Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nik Adžič

Inteligentni pomočnik za pridobivanje delno strukturiranih spletnih podatkov

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2016

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Nik Adžič sem avtor magistrskega dela z naslovom:

Inteligentni pomočnik za pridobivanje delno strukturiranih spletnih podatkov

S svojim podpisom zagotavljam, da:

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

V Ljubljani, 8. septembra 2016 Podpis avtorja:

Zahvaljujem se svojemu mentorju doc. dr. Dejanu Lavbiču za zelo dobro strokovno usmerjanje in spodbujanje pri izdelavi magistrskega dela. Prav tako bi se rad zahvalil vsem članom laboratorija za podatkovne tehnologije, ki so mi omogočili odlične delovne pogoje za opravljanje magistrskega dela v laboratoriju.

Posebno zahvalo namenjam še svojim staršem. Hvala vama za vso vajino materialno in moralno podporo tekom študija.

Kazalo

1 Uvod **1**

1.1 Motivacija 2

1.2 Cilji 2

1.3 Struktura magistrske naloge 3

2 Tehnike za pridobivanje spletnih podatkov in semantični splet **7**

2.1 Pridobivanje spletnih podatkov 7

2.2 Tehnike pridobivanja spletnih podatkov 8

2.2.1 Ročno kopiranje in prilepljanje 8

2.2.2 Iskanje po besedilu in regularni izrazi 9

2.2.3 HTTP zahtevki 9

2.2.4 DOM razčlenjevanje 10

2.2.5 HTML razčlenjevalniki 10

2.2.6 Vertikalne agregracijske platforme 10

2.2.7 Semantično prepoznavanje anotacij 10

2.2.8 Programska orodja, ki pridobivajo podatke s pomočjo računalniškega vida 11

2.2.9 Naša tehnika pridobivanja spletnih podatkov 11

2.3 Semantični splet 12

2.3.1 Linked Data oblak 12

2.3.2 Spletni jezik ontologij (ang. OWL) 13

2.3.3 Semantika 14

2.3.4 Ontologije 14
2.3.5 Eksplicitni metapodatki 16

3 Pravni in varnostni vidiki pridobivanja spletnih podatkov 19
3.1 Pravni vidik pridobivanja spletnih podatkov 19
3.2 Varnost pridobivanja spletnih podatkov 20
3.2.1 HTML iframe atribut sandbox 21
3.2.2 Dodatne tehnike za blokiranje robotov 22

4 Pregled uporabljenih tehnologij 25
4.1 HtmlUnit 25
4.2 Greasemonkey 26
4.3 SPARQL 27
4.4 Arhitekturna rešitev Virtuoso 28
4.5 Komponenta Virtuoso Sponger 29
4.6 Kartuša Virtuoso Sponger 29
4.7 Hramba trojčkov (angl. triplestore) 31
4.8 Uporabljeni programski jeziki 33
4.8.1 PL/SQL 33
4.8.2 XPath 34
4.8.3 Java 35
4.8.4 PHP 35
4.8.5 JavaScript in jQuery 35

5 Pregled obstoječih rešitev 37
5.1 Semantic Fire 37
5.2 Visual Web Ripper 39
5.3 Fake 40
5.4 IRobotSoft 41
5.5 Selenium IDE 42
5.6 Apache Any23 43

6 Konceptualni predlog rešitve inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov 45
6.1 Hiter razvoj prepletene storitve aplikacije 46
6.1.1 Faza 1 - Zahteva po podatkih 47
6.1.2 Faza 2 - Priprava kartuš in meta-kartuš 47
6.1.3 Faza 3 - Analiza SPARQL poizvedbe 47
6.1.4 Faza 4 - Zahteva po RDF podatkih 48
6.1.5 Faza 5 - Preverjanje RDF podatkov 48
6.1.6 Faza 6 - Združitev v RDF graf 48
6.2 Komponenta, ki uporabniku pomaga pridobivati spletne podatke med brskanjem po spletni strani 48
6.2.1 Podrobna opredelitev implicitne gradnje ovojnice 50
6.3 Umestitev konceptualnega predloga rešitve znotraj komponente Virtuoso Sponger 55

7 Tehnične podrobnosti implementacije 57
7.1 Čelni del sistema - priprava ovojnice 59
7.1.1 Algoritmi za ustvarjanje generičnih XPath izrazov 60
7.1.2 Algoritem za samodejno predlaganje imena označenega podatka 66
7.1.3 Lokalni iskalnik 66
7.1.4 Seznam zadetkov 69
7.1.5 Podrobnosti posameznega zadetka 82
7.2 Zaledni del sistema 85
7.2.1 Skripta, ki shranjuje dele ovojnice na trdi disk 86
7.2.2 Kartuša, ki kliče namestniško javansko storitev 87
7.2.3 Namestniška javanska storitev 89

8 Vrednotenje predlagane metode in vizualizacija pridobljenih delno strukturiranih spletnih podatkov 93
8.1 Testna množica spletnih strani 94

8.2 Testiranje predlagane metode na podlagi podanih metrik 94
8.2.1 Dinamične spletne strani 94
8.2.2 Zahtevano tehnično znanje poslovnega uporabnika 97
8.2.3 Čas za pridobitev delno strukturiranih spletnih podatkov . 98
8.2.4 Pravilnost delovanja algoritma za iskanje ponovljenih vzorcev zadetkov 100
8.2.5 Zaščitenost spletnih strani 102
8.3 Vizualizacija pridobljenih podatkov 104

9 Sklepne ugotovitve **109**

Slike

2.1 Hierarhična razporeditev nekaterih tehnik pridobivanja spletnih podatkov glede na človeka in računalnik. 9
2.2 Linked Open Data oblak [36], avgust 2014. 13
2.3 Primer hierarhije pri ontologijah. 15

4.1 Umestitev komponente Sponger znotraj arhitekturne rešitve Virtuoso. 30

5.1 Označevanje ponavljajočih vzorcev zadetkov z orodjem Visual Web Ripper. 39
5.2 Primer snemanja akcij s pomočjo aplikacije Fake. 41
5.3 Primer snemanja akcij s pomočjo orodja Selenium IDE. 42

6.1 Hiter razvoj prepletene storitve aplikacije. 47
6.2 Priprava ovojnice za pretvorbo delno strukturiranih HTML podatkov v strukturirano obliko RDF podatkovnega modela. . 49
6.3 Koraki pri pripravi ovojnice. 50
6.4 Identifikacija ponavljajočih vzorcev zadetkov, številčenja strani in zagotovitev semantičnih informacij. 52
6.5 Identifikacija podatkov in samodejno priporočanje imen teh podatkov pri večih ponavljajočih vzorcih zadetkov na spletni strani. 53
6.6 Identifikacija podatkov in priporočanje imen teh podatkov pri podrobnostih posameznega zadetka. 54

6.7 Umestitev konceptualnega predloga rešitve znotraj komponente Virtuoso Sponger. 56

7.1 Prikaz arhitekture in delovanja inteligetnega pomočnika. . . . 58
7.2 Primer samodejnega predlaganja imena (Občina) pri izbiri podatka Ljutomer. 67
7.3 Koraki pri delovanju prototipa lokalnega iskalnika. 67
7.4 Koraki pri delovanju prototipa seznama zadetkov. 70
7.5 Predstavitev HTML strani v obliki HTML drevesa. 74
7.6 Z rdečo in turkizno zeleno označena enaka HTML poddrevesa. 74
7.7 Najdeni ponovljeni zadetki, ki imajo nastavljeno rumeno obrobo. 78
7.8 Najdeni in označeni elementi znotraj zadetkov, ki imajo nastavljeno rumeno ozadje. 80
7.9 Koraki pri gradnji ovojnice na strani posameznega zadetka. . . 84
7.10 Primer vhodnih in izhodnih podatkov namestniške javanske storitve. 91

8.1 Primer preproste spletne strani z izklopljenim izvajanjem JavaScripta na strani odjemalca. 96
8.2 Primer preproste spletne strani z vklopljenim izvajanjem JavaScripta na strani odjemalca. 96
8.3 Primer dveh prehodov algoritma čez spletno stran, vsak od njiju predstavlja poskus. 101
8.4 Primer, kako se nam zaradi izklopa skript znotraj HTML značke iframa preko AJAX klicev ne naložijo vrednosti opcijskega izbirnega polja pri regiji. 103
8.5 Cene stanovanj v občinah Lendava in Ljutomer, podatki pridobljeni iz spletne strani http://www.nepremicnine.net. . . 106
8.6 Velikost stanovanj v občinah Lendava in Ljutomer, podatki pridobljeni iz spletne strani http://www.nepremicnine.net. . . 106

8.7 Osnovni podatki občin Lendava in Ljutomer, podatki pridobljeni iz spletne strani http://www.stat.si. 107
8.8 Prikaz občin Lendava in Ljutomer na zemljevidu, podatki pridobljeni iz spletne strani http://wiki.dbpedia.org. 107

Tabele

3.1 Vrednosti atributa sanbox 22

8.1 Zahtevana tehnična znanja obstoječih pristopov 98
8.2 Časi pridobivanja delno strukturiranih spletnih podatkov 99
8.3 Število poskusov, da algoritem najde ponovljene vzorce zadetkov 100

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Program Interface	Aplikacijski uporabniški vmesnik
CSV	Comma Separated Values	Z vejico ločene vrednosti
DBMS	Database Management Systems	Sistemi za upravljanje podatkovnih baz
DNSRBL	Domain Name System Real-time Blackhole List	Črni seznam domenskega imenskega sistema v realnem času
HTML	HyperText Markup Language	Hipertekstovni označevalni jezik
JSON	JavaScript Object Notation	JavaScript objektna notacija
NoSQL	Not only Structured Query Language	Nerelacijski Povpraševalnik Jezik
NTLM	NT LAN Manager	NT upravljalec omrežne povezave
OWL	Web ontology language	Spletni jezik ontologij
RDF	Resource Data Framework	Ogrodje podatkovnih virov
TTL	Time to live	Življenjska doba

Povzetek

V pregledani literaturi nismo zasledili, da bi obstajal pristop, s pomočjo katerega bi lahko pretvorili podatke iz delno strukturiranih (spletne strani) ali nestrukturiranih spletnih virov v strukturirano RDF obliko in posledično integracijo v Linked Data oblak. Zato smo si postavili cilj izdelati inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov. Ta inteligentni pomočnik bi določene spletne podatke samodejno prepoznal in jih označil, nekatere dodatne spletne podatke pa bi lahko po potrebi označil poslovni uporabnik brez zahtevanega tehničnega znanja, in že bi imeli pripravljeno ovojnico za luščenje teh spletnih podatkov.

Implementirali smo prototip, ki s pomočjo posebnih algoritmov samodejno identificira glavni iskalni obrazec, ponovljene zadetke na spletni strani, podatke znotraj identificiranih zadetkov in podrobnosti posameznega zadetka, omogoča pa po potrebi dodatno označevanje podatkov in samodejno identificiranje imena podatka, prav tako pa uporabniku omogoča prikaz in izvoz teh podatkov v strukturirano RDF obliko. Inteligentni pomočnik omogoča tudi luščenje iz zelo dinamčnih spletnih strani (spletne strani, ki vsebujejo veliko JavaScripta in AJAX-a), kjer imajo podobni pristopi precej težav.

Delovanje inteligetnega pomočnika smo preverili tako, da smo poskusili izluščiti podatke iz čim več različnih spletnih strani, ki so zelo dinamične, statične, spletne strani zavarovane, proti luščenju ipd. Ugotovili smo, da ima pred ostalimi pristopi prednost v tem, da se na zelo dinamičnih straneh obnese precej bolje kot drugi in da omogoča eksplicitno pretvorbo podatkov v četrto oz. celo v peto stopnjo na petstopenjski Linked Data lestvici,

kjer drugi pristopi v večini primerov dosežejo le tretjo stopnjo. Prav tako pa s pomočjo predlaganega algoritma samodejno identificira ponavljajoče zadetke na spletni strani, kar je ena izmed funkcionalnosti, ki jo večina drugih pristopov ne ponuja.

Ključne besede: inteligentni pomočnik, delno strukturirani spletni podatki, RDF, Linked Data oblak, pridobivanje podatkov, spletne strani, spletni viri

Abstract

In the revised literature we have not identified any existing approach, which could convert data from semi-structured (websites) or unstructured web sources to the RDF form and consequently integrate into a Linked Data cloud. Therefore, our motivation and objective was to develop intelligent assistant for extracting semi-structured web data. This intelligent assistant should automatically identify and select part of web data, some of those web data should be selected by business user without any technical skills and we have automatically prepared wrapper for extracting these web data.

We implemented the prototype, which automatically identifies main search form, repeated results with specific algorithms on the website, identifies data inside these results and their details data. It also allows additional selecting data and automatically propose name of those data. With intelligent assistant we can also export data to the RDF form. Intelligent assistant allows us extracting data from very dynamic websites (websites with many lines of JavaScript and AJAX code), where similar approaches have many issues.

We have evaluated the functioning of intelligent assistant in such a way that we tried to extract web data from many different websites. As different websites we consider very dynamic, static and secured against extracting websites, etc. We have found out that our approach has advantages over others in extracting web data from very dynamic websites and it allows explicit conversion of web data in the forth or fifth level on five star Linked Data ranking, where others in most cases convert web data in third level only. Besides that it allows automatic identification of repeated results on

website with specific algorithm, which is one of the features of our approach and most of others do not offer this option.

Keywords: intelligent assistant, semi-structured web data, RDF, Linked Data cloud, extracting data, websites, web sources

Poglavje 1

Uvod

Svetovni splet, poznan tudi pod imenom World Wide Web, je dandanes nekaj vsakdanjega in je z vsakim dnem bolj razširjen in izpopolnjen. Predstavlja tako rekoč veliko množico med seboj povezanih dokumentov, ki so dostopni preko spleta. Ljudje moramo sami poiskati in povezati te medsebojne vire, ter si jih sami interpretirati. S tem namenom pa se v ospredje vse bolj postavlja semantični splet, ki poskuša izboljšati svetovni splet in se osredotoča na pretvorbo delno strukturiranih in nestrukturiranih dokumentov v podatkovno mrežo. Tako kot je svetovni splet velika množica porazdeljenih povezav, nam semantični splet zagotavlja porazdeljeno bazo znanja. Semantični splet temelji na meta podatkovnem modelu Resource Data Framework (RDF), prav tako pa omogoča serializirane formate, kot so trojčki, četvorčki, ipd.

Pogosto se zgodi, da želi poslovni uporabnik pretvoriti podatke iz delno strukturirane ali nestrukturirane oblike v strukturirano RDF obliko in jo posledično integrirati v Linked Data oblak. Vendar je pri tem zahtevana visoka raven tehničnega znanja, zato je bila naša glavna motivacija v okviru magistrskega dela razviti prototip inteligentnega pomočnika, ki omogoča poslovnemu uporabniku brez tehničnega znanja ustvariti ovojnico, na podlagi katere se potem podatki iz delno strukturiranih ali nestrukturiranih spletnih virov pretvorijo v strukturirano povezano RDF obliko.

1.1 Motivacija

Na področju pridobivanja podatkov iz delno strukturiranih (spletne strani) ali nestrukturiranih virov obstajajo že številni pristopi. Nekatere izmed teh obstoječih sorodnih pristopov smo pregledali in opisali v petem poglavju. Vendar pri teh pristopih naletimo na težavo, saj večinoma vsi ti pristopi omogočajo samo pretvorbo izvorne spletne strani v strukturirano obliko do tretje stopnje v okviru petstopnjske lestvice Linked Data, ki jo je opredelila organizacija W3C. V literaturi ni mogoče zaslediti pristopov, ki bi omogočali objavo podatkov v četrti (RDF) ali celo v peti stopnji (Linked RDF) z dodano semantiko. Torej ni samodejne povezave opredeljene sheme ovojnice z obstoječimi Linked Data shemami (5*).

Uporabljeni pristopi pa imajo tudi težave pri zajemanju dinamično generirane vsebine na strani odjemalca (AJAX, JavaScript). Velikokrat so AJAX klici prisotni na spletnih straneh in se tega niti ne zavedamo, ravno to nam pa povzroča težave pri zajemanju spletnih podatkov, ker se spletna stran neprestano spreminja.

