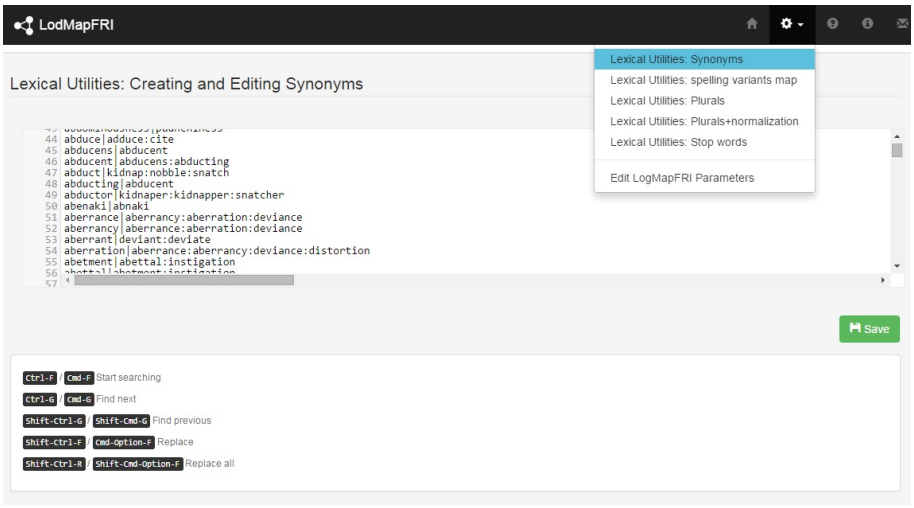
drobneje predstavljena v poglavju 2.23). V primeru preslikave trojic v omenjeno shrambo je potrebno navesti njeno pot.



Slika 4.31: Vnos parametrov za DB2Triples za preslikavo podatkov v RDF in vnos parametrov za usklajevanje (LogMapFRI).

Drugi del spletnega obrazca, ki je prikazan na spodnjem delu slike 4.31 je namenjen področju usklajevanja ontologij z orodjem LogMapFRI. Uporabnik v obrazcu najprej izbere vir oz. ontologijo s katero se bo uskladila njegova ontologija, ki je bila zgrajena v predhodnih korakih. Uporabnik lahko iz nabora ontologij, ki so na voljo v spletni aplikaciji izbere poljubno ali pa naloži svojo. S pomočjo obrazca lahko uporabnik naloži tudi v naprej pripravljena pravilna ujemanja – prej omenjena ujemanja zlatega standarda. Na podlagi njega se preračunajo vse mere, ki so definirane v prejšnjem poglavju 4.3.1.

Pred začetkom procesa usklajevanja ontologij z aplikacijo LogMapFRI, lahko uporabnik uredi različne slovarje. Z njimi si orodje LogMapFRI pomaga pri semantični interpretaciji različnih entitet znotraj ontologij, ki so zgrajene in izbrane za usklajevanje. V glavi spletne aplikacije se nahaja orodna vrstica, preko katere lahko dostopamo do spustnega menija. V tem meniju so dostopni različni slovarji: seznam sinonimov, seznam nedoločnikov, seznam neštevnikov, seznam mašil. Vir omenjenih seznamov je podatkovna baza Wordnet (predstavljena v poglavju 3.1.1),



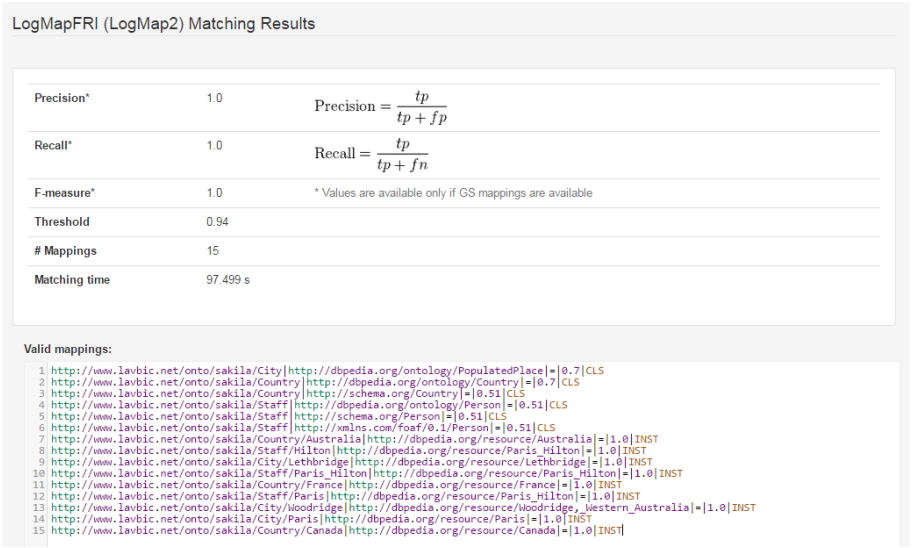
Slika 4.32: Dostop do nastavitev za urejanje leksikonov (sinonimi, mašila, itd.) in LogMapFri nastavitev.

ki smo jo že predstavili v prejšnjih poglavjih. Poleg slovarjev je v spustnem meniju nastavitev na voljo možnost, ki uporabniku omogoča dostop do obrazca nastavitev za orodje LogMapFRI. Nastavitve zajemajo različne parametre, kot so:

- vrednostni pragovi (ang. Threshold), s katerimi lahko vpliva na rezultat usklajevanja,
- omejitve (maksimalno število sinonimov, ki jih lahko orodje upošteva za en niz),
- definicija URI-jev za podporne razrede in relacije med njimi idr.

Znotraj nastavitev lahko upravljamo nad različnimi funkcijami, ki so bile razvite kot izboljšave orodja LogMap. Na takšen način je bila izvedena evalvacija pristopa, ki je predstavljena v naslednjem poglavju. Da povzamemo: po končani uskladitvi ontologij oz. po končanem procesu bogatenja podatkov v relacijski podatkovni bazi, so uporabniku na voljo rezultati v RDF, OWL in TXT obliki - v slednji so trditve ločene s pokončnico (znak "|"). Poleg ujemanj, ki jih je algoritem LogMapFRI označil kot pravilne, so uporabniku na voljo tudi ujemanja, ki jih je algoritem označil kot neustrezna oz. nepravilna. S potrditvijo rezultatov oz.

ujemanj, se trditve zapišejo v triplestore podatkovno bazo Fuseki (predstavljena v poglavju 2.23).



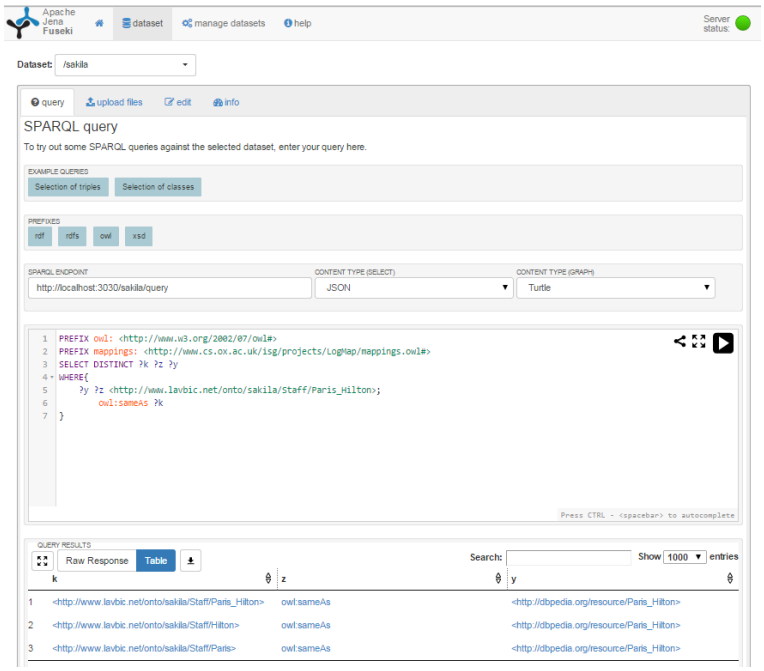
Slika 4.33: Prikaz pravilnih ujemanj ločenih z navpičnico in vrednosti mer natančnosti, preklica in F-mere.

Slika 4.34: Ujemanja, ki so označena kot nepravilna, v .txt formatu (vrednosti so ločene z navpičnico).

4.3.3 Predstavitev rezultatov pristopa in njihova analiza

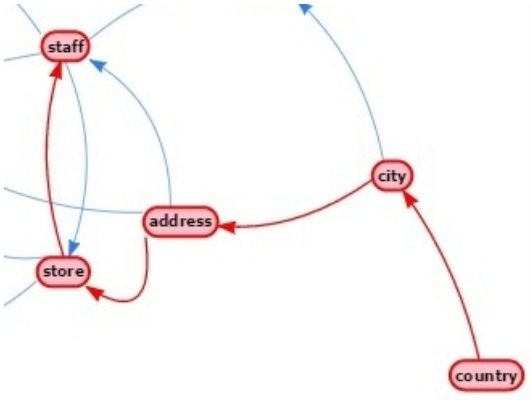
V prejšnjem poglavju je bilo predstavljano orodje, ki je nastalo v okviru verifikacije zastavljene metode. V nadaljevanju bomo predstavili rezultate bogatitve

Slika 4.35: Prikaz rezultata usklajevanja v OWL obliki.



Slika 4.36: Poizvedovanje s pomočjo orodja Apache Jena Fuseki.

podatkov testne relacijske podatkovne baze Sakila s prosto dostopnim repozitorijem DBPedia. Izbrali smo si pot znotraj usmerjenega grafa, ki predstavlja shemo relacijske podatkovne baze. Pot se začne pri vozlišču oz. tabeli "Country" in konča v vozlišču "Staff". Usmerjen graf z izbrano potjo je prikazan na sliki 4.36.



Slika 4.37: Izbrana pot v grafu na katere so prikazani spodnji rezultati.

Podatkovno bazo Sakila je razvil Mike Hilyer. Z njo je želel javnosti ponuditi standardno shemo uporabno na različnih področjih, ki imajo opravka z knjigami, filmi, tečaji, članki ipd. [75].

DBpedia [10] je ena največjih zbirk podatkov v RDF obliki. Gre za zelo pomembno zbirko podatkov, saj se znotraj oblaka Linked data veliko podatkovnih virov poveže prav z njo.

V okviru verifikacije zastavljene metode smo v orodju LodMapFRI v ta namen definirali tri primere, s katerimi smo želeli ovrednotiti prispevek našega pristopa na področju usklajevanja ontologij.

1. V prvem primeru smo izklopili vse izboljšave, ki smo jih aplicirali na orodje LogMap. Torej proces uskladitve podatkov v relacijski podatkovni bazi smo izvedli brez upoštevanja konteksta primerkov (posledično brez upoštevanja podpornih razredov), brez uporabe tehnike lematizacije (uporabljeno krnenje) in z upoštevanjem prvotnega slovarja iz področja biomedicine UMLS [61].

2. V drugem primeru smo ponovno vklopili vse izboljšave razen lematizacije,

namesto katere smo uporabili krnenje. Poleg lematizacije smo prav tako izločili uporabo tehnik, ki temeljijo na zunanjih virih oz. brez upoštevanja sinonimov, mašil in drugih slovarjev, ki jih pridobimo iz okolja WordNet [42] (slovar UMLS je tudi izključen).

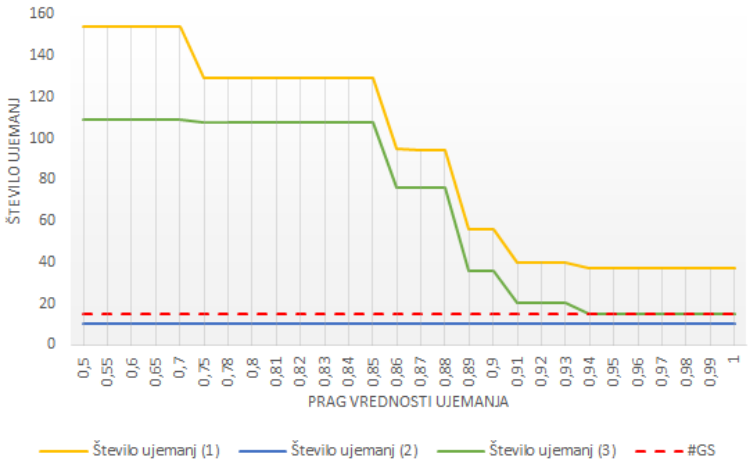
3. V tretjem primeru pa smo upoštevali vse izboljšave, vključno z lematizacijo in Wordnet slovarji.

Vsak definiran scenarij smo izvedli tridesetkrat. Pri vsakem zagonu smo v intervalu od 0 do 1 zviševali vrednostni prag, ki določa veljavnost posamezne uskladitve. Koeficient ujemanja med razredi se določi z mero zaupnosti v kombinaciji z mero podobnosti nizov SMOA (predstavljeno v poglavju 4.2.1), medtem ko se koeficient ujemanja na nivoju primerkov določi s pomočjo mere konteksta, ki je predstavljena v poglavju 4.2.2. Pri preverjanju smo upoštevali mere uspešnosti, ki so predstavljene v poglavju 4.3.1. Seznam vnaprej ročno definiranih pravilnih uskladitev oz. GS ujemanja smo definirali na podlagi ročnih poizvedb izvedenih neposredno v iskalniku vira DBpedije[13] in na podlagi večkrat pregledanega rezultata usklajevanja ontologij z algoritmom LogMapFRI. Rezultat predstavljajo tako nezadostna kot pravilna ujemanja, ki jih dobimo po končanem procesu usklajevanja z orodjem LogMapFRI.

Na podlagi vseh izvedenih primerov smo dobili rezultate, ki so prikazani na slikah 4.38, 4.39, 4.40, 4.41, 4.42.