Težave nam predstavlja tudi zahtevana visoka raven tehničnega znanja za poslovnega uporabnika pri gradnji ovojnice, na katero bomo aplicirali delno struktuririane podatke iz poljubne HTML strani v Linked Data oblak.

1.2 Cilji

Cilj magistrske naloge je aplicirati obstoječe znanstvene metode iz različnih interdisciplinarnih področij, povezanih z našo raziskavo. Največ poudarka pri implementaciji je bilo na pristopih za pridobivanje delno strukturiranih podatkov iz različnih virov, zato smo analizirali učinkovitost deleža uspešno samodejno identificiranih gradnikov na spletni strani in čas izvajanja ovojnice. V tem delu smo pričakovali, da bomo s pomočjo spletnih testov enot, dosegli višjo učinkovitost samodejne identifikacije gradnikov kot pri že obstoječih pristopih. Prav tako smo pri evalvaciji naše metode ovrednotili raven zahtevanega tehničnega znanja povprečnega spletnega uporabnika za gradnjo

ovojnice. Naš delno samodejni pristop pridobivanja delno strukturiranih podatkov iz različnih virov smo primerjali s pristopi samodejne gradnje ontologij, ki na splošnih domenah ne delujejo tako dobro, zato smo s tem prikazali dodano vrednost našega pristopa. Namen prototipa orodja je bil predvsem podpreti analize in hipoteze teoretičnega dela magistrske naloge.

Kot primer lahko podamo problemsko domeno načrtovanja potovanja, kjer moramo obiskati več spletnih strani, saj vseh željenih podatkov ne najdemo na enem mestu (npr. nakup letalske karte, ureditev prenočišča, rezerviranje lokalnega prevoza ipd.). V tem primeru bi nam pomagalo orodje, s katerim bi na izbranih spletnih straneh označili podatke, ki nas zanimajo (ustvarili bi ovojnico in pripadajočo ontologijo, povezano v Linked Data oblak), sistem pa bi nam kasneje omogočal samodejno pridobivanje podatkov iz več različnih izbranih spletnih strani s pomočjo ovojnic, ter nam na podlagi opredeljene ontologije omogočal tudi sklepanje o novih podatkih.

1.3 Struktura magistrske naloge

V 2. poglavju smo pregledali katere tehnike za pridobivanje spletnih podatkov obstajajo, prav tako pa tudi semantični splet in pojme povezane z njim. V poglavju 2.3 smo opisali in razložili pojem semantični splet in pojme Linked Data oblak, spletni jezik ontologij, semantika in eksplicitni metapodatki, ki jih lahko srečamo znotraj semantičnega spleta.

V 3. poglavju smo utemeljili pravne in varnostne vidike pridobivanja spletnih podatkov.

V nadaljevanju smo v 4. poglavju opisali tehnologije, ki smo jih uporabili pri razvoju prototipa orodja inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov. Podrobneje smo v poglavju 4.8 tudi opisali posamezne uporabljene programske jezike, ter razložili njihov namen pri našem razvoju.

Potem smo v 5. poglavju razširili pregled že obstoječih pristopov (Virtuoso Sponger [1], Semantic Fire [2], Visual Web Ripper [3], Fake [4], IRbobot-

Soft [5], Selenium IDE [6], Apache Any23 [7] idr.), ki smo jih preliminarno, za potrebe opredelitve raziskave že izvedli in se osredotočili na njihove pomanjkljivosti. Opisali smo tudi nekatere pojme, ki so povezani z pridobivanjem spletnih podatkov in s semantičnim spletom.

Po pregledanih že obstoječih pristopih smo v 6. poglavju predlagali svoj pristop pridobivanja delno strukturiranih podatkov iz spletnih strani, kjer smo prilagodili in integrirali obstoječe znanstvene metode iz različnih interdisciplinarnih področij, povezanih z našo raziskavo:

- (i) pristopi za pridobivanje delno strukturiranih podatkov iz različnih virov (npr. z uporabo XPath, regularnimi izrazi ipd.) [8],
- (ii) pristopi, ki omogočajo objavo podatkov v 4* (RDF) [9] ali celo 5* (Linked RDF) stopnji Linked Data,
- (iii) pristopi testiranja aplikacij programskega inženiringa [10] (angl. Software Engineering), kjer se uporabljajo spletni testi enot (angl. Web Unit Tests), za rešitev problema dinamično generirane vsebine na strani odjemalca.

Po uspešno zasnovanem konceptualnem predlogu rešitve smo v 7. poglavju še prikazali tehnične podrobnosti implementacije prototipa informacijske rešitve, s pomočjo katere smo tudi evalvirali naš pristop. Inteligentni pomočnik nam omogoča implicitno gradnjo ovojnice ter funkcionalnost komponente za integracijo podatkov iz različnih virov. Ovojnica [11] skrbi za samodejno pretvorbo delno strukturiranih HTML podatkov v strukturirano RDF obliko in posledično integracijo v Linked Data oblak ter za izvajanje obogatenih poizvedb nad izbranimi spletnimi stranmi in obstoječimi strukturiranimi viri podatkov. Temeljili smo na znanstvenih metodah za pridobivanje delno strukturiranih podatkov iz literature in jih prilagodili, ter aplicirali na problem objave ovojnice za poljubno HTML spletno stran v Linked Data oblak.

V 8. poglavju smo predlagano metodo ovrednotili na izbrani testni množici spletnih strani. Testiranje smo izvedli na podlagi podanih metrik, na koncu pa še pridobljene podatke vizualizirali in s tem pokazali dodano vrednost predlaganega pristopa. V 9. poglavju sledi zaključek s sklepnimi ugotovitvami.

Poglavje 2

Tehnike za pridobivanje spletnih podatkov in semantični splet

Za namene testiranja magistrskega dela smo razvili prototip orodja inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov, zato smo v tem poglavju pregledali določene pojme, ki so povezani z našo raziskavo in predlaganim pristopom.

Pregledali smo teoretično ozadje pridobivanja spletnih podatkov in semantičnega spleta. Našteli in opisali smo, katere tehnike se uporabljajo pri pridobivanju spletnih podatkov in podrobneje opisali semantični splet. Uvedli smo pojme, kot so Linked Data Oblak, jezik spletnih ontologij, semantika, ontologije in eksplicitni metapodatki, ki predstavljajo jedro semantičnega spleta.

2.1 Pridobivanje spletnih podatkov

V današnjem času so informacije ključne v poslovnem svetu. Veliko podjetij temelji na zbranih podatkih iz spleta, saj se ravno na podlagi teh lahko odločajo in sprejemajo odločitve.

Pridobivanje spletnih podatkov (angl. Web scraping) [33] je računalniška tehnika za pridobivanje podatkov iz spletnih virov oz. spletnih strani. Zelo pogosto gre za programe, ki simulirajo človeško raziskovanje svetovnega spleta preko HTTP protokola ali preko spletnega brskalnika kot je npr. Google Chrome ali Safari.

Pridobivanje spletnih podatkov se osredotoča na transformacijo delno strukturiranih ali nestrukturiranih podatkov na spletu, največkrat je to HTML ali XHTML format, v strukturirane podatke, katere lahko shranimo, analiziramo in vizualiziramo.

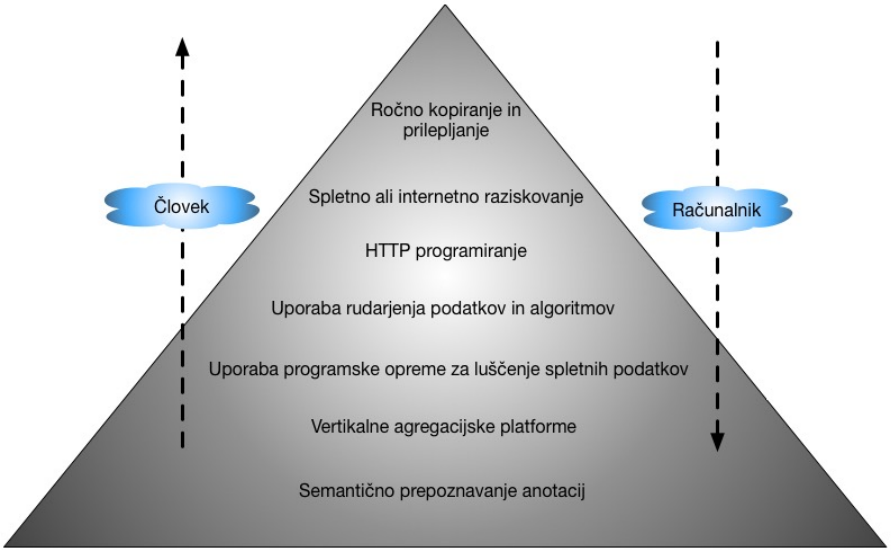
Programska orodja, ki se ukvarjajo s pridobivanjem spletnih podatkov, v veliki meri pridobivajo podatke iz delno strukturiranih spletnih virov. To pomeni, da imajo ti spletni viri neko določeno strukturo, ki jo določa standard, za katerim stoji organizacija W3C. Vendar pa nekateri razvijalci spletnih strani ne upoštevajo tega standarda, zato nekatere spletne strani nimajo pravilne strukture (sintaktično nepravilne spletne strani) in zato na teh spletnih straneh ta orodja za pridobivanje spletnih podatkov ne delujejo pravilno, kar predstavlja največji izziv pridobivanja spletnih podatkov.

2.2 Tehnike pridobivanja spletnih podatkov

Pridobivanje spletnih podatkov je še vedno področje zelo aktivnega razvoja s skupnim ciljem vizije semantičnega spleta, ki zahteva znanja s področja obdelovanja teksta, razumevanja semantike, umetne inteligence in interakcije človek-računalnik. Obstaja več različnih tehnik pridobivanja spletnih podatkov, katere smo opisali v tem poglavju, prav tako pa smo podrobneje predstavili tudi našo uporabljeno tehniko pridobivanja spletnih podatkov.

2.2.1 Ročno kopiranje in prilepljanje

Včasih se najbolje obnese ročna tehnika kopiraj-prilepi, pri kateri mora večino dela opraviti človek, saj se pridobivanje podatkov s pomočjo računalnikov v nekaterih primerih ne obnese najbolje. Tukaj lahko poljubno označimo po-



Slika 2.1: Hierarhična razporeditev nekaterih tehnik pridobivanja spletnih podatkov glede na človeka in računalnik.

datke, ki nas zanimajo, jih skopiramo in prilepimo na željeno mesto. Ročna tehnika kopiraj-prilepi je časovno potratna, saj se moramo, če želimo postopek ponoviti, vedno znova vračati na vir in ponavljati postopek kopiraj-prilepi.

2.2.2 Iskanje po besedilu in regularni izrazi

Pri tej tehniki gre za UNIX ukaz grep, ki omogoča pridobivanje podatkov (niza znakov), ki se ujemajo z določenim vzorcem. Ta vzorec, ki ga sestavimo je regularni izraz. Regularne izraze lahko uporabljamo tudi pri nekaterih drugih programskih jezikih, kot so npr. XPath, Java, ipd.

2.2.3 HTTP zahtevki

Podatke iz dinamičnih in statičnih spletnih strani lahko pridobimo s pomočjo GET in POST zahtevkov v različnih programskih jezikih, npr. Java, Python, ipd.

2.2.4 DOM razčlenjevanje

Z vključitvijo v celoti razvitega spletnega brskalnika, kot je npr. Internet Explorerjeva ali Mozillina kontrola, programi lahko pridobijo dinamično vsebino [35], katero so ustvarile skripte na strani odjemalca. Te kontrole spletnega brskalnika lahko razčlenijo spletne strani v DOM drevo, na podlagi katerega lahko potem programi pridobijo dele spletnih strani.

2.2.5 HTML razčlenjevalniki

Pri tehniki HTML razčlenjevalnikov se uporabljajo poizvedovalni jeziki, ki omogočajo poizvedovanje po delno strukturiranih podatkih. Taka jezika sta XQuery in XPath. V naši rešitvi eno od ključnih vlog pridobivanja delno strukturiranih spletnih podatkov predstavlja ravno poizvedovalni jezik XPath.

2.2.6 Vertikalne agregracijske platforme

Vertikalne agregacijske platforme so namenjene ustvarjanju in spremljanju botov, ki so namenjeni za specifično vertikalo. Pri tej tehniki se ustvari celotna podatkovna baza znanja določene vertikale, potem pa se samodejno ustvarjajo platforme. Pri teh platformah merimo kakovost informacij, ki so bile pridobljene.

2.2.7 Semantično prepoznavanje anotacij

Spletne strani, iz katerih pridobivamo podatke, lahko vsebujejo metapodatke in anotacije, katere so nam v pomoč, kadar želimo zajeti specifične dele podatkov na spletni strani. Te anotacije so lahko vključene v spletne strani in so vidne pri DOM razčlenjevanju.

Po drugi strani pa so lahko anotacije na posebni semantični plasti, ločeni od spletnih strani, zato programska orodja za pridobivanje podatkov lahko

pridobijo podatkovno shemo in navodila preden se lotijo pridobivanja podatkov iz spletnih strani.

2.2.8 Programska orodja, ki pridobivajo podatke s pomočjo računalniškega vida

Ta orodja uporabljajo pristope strojnega učenja in računalniškega vida, s pomočjo katerih poskušajo identificirati in pridobiti podatke iz spletne strani z enakim interpretiranjem, kot bi to naredil človek.

2.2.9 Naša tehnika pridobivanja spletnih podatkov

Pri izdelavi prototipa orodja inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov smo izbrali in združili več tehnik pridobivanja spletnih podatkov, saj smo tako dosegli večjo učinkovitost. Uporabili smo tehniko HTML razčlenjevalnikov, dopolnili pa smo jo z uporabo regularnih izrazov.

Večina spletnih strani, ki imajo dinamično generirano vsebino, pobirajo podatke iz nekega vira; največkrat je to iz programskega vmesnika, kateri servira podatke iz podatkovne baze. Ti podatki so potem prikazani na spletni strani s pomočjo predloge. Pri podatkovnem rudarjenju želimo z različnimi programskimi orodji prepoznati sestavo te predloge in glede na njeno sestavo pridobiti podatke iz nje. To je tehnika HTML razčlenjevalnikov, kjer smo v okviru skriptnega jezika JavaScript uporabili poizvedovalni jezik XPath (lahko bi tudi XQuery) za pridobivanje delno strukturiranih spletnih podatkov. Poizvedovalni jezik XPath smo dopolnili z regularnimi izrazi, saj nam ti omogočajo pridobivanje bolj zapletenih delno strukturiranih spletnih podatkov.

2.3 Semantični splet

Semantični splet [31] je razširitev svetovnega spleta, ki omogoča ljudem, da delijo vsebino zunaj meja aplikacij in spletnih strani. Lahko si ga predstavljamo kot učinkovito predstavitev podatkov na svetovnem spletu ali kot globalno povezano podatkovno bazo. Omogoča, da so podatki iz enega vira povezani z drugimi in da jih razumejo tudi računalniki, s tem pa lahko ti vse bolj opravljajo naloge v našem imenu. Semantični splet je še trenutno v povojih, njegova trenutna različica pa je Web 3.0.

Ključne tehnologije, ki jih zajema semantični splet, so eksplicitni metapodatki, ontologije, logika in sklepanje ter inteligentni agenti.

Semantični splet temelji na sintaksi, katera uporablja enolične identifikatorje virov (ang. krat. URI) za predstavitev podatkov, pogosto je to podatkovna struktura trojčki. Npr. neko določeno število trojčkov je lahko shranjeno v podatkovni bazi ali pa se uporabljajo za izmenjavo podatkov na svetovnem spletu s točno določeno sintakso za ta namen. Ta sintaksa se imenuje ogrodje podatkovnega vira (ang. krat. RDF).

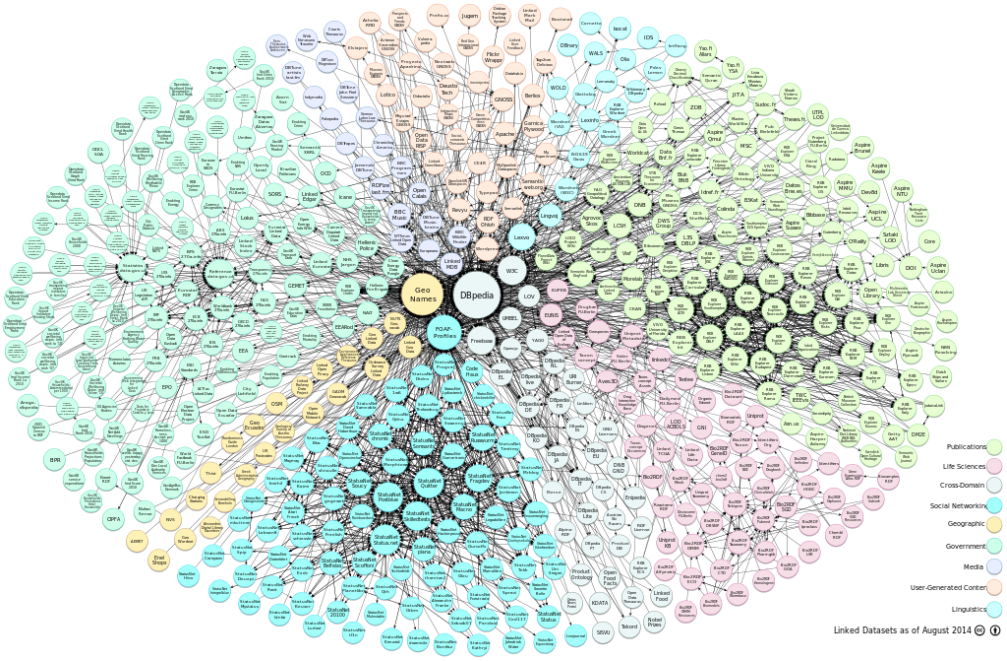
2.3.1 Linked Data oblak

Pojem Linked Data [34] se nanaša na svetovni splet, ki povezuje povezane podatke, kateri prej med sabo niso bili povezani. Lahko bi tudi rekli, da se nanaša na splet, kateri poskuša odstraniti meje za povezovanje podatkov med sabo. Posledično, ker so ti podatki strukturirani, so lahko med seboj povezani in postanejo zelo uporabni za semantične poizvedbe. Linked Data temelji na standardnih spletnih tehnologijah, kot so RDF, HTTP in URI, vendar imajo spletne strani v večini primerov strukturo, ki je primerna za človeško branje, zato te tehnologije prispevajo določen del podatkov (metapodatke) k tem spletnim stranem, da jih lahko prebirajo tudi računalniki.

Te povezane podatke lahko predstavimo v obliki oblačkov in povezav med njimi in ravno to potem predstavlja Linked Data oblak.

Oblak, ki je prikazan spodaj na Sliki 2.2, predstavlja Linked Open Data,

kar pomeni, da so ti podatki prosto dostopni za vsakogar - lahko se jih izkorišča in ponovno objavlja brez kakršnihkoli omejitev.



Slika 2.2: Linked Open Data oblak [36], avgust 2014.

2.3.2 Spletni jezik ontologij (ang. OWL)

RDF zagotavlja preprost model za podatke v grafu, vendar ne določa kompleksne semantike za razmerja in napredne podatkovne modele. V ta namen je bil uveden spletni jezik ontologij, ki je pravzaprav razširitev RDF podatkovnega modela, kateri zagotavlja zelo bogato semantiko za kompleksne podatkovne modele in logiko.

Spletni jezik ontologij zagotavlja objektno orientirano ogrodje, ki povezuje RDF trojčke v razrede, združenja in ostala kompleksna razmerja. Preprosto povedano, z njim lahko opisujemo ontologije na spletu.

Namen spletnega jezika ontologij je podoben kot pri RDF podatkovnem modelu; zagotoviti XML besednjak za definiranje razredov, lastnosti in nji-

hovih relacij, vendar v primerjavi z RDF podatkovnim modelom omogoča veliko bolj bogat nabor relacij, s čimer pa posledično zagotavlja tudi precej bolj napredno možnost sklepanja.

2.3.3 Semantika

Semantika je poddisciplina lingivstike, katera se osredotoča na študijo o pomenu. Zelo je povezana z drugo poddisciplino lingvistike, pragmatiko, katera ravnotako dokaj na široko govori o študiju pomena.

Semantika se posebej osredotoča na pomen v nekem jeziku. Lahko bi rekli, da se računalniki več ne ukvarjajo toliko s strukturo dokumenta, temveč se bolj osredotočajo na vsebino in na podlagi le-te pridobivajo podatke. Ravno ta vsebina pa predstavlja velik izziv za računalnike, zato so strokovnjaki s tega področja razvili dva pristopa za lažje delo:

- ontologije in
- eksplicitne metapodatke.

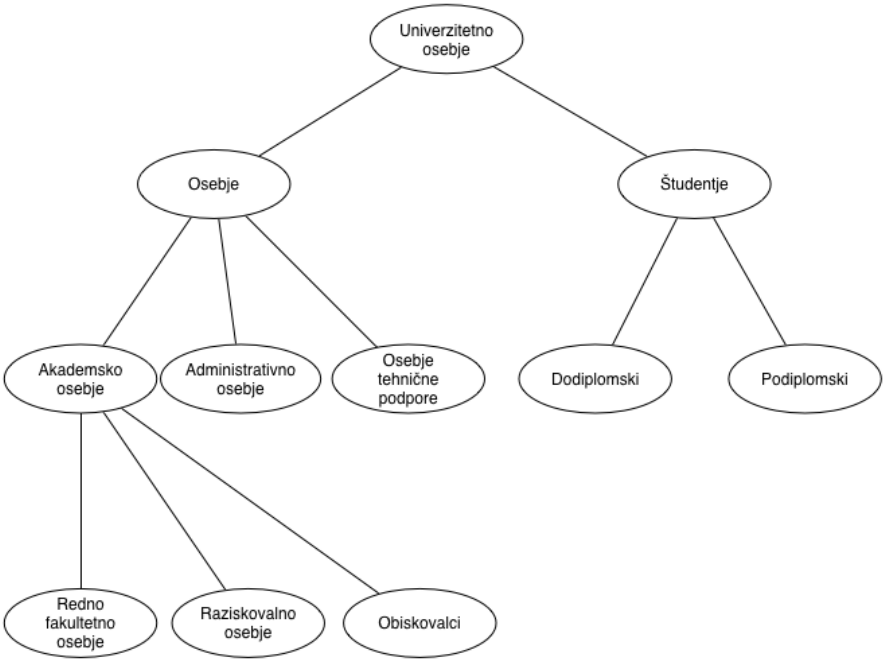
2.3.4 Ontologije

Beseda ontologija [30] prihaja iz filozofije in pomeni študijo nečesa kar res obstaja. Prav tako se je v zadnjih letih beseda ontologija prevzela v računalništvu in dodeljen ji je bil specifičen tehnični pomen, ki je dokaj različen od prvotnega.

Na področju umetne inteligence, semantičnega spleta, sistemskega in programskega inženiringa, biomedicinske informatike in informacijske arhitekture, ter še na nekaterih drugih področjih se ustvarjajo ontologije za omejevanje kompleksnosti in za organizacijo informacij. Ontologije se lahko uporabljajo tudi za reševanje problemov.

Tipično ontologija vsebuje končen seznam izrazov in relacij med temi izrazi. Izrazi označujejo pomembne koncepte (razredi objektov) v domeni. Kot primer lahko podamo: vsi znotraj univerze (člani osebja, študentje, predmeti, predavalnice in discipline) so pomembni koncepti. Ta razmerja so

sestavljena v obliki hierarhije razredov. Vsi na fakulteti so npr. člani osebja fakultete.



Slika 2.3: Primer hierarhije pri ontologijah.

Poleg relacij med razredi, lahko ontologije vključujejo informacije, kot so:

- lastnosti (X poučuje Y),
- omejitve vrednosti (en član fakultete lahko poučuje predmete),
- disjunktivne izjave in
- specifikacije logičnih izjav med objekti.

V kontekstu spleta, ontologije zagotavljajo skupno razumevanje domene. Ontologije so koristne za organizacijo in navigacijo spletnih strani, prav tako pa so koristne za izboljšanje natančnosti spletnega iskanja. Trenutno najbolj pomembni jeziki za ontologije so:

- XML, ki zagotavlja sintakso za strukturirane dokumente, vendar ne določa omejitev za te dokumente;

- XML Schema, ki se uporablja za omejitev strukture XML dokumentov;
- RDF, ki je podatkovni model za objekte in relacije med njimi; zagotavlja preprosto semantiko za ta podatkovni model;
- RDF Schema je besedno opisni jezik za opis lastnosti in razredov RDF virov s semantiko za posplošenje hierarhije lastnosti in razredov;
- OWL, ki je bogatejši opisni jezik za opisovanje entitet in razredov, kot so relacije med razredi, enakost, ipd.

2.3.5 Eksplicitni metapodatki

Večina spletnih strani, ki jih srečamo na svetovnem spletu, so formatirane v formatu, ki je primeren za človeško branje, ne pa v formatu, ki ga za branje uporabljajo računalniki. HTML je prevladujoči jezik, kateri se uporablja za spletne strani. Npr. del spletne strani izgleda sledeče:

```
<h2>Domaca fizioterapija</h2>
<p>Dobrodosli na spletni strani Domaca fizioterapija.
Ukvarjamo se s sportno masazo in se z mnogimi
drugimi aktivnostnmi. Za vas bo poskrbel doktor
Matej Serec in fizioterapevtki Maja Balazic in Jana
Gregorij.</p>
```

Iz zgornjega primera lahko vidimo, da so ti podatki človeku berljivi, vendar bi imeli računalniški stroji probleme z njihovim branjem. Z iskanjem po ključnih besedah bi npr. brez problema lahko našli besedo fizioterapevtki, vendar bi se težava pojavila pri ločevanju med doktorjem in fizioterapevti. Vemo, da ni namen semantičnega spleta, da bi poskušal ustvariti zelo inteligente agente, ki bi bili zmožni ločiti med slednjimi primeri.

Namesto tega se semantični splet zavzema, da bi bili podatki predstavljeni na način, da bi jih lahko prebirali ljudje, prav tako pa bi vsebovali še dodatne informacije o vsebini, da bi jih lahko brali tudi računalniki.

V našem primeru bi bile te informacije podane na naslednji način:

```
<podjetje>
  <ponudbaZdravljenja>Fizioterapija</ponudbaZdravljenja
     >
  <imePodjetja>Domaca fizioterapija</imePodjetja>
  <osebje>
    <doktor>Matej Serec</doktor>
    <fizioterapevt>Maja Balazic</fizioterapevt>
    <fizioterapevt>Jana Gregorij</fizioterapevt>
  </osebje>
</podjetje>
```

Slednjo predstavitev lahko računalniki precej lažje obdelajo. Tukaj ne bi bilo problema pri ločevanju med doktorjem in fizioterapevti. Termin metapodatki se navezuje na informacije kot so podatki o podatkih. Metapodatki zajemajo del pomena podatkov, podobno kot semantika pri semantičnem spletu.

Prvi uporabniki so sprejeli jezik HTML, ker je bil takrat sprejet kot standard in so kot prvi pričakovali prednosti te prilagoditve.

Podobno lahko danes spremljamo uveljavljanje XML jezika. To je eden ključnih korakov pri uresničevanju vizije semantičnega spleta. Še eno stopnjo naprednejši pa je RDF standard, kateri se uporablja izključno pri semantičnem spletu.

Poglavje 3

Pravni in varnostni vidiki pridobivanja spletnih podatkov

V prejšnjem poglavju smo si ogledali tehnike za pridobivanje spletnih podatkov in semantični splet ter pojme povezane z njim, v tem poglavju pa bomo dali poudarek na pravne in varnostne vidike teh spletnih podatkov. Sami vemo, da lastniki spletnih strani niso najbolj zadovoljni s tem, da se po njihovih straneh sprehajajo roboti, ki s pomočjo posebnih tehnik pridobivajo podatke iz le-teh. Podatke na njihovih spletnih straneh pred temi roboti delno varuje zakonodaja, nekatere bolj napredne spletne strani pa uporabljajo posebne tehnike za blokiranje oz. čim večje onemogočanje avtomatskega pridobivanja spletnih podatkov.

Med razvojem in testiranjem prototipa inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov smo se srečali z različnimi varnostnmi računalniškimi koncepti, ki so nam precej otežili delo.

3.1 Pravni vidik pridobivanja spletnih podatkov

Po pregledu zakonodaje smo ugotovili, da legalnost pridobivanja spletnih podatkov v Sloveniji ni natančno definirana. Situacija je podobna je tudi v

drugih sosednjih državah. Dostikrat se lastniki spletnih strani odločijo, da objavijo pogoje uporabe za svojo spletno stran in v teh pogojih navedejo, da ne dovolijo pridobivanja spletnih podatkov iz te strani. V Uradnem listu imamo v 221. členu *Napad na informacijski sistem*, navedeno naslednje:

1. Kdor neupravičeno vstopi ali vdre v informacijski sistem ali kdor neupravičeno prestreže podatek ob nejavnem prenosu v informacijski sistem ali iz njega, se kaznuje z zaporom do enega leta.
2. Kdor podatke v informacijskem sistemu neupravičeno uporabi, spremeni, preslika, prenaša, uniči ali v informacijski sistem neupravičeno vnese kakšen podatek, ovira prenos podatkov ali delovanje informacijskega sistema, se kaznuje z zaporom do dveh let.
3. Poskus dejanja iz prejšnjega odstavka je kazniv.
4. Če je z dejanjem iz drugega odstavka tega člena povzročena velika škoda, se storilec kaznuje z zaporom od treh mesecev do petih let.

3.2 Varnost pridobivanja spletnih podatkov

Naš primarni namen je bil, da bi željeno spletno stran, iz katere bomo pridobivali spletne podatke odprli znotraj HTML značke iframe. S tem smo želeli doseči to, da nam za gradnjo ovojnic več spletnih strani ne bi bilo potrebno konstantno zamenjevati naslova spletne strani znotraj brskalnika, da bi lahko znotraj spletne strani na našem strežniku, s pomočje HTML značke iframe, odpirali željene spletne strani. Tu se nam je pojavila težava, saj lahko do podatkov spletne strani, odprte znotraj značke HTML iframe, dostopamo le, če sta spletni strani na enaki domeni. To definira zelo pomemben koncept politike enakega izvora (ang. same-origin policy).

Politika enakega izvora je koncept v varnostnem modelu spletnih aplikacij. V skladu s to politiko, spletni brskalnik dovoljuje da skripte prve spletne strani dostopajo do podatkov druge spletne strani le, če imata obe spletni strani enak izvor. Izvor je definiran s kombinacijo URI sheme, z imenom gostitelja in številko vrat.

Problem koncepta politike enakega izvora smo rešili tako, da smo v našo rešitev vključili Firefoxov dodatek Greasemonkey [13], ki omogoča, da izvajamo JavaScript programsko kodo pri sebi, v brskalniku nad spletno stranjo iz druge domene in tako posledično lahko dostopamo do podatkov spletne strani iz druge domene.

Kasneje se je izkazalo, da so nekatere spletne strani zaščitene tako - ko jih odpremo znotraj HTML značke iframe - da nas takoj preusmerijo na drugo spletno stran. Tu smo si poskušali pomagati z HTML iframe atributom sandbox.

3.2.1 HTML iframe atribut sandbox

Atribut sandbox nam omogoča posebno množico omejitev za vsebino, prikazano znotraj značke iframe. Ko je atribut sandbox prisoten, nam omogoča naslednje:

- privzema vsebino, da je iz unikatnega izvora,
- blokira predlaganje obrazcev,
- blokira izvajanje skript,
- izključuje programske vmesnike,
- preprečuje povezave iz ostalih ciljnih brskalniških kontekstov,
- preprečuje vsebine pred uporabo vtičnikov (<embed>, <object>, <applet>, ali druge),
- preprečuje vsebini krmiljenje njene vrhnje brskalne vsebine,
- blokira samodejno sprožene funkcije (kot npr. samodejno predvajanje videa).