Graf na sliki 4.38 prikazuje število vseh ujemanj po posameznih primerih. Na grafu je prikazano tudi število GS uskladitev. Iz grafa lahko vidimo, da zgolj zadnji primer, v katerem so upoštevane vse izboljšave, doseže enako število ujemanj, kot jih je definirano v seznamu GS ujemanj. Krivulji v prvem in tretjem primeru sta enake oblike. Razlog tiči v tem, da izhajamo iz enakega orodja, le da je v tretjem primeru upoštevana izboljšava iskanja konteksta primerkov pri katerem ne iščemo novih ujemanj, ampak preverjamo kontekst obstoječim. Če se ta ne ujema se ujemanja zgolj izločijo. Slednje bolj jasno prikazuje drugi primer, pri katerem so rezultati ujemanj konstantni, in to prav zato, ker se iz množice kandidatov (vezi) izločijo vsa ujemanja, pri katerih se kontekst ne ujema. V primerjavi rezultatov drugega in tretjega primera lahko opazimo, da iz naslova alternativnih nizov (pred-

[13] `http://dbpedia.org/fct/`



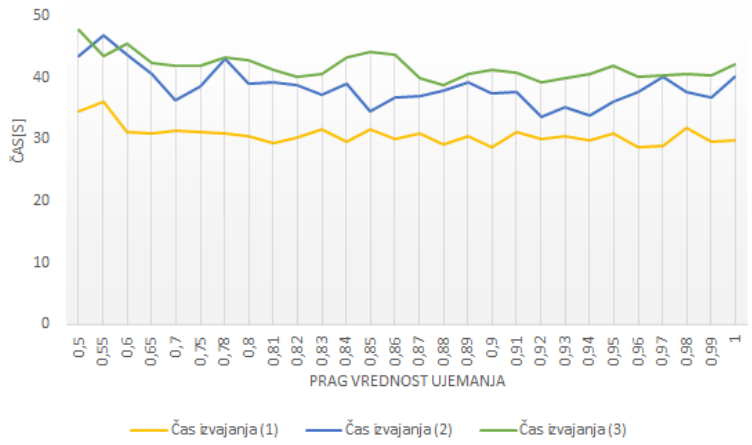
Slika 4.38: Graf, število najdenih ujemanj po posameznih primerih.

vsem sinonimov), dobimo večje število ujemanj. Na podlagi tega lahko sklepamo, da sinonimi pri nizki progovni vrednosti ujemanj povzročijo večjo dvoumnost oz. večje število nepravilnih ujemanj. Pri višanju praga pa vidimo, da nam pripomorejo k uskladitvi primerov, ki so si kontekstno ekvivalentni, ampak leksikografsko zelo različni (Primer: $Staff \equiv Person$).

Na podlagi prvega primera lahko iz grafa na sliki 4.38 ugotovimo, da obstajajo primeri, ki jih brez ugotavljanja njihovega konteksta ne moremo izločiti, saj so si med seboj sintaktično zelo podobni.

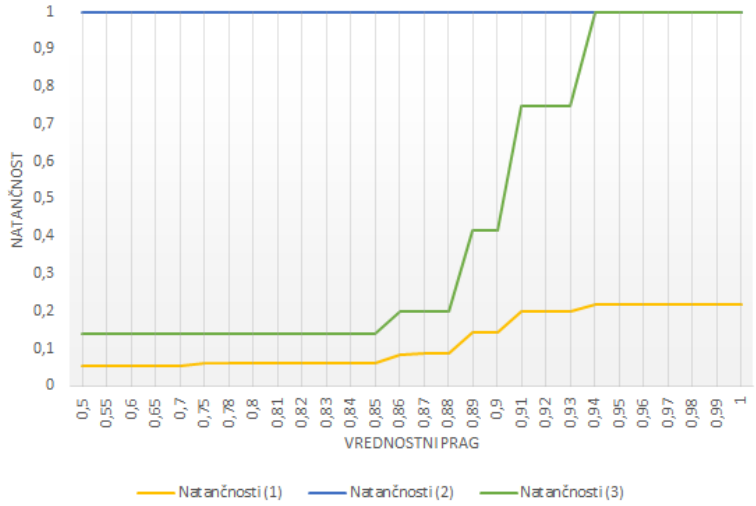
Konstantno število ujemanj v drugem primeru lahko pripišemo koraku iskanja oz. ugotavljanja konteksta primerkov, saj nam orodje izloči vsa tista ujemanja, ki ne sodijo v isti kontekst. Kljub nizki pragovni vrednosti nam kot rezultat proces vrne ujemanja s tako visoko vrednostjo, da imamo skozi celoten postopek enako število pravilnih ujemanj.

Graf na sliki 4.39 prikazuje razliko med primeri glede na porabljen čas za izvedbo uskladitve. Iz grafa lahko razberemo, da se je na račun izboljšav (preiskava strukture obeh ontologij, dodajanja alternativnih nizov itd.) časovna kompleksnost povišala, saj smo v povprečju na aktualnem vsebinskem problemu izgubili 10 sekund, kar predstavlja 1/3 tretjino porabljenega časa osnovnega algoritma. Povišano časovno kompleksnost lahko, na podlagi drugega in tretjega primera, pripišemo procesu pridobivanja alternativnih nizov elementov iz slovarjev Wor-



Slika 4.39: Graf prikazuje čas usklajevanja po posameznih primerih.

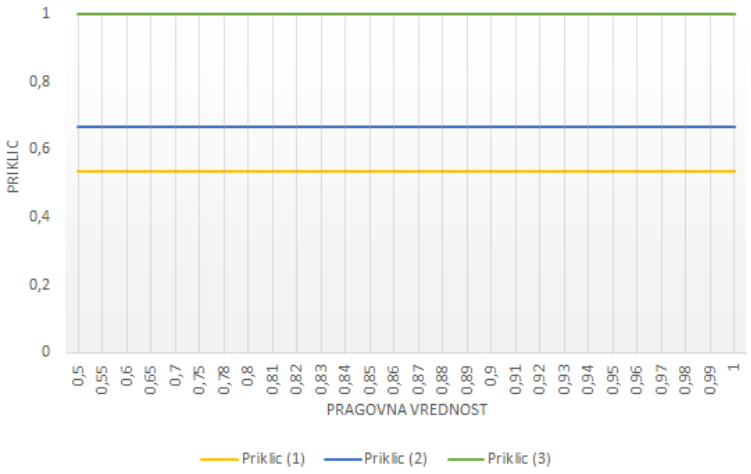
dNet. Iz vrednosti mer število ujemanj, natančnost in priklic prikazanih na grafu na sliki 4.39 ne moremo razbrati nobenega vzorca. Velja pa upoštevati, da smo preizkuse izvajali na osebnem računalniku (CPU: Intel Core i7-4600 2.70GHz; RAM: 12GB), in zato krivulje, ki prikazujejo porabljen čas zastavljenih problemov, niso ravno zglajene.



Slika 4.40: Graf prikazuje rezultat ujemanja glede na mero natančnosti za vse tri definirane primere.

Na grafu na sliki 4.40 je prikazana razlika med tremi primeri in sicer v meri natančnost, ki nam pove kakšno je razmerje med pravilno najdenimi uskladitvami in vsemi najdenimi ujemanji. Zelo velik vpliv pri omenjeni meri ima število vseh uskladitev - če imamo veliko nepravilno določenih uskladitev nam bo ta mera pokazala nizke vrednosti. Število vseh najdenih ujemanj je prikazano v grafu na sliki 4.39. Opazimo lahko, da sta drugi in tretji primer, v katerih upoštevamo predlagane izboljšave, v veliki prednosti pred orodjem brez izboljšav.

Najbolje se izkaže primer v katerem ne uporabljamo zunanjih virov za pridobivanje alternativnih nizov. Iz grafa je razvidno, da so v tem primeru vsa najdena ujemanja vsebovana v množici preddefiniranih pravilnih ujemanj oz. v GS, vendar moramo upoštevati tudi, da v tem primeru orodje ni našlo vseh pravilnih ujemanj, kar vidimo na grafu slike 4.41. Iz grafa slike 4.41 lahko sklepamo tudi, da orodje z upoštevanjem konteksta primerkov (oz. njihovih tipov) izloči vse tiste uskladitve, ki so si sicer sintaktično zelo podobne, vendar semantično (kontekstualno) zelo različne.



Slika 4.41: Graf prikazuje rezultat ujemanja glede na mero priklica za vse tri definirane primere.

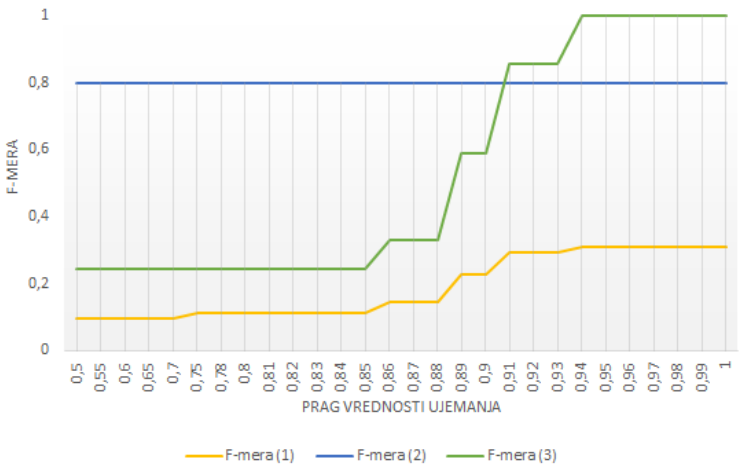
Na grafu 4.41 je prikazana razlika med vsemi tremi primeri na podlagi priklica, ki nam pove kakšno je razmerje med pravilno najdenimi uskladitvami in v naprej definiranimi pravilnimi uskladitvami (GS ujemanja). V zgornjem grafu

lahko opazimo, da so vrednosti priklica vseh treh primerov konstantne.

Na istem grafu vidimo tudi, da so pri obeh primerih, kjer so upoštevane oz. delno upoštevane izboljšave (tj. drugi in tretji primer), rezultati boljši oz. je vrednost priklica višji od primera, pri katerem izboljšave niso upoštevane (tj. prvi primer). Na podlagi rezultatov drugega primera lahko opazimo, da brez upoštevanja zunanjih virov ni bilo mogoče poiskati vseh predpisanih ujemanj (ujemanj iz GS). Kljub temu so rezultati v drugem primeru zadovoljivi, saj je orodje sposobno najti kompleksnejše povezave, ki so sestavljene iz več entitet posamezne ontologije (primer tovrstnih povezav je prikazana v tabeli 4.7).

Element ontologije Sakila	**Relacija**	**Element DBpedije**
"lavbic-owl:Paris"	≡	"dbpedia:Paris_Hilton"
"lavbic-owl:Woodridge"	≡	"dbpedia:Woodridge,_Western_Australia"

Tabela 4.7: Primer kompleksnih povezav.



Slika 4.42: Graf prikazuje rezultat ujemanja glede na F-mero za vse tri definirane primere.

Sistemov za usklajevanje ontologij večkrat ni mogoče primerjati med seboj samo na podlagi ene posamezno mere (primer: primerjava sistemov samo glede na priklic). Če imajo taki sistemi višji priklic, imajo ponavadi manjšo natančnost in obratno. Prav zato uporabimo F-mero, ki združuje obe prej obravnavani meri in

nam poda realen koeficient za primerjavo sistemov med seboj. Mi bomo F-mero uporabili zgolj za skupni pregled prejšnjih ugotovitev. Zgornji graf potrjuje ugotovitve iz prejšnjih dveh grafov. Vrednosti F-mere za drugi in tretji primer, kjer smo upoštevali izboljšave, so višje od prvega, pri katerem izboljšave niso uporabljene. Iz zgornjega grafa lahko spet potrdimo doprinos lematizacije k iskanju pravilnih ujemanj, saj je tretji primer spet nekoliko boljši od drugega.

V nadaljevanju bomo orodje LogMapFRI preizkusili še na dveh testih organizacije OAEI iz leta 2014, in sicer na testu iz področja velikih ontologij in na iz področja usklajevanja primerkov.

4.3.4 Evalvacija orodja za usklajevanje ontologij LogMapFRI v okviru OAEI 2014

Naše orodje smo preizkusili v dveh področjih OAEI 2014, in sicer:

- Large Biomedical Ontologies FMA-NCI matching problem in
- Instance Matching.

Ontology Alignment Evaluation Initiative[14] (OAEI) je mednarodna pobuda, ki organizira ocenjevanje večjega števila sistemov namenjenih usklajevanju ontologij. Glavni cilj OAEI je primerjati sisteme in algoritme, in sicer tako, da se vsem sodelujočim zagotovi enake pogoje za evalvacijo posameznih orodij. S tovrstno evalvacijo je moč pridobiti informacije o najboljših pristopih na področju usklajevanja ontologij. Namen organizacije je takšne informacije oz. na novo odkrite smernice ponuditi razvijalcem, in jim tako omogočiti da bi z novimi, že veljavnimi pristopi, izboljšali svoje algoritme oz. sisteme.

Dogodki, ki jih organizacija OAEI izvaja, segajo v leto 2004. Do leta 2011 so evalvacije orodij potekale v okviru različnih delavnic. Po letu 2011 se je evalvacija orodij nekoliko avtomatizirala. Začelo se je uporabljati okolje za samodejno evalvacijo orodij, ki je bila razvita v okviru projekta SEALS (Semantic Evaluation At Large Scale). SEALS[15] ponuja programsko infrastrukturo za samodejno izvajanje evalvacije orodij na področju semantičnega spleta, v katero spadajo tudi

[14] `http://oaei.ontologymatching.org/`
[15] `http://www.seals-project.eu/`

orodja za usklajevanje ontologij. V času, ko smo mi evalvirali naše orodje, testne ontologije žal niso bile na voljo, smo pa evalvacijo zato izvedli na ročno izvoženih podatkovnih množicah.

Large Biomedical Ontologies FMA-NCI matching problem

Gre za evalvacijo, pri kateri iščemo ujemanja med velikimi ontologijami kot so: Foundational Model of Anatomy (FMA), National Cancer Institute Thesaurus (NCI) in druge. Ontologije, ki jih usklajujemo so semantično bogate in vsebujejo več deset tisoč objektov. Rezultat usklajevanja je prikazan v spodnji tabeli. Vrstica "Average" prikazuje povprečje vseh orodij, ki nastopajo pri ocenjevanju.

Konkretno, spodnji tabeli predstavljata rezultat pri usklajevanju ontologij FMA in NCI. Prvo opravilo sestavljata dva relativno majhna dela obeh ontologij. Prvi del ontologije FMA vsebuje 3696 razredov (5% celote), medtem ko del ontologije NCI vsebuje 6488 (10% celotne ontologije). Iz spodnje tabele lahko razberemo, da se je naše orodje izkazalo kot najbolj natančno. Izboljšavo lahko pripišemo dejstvu, da smo upoštevali kontekst posameznega objekta v ontologijah, ki jih usklajujemo. Na spodnji tabeli LogMapFRI primerjamo z različico brez izboljšav in vidimo, da je to orodje pod mejo povprečja.