Vrednost atributa sandbox je lahko samo prazno polje, kar pomeni, da so vse omejitve prisotne. Lahko pa vsebuje tudi s presledkom ločene in naštete vrednosti, kjer te naštete vrednosti odstranijo te omejitve. Atribut sandbox je nov v HTML5 verziji. Njegova sintaksa je sledeča:

```
<iframe sandbox="vrednost/i">
```

Vrednost	Opis
(brez vrednosti)	Upoštevajo se vse omejitve
allow-forms	Ponovno omogočanje predložitve obrazcev
allow-pointer-lock	Ponovno omogočanje API-jev
allow-popups	Ponovno omogočanje pojavnih oken
allow-same-origin	Omogočanje vsebini iframea, da jo privzemamo kot, da je enakega izvora
allow-scripts	Ponovno omogočanje skript
allow-top-navigation	Omogočanje vsebini krmiljenje njene vrhnje brskalne vsebine

Tabela 3.1: Vrednosti atributa sanbox

Vrednosti atributa sandbox imamo podane v Tabeli 3.1.

3.2.2 Dodatne tehnike za blokiranje robotov

Nekateri lastniki spletnih strani uporabljajo dodatne tehnike za blokiranje robotov. Te so naštete in na kratko razložene spodaj:

- blokiranje IP naslova bodisi ročno ali na podlagi meril, kot sta geolokacija ali DNSRBL. Blokiranje IP naslova onemogoči tudi brskanje po spletni strani iz tega IP naslova;
- onemogočanje katerekoli spletne storitve programskega vmesnika, ki jo ponuja spletna stran;
- roboti se včasih sami izdajo kdo so, zato jih na podlagi tega lahko blokirajo (uporaba datoteke robots.txt). Tak primer razglasitve je 'googlebot'. Nekateri roboti pa so že tako napredni, da ne moremo ugotoviti, ali po spletni strani brska robot ali človek;
- uporaba CAPTCHE, ki je vrsta testa, kjer se na podlagi odgovora ugotovi ali je uporabnik človek;
- robote se lahko blokira tudi s pomočjo nadzorovanja prometa na spletni strani, saj ustvarijo veliko količino prometa na izbrani spletni strani.

Če presežejo dovoljeno mejo prometa na spletni strani, so blokirani;

- nekatera podjetja ponujajo anti-bot in storitve za spletne strani. Tudi nekateri boljši požarni zidovi dobro zaznavajo robote in jih blokirajo;
- s pomočjo uporabe tehnologije CSS sprite, kjer zmanjšamo število klicev na strežnik s tem, da več manjših slik ali ikon združimo v eno. Pridobljeno sliko nato s pomočjo CSS-ja razrežemo in razporedimo na željena mesta na spletni strani;
- detekcija robotov s pomočjo pasti, tako imenovanih honeypotov. Robotom podtaknemo povezave do spletnih strani, katerih naj ne bi obiskali (prepovedane spletne strani), a jih roboti vseeno obiščejo, zato se pridobi njihov IP naslov in se jih blokira.

Poglavje 4

Pregled uporabljenih tehnologij

Pri izdelavi prototipa inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov smo se dotaknili precejšnjega nabora različnih tehnologij. Glavni razlog je v tem, da smo samo pridobivanje golih delno strukturiranih spletnih podatkov izvedli v dveh fazah; v prvi fazi smo ustvarili ovojnico, v drugi pa smo pridobili podatke na podlagi ustrezno ustvarjene ovojnice, potem pa smo te podatke še prikazali. Izbira ustrezne tehnologije je odvisna od primernosti in zahtevnosti problema. V nadaljevanju smo na kratko opisali vsako od uporabljenih tehnologij, za lažjo predstavo delovanja prototipa inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov.

4.1 HtmlUnit

HtmlUnit [12] bi lahko z drugimi besedami poimenovali tudi grafični uporabniški vmesnik za programe, napisane v programskem jeziku Java. Njegova funkcija je modeliranje HTML dokumentov in zagotavljanje programskega vmesnika, ki omogoča sklicevanje na spletne strani, izpolnjevanje obrazcev, klikanje povezav ipd. To pomeni, da ponuja določen nabor akcij, kot jih lahko najdemo tudi v spletnem brskalniku.

Glavne lastnosti HtmlUnita so:

- podpora HTTP in HTTPS protokolom,
- podpora piškotom,
- podpora predložitve spletnega obrazca v obliki POST ali GET metode,
- podpora HTML odgovorom,
- podpora namestniškega strežnika,
- podpora za osnovno in NTLM avtentifikacijo,
- zmožnost prilagoditve zahtevanih glav, poslanih na strežnik,
- zmožnost specificiranja neuspešnih odgovorov strežnika, ki vrnejo izjeme.

Spletni brskalnik HtmlUnit poleg navedenega zagotavlja še izredno podporo JavaScriptu ter ima sposobnost simulacije spletnih brskalnikov (Google Chrome, Mozilla Firefox, Microsoft Internet Explorer), pri tem pa kot osnovni jezik uporablja Rhino (JavaScript pogon). Za HtmlUnit smo se odločili, ker podpira JavaScript, prav tako pa temelji na Javi in je posledično neodvisen od platforme. Ker podpira JavaScript, lahko tudi poljubno izključimo JavaScript na spletni strani, iz katere pridobivamo podatke. Je tudi najhitrejša verzija WebDriverja, ki je gonilnik za brskalnikovo domorodno podporo avtomatizacije.

4.2 Greasemonkey

Greasemonkey [13] je dodatek v spletnem brskalniku Mozilla Firefox, ki omogoča uporabnikom, da namestijo skripte za spreminjanje vsebine spletne strani, in to pred ali po začetku brskanja po sami spletni strani.

Spremembe spletne strani so vidne vsakič, ko je le-ta osvežena in so stalne za odjemalca, ki zaganja Greasemonkey skripto.

Greasemonkey se lahko uporablja za dodajanje novih funkcionalnosti spletni strani, kot je npr. dodajanje poslušalcev dogodkov, omogoča tudi prilagoditev izgleda spletne strani, kombiniranje podatkov iz več spletnih strani

ipd. S tehničnega vidika so uporabniške Greasemonkey skripte napisane v programskem jeziku JavaScript in manipulirajo z DOM vmesnikom. Pisanje tovrstnih skript je zato dokaj podobno pisanju JavaScripta za spletno stran, vendar s peščico dodatnih pravic, kot je npr. vrivanje kode preko XMLHttpRequest zahtevkov ipd.

Če želimo začeti uporabljati dodatek Greasemonkey, ga moramo namestiti v brskalnik Mozilla Firefox; leta 2010 ga je Google integriral v svoj brskalnik Google Chrome kot domorodno podporo za Greasemonkey skripte, ki jih znotraj spletnega brskalnika pretvori v dodatke. Izjema ni niti Microsoft Internet Explorer, za katerega je prav tako podprta podobna funkcionalnost, ki jo zagotavljajo IE7Pro [26], Sleipnir 2 [27] in iMacros [28]. Poleg vsega tega pa obstaja še dodatek GreaseKit [29], ki je podprt v spletnem brskalniku Safari, privzetem brskalniku operacijskega sistema Macintosh.

Ko zajemamo podatke iz spletne strani na drugi domeni, odpremo to spletno stran znotraj naše spletne strani (HTML značka iframe), uporabimo dodatek Greasemonkey (in Greasemonkey skripto), saj brez njega ne moremo spreminjati in označevati podatkov spletne strani iz druge domene. Zaščitenih spletnih strani ne odpiramo znotraj HTML značke iframe, ampak neposredno na njih poženemo samo našo Greasemonkey skripto in pripravimo ovojnico, ki je predpogoj za pridobivanje delno strukturiranih spletnih podatkov.

4.3 SPARQL

SPARQL je poizvedovalni jezik za RDF format, ki je označen in usmerjen graf za predstavitev podatkov v svetovnem spletu, iz tega pa se da izpeljati, da SPARQL omogoča poizvedovanje po podatkovnih bazah.

SPARQL na ta način dopušča iskanje sestavljenih trojčkov, konjunkcij, disjunkcij in opcijskih vzorcev. Poleg tega omogoča uporabnikom, da vnesejo zahtevke po podatkih, ki jih lahko imenujemo ključ-vrednost podatki, ki so podrobnejši podatki in sledijo specifikaciji RDF-W3C. Celotna baza lahko

temelji na ključ-vrednost podatkih, rezultat tega je pa množica tipa trojčkov predmet-predikat-objekt. To je analogno nekaterim NoSQL podatkovnim bazam v smislu pojma dokument-ključ-vrednost, kot je npr. MongoDB [25].

V naši magistrski nalogi smo za potrebe testiranja inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov uporabili Virtuoso SPARQL poizvedovalni servis arhitekturne rešitve Virtuoso. Ta servis omogoča poizvedovanje po trojčkih, ki so zapisani v podatkovni shrambi. Implementacija te storitve razširja standardni protokol z mnogimi izhodnimi formati skupaj s standardno XML in JSON serializacijo rezultatov.

4.4 Arhitekturna rešitev Virtuoso

Virtuoso [14] je edinstvena arhitekturna strežniška in hibridna rešitev, ki omogoča tradicionalno izrazito funkcionalnost strežnika znotraj samostojnega produkta. Lahko ga najdemo kot odprtokodno rešitev, prav tako pa tudi komercialno, plačniško verzijo.

Pokriva naslednja področja:

- SQL relacijske tabele v okviru obvladovanja podatkovnih baz (SQL RDBMS),
- RDF relacijske lastnosti grafov v okviru obvladovanja podatkovnih baz (RDF shramba trojčkov ali četvorčkov),
- vsebinski management (HTML, TEXT, TURTLE, RDF/XML, JSON, JSON-LD, XML),
- spletne in ostale datotečne storitve (spletni dokument ali datotečni strežnik),
- petzvezdična Linked Open data postavitev (RDF-baziran povezan podatkovni strežnik),
- spletni aplikacijski strežnik (SOAP ali RESTFUL iteracijski modeli).

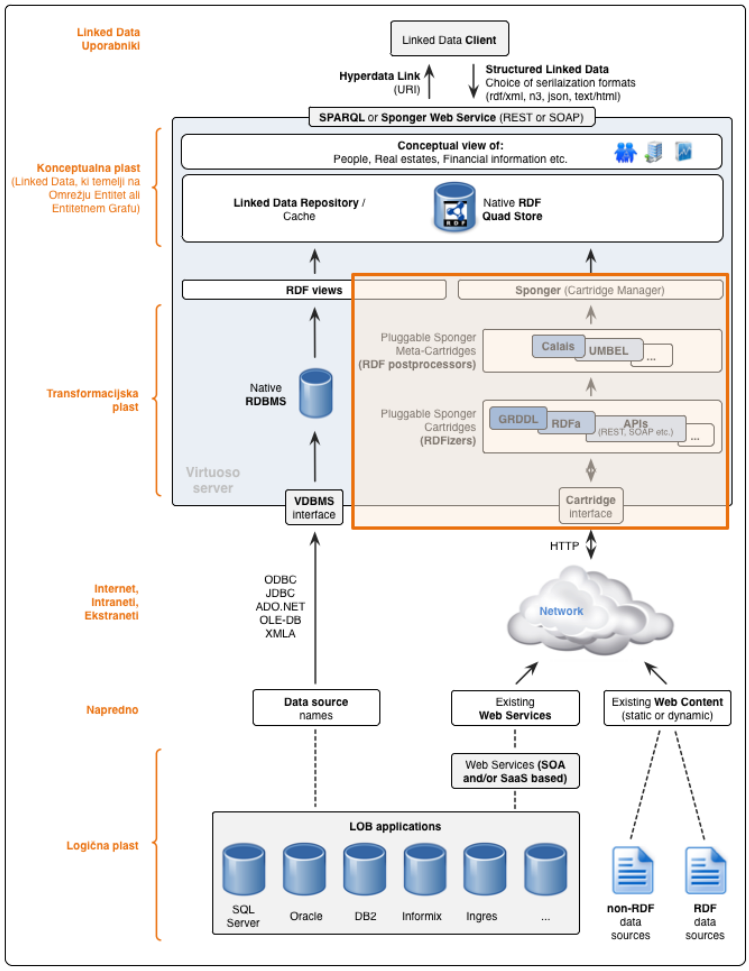
4.5 Komponenta Virtuoso Sponger

Komponenta Virtuoso sponger je vmesna komponenta arhitekturne rešitve Virtuoso. Ta generira povezane podatke iz različnih podatkovnih virov in podpira veliko raznih reprezentacij podatkov in serializiranih formatov. Orisana komponenta je transparentno integrirana v Virtuoso SPARQL Query Processor, kjer prinaša dereferenciranje znotraj SPARQL poizvedovalnih vzorcev čez neenake podatkovne prostore [32].

Komponenta Sponger omogoča poizvedovanje po nepovezanih podatkovnih spletnih virih in pretvorbo njihove vsebine v povezane podatke, izpostavlja pa, da podatki v kanonični obliki za poizvedovanje in sklepanje omogočajo hitro in preprosto gradnjo povezanih, podatkovno usmerjenih in prepletnih storitev. Funkcionalnost komponente Sponger je zagotovljena s kartušami. Vsaka izmed teh vsebuje luščilnike podatkov (ang. Data Extractors), katerih cilj je pridobivanje podatkov iz enega ali več virov, ki preslikajo podatke v eno ali več ontologij oz. shem, poleg tega pa se usmerjajo v produciranje RDF povezanih podatkov (ang. RDF Linked Data). Bistvo te komponente je uporaba ne-RDF spletnih podatkovnih virov (delno strukturirani ali nestrukturirani podatkovni viri) kot vhod, npr. samostojna (X)HTML spletna stran ali več (X)HTML spletnih strani, ki strežejo mikroformate in spletne storitve iz npr. Googla, Flickerja ipd., in generirajo RDF podatkovni model kot izhod. Slika 4.1 prikazuje umestitev komponente Sponger znotraj arhitekturne rešitve Virtuoso.

4.6 Kartuša Virtuoso Sponger

Komponenta Virtuoso sponger je sestavljena iz kartuš, ki so iz dveh delov, in sicer iz ekstraktorja entitete in ontološkega preslikovalnika. Entitete, pridobljene iz ne-RDF podatkovnih virov (delno strukturirani ali nestrukturirani podatkovni viri) se uporabljajo kot osnova za ustvarjanje strukturiranih podatkov s preslikavo v primerno ontologijo. Ekstrator entitete poskuša pridobiti podatke v RDF obliki. Če jih dobi določen Sponger, ta pošlje podatke



Slika 4.1: Umestitev komponente Sponger znotraj arhitekturne rešitve Virtuoso.

skozi entitetni ekstraktorski cevovod, pridobljeni podatki se zatem transformirajo v RDF obliko preko preslikovalnega cevovoda, RDF instanca podatkov pa je ustvarjena preko ujemanja ontologije in preslikovanja, ter se kot rezultat vrne odjemalcu.

Ontologija preslikovalnikov opravlja nalogo ustvarjanja instance RDF podatkov iz pridobljenih entitet (ne-RDF) z uporabo ontologije, povezane z določeno vrsto virov podatkov.

Vsako kartušo je potrebno pred uporabo registrirati v registru kartuš z

dodajanjem zapisa v tabelo DB.DBA.SYS_RDF_MAPPERS, registracija pa je relevantna tudi iz razloga, da jo prepozna SPARQL stroj (angl. SPARQL engine).

Za potrebe našega inteligentnega pomočnika smo razvili preprosto kartušo, katera ob klicu kliče zunanjo javansko storitev, ta pa s pomočjo ovojnice pridobi delno strukturirane podatke iz spletne strani.

4.7 Hramba trojčkov (angl. triplestore)

Hrambe tročkov [37] so sistemi za upravljanje baz podatkov, oblikovanih v skladu z RDF podatkovnim modelom. Nasprotno kot relacijski podatkovni modeli, ki shranjujejo podatke v tabelah in uporabljajajo poizvedovalni jezik SQL, hrambe trojčkov shranjujejo RDF trojčke in uporabljajo poizvedovalni jezik SPARQL. Ključna funkcija večine hramb trojčkov je zmožnost sklepanja. Pomembno je omeniti, da sistemi za upravljanje podatkovnih baz (ang. krat. DBMS) tipično ponujajo sposobnost vzporednosti, varnosti, logiranja, obnovitve in nadgrajevanja zraven nalaganja ter shranjevanje shranjenih podatkov. Hrambo trojčkov lahko razdelimo tipično v tri kategorije:

- domorodni pomnilnik trojčkov,
- RDBMS-zaledni pomnilnik trojčkov,
- NoSQL pomnilnik trojčkov.