Orodje	**Čas (s)**	**Število ujemanj**	**Natančnost**	**Priklic**	**F-mera**
AML	112	2931	0,832	0,856	0,844
LogMapFRI	**323**	**2694**	**0,868**	**0,773**	**0,818**
LogMap-Bio	1226	3412	0,724	0,874	0,792
XMap	144	2571	0,835	0,745	0,787
Average	**142**	**749**	**0,81**	**0,745**	**0,761**
LogMap-C	289	2124	0,877	0,65	0,747
LogMapFRI - brez izboljšav	**235**	**3146**	**0,726**	**0,761**	**0,743**
LogMapLite	44	3467	0,675	0,819	0,740
OMReasoner	36369	1403	0,964	0,466	0,628

Tabela 4.8: Rezultat usklajevanju manjših delov ontologij FMA in NCI

Drugo opravilo zajema usklajevanje celotnih ontologij FMA in NCI, ki vsebujeta 78989 in 66724 razredov. Rezultat je prikazan v tabeli 4.9, iz katere je razvidno

da se je orodje LogMap-Bio bolje odrezal od naše izboljšave. Takšen rezultat lahko pripišemo namenskim slovarjem, jih tovrstna orodja uporabljajo. Orodje LogMap-Bio je namreč orodje prilagojeno usklajevanje ontologij na področju biomedicine.

Orodje	**Čas (s)**	**Število ujemanj**	**Natančnost**	**Priklic**	**F-mera**
AML	27	269	0,96	0,899	0,928
LogMap-Bio	975	2892	0,914	0,918	0,916
LogMapFRI	**46**	**2716**	**0,956**	**0,858**	**0,905**
LogMapFRI - brez izboljšav	**37**	**2816**	**0,923**	**0,867**	**0,894**
XMap	17	2657	0,932	0,848	0,888
LogMapLite	5	2479	0,967	0,819	0,887
LogMap-C	81	2153	0,962	0,724	0,826
MaasMatch	1460	2981	0,808	0,840	0,824
Average	**101,992**	**2178**	**0,914**	**0,729**	**0,779**
AOT	9341	3696	0,662	0,855	0,746
OMReasoner	82000	1362	0,995	0,466	0,635
RSDLWB	2216	728	0,962	0,236	0,380
AOTL	20908	790	0,902	0,237	0,375

Tabela 4.9: Rezultat usklajevanju celotnih ontologij FMA in NCI

Instance Matching

Namen ocenjevanja usklajevanja ontologij je evalvacija orodij, ki so sposobna prepoznati podobnosti med različnimi RDF in OWL nabori objektov (owl razredi in primerki). Ena izmed nalog ocenjevanja je prepoznavanje identičnosti (ang. id-rec task), pri kateri udeleženci prejmejo dva nabora podatkov, ki so označeni kot vir in cilj. Cilj naloge je odkriti ujemanja parov (tj. preslikave) med primerki v izvornem nizu in ciljnem OWL nizu. Pričakovana oz. pravilna ujemanja udeležencem ponavadi niso na voljo takoj. V času evalvacije našega orodja pa so bili rezultati pričakovanih ujemanj že na voljo, tako da smo lahko prišli do izračuna mer (natančnost, priklic in F-mera) in kasneje naše orodje primerjali z drugimi orodji, ki so se evalvirala na isti nalogi.

Kot smo že omenili, na voljo sta dva nabora objektov za ocenjevanje. Prvi vsebuje 1330 primerkov, ki so opisane z štirimi razredi, petimi lastnostmi podat-

kovnega tipa in z eno lastnostjo oznake. Ciljni nabor pa sestavlja 2649 primerkov, ki so opisane z štirimi razredi, štirimi lastnostmi podatkovnega tipa, eno lastnostjo objekta in eno lastnostjo oznak.

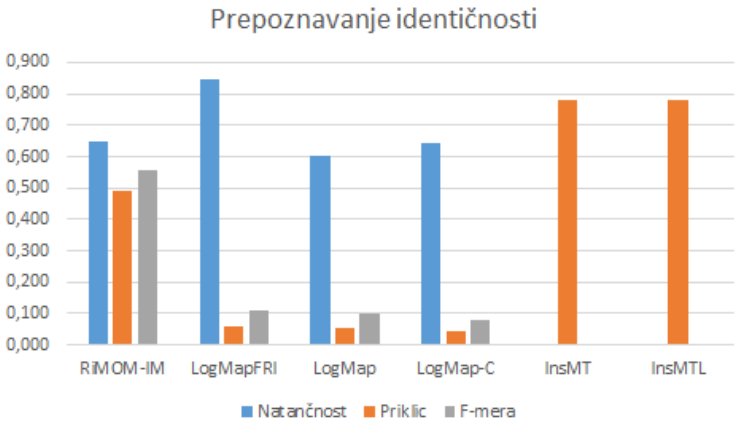
Na prvi pogled se rezultati v tabeli 4.10, na splošno, ne zdijo zadovoljivi, predvsem zaradi nezadovoljive vrednosti priklica. To lahko pripišemo skriti nalogi preizkusa, kajti večina izrazov iz ene ontologije je bilo prevedenih iz angleškega v italijanski jezik. Ker v našem pristopu (in tudi v vseh LogMap različicah) ni vključenih večjezičnih tehnik, ki bi vse izraze, nad katerimi se izvaja proces usklajevanja, prevedle v enoten jezik (npr. angleščina), je rezultat priklica zelo nizek. Vendar pa se lahko pohvalimo z visoko natančnostjo. Na podlagi poznavanja osnovnega delovanja orodja LogMap in na podlagi predstavljenih izboljšav (poglavje 4.2.2) namreč vemo, da za vsako ujemanje iz začetnega nabora (vezi) dodatno preverjamo njihov kontekst in v primeru neujemanja sledi izločitev. Pri semantično revnih ontologijah se konteksta ne da razbrati in takrat se preverjanje ujemanja izvede zgolj na podlagi tehnik na nivoju nizov oz. tehnik, ki določajo mero podobnosti med nizi. Pri orodjih InsMT in InsMTL, pa je razvidno, da je množica ujemanj, ki jo omenjeni orodji vrneta kot rezultat ujemanja, preobilna, kar se pozna na visokem rezultatu priklica in nizki natančnosti ter posledično tudi na f-meri.

V tabeli 4.10 lahko vidimo, da se je algoritem LogMapFRI izkazal za bolj natančnega kot LogMap. Je pa pri F-meri LogMapFRI zaostal za RiMOM-IM, ki je namensko orodje za usklajevanje primerkov. Ker je naš algoritem namenjen usklajevanju tako razredov kot primerkov, menimo da bi se v primeru usklajevanja z semantično bogatejšimi ontologijami (z razmeroma večjim številom razredov) naša metoda bolje izkazala. Pri usklajevanju primerkov namreč izkoriščamo tudi podatek o kontekstu primerkov (na podlagi njihovih definicij, kot je npr. tip).