Domorodni pomnilniki trojčkov so tisti, ki so implementirani od začetka, in izkoriščjo RDF podatkovni model za učinkovito shranjevanje in dostopanje do RDF podatkov. Primer takega domorodnega pomnilnika trojčkov je AllegroGraph [23].

RDBMS-zaledni pomnilniki trojčkov so zgrajeni z dodajanjem specifične RDF plasti na že obstoječi sistem za upravljanje relacijskih podatkovnih baz (ang. krat. RDBMS). Primer takega RDBMS zalednega pomnilnika trojčkov je Apache Jena SDB [22].

NoSQL pomnilniki trojčkov so pred kratim bili naznanjeni kot možni upravljalci za RDF. Npr. CumulusRDF [15], ki temelji na vrhnjem delu porazdeljenega odprtokodnega podatkovnega sistema Cassandra [24].

Če povzamemo, je glavna funkcija, ki jo ponujajo pomnilniki trojčkov, sklepanje (ang. inferencing) za poizvedbe. Primer podane ontologije spletnega ontološkega jezika (ang. krat. OWL):

```
ex1:FullProfessor  rdf:subClassOf  ex1:Professor.
ex1:AssistantProfessor  rdf:subClassOf  ex1:Professor.
ex1:Professor  owl:equivalentClass  ex2:Teacher
```

Primer navaja, da sta razreda FullProfessor in AssitantProfessor podrazreda razreda Professor. Potem imamo še eno ontologijo, ki ima razred Teacher, kateri je enak razredu Professor. Upoštevamo naslednjo ontologijo:

```
ex1:Bob  rdf:type  ex1:FullProfessor .
ex1:Alice  rdf:type  ex1:AssistantProfessor .
ex2:Mary  rdf:type  ex2:Teacher
```

Ta navajaja, da je Bob tipa FullProfessor, Alice AssistantProfessor in Mary Teacher.

Z naslednjo SPARQL poizvedbo bi dobili prazno poizvedbo, če ne bi bilo vključenega sklepanja:

```
SELECT ?x WHERE {
   ?x rdf:type ex1:Professor
}
```

Vendar je mogoče sklepati, da ex1:Bob, ex1:Alice in ex2:Mary vsi spadajo pod razred Professor, saj sta razreda FullProfesor in AssistantProfessor podrazreda razreda Professor in je razred Teacher enak razredu Professor.

4.8 Uporabljeni programski jeziki

V tem poglavju smo našteli in na kratko opisali programske jezike, prav tako pa tudi utemljili, s katerim namenom smo vsakega od njih uporabili. Predlagana in razvita rešitev je precej obsežna, zato smo se dotaknili kar precej različnih programskih jezikov.

4.8.1 PL/SQL

PL/SQL je kombinacija SQL programskega jezika in proceduralnih funkcij, ki razširjajo jezik SQL. Virtuoso Sponger uporablja PL/SQL kot primarni jezik, v sklopu katerega lahko razvijamo kartuše, katere omogočajo pridobivanje spletnih podatkov. Primer podane preproste kartuše, s katero kličemo javansko storitev in podatke pridobljene s pomočjo javanske storitve, shranimo v graf:

```
create procedure DB.DBA.RDF_PROXY_CLASS (in graph_iri
   varchar, in new_origin_uri varchar,  in dest varchar
   ,    inout _ret_body any, inout aq any, inout ps any
   , inout _key any, inout opts any)
{
  declare ret nvarchar;
  java_load_class ('Form', file_to_string ('Form.class
     '));
  java_load_class ('ResultList', file_to_string ('
     ResultList.class'));
  java_load_class ('ResultDetails', file_to_string ('
     ResultDetails.class'));
  java_load_class ('Proxy', file_to_string ('Proxy.
     class'));
  ret := java_call_method ( 'Proxy', NULL, 'parse', '
     Ljava/lang/String;');
  DB.DBA.TTLP (cast (ret as varchar), '', 'http://
     localhost:8899/DAV/customwebsites', 32);

  return 1;
}
```

4.8.2 XPath

XPath je poizvedovalni jezik, katerega glavni namen je naslavljanje vozlišč v XML dokumentu. Zraven naslavljanja vozlišč zagotavlja tudi osnovne storitve uporabe nizov, števil in logičnih podatkovnih tipov. Vsebuje čez 100 vgrajenih funkcij, katere so nam v pomoč pri naslavljanju vozlišč.

Primarni sintaktični konstrukt v XPathu je izraz. XPath je tudi glavni element v XSLT standardu. Brez njegovega znanja ne moremo ustvariti XSLT dokumentov. Uporabimo ga lahko v kombinaciji z drugimi program-

skimi jeziki, npr. Java, Ruby, Python, ipd. Mi smo ga uporabili v kombinaciji z ogrodjem za testiranje aplikacij HtmlUnitom, kjer se je zelo dobro izkazal.

4.8.3 Java

Java, ki je objektno orientiran programski jezik in je neodvisen od računalniškega okolja, omogoča večnitno izvajanje in je oblikovan za ustvarjanje programov in aplikacij za internet in intranet.

Ker Virtuoso pri programiranju kartuš omogoča klice namestniških storitev napisanih v programskih jezikih Java, C++ in Python, smo se odločili, da bomo klicali namestniško javasko storitev, saj ravno Java omogoča integracijo HtmlUnita, katerega uporabljamo v naši javanski storitvi.

4.8.4 PHP

PHP je skriptni programski jezik na strani strežnika, ki je bil razvit za spletne aplikacije, vendar ga lahko uporabljamo tudi za splošne namene.

V naši aplikaciji smo ga uporabili z namenom, da sprejema AJAX zahtevke Greasmonkey skripte in shranjuje podatke v obliki JSON formata na trdi disk. PHP skripta, ki smo jo pripravili za potrebe našega prototipa, deluje kot nek posrednik med odjemalčevo Greasemonkey skripto in trdim diskom, kamor shranimo ustvarjeno ovojnico izbranega spletnega vira.

4.8.5 JavaScript in jQuery

JavaScript je najbolj razširjen programski jezik v spletnih brskalnikih, katerega implementacija omogoča odjemalčeve skripte za interakcijo z uporabnikom, kontroliranjem brskalnika in spreminjanje vsebine. S tem namenom smo v naši aplikaciji uporabili JavaScript, saj smo potrebovali nek programski jezik, s katerim lahko manipuliramo in spreminjamo vsebino. Pomagali smo si z JavaScript knjižnico jQuery, ki precej olajša programiranje, saj vsebuje že mnogo preddefiniranih funkcij.

Skriptni jezik JavaScript in knjižnica jQuery sta torej ključnega pomena pri našem prototipu, saj ravno z njuno pomočjo lahko ustvarimo ovojnico, ki je predpogoj za kasnejše pridobivanje podatkov iz različnih spletnih virov.

Poglavje 5

Pregled obstoječih rešitev

Za pomoč pri pripravi konceptualnega predloga rešitve prototipa inteligentnega pomočnika smo pregledali že obstoječe rešitve za pridobivanje delno stukturiranih spletnih podatkov.

Vsaka od teh rešitev ima določene prednosti in slabosti. Pri načrtovanju in implementaciji našega prototipa smo med pregledom obstoječih rešitev analizirali prednosti, ki jih vsaka od teh rešitev ponuja, ter tudi te vpeljali v naš prototip. Omenili smo tudi negativne lastnosti vsake obstoječe rešitve, katere smo poskušali izboljšati oz. jih implementirati od začetka.

5.1 Semantic Fire

Orodje [2], ki nam omogoča, da pridobivamo delno strukturirane podatke iz spletnih strani v obliki RDF podatkovnega modela; te podatke lahko kasneje tudi shranimo v podatkovno shrambo v obliki trojčkov. To nam omogoča, da lahko kasneje poizvedujemo po teh trojčkih. Ti podatki so lahko povezani z drugimi (podatki), kot je npr. DBPedia [16], zato lahko pridobimo še bolj podrobne podatke od vsakega našega podatka.

Predpogoj uporabe orodja Semantic Fire je nastavitev Python okolja in prav tako nekaterih potrebnih Python knjižnic (python-yaml, python-lxml). V primeru uporabe orodja Semantic Fire ustvarimo yaml datoteko, ki nam

omogoča preslikovanje vsebine spletne strani v semantične spletne tehnologije. Yaml datoteka vsebuje ključne tipe elementov, podatkovne postavke in podatkovne vreče. Primer podatkovne postavke je naslov ali datum, medtem ko je podatkovna vreča ponavljajoča struktura, kjer je vsak element člen s HTML značkami (npr. div, li, tr). Spodaj si lahko ogledamo primer yaml datoteke:

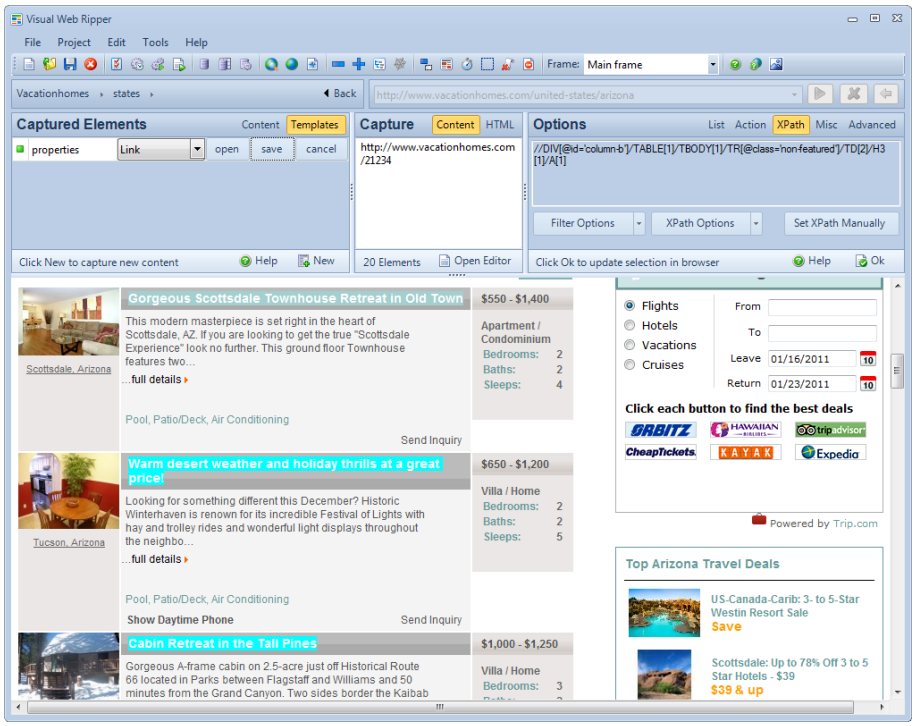
```
url: http://digg.com/
require_id: Yes
dataitems:
  - predicate: dc:title
    xpath: id('title')/a
  - predicate: dc:creator
    xpath: //a[@rel='dc:creator']
  - predicate: foaf:primaryTopic
    xpath: id('title')/a[@href]/@href
databags:
  - predicate: sioc:has_container
    xpath: //*[@id="p-main"]
    databag_class: sioc:Thread
    subpredicate: sioc:container_of
    datarow_class: sioc:Post
    p-list: ['dc:creator', 'dc:created', 'sioc:content
       ', 'rev:rating']
    skip_end_rows: 2
    row_separator: li
```

Iz zgornje yaml datoteke lahko razberemo, da podatke na spletni strani iščemo s pomočjo poizvedovalnega jezika XPath. Posebej pomembna za prototip našega inteligentnega pomočnika je bila ideja, na kateri temelji orodje Semantic Fire, saj pri njem podobno kot pri naši rešitvi, pridobivamo delno strukturirane podatke iz spletnih strani s pomočjo poizvedovalnega jezika XPath in jih vrnemo v obliki RDF trojčkov.

5.2 Visual Web Ripper

Visual Web Ripper [3] je orodje, ki nam omogoča na enostaven način pridobivati delno strukturirane spletne podatke, kot so katologi produktov, oglasi, spletne strani finančnih institucij ali iz katerih drugih spletnih strani, ki vsebujejo nam zanimive podatke.

Visual Web Ripper pobira vsebino iz ciljnih spletnih strani avtomatično in zagotavlja podatke v obliki strukturiranih podatkov, kot so podatkovne baze, razpredelnice, CSV ali XML datoteke. Ne omogoča pa neposrednega pridobivanja podatkov v obliki RDF podatkovnega modela npr. trojčkov, kar pa je bil eden izmed ciljev pri izdelavi našega prototipa inteligentnega pomočnika.



Slika 5.1: Označevanje ponavljajočih vzorcev zadetkov z orodjem Visual Web Ripper.

Orodje Visual Web Ripper deluje kot samostojna aplikacija, ni spletna

aplikacija kot večina drugih, kar pa nam lahko povzroča težave, saj je razvita samo za operacijski sistem Windows. Tudi licenca je plačljiva, zato smo precej omejeni z njegovo uporabo.

Ena izmed ključnih prednosti, ki jo ponuja orodje Visual Web Ripper je ta, da lahko delno strukturirane spletne podatke pridobivamo iz visoko dinamičnih spletnih strani (JavaScript, AJAX), kjer ima večina drugih orodij precej težav. Ena izmed njegovih funkcionalnosti je, da ponavljajoče potrjuje spletne obrazce z vsemi možnimi vnosnimi vrednostmi.

Če imamo znanje s področja s programiranja .NET programskega ogrodja, lahko povežemo orodje Visual Web Ripper neposredno z našo aplikacijo. Programski vmesnik orodja Visual Web Ripper omogočna popoln dostop do vsake samostojne funkcionalnosti, zato lahko ustvarimo in urejamo projekte ali pa zaženemo projekt in dostopamo do zbranih podatkov neposredno, še preden so shranjeni v zunanji podatkovni vir.

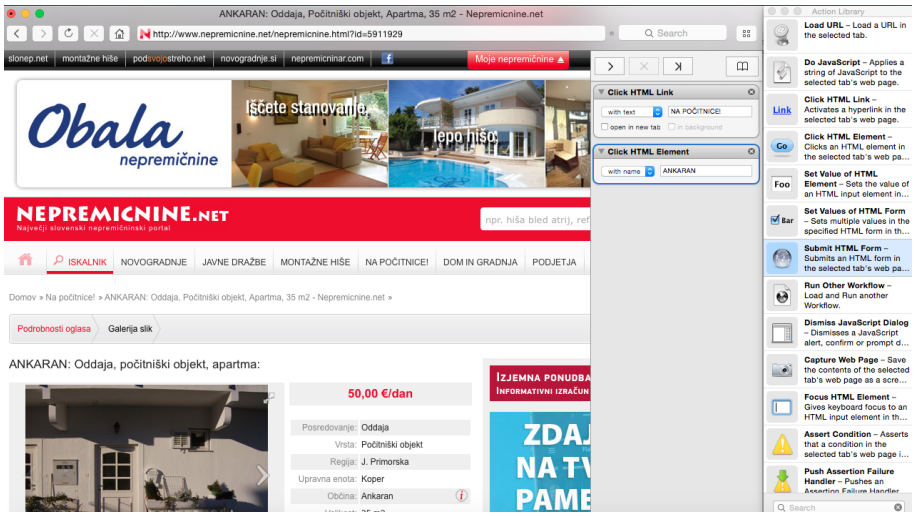
5.3 Fake

Orodje Fake [4] je samostojna aplikacija, katera je trenutno razvita samo za operacijski sistem Macintosh in nam omogoča zelo preprosto spletno avtomatizacijo. Omogoča, da izvajamo akcije brskalnika v grafičnem poteku delovanja - te lahko poženemo ponovno brez človeške interakcije. Ta potek lahko shranimo, ponovno odpremo in tudi delimo z drugimi.

Vse funkcionalnosti aplikacije Fake v ozadju uporabljajo Macintoshov skripti jezik AppleScript. Posledično lahko Fake uporabimo za vključitev v spletne avtomatizacije v veliko ostalih Macintoshovih skriptnih opravil.

Razvijalci lahko uporabljajo Fake za grafično konfigurabilne avtomatske teste za spletne aplikacije, vključujoč trditve, opravilne trditve okvar in opravilnike napak. Prav tako pa lahko navadni uporabniki uporabljajo Fake za avtomatska spletna opravila, kot so npr. izpolnjevanje dolgih obrazcev, zajemanje slik zaslona ipd.

Z analiziranjem delovanja orodja Fake smo si pomagali pri naši rešitvi



Slika 5.2: Primer snemanja akcij s pomočjo aplikacije Fake.

tako, da smo si pogledali, kako Fake ustvari skripto in posnema akcije. To skripto smo preučili z primi in potegni (ang. drag and drop) komponentami, ter si tako pomagali pri zasnovi naše ovojnice.

5.4 IRobotSoft

IRobotSoft [5] je programsko orodje za inteligentno spletno avtomatizacijo. Z njim lahko na preprost način ustvarimo robote in jih naučimo, da avtomatično opravljajajo naše dnevne aktivnosti. Ti roboti lahko klikajo na povezave, potrjujejo obrazce, se povezujejo na podatkovne baze, zaganjajo prilagojeno programsko kodo in analizirajo podatke.

S programsko opremo IRobotSoft lahko na preprost način izvozimo podatke v CSV datoteko, XML datoteko ali relacijske podatkovne baze z le nekaj preprostimi kliki.

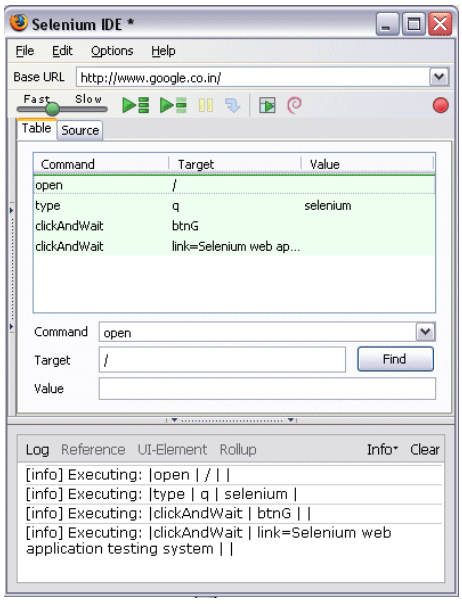
Če smo ekspert na področju obdelovanja podatkov, nam lahko IRobotSoft pride zelo prav. Z zelo močnim podatkovnim pogonom lahko na preprost način pridobimo precej zapletene delno strukturirane spletne podatke. Zagotavlja nam namestnike, paralelno procesiranje, podatkovne baze v po-

mnilniku, razporejanje in klice opravil. Omogoča nam tudi programski jezik za opravljanje kompleksne podatkovne logike.

Tudi testerji imajo možnost ustvarjanja robotov za testne akcije. Robote lahko prenesemo na drugo napravo s kopiranjem programske opreme IRobotSoft in robota. Lahko jih zaganjamo in kontroliramo na oddaljen način, s preprostimi Pythonovimi uporabniškimi vmesniki.

5.5 Selenium IDE

Selenium IDE [6] je integracijsko razvijalsko okolje za zagon Selenium skript. Implementirano je kot Mozilla Firefox dodatek (ang. add-on) in omogoča snemanje, spreminjanje in razhroščevanje testov. Selenium IDE vključuje celoten Selenium Core, ki omogoča preprosto, hitro snemanje in zaganjanje testov v trenutnem okolju.



Slika 5.3: Primer snemanja akcij s pomočjo orodja Selenium IDE.

Selenium IDE ni samo orodje za snemanje, ampak omogoča mnogo več, npr. ročno spreminjanje skript ipd. Je idealno orodje za ustvarjanje Sele-

nium testov, saj omogoča samodejno dopolnjevanje (ang. autocomplete) in še nekatere druge ukaze. Značilnosti:

- preprosto snemanje in predvajanje,
- pametna izbira polja, ki uporablja ID izbirnike, imena ali XPath izraze,
- samodokončanje za vse običajne Selenium ukaze,
- sprehod skozi teste,
- razhroščevanje in postavljanje prelomnih točk,
- shranjevanje testov kot HTML in Ruby skript ali kateri drugi format,
- podpora za Selenium user-extension.js datoteko,
- možnost vstavljanja naslova vsake strani,
- enostavna prilagoditev preko vtičnikov.

5.6 Apache Any23

Apache Any23 [7] oz. Anything To Triples je knjižnica, spletna storitev in ukazna vrstica, ki omogoča pridobivanje delno strukturiranih spletnih podatkov iz različnih spletnih dokumentov, kot rezultat pa jih vrne v obliki RDF podatkovnega modela. Trenutno podpira naslednje vhodne formate:

- RDF/XML, Trojčki, Notacija 3,
- RDFa z RDFa1.1 predponskim mehanizmom,
- mikroformati: Adr, Geo, hCalendar, hCard, hListing, hRecipe, hReview, License, XFN in Species,
- HTML5 Mikropodatki: Schema.org,
- JSON-LD: JSON za Linking Data, preprosti Linked Data format, ki temelji na že uspešnem JSON formatu in zagotavlja JSON podatkom interoperabilnost pri spletnem skaliranju,
- CSV: z vejico ločene vrednosti,
- besedni zaklad: podpora luščenja za CSV, Dublin Core Terms, opis kariere, opis projekta, prjatelj od prijatelja, GEO imena, CAL, lkif-core, Open Graph Protocol, BBC Programmes Ontology, RDF Review Vocabulary, schema.org, VCard, BBC Wildlife Ontology in XHTML.

Kot smo že omenili, lahko Any23 uporabimo na tri različne načine, in sicer kot:

- knjižnico v Javanskih aplikacijah, ki pridobiva delno strukturirane podatke iz spleta;
- ukazno vrstico za pridobivanje in pretvarjanje podatkov med podprtimi formati;
- aplikacijsko spletno storitev dostopno na http://any23.org.

Poglavje 6

Konceptualni predlog rešitve inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov

Obstoječi pristopi s področja pridobivanja delno strukturiranih spletnih podatkov vsebujejo nekatere pomanjkljivosti, zato nam je bila glavna motivacija to, da smo predlagali konceptualni predlog rešitve, katera bi odpravila te pomanjkljivosti. V predlaganem konceptualnem predlogu rešitve smo združili prednosti posameznega pristopa, ravno tako pa smo odpravili tudi pomanjkljivosti posameznega pristopa. Glavne pomanjkljivosti obstoječih pristopov, ki so nas motivirale za delo, so bile:

- Obstoječi pristopi v večini omogočajo pretvorbo izvorne spletne strani v strukturirano obliko do tretje stopnje v okviru pet stopenjske lestvice Linked Data, ki jo je opredelila organizacija W3C. Pristopov, ki bi omogočali objavo podatkov v četrti (RDF) ali celo peti (Linked RDF) obliki, z dodano semantiko, v literaturi ni mogoče zaslediti;

- nekateri od pristopov zahtevajo tehnično znanje, npr. s področja programiranja, da lahko pridobivamo podatke iz delno strukturiranih virov;
- nekateri pristopi imajo probleme z pridobivanjem podatkov iz zelo dinamičnih spletnih strani (JavaScript, AJAX).

Naš konceptualni predlog rešitve inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov odpravlja vse tri glavne pomanjkljivosti, ki smo jih navedli. S predlogom konceptualnega predloga rešitve smo želeli prikazati kako lahko apliciramo teoretično znanje s področja pridobivanja delno strukturiranih spletnih podatkov na praktični domeni.

Opisali smo dva dela načrtovanja razvoja:

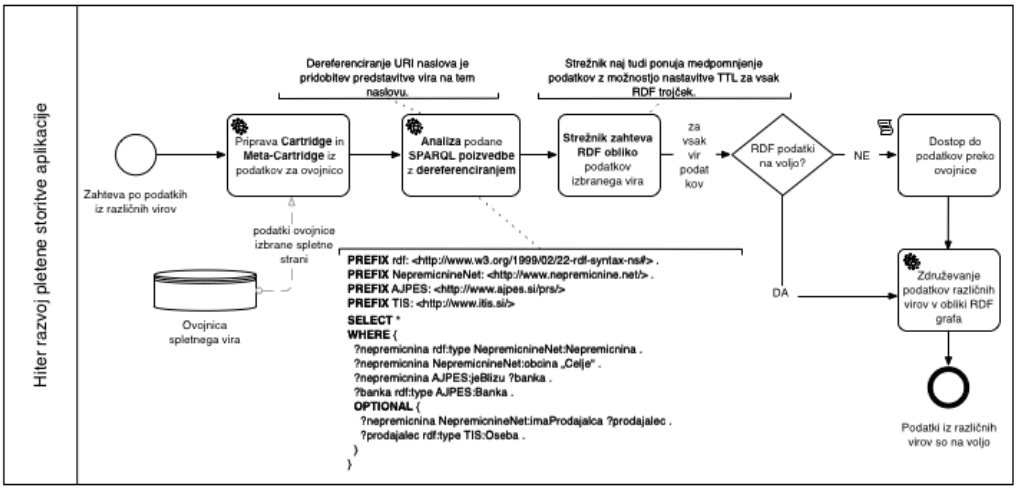
- hiter razvoj prepletene storitve inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov,
- komponento, ki uporabniku pomaga pridobivati spletne podatke med brskanjem po spletni strani.

Na koncu smo predstavili še, kako smo umestili konceptualni predlog rešitve znotraj arhitekturne rešitve Virtuoso.

6.1 Hiter razvoj prepletene storitve aplikacije

V prvem delu načrtovanja razvoja smo podrobneje opisali hiter razvoj prepletene storitve aplikacije. S tem smo pokazali sestavo konceptualnega modela prototipa naše rešitve.

Pridobivanje delno strukturiranih spletnih podatkov ni preprost postopek in pogosto ga moramo izvesti v več fazah. To smo predstavili v šestih fazah, pri vsaki pa podrobneje opisali, kaj se dogaja. Na Sliki 6.1 lahko vidimo hiter razvoj prepletene storitve aplikacije.



Slika 6.1: Hiter razvoj prepletene storitve aplikacije.

6.1.1 Faza 1 - Zahteva po podatkih

V prvi fazi dobimo od navadnega poslovnega uporabnika zahtevo po podatkih iz več različnih virov. To pomeni, da npr. želi poslovni uporabnik pridobiti podatke iz več spletnih virov. Tu se srečamo s pojmom semantičnega spleta, kjer se podatki med seboj povezujejo in posledično se gradi velika podatkovna baza.

6.1.2 Faza 2 - Priprava kartuš in meta-kartuš

V drugi fazi pripravimo kartuše (ang. Cartridge) in meta-kartuše (ang. Meta-Cartridge), na katerih apliciramo podatke iz ovojnice. V tej fazi imamo že pripravljeno ovojnico spletnega vira, s pomočjo katere pridobimo delno strukturirane spletne podatke iz izbranega spletnega vira.

6.1.3 Faza 3 - Analiza SPARQL poizvedbe

Sledi analiziranje SPARQL poizvedbe z dereferenciranjem URI naslova, kar pomeni, da dobimo predstavitev vira na tem naslovu. Primer take poizvedbe

imamo podan na Sliki 6.1, kjer gre za poizvedovanje po strukturiranih podatkih, npr. trojčkih.

6.1.4 Faza 4 - Zahteva po RDF podatkih

Po uspešno analizirani SPARQL poizvedbi, strežnik poda zahtevo po RDF podatkih (trojčkih) izbranega spletnega vira. Medtem strežnik omogoča tudi medpomnjenje RDF trojčkov z možnostjo nastavitve življenjske dobe (angl. krat. TTL) vsakega trojčka.

6.1.5 Faza 5 - Preverjanje RDF podatkov

V tej fazi se za vsak vir podatkov preveri, ali so podatki v obliki RDF podatkovnega modela že na voljo. V primeru, da niso, se vrnemo nazaj na drugo fazo, kjer smo si vnaprej pripravili ovojnico za točno določeni spletni vir, s pomočjo katere lahko pridobimo delno strukturirane podatke in jih vrnemo v obliki RDF podatkovnega modela. V nasprotnem primeru - da imamo podatke v obliki RDF podatkovnega modela že na voljo - pa sledi zadnja faza.

6.1.6 Faza 6 - Združitev v RDF graf

V zadnji fazi imamo že uspešno pridobljene vse podatke, zato jih lahko združimo v obliko RDF povezanega grafa. Te podatke lahko npr. uporabimo tudi za vizualizacijo in s tem pokažemo dodano vrednost.

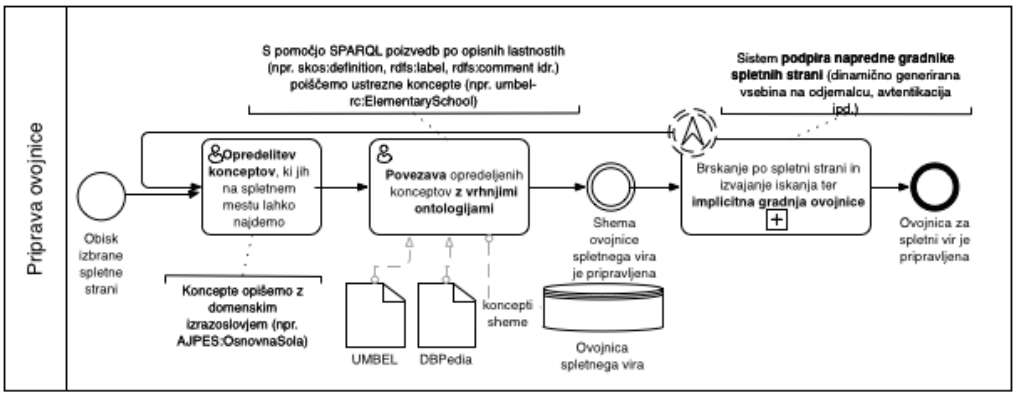
6.2 Komponenta, ki uporabniku pomaga pridobivati spletne podatke med brskanjem po spletni strani

V drugem delu načrtovanja razvoja smo podrobneje razdelali komponento, ki uporabniku pomaga pridobivati spletne podatke med brskanjem po spletni

strani.

V prvem koraku smo analizirali in določili, na kakšen način bo definirana ta komponenta oz. ovojnica, da jo bomo najlažje uporabili znotraj prototipa arhitekturne rešitve inteligentnega pomočnika.

Ovojnico smo definirali kot tip eksplicitnih metapodatkov, ki skrbijo za pretvorbo delno strukturiranih HTML podatkov v strukturirano obliko RDF podatkovnega modela, katero lahko kasneje posledično vključimo v Linked Data oblak in jo umestimo v štiri- ali petzvezdično obliko na petstopenjski lestvici. S temi podatki, ki jih imamo v ustrezni obliki (RDF ali Linked RDF podatkovni model), lahko potem izvajamo obogatene pozivedbe nad več različnimi podatkovnimi viri.



Slika 6.2: Priprava ovojnice za pretvorbo delno strukturiranih HTML podatkov v strukturirano obliko RDF podatkovnega modela.

Priprava ovojnice poteka v naslednjih korakih:

- obisk izbrane spletne strani,
- opredelitev konceptov na spletnem mestu, katere opišemo z domenskim izrazoslovjem (npr. AJPES:OsnovnaSola);
- povezovanje opredeljenih konceptov z vrhnjimi ontologijami (npr. UMBEL, DBPedia, shema ovojnice spletnega vira ipd.); poizvedba s pomočjo SPARQL poizvedovalnega jezika po lastnostih

(npr rdf:definition), ki nam vrne ustrezne koncepte
(npr. umbel-rc:ElementarySchool);

- nadaljnje brskanje po spletni strani in implicitna gradnja ovojnice, ter nenehno vračanje na drugi korak. Ko pa se zaključi brskanje po spletni strani, je ovojnica spletnega vira pripravljena in se lahko začne uporabljati.

6.2.1 Podrobna opredelitev implicitne gradnje ovojnice

Do sedaj smo predstavili le osnutek priprave in vključitve ovojnice v prototip inteligetnega pomočnika za luščenje delno strukturiranih spletnih podatkov, v tem delu pa smo se bolj podrobno osredotočili na podrobnosti zgradbe in delovanja ovojnice.