Na podlagi predstavljenih rezultatov iz tabel in grafa na sliki 4.43, lahko sklepamo, da so izboljšave aplicirane v orodju LogMapFRI zelo pozitivno vplivale na preiskovanje pravilnih uskladitev oz. na tista ujemanja, ki ne sodijo v kontekst. Pomembno vlogo je imela tudi lematizacija, s katero smo uspešneje določili normalno obliko zapisov in lažje uskladili tudi tiste nize, ki jih z običajnimi tehnikami za ugotavljanje podobnosti med nizi ne bi mogli.

Orodje	Natančnost	Priklic	F-mera
RiMOM-IM	0,649	0,489	0,558
LogMapFRI	**0,848**	**0,057**	**0,108**
LogMap	0,603	0,054	0,099
LogMap-C	0,642	0,042	0,078
InsMT	0,001	0,779	0,002
InsMTL	0,001	0,779	0,002

Tabela 4.10: Prikaz rezultata preizkušnje iz področja usklajevanja ontologij in primerjava orodja LogMapFRI z ostalimi aktivnimi orodji na tem področju



Slika 4.43: Rezultat preizkusa na področju usklajevanja primerkov: prepoznavanje identičnosti.

Poglavje 5

Sklepne ugotovitve

Pri relacijskih podatkovnih bazah smo zaradi specifičnosti podatkov, ki se v njih hranijo, precej omejeni pri poskusih izboljšanja njihove kvalitete. To težko izboljšamo samo s čiščenjem in povezovanjem obstoječih podatkov, temveč predvsem z njihovim bogatenjem [76]. Vir podatkov, ki bi lahko bil uporabljen za bogatenje prej izoliranih podatkov v relacijski bazi, je oblak strukturiranih podatkov LOD. Ti podatki so strukturirani s pomočjo tehnologij semantičnega spleta. Preden bi podatke iz relacijske baze sploh lahko povezali, jih moramo naprej na pravi način predstaviti. Šele take lahko povezujemo z viri iz oblaka LOD, in jih obogatimo, tako da jim izboljšamo semantično izraznost. Ker v literaturi nismo identificirali ustreznega postopka, ki bi celostno reševal to problematiko, smo ga v magistrski nalogi predstavili, ter nato še verificirali in evalvirali sami.

Proces predlagane rešitve je sestavljen iz dveh krovnih procesov oz. korakov. Prvi korak zajema preslikavo sheme oz. podsheme poljubne vhodne relacijske podatkovne baze v RDF obliko. V drugem koraku pa problematiko bogatenja podatkov rešujemo s tehnikami usklajevanja ontologij. Naredimo namreč preslikavo in povezovanje ontologije in konkretnih primerkov vhodne podatkovne baze z drugimi ontologijami. Tu gre za prosto dostopne repozitorije iz LOD oblaka (DBpedia, Freebase, itd.), ki poleg podatkov o strukturi zajemajo tudi konkretne primerke. Po končanem procesu bogatitve izbrane relacijske podatkovne baze naša začetna shema pridobi dodatne trditve v obliki trojic oz. povezave z obstoječimi shemami oz. ontologijami. Npr. razredi pridobijo relacije tipa relacije tipa owl:equivalentClass, rdfs:subClassOf ipd., primerki pa pridobijo relacije tipa

owl:sameAs, rdfs:seeAlso ipd.

Drugi korak naše rešitve podrobneje obravnavamo v poglavju 4.2, kjer definiramo korake usklajevanja naše ontologije narejene na osnovi sheme naše relacijske podatkovne baze in njenih primerkov, z javno dostopnimi ontologijami oz. repozitoriji iz linked data oblaka – v našem primeru je to DBpedia. Za izpolnitev tega koraka smo morali izbrati ustrezen pristop, zato smo v poglavju 3 naredili pregled sodobnih tehnik in pristopov na področju usklajevanja ontologij. Z razumevanjem tehnik smo lahko nato v poglavju 3.2 opravili pregled in primerjavo orodij oz. sistemov iz omenjenega področja, na katerem bi lahko osnovali svojo implementacijo. Na podlagi naših specifičnih potreb smo se odločili za odprtokodno orodje LogMap. Kljub temu, da orodje podpira širok nabor sodobnih tehnik iz področja usklajevanja ontologij, so bile za reševanje našega problema (pričakovano) potrebne izboljšave, ki jih ravno tako opisujemo v poglavju 4.

Praktični del naše rešitve je prilagojeno orodje za usklajevanje ontologij LogMap, imenovano LogMapFRI. Pri tem orodju smo v prvem koraku, kjer se izvede indeksiranje leksikalnih elementov, izboljšali bogatenje nizov elementov s sinonimi iz splošnega tezavra WordNet, izboljšali tudi čiščenje nizov in z uporabo večjezične lematizacije (LemmaGen) zagotovili boljšo normalizacijo nizov. Že v fazi indeksiranja nizov primerkov povežemo skupaj primerke, ki spadajo v isti kontekst oz. predstavljajo isto entiteto iz relacijske podatkovne baze. V fazi indeksiranja strukture ontologij smo, pri zajemu tabel iz relacijske podatkovne baze, ohranili povezave med njimi in tako izboljšali zajem konteksta podatkov, ki jih bogatimo.

S pomočjo orodja LogMapFRI smo nato naš pristop evalvirali, in sicer na konkretnem primeru povezovanja. Delovanje izboljšanega orodja LogMapFRI pa smo preizkusili še na področju usklajevanja velikih ontologij in primerkov, in sicer s testnimi ontologijami iz preizkušnje OAEI 2014.

Pri primerjavi rezultatov orodja brez upoštevanja izboljšav in orodja z našimi izboljšavami smo videli, da v vseh merjenih kazalcih slednje izstopa. Smo pa na ta račun žrtvovali čas, saj so testi pokazali, da so tovrstni pristopi časovno potratni. Na podlagi rezultatov evalvacije iz poglavja 4.3.3 smo ugotovili, da se nam je na račun vseh izboljšav časovna kompleksnost povišala, saj smo v povprečju na aktualnem vsebinskem problemu izgubili 1/3 tretjino porabljenega časa osnovnega algoritma. Vseeno pa ima dodaten pribitek časa pri poganjanju orodja LogMapFRI

svojo vrednost. Z razširitvijo upoštevanja konteksta primerkov v odvisnosti od njegovih sosedov in z razširitvijo sintaktičnih pristopov smo veliko doprinesli k natančnosti samega procesa. To je bilo ključno za boljši zajem podatkov, saj je zelo vplivalo na vrednosti mer, ki smo jih spremljali.

Orodje ima po našem mnenju precejšnjo praktično vrednost. Vendar, če bi ga želeli implementirati kot spletno storitev, bi bilo najprej treba preučiti možnosti serviranja velikega števila hkratnih uporabnikov.

Pri delu z večjimi javnimi repozitoriji iz LOD oblaka, bi bilo tudi smiselno omogočiti dostop do podatkov s pomočjo API vmesnikov. Tako nam ne bi bilo treba z njimi operirati iz lokalnih posnetkov repozitorijev (ang. "RDF dump").