Slika 6.3: Koraki pri pripravi ovojnice.

Pri pripravi ovojnice za pridobivanje delno strukturiranih spletnih podatkov iz spletne strani so predvideni trije koraki, ni pa nujno, da vse spletne

strani, iz katerih pridobivamo delno strukturirane spletne podatke, vsebujejo vse te tri korake:

- V prvem koraku inteligentni pomočnik s pomočjo posebnih algoritmov samodejno poišče in prepozna glavni iskalni obrazec na spletni strani. Ko je glavni iskalni obrazec najden, ga označi in iz njega pridobi vrednosti vseh vhodnih elementov. S pomočjo vrednosti vhodnih elementov lahko sestavimo začetni spletni naslov, iz katerega bomo pridobivali delno strukturirane spletne podatke. Po uspešno izbranem gumbu za iskanje zadetkov, se izvede AJAX klic do programskega vmesnika in podatki vhodnih elementov se shranijo v obliki prvega dela ovojnice na trdi disk.

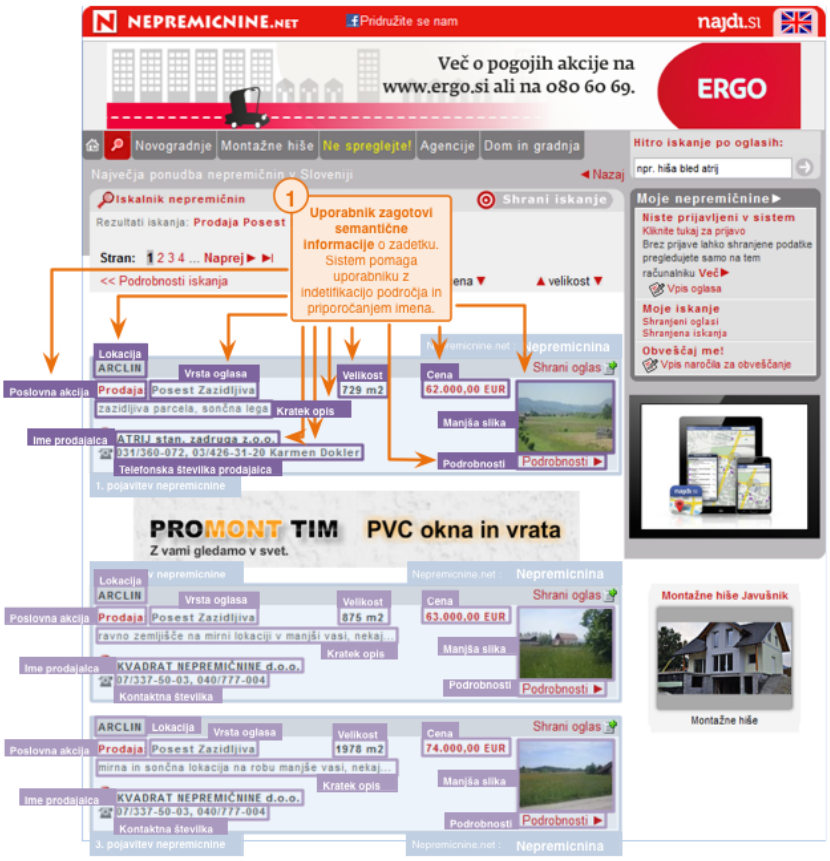
- V drugem koraku inteligentni pomočnik, s pomočjo prilagojenega algoritma iskanja v širino, samodejno identificira ponavljajoče vzorce kot individualne zadetke. Ta nas vpraša, če smo zadovoljni s trenutno identificiranimi ponavljajočimi vzorci zadetkov, in če potrdimo, potem samodejno, znotraj teh zadetkov poišče podatke in vrne njihove XPath izraze. V nasprotnem primeru se, če z izbranimi identificiranimi ponavljajočimi vzorci zadetkov nismo zadovoljni, pomakne na naslednjo izbiro ponavljajočih vzorcev zadetkov in nam jih ponudi kot naslednjo možnost. Po polju identificiranih ponavljajočih vzorcev zadetkov se sprehaja in nam jih ponuja tako dolgo, dokler nismo zadovoljni z označenimi identificiranimi zadetki. V primeru, da inteligentni pomočnik ne uspe poiskati vseh podatkov in njihovih XPath izrazov znotraj ponavljajočih zadetkov, lahko podatke, ki so nam zanimivi, ročno označimo in inteligentni pomočnik nam vrne njihove XPath izraze, ki so dodani v objekt, kateri je v drugem delu ovojnice.

  Pred samim identificiranjem ponavljajočih vzorcev individualnih zadetkov pa inteligentni pomočnik s pomočjo preprostega algoritma na strani prepozna vzorec številčenja strani. V objekt shrani XPath izraz povezave na naslednjo stran in XPath izraz povezave na zadnjo spletno

stran.

Slika 6.4: Identifikacija ponavljajočih vzorcev zadetkov, številčenja strani in zagotovitev semantičnih informacij.

Ko smo zadovoljni in imamo označene željene ponavljajoče zadetke in podatke znotraj njih, izberemo posamezni zadetek, da pridemo na stran podrobnosti le-tega. Pred tem se izvede AJAX klic do programskega vmesnika in XPath izrazi ponavljajočih zadetkov, podatkov zadetkov in vzorcev številčenja strani se shranijo v obliki drugega dela ovojnice na trdi disk.



Slika 6.5: Identifikacija podatkov in samodejno priporočanje imen teh podatkov pri večih ponavljajočih vzorcih zadetkov na spletni strani.

- V tretjem koraku, ko se nahajamo na spletni strani podrobnosti posameznega zadetka, poskuša inteligentni pomočnik s pomočjo algoritmov samodejno poiskati podatke, ki podrobneje opisujejo posamezni ponavljajoči zadetek. Različne spletne strani imajo na različen način prikazane podrobne podatke posameznega zadetka, npr. nekatere s pomočjo tabel, druge s pomočjo ponavljajočih HTML značk ipd. V primeru, da inteligentni pomočnik ne uspe poiskati vseh podatkov in njihovih XPath izrazov samodejno, lahko podatke, ki so nam zanimivi, ročno označimo in inteligentni pomočnik nam vrne njihove XPath izraze, ki so dodani v objekt, kateri je tretji del ovojnice. Ko smo zadovoljni z označenimi

podatki, zapustimo spletno stran, pred tem pa se izvede AJAX klic do programskega vmesnika in XPath izrazi podrobnosti zadetkov se shranijo v obliki tretjega dela ovojnice na trdi disk.

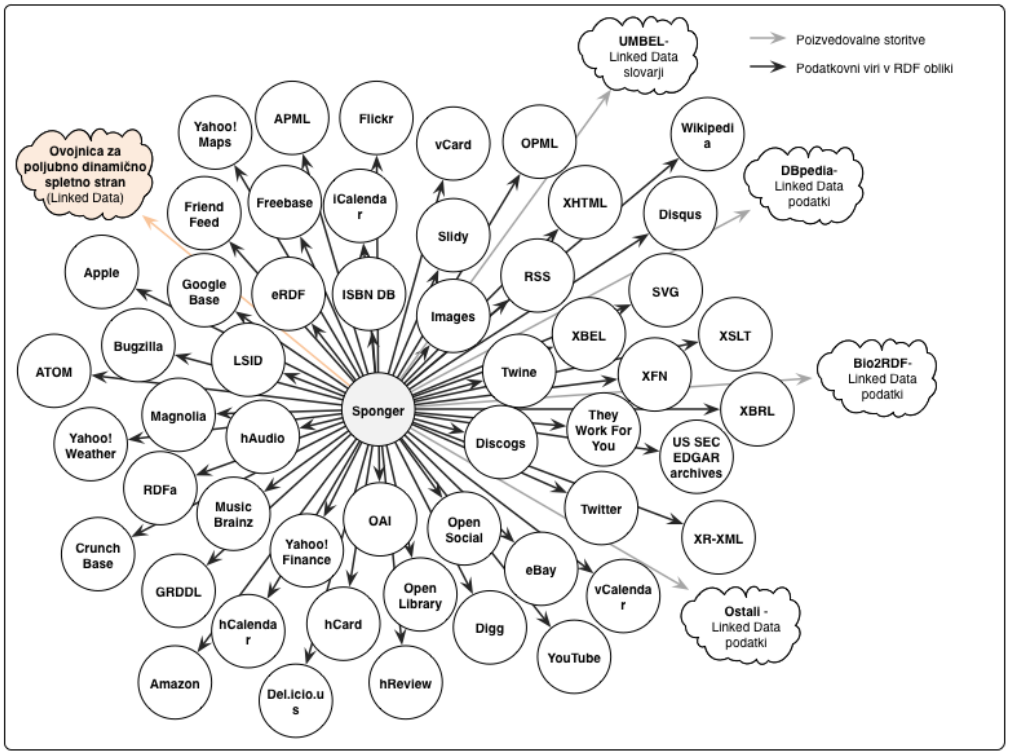
Slika 6.6: Identifikacija podatkov in priporočanje imen teh podatkov pri podrobnostih posameznega zadetka.

6.3 Umestitev konceptualnega predloga rešitve znotraj komponente Virtuoso Sponger

Prototip inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov smo umestili znotraj arhitekturne rešitve Virtuoso Sponger in s tem prikazali dodano vrednost. Z namestitvijo komponente Sponger pri arhitekturini rešitvi Virtuoso se samodejno že naložijo nekatere kartuše in meta-kartuše. Te so že vnaprej pripravili programerji nekaterih bolj znanih podjetij in organizacij in lahko jih prosto uporabljamo. Primeri takih podjetij so: eBay, Apple, Flickr in še nekatera druga znana podjetja in organizacije.

Konceptualni predlog naše rešitve spada v enako področje pri komponenti Virtuoso Sponger kot kartuše in meta-kartuše teh podjetij, vendar to ni kartuša oz. meta-kartuša za specifično podjetje oz. spletno stran, ampak splošna kartuša, na katero lahko apliciramo ovojnico za poljubno dinamično spletno stran (ang. Linked data).

Če si pogledamo rešitev Virtuoso Sponger na Sliki 6.7, lahko vidimo, da naš prispevek spada v oranžno obarvan del slike.



Slika 6.7: Umestitev konceptualnega predloga rešitve znotraj komponente Virtuoso Sponger.

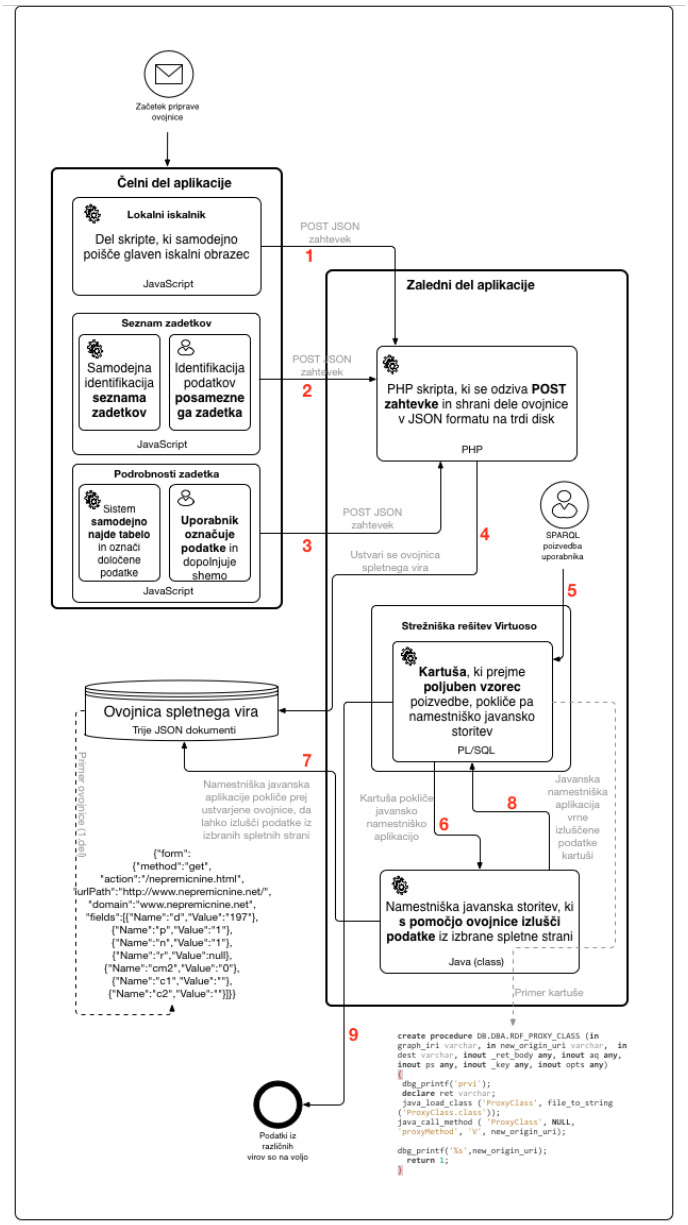
Poglavje 7

Tehnične podrobnosti implementacije

Razvoj inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov lahko iz tehničnega vidika aplikacij razdelimo na dva glavna dela:

- čelni del sistema in
- zaledni del sistema.

Tudi ta lahko razdelimo še na manjše enote, saj smo vsakega od njiju implementirali po delih. Glede na to delitev, smo predstavili tehnične podrobnosti implementacije. V prvem delu smo predstavili čelni del sistema, ki je po načinu delovanja bližje uporabniku, kasneje pa zaledni del, kjer se stvari dogajajo v zaledju, nevidno za uporabnika. Podali smo tudi delčke programske kode [39] ter psevdokode, ki smo jih razvili za potrebe delovanja našega inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov. Na sliki 7.1 lahko vidimo grafični prikaz delovanja inteligetnega pomočnika, prav tako pa tudi njegovo celotno arhitekturno shemo in tehnične podrobnosti implementacije. Na Sliki 7.1 smo vsaki komponenti pripisali programski jezik, v katerem je implementirana.



Slika 7.1: Prikaz arhitekture in delovanja inteligetnega pomočnika.

7.1 Čelni del sistema - priprava ovojnice

V tem poglavju smo s tehničnega vidika predstavili gradnjo trodelne ovojnice in razložili delovanje algoritmov, ki smo jih razvili za potrebe gradnje ovojnice. Trodelna gradnja ovojnice je sestavljena iz naslednjih treh delov:

- lokalnega iskalnika,
- seznama zadetkov,
- podrobnosti posameznega zadetka.

Pogramski jezik, ki smo ga uporabili za razvoj skripte, s pomočjo katere gradimo dele ovojnice, je JavaScript, saj nam ravno ta omogoča interakcijo z uporabnikom, ki je zahtevana pri tem delu prototipa inteligentnega pomočnika.

Eno od pomembnih vprašanj, ki se nam je pojavilo pri pripravi ovojnice, je bil format v katerem bomo hranili dele ovojnice. Odločali smo se med dvema, in sicer, da bi imeli ovojnico shranjeno v obliki XML dokumenta ali pa JSON dokumenta. Lahko bi uporabili katerega koli od zgoraj navedenih, saj bi z obema prišli do podobne rešitve. Po preliminarnih raziskavah in razmisleku, smo se odločili, da bomo ovojnico raje hranili v obliki JSON dokumenta, kot pa XML dokumenta.

Za JSON format smo se odločili iz dveh glavnih razlogov:

- dokumenti v JSON formatu zavzamejo manj pomnilniškega prostora, kot pa XML dokumenti,
- JSON je trenutno najbolj razširjen in poznan format za izmenjavo podatkov, prav tako pa ima tudi zelo dobro podporo pri skoraj vseh spletnih programskih jezikih, tudi pri JavaScriptu in Javi, kjer na preprost način lahko upravljamo z JSON objekti.

7.1.1 Algoritmi za ustvarjanje generičnih XPath izrazov

V tem poglavju smo podrobneje predstavili dva algoritma, katera smo uporabili na čelnem delu aplikacije za ustvarjanje generičnih XPath izrazov iz najdenih HTML elementov. Ta dva algoritma sta ključna pri gradnji ovojnice, saj na podlagi njunih izhodnih podatkov lahko pridobivamo delno strukturirane spletne podatke.