Kar se tiče samega orodja, bi bilo vsekakor smiselno preučiti še možnost vključitve lastnosti (lastnosti objektov in lastnosti podatkov), s pomočjo katerih bi lahko morebiti še bolje definirali kontekst. Usklajevanje bi bilo lahko tako še natančnejše. Tekom procesa usklajevanja oz. obogatitve bi bilo smiselno uporabnikom omogočiti višji nivo interakcije. Natančneje, omogočiti bi jim bilo potrebno sodelovanje v samem procesu usklajevanja tako, da bi z uporabniškimi odločitvami sproti pregledovali in spreminjali ujemanja, ki so bila označena kot neveljavna. Tekom evalvacije pristopa smo ugotovili da bi bilo smiselno v procesu usklajevanja uporabiti večjezično tehniko, ki bi zmogla elemente nizov prevesti v enoten jezik, in tako bi se nad takšnimi nizi izvedle druge tehnike na nivoju nizov (v našem primeru SMOA).

Pri orodju bi se dalo izboljšati tudi algoritem zaznave, kar bi pripomoglo k zaznavanju kompleksnejših relacij med tabelami v relacijski podatkovni bazi. Na ta način bi se lahko pridobilo še več informacij o kontekstu posameznih elementov, ki jih usklajujemo.

Literatura

[1] A. Doan and A. Y. Halevy, "Semantic integration research in the database community: A brief survey.", *AI Magazine*, vol. 26, no. 1, pp. 83–94, 2005.

[2] Linked Data Community, "Linked data - connect distributed data across the web", Dostopno na: `http://linkeddata.org/home`, Poskus dostopa: Junij 2015.

[3] E. Miller, Swick R., and D. Brickley, *Resource description framework (RDF)*, World Wide Web Consortium W3C., 2003.

[4] F. Manola, E. Miller, and B. McBride, "RDF 1.1 Primer", Dostopno na: `http://www.w3.org/TR/rdf11-primer/`, Poskus dostopa: Junij 2015.

[5] D. Brickley and R.V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema", Dostopno na: `http://www.w3.org/TR/rdf-schema/`, Poskus dostopa: Junij 2015.

[6] S. Das, S. Sundara, and R. Cyganiak, "R2RML: RDB to RDF Mapping Language", Dostopno na: `http://www.w3.org/TR/r2rml/`, Poskus dostopa: Junij 2015.

[7] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language overview", Dostopno na: `http://www.w3.org/TR/owl-features/`, Poskus dostopa: Junij 2015.

[8] S. Harris and A. Seaborne (eds), "SPARQL 1.1 query language", Dostopno na: `http://www.w3.org/TR/sparql11-query`, Poskus dostopa: Junij 2015.

[9] A. Jentzsch and R. Cyganiak and C. Bizer, "State of the lod cloud", Dostopno na: `http://lod-cloud.net/state/`, Poskus dostopa: Junij 2015.

[10] "DBpedia", Dostopno na: http://wiki.dbpedia.org/, Poskus dostopa: Junij 2015.

[11] "Freebase", Dostopno na: https://www.freebase.com/, Poskus dostopa: Junij 2015.

[12] Structural Informatics Group, University of Washington, "Foundational model of anatomy ontology", Dostopno na: http://sig.biostr.washington.edu/projects/fm/, Poskus dostopa: Junij 2015.

[13] National Centers for Biomedical Computing, "National cancer institute thesaurus", Dostopno na: http://bioportal.bioontology.org/ontologies/NCIT, Poskus dostopa: Junij 2015.

[14] V. Bush, "As we may think", *The Atlantic Monthly*, 1945.

[15] T. Berners-Lee and R. Cailliau, "WorldWideWeb: Proposal for a HyperText Project", Dostopno na: http://www.w3.org/Proposal.html, November 1990, Poskus dostopa: Junij 2015.

[16] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", *Scientific American Magazine*, vol. 284, no. 5, pp. 34–43, 2001.

[17] N. Spivack (Radar Networks), "How the webos evolves?", Dostopno na: www.radarnetworks.com, Poskus dostopa: Junij 2015.

[18] World Wide Web Consortium (W3C), "W3C Semantic Web Activity", Dostopno na: http://www.w3.org/2001/sw/, Poskus dostopa: Junij 2015.

[19] T. R. Gruber, "A translation approach to portable ontology specifications", *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[20] T. Berners-Lee, R. T. Fielding, and L. Masinter, "Uniform resource identifier (uri): Generic syntax", *Network Working Group*, vol. 66, no. 3986, pp. 1–61, 2005.

[21] F. Gandon and G. Schreiber, *RDF 1.1 XML Syntax*, World Wide Web Consortium W3C., 2014, Poskus dostopa: Junij 2015.

[22] M. Južna, "Pretvorba xml podatkov v owl ontologijo", diplomsko delo, FRI - Fakulteta za računalništvo in informatiko, 2009.

[23] E. Simperl, M. Acosta, M. Dimitrov, J. Domingue, P. Haase, M. Maleshkova, A. Mikroyannidis, B. Norton, and M. E. Vidal, "Euclid: Educational curriculum for the usage of linked data", Dostopno na: `http://www.euclid-project.eu/`, Poskus dostopa: Junij 2015.

[24] L. Feigenbaum, "Sparql by example", Dostopno na: `http://www.w3.org/2009/Talks/0615-qbe/`, Poskus dostopa: Junij 2015.

[25] W3C, "Popular SPARQL Endpoints", Dostopno na: `http://www.w3.org/wiki/SparqlEndpoints`, Poskus dostopa: Junij 2015.

[26] The Apache Software Foundation, "Apache jena", Dostopno na: `https://jena.apache.org`, Poskus dostopa: Junij 2015.

[27] T. Berners-Lee, "Linked data (5 star open data)", Dostopno na: `http://www.w3.org/DesignIssues/LinkedData`, Poskus dostopa: Junij 2015.

[28] A. Dimou, M. Vander Sande, P. Colpaert, E. Mannens, and R. Van de Walle, "Extending r2rml to a source-independent mapping language for rdf", in *International Semantic Web Conference, Proceedings*, 2013, pp. 237–240.

[29] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda, "A Direct Mapping of Relational Data to RDF", Dostopno na: `http://www.w3.org/TR/rdb-direct-mapping/`, Poskus dostopa: Junij 2015.

[30] J. Vidmar, *Preslikava in obogatitev podatkov iz relacijskih podatkovnih baz v RDF obliko: diplomsko delo*, J. Vidmar, 2014.

[31] The Apache Software Foundation, "Apache Jena Fuseki", Dostopno na: `https://jena.apache.org/documentation/fuseki2/`, Poskus dostopa: Junij 2015.

[32] B. Villazón-Terrazas and M. Hausenblas, "R2RML and Direct Mapping Test Cases", Dostopno na: `http://www.w3.org/2001/sw/rdb2rdf/test-cases/`, Poskus dostopa: Junij 2015.

[33] B. Villazón-Terrazas and M. Hausenblas, "RDB2RDF Implementation Report", Dostopno na: `http://www.w3.org/TR/rdb2rdf-implementations/`, Poskus dostopa: Junij 2015.

[34] F. Michel, J. Montagnat, and C. Faron-Zucker, "A survey of RDB to RDF translation approaches and tools", Research report, I3S, May 2014, ISRN I3S/RR 2013-04-FR 24 pages.

[35] LATC, "5 star open data", Dostopno na: `http://5stardata.info/`, Poskus dostopa: Junij 2015.

[36] M. Horridge, S. Jupp, G. Moulton, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using protégé 4 and co-ode tools edition1. 2", *The University of Manchester*, 2009.

[37] J. Euzenat and P. Shvaiko, *Ontology Matching, Second Edition.*, Springer, 2013.

[38] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching", *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[39] V. Levenshtein, "Binary Codes Capable of Correcting Deletions and Insertions and Reversals", *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[40] G. Stoilos, G. Stamou, and S. Kollias, "A String Metric For Ontology Alignment", in *Proceedings of the 4rd International Semantic Web Conference (ISWC)*, Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, Eds., Berlin, H., November 2005, vol. 3729 of *Lecture Notes in Computer Science*, pp. 624–637, Springer.

[41] D. Maynard and S. Ananiadou, "Term extraction using a similarity-based approach", in *Recent Advances in Computational Terminology*. John Benjamins, 1999.

[42] Princeton University, "About wordnet - wordnet - about wordnet", Dostopno na: `https://wordnet.princeton.edu/wordnet/`, Poskus dostopa: Junij 2015.

[43] IJS - Institut Jozef Stefan, "slowtool", Dostopno na: `http://nl.ijs.si/slowtool/`, Poskus dostopa: Junij 2015.

[44] D. N. Christodoulakis, "Balkanet - design and development of a multilingual balkan wordnet", Dostopno na: `http://www.dblab.upatras.gr/balkanet/`, Poskus dostopa: Junij 2015.

[45] J. Toporišič, *Slovenska slovnica. 4., prenovljena in razširjena izd.*, Maribor : Obzorja, 2000 (Ljubljana : Delo), 2000.

[46] M. Lesk, "Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone", in *Proceedings of SIGDOC*, 1986.

[47] N. Ide and J. Veronis, "Word sense disambiguation: The state of the art", *Computational Linguistics*, vol. 24, no. 1, pp. 1–40, 1998.

[48] A. Miles and S. Bechhofer, "SKOS simple knowledge organization system reference", Working draft, W3C, 2009.

[49] C. Trojahn dos Santos, P. Quaresma, and R. Vieira, "An api for multilingual ontology matching", in *Proc. 7th conference on Language Resources and Evaluation Conference (LREC), Valletta (MT)*, N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odjik, S. Piperidis, M. Rosner, and D. Tapias, Eds., 2010, pp. 3830–3835.

[50] W. F. Dowling and J. H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional horn formulae.", *J. Log. Program.*, vol. 1, no. 3, pp. 267–284, 1984.

[51] G. Gallo and G. Urbani, "Algorithms for testing the satisfiability of propositional formulae.", *J. Log. Program.*, vol. 7, no. 1, pp. 45–61, 1989.

[52] Antidot, "Open source software: db2triples", Dostopno na: `http://www.antidot.net/en/ecosystem/db2triples/`, Poskus dostopa: Junij 2015.

[53] J. Currier, "Schemaspy - graphical database schema metadata browser", Dostopno na: `http://schemaspy.sourceforge.net/`, Poskus dostopa: Junij 2015.

[54] AT&T Research, "Graphviz - graph visualization software", Dostopno na: http://www.graphviz.org/, Poskus dostopa: Junij 2015.

[55] R. Radev, "Representing a relational database as a directed graph and some applications.", in *BCI (Local)*, Christos K. Georgiadis, Petros Kefalas, and Demosthenes Stamatis, Eds. 2013, vol. 1036 of *CEUR Workshop Proceedings*, p. 1, CEUR-WS.org.

[56] S. Skiena, *The Algorithm Design Manual (2. ed.).*, Springer, 2008.

[57] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, 3rd edition, 2009.

[58] "Graph500 benchmark specification (for petascale-era supercomputer performance evaluation.", Dostopno na: http://www.graph500.org/specifications#sec-5, Poskus dostopa: Junij 2015.

[59] "Depth-first search and breadth-first search in python.", Dostopno na: http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/, Poskus dostopa: Junij 2015.

[60] R. Sokol, "Vizualizacija iskanja z algoritmom lrta*", diplomsko delo, FRI - Fakulteta za računalništvo in informatiko, 2013.

[61] U.S. National Library of Medicine, "Unified medical language system", Dostopno na: http://www.nlm.nih.gov/research/umls/, Poskus dostopa: Junij 2015.

[62] R. Agrawal, A. Borgida, and H. V. Jagadish, "Efficient management of transitive relationships in large data and knowledge bases", in *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 1989, pp. 253–262, ACM Press.

[63] V. Nebot and R. B. Llavori, "Efficient retrieval of ontology fragments using an interval labeling scheme.", *Inf. Sci.*, vol. 179, no. 24, pp. 4151–4173, 2009.

[64] V. Christophides, D. Plexousakis, M. Scholl, and S. Tourtounis, "On Labeling Schemes for the Semantic Web", in *Proc. of the 12th International World Wide Web Conference (WWW'03)*, Budapest, Hungary, 2003, pp. 544–555.

[65] E. Jiménez-Ruiz and B. C. Grau, "Logmap: Logic-based and scalable ontology matching.", in *International Semantic Web Conference (1)*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, Eds. 2011, vol. 7031 of *Lecture Notes in Computer Science*, pp. 273–288, Springer.

[66] M. Cheatham and P. Hitzler, "String similarity metrics for ontology alignment", in *The Semantic Web – ISWC 2013*, H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, and K. Janowicz, Eds., vol. 8219 of *Lecture Notes in Computer Science*, pp. 294–309. Springer Berlin Heidelberg, 2013.

[67] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage", in *Proceedings of the Section on Survey Research*, 1990, pp. 354–359.

[68] Inštitut Jožef Štefan, "LemmaGen - multilingual opensource lemmatization", Dostopno na: `http://lemmatise.ijs.si/`, Poskus dostopa: Junij 2015.

[69] V. Laharnar, *Kontekstualno ujemanje in iskanje na modelu spletne oglasne deske: diplomsko delo na univerzitetnem študiju*, V. Laharnar, 2013.

[70] E. Jerome, "Ontology alignment evaluation initiative", Dostopno na: `http://oaei.ontologymatching.org/`, Poskus dostopa: Junij 2015.

[71] J. Euzenat, "Semantic precision and recall for ontology alignment evaluation.", in *IJCAI*, Manuela M. Veloso, Ed., 2007, pp. 348–353.

[72] H. Do, S. Melnik, and E. Rahm, "Comparison of schema matching evaluations", in *Proceedings of the second international workshop on Web Databases (German Informatics Society)*, 2002.

[73] D. Balija, "LodMapFRI", Dostopno na: `https://bitbucket.org/dlavbic/lodmapfri`, Poskus dostopa: Junij 2015.

[74] Almende B. V., "vis.js - A dynamic, browser based visualization library.", Dostopno na: http://visjs.org/, Poskus dostopa: Junij 2015.

[75] M. Hillyer, "Sakila Sample Database", Dostopno na: https://dev.mysql.com/doc/sakila/en/, Poskus dostopa: Junij 2015.

[76] P. Brooks and L. Ellams and T. Kilburn and R. Roussell and T. Hughes, "5 best practices for data enhancement and data cleansing", Dostopno na: http://www.connection2.com/5-best-practices-for-data-enhancement-and-data-cleansing/, Poskus dostopa: Junij 2015.