Ker algoritmi za ustvarjanje XPath izrazov iz podanih HTML elementov že obstajajo, smo uporabili te obstoječe [17][18], vendar smo jih nekoliko spremenili in prilagodili za naše potrebe. Najbolj primerna algoritma za naše potrebe smo našli kar znotraj dveh spletnih brskalnikov, in sicer znotraj Mozille Firefox in Googla Chrome. S kombinacijo teh dveh algoritmov smo lahko razvili skripto, ki nam omogoča implicitno gradnjo ovojnice.

Algoritem za ustvarjanje XPath izrazov v spletnem brskalniku Mozilla Firefox

Prvi algoritem, katerega smo predstavili, je algoritem za iskanje XPath izrazov znotraj spletnega brskalnika Mozilla Firefox. Pred samim začetkom uporabe algoritma smo ga testirali, na kakšen način generira XPath izraze iz elementov. Tudi te XPath izraze smo morali preveriti na različnih primerih, da smo ugotovili, če algoritem zadošča našim potrebam. Spodaj smo podali primarno implementacijo algoritma v programskem jeziku JavaScript.

```
function createXPathFromElementFirefox( element ) {
  var xpath = '';
  for ( ; element && element.nodeType == 1; element =
     element.parentNode ) {
    var id = $(element.parentNode).children(element.
       tagName).index(element) + 1;
  id > 1 ? (id = '[' + id + ']') : (id = '');
  xpath = '/' + element.tagName.toLowerCase() + id +
     xpath;
  }
  return xpath;
};
```

Funkcija algoritma dobi kot vhodni podatek HTML element, od katerega želimo pridobiti XPath izraz. Ta se potem sprehodi po HTML drevesu navzgor, dokler ne pride do korenskega vozlišča (ang. root node). Med temi vmesnimi koraki sprehoda za vsako HTML vozlišče ugotavlja, katero po vrsti je na tem nivoju drevesa in mu doda to številko. Spodaj imamo podan primer sprehoda algoritma po drevesu navzgor, pri podrobnostih zadetka na spletni strani http://www.nepremicnine.net.

```
1. iteracija: /table
2. iteracija: /a/table
3. iteracija: /div[2]/a/table
4. iteracija: /div[2]/div[2]/a/table
5. iteracija: /div[4]/div[2]/div[2]/a/table
6. iteracija: /div[3]/div[4]/div[2]/div[2]/a/table
7. iteracija: /div[2]/div[3]/div[4]/div[2]/div[2]/a/
   table
8. iteracija: /div[3]/div[2]/div[3]/div[4]/div[2]/div
   [2]/a/table
9. iteracija: /body/div[3]/div[2]/div[3]/div[4]/div
   [2]/div[2]/a/table
10. iteracija: /html/body/div[3]/div[2]/div[3]/div
   [4]/div[2]/div[2]/a/table
```

Končni XPath izraz algoritma je:

```
/html/body/div[3]/div[2]/div[3]/div[4]/div[2]/div[2]/a/
   table
```

Algoritem gradi XPath izraze od desne proti levi, kar v HTML drevesu pomeni od spodaj navzgor. V zadnji iteraciji nam vrne končni XPath izraz, ki vsebuje tudi korenski element celotnega HTML drevesa spletne strani.

S testiranjem algoritma na praktičnem primeru gradnje ovojnice, na spletni strani Statističnega urada Slovenije [19], smo ugotovili, da algoritem ne deluje čisto pravilno na vseh primerih. Kot lahko razberemo iz implementacije algoritma v programskem jeziku JavaScript, vidimo, da algoritem v primeru, če je na nekem nivoju HTML element prvi na tem nivoju drevesa, ne doda številke 1. S tem generira XPath izraz, ki izbere vse elemente na tem nivoju HTML drevesa, ki ustrezajo podanemu izrazu. Algoritmu smo dodali eno vrstico programske kode, s katero smo elementu na najnižjem nivoju HTML drevesa dodali vrednost 1. Koda spremenjenega algoritma je

podana spodaj.

```
function createXPathFromElementFirefox( element ) {

  var xpath = '';

  for ( ; element && element.nodeType == 1; element =
     element.parentNode ) {
    var id = $(element.parentNode).children(element.
       tagName).index(element) + 1;
  id > 1 ? (id = '[' + id + ']') : (id = '');
  xpath = '/' + element.tagName.toLowerCase() + id +
     xpath;
  }
  xpath = xpath + '[1]';

  return xpath;
};
```

S to spremembo algoritma dobimo končni izraz zgornjega primera iteracij:

```
/html/body/div[3]/div[2]/div[3]/div[4]/div[2]/div[2]/a/
   table[1]
```

Algoritem za ustvarjanje XPath izrazov v spletnem brskalniku Google Chrome

Drugi algoritem, katerega smo predstavili, je algoritem za iskanje XPath izrazov znotraj spletnega brskalnika Google Chrome. Do izvorne kode primarnega algoritma za iskanje XPath izrazov smo prišli s pomočjo Google Chrome dodatka Chrome extension source viewer. Ta nam omogoča, da si ogledamo izvorne kode nekaterih Google Chrome razširitev.

Tudi ta algoritem smo preverili na različnih primerih, ter ga na podlagi rezultatov dopolnili, da deluje na čim več različnih primerih in je čim bolj generičen. Spodaj smo podali implementacijo že nadgrajenega algoritma v programskem jeziku JavaScript.

```
function createXPathFromElementChrome( element ) {

  var allNodes = document.getElementsByTagName('*');

  for (var segs = []; element && element.nodeType == 1;
      element = element.parentNode) {
    if (element.hasAttribute('id')) {
      var uniqueIdCount = 0;
        for (var n=0;n < allNodes.length;n++) {
      if (allNodes[n].hasAttribute('id') && allNodes[n
         ].id == element.id)
        uniqueIdCount++;
      if (uniqueIdCount > 1)
        break;
    };
    if ( uniqueIdCount == 1) {
      if(element.getAttribute('class') != null )
        segs.unshift('//*[@class="' + element.
           getAttribute('class') + '"]');
      else
      segs.unshift('/');
        return segs.join('/');
    } else {
      segs.unshift(element.localName.toLowerCase() + '[
         @class="' + element.getAttribute('class') + '
         "]');
    }
  }
```

```
  else if (element.hasAttribute('class')) {
    segs.unshift(element.localName.toLowerCase() + '[
       @class="' + element.getAttribute('class') + '"]'
       );
  } else {
    for (i = 1, sib = element.previousSibling; sib; sib
        = sib.previousSibling) {
      if (sib.localName == element.localName)
        i++;
    };
      segs.unshift(element.localName.toLowerCase() + '[
         ' + i + ']');
  };
 };

  return segs.length ? '/' + segs.join('/') : null;
};
```

Tudi algoritem znotraj spletnega brskalnika Google Chrome gradi XPath izraz podanega HTML elementa od desne proti levi. Deluje na podoben način kot Mozillin algoritem, vendar poskuša v končni XPath izraz dodati tudi izbirnike razredov. S tem algoritmom dobimo bolj pregleden končni XPath izraz, saj ni nujno, da vsebuje samo HTML značke in njihove številke, temveč vsebuje tudi imena razredov, kjer je le-to možno.

Primer končnega XPath izraza, ki ga dobimo s tem algoritmom:

```
//a[@class="pt"]/table[1]
```

7.1.2 Algoritem za samodejno predlaganje imena označenega podatka

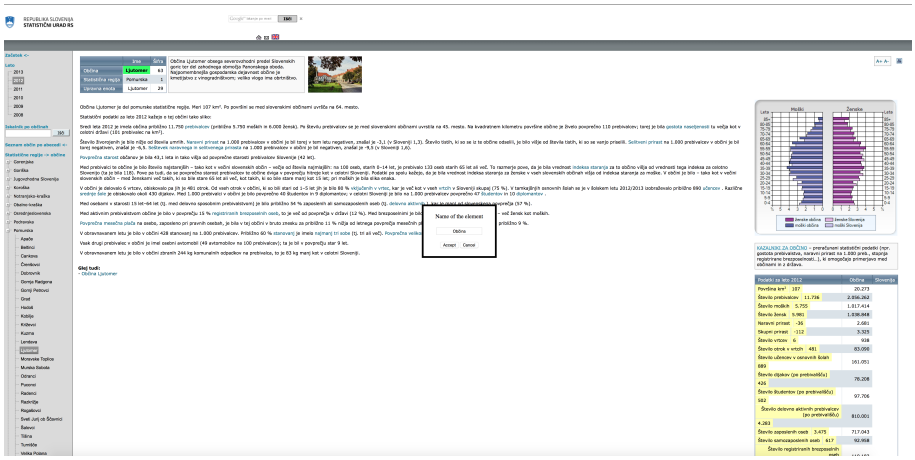
Z namenom, da bi bil prototip inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov čim bolj prijazen do uporabnika, smo implementirali algoritem za samodejno predlaganje imena označenega podatka. S tem smo poskušali prihraniti uporabnikov čas in trud, ki bi bil potreben za vnos imena izbranega podatka. Algoritem smo testirali na več različnih spletnih straneh, na različnih primerih podatkov in ga poskušali razviti generično do neke mere. Spodaj smo podali njegovo psevdokodo, na Sliki 7.2 pa lahko vidimo primer delovanja algoritma.

```
FUNKCIJA samodejnoPredlagajImeOznacenegaPodatka(
   izbraniTekst ):
  Pridobi HTML element, ki vsebuje izbraniTekst;
  Ce je ta HTML element tipa znacke TD:
  Potem za ime izbranega podatka predlagaj tekst
     prejsnje TD celice;
  Ce je ta HTML element drugega tipa znacke:
    Potem za ime izbranega podatke predlagaj ime
       razreda tega HTML elementa;
KONEC.
```

Vidimo, da je algoritem prototipa orodja precej preprost. Pridobi HTML element na podlagi izbranega teksta, potem pa glede na tip HTML značke elementa predlaga različna imena podatka.

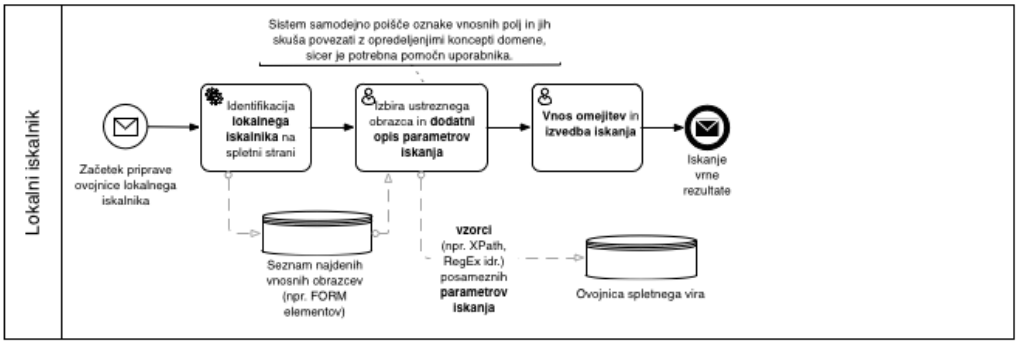
7.1.3 Lokalni iskalnik

Prvi od treh delov ustvarjanja ovojnice je lokalni iskalnik, kateri nam kot rezultat poskuša ustvariti del ovojnice, ki vsebuje vrednosti vhodnih polj, na podlagi katerih lahko ustvarimo začetni spletni naslov za pridobivanje podatkov na spletni strani. Na Sliki 7.3 imamo na grafičen način prikazane



Slika 7.2: Primer samodejnega predlaganja imena (Občina) pri izbiri podatka Ljutomer.

korake delovanja lokalnega iskalnika.



Slika 7.3: Koraki pri delovanju prototipa lokalnega iskalnika.

Jedro delovanja lokalnega iskalnika predstavlja izbira ustreznega iskalnega obrazca na spletni strani. Ponavadi je na začetni spletni strani več obrazcev, vendar je samo eden od njih tisti, s katerim ob vnosu ustreznih vrednosti polj in ob izbiri gumba za iskanje, vrne spletno stran s seznamom zadetkov. Iskanje glavnega iskalnega obrazca izmed vseh obrazcev na začetni spletni strani smo rešili z implementacijo algoritma, ki poišče obrazec na spletni strani z največ HTML vozlišči. Njegova psevdokoda je podana v naslednjem

razdelku.

```
FUNKCIJA poisciGlavniObrazec:
  Vrni vse obrazce na zacetni spletni strani;
  Izmed vseh obrazcev najdi tistega,
  ki vsebuje najvec HTML vozlisc;
  Iz najdenega obrazca poberemo ven
  vrednosti vhodnih polj (potrditvena polja
  izbirna polja, tekstovna vnosna polja, ipd.);
  Te vrednosti hranimo v pomnilniski enoti
  v obliki JSON objekta, dokler ne naredimo
  AJAX POST zahtevka in ga shranimo na streznik;
KONEC.
```

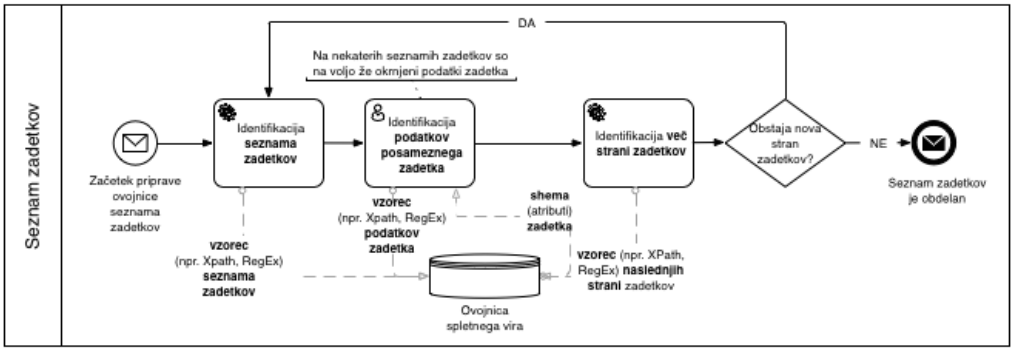
Psevdokodo zgornjega algoritma za iskanje glavnega iskalnega obrazca smo implementirali v programskem jeziku JavaScript in testirali na praktičnem primeru spletne strani. Kot rezultat po AJAX klicu smo dobili JSON dokument s spodnjo vsebino.

```
{"data":
  {"method":"get",
   "action":"/nepremicnine.html",
   "urlPath":"http://www.nepremicnine.net/",
   "domain":"www.nepremicnine.net",
   "fields":
     [{"Label":"Drzava","Name":"d","Value":"197"},
      {"Label":"Posredovanje","Name":"p","Value":"1"},
      {"Label":"Nepremicnina","Name":"n","Value":"1"},
      {"Label":"Regija","Name":"r","Value":"14"},
      {"Label":"Cena:","Name":"cm2","Value":"0"},
      {"Label":"Cena:","Name":"c1","Value":"100000"},
      {"Label":"Cena:","Name":"c2","Value":"102000"}]
  }
}
```


7.1.4 Seznam zadetkov

Drugi od treh delov ovojnice je seznam zadetkov, kjer s prototipom orodja poskušamo ustvariti del ovojnice, ki vsebuje XPath izraza številčenja strani, XPath izraz zadetka in XPath izraze znotraj zadetkov, s pomočjo katerih lahko pridobivamo delno strukturirane spletne podatke ponovljenih vzorcev zadetkov.

Prikazali smo tudi tri algoritme, ki so ključni pri gradnji dela ovojnice seznama zadetkov. Na Sliki 7.4 imamo na grafičen način prikazane korake delovanja prototipa pri seznamu zadetkov.



Slika 7.4: Koraki pri delovanju prototipa seznama zadetkov.

Primer JSON dokumenta, ki smo ga ustvarili s pomočjo prototipa inteligetnega pomočnika s pomočjo AJAX klica na programski vmesnik imamo podan spodaj.

```
{"data":
  {"domain":"www.nepremicnine.net",
   "url":"http://www.nepremicnine.net/nepremicnine.html
      ?d=197&p=1&n=1&r=15&c1=&c2=&cm2=0",
   "paginationNext":"//div[@class="headbar"]/div[@class
      ="top fr"]/ul/li/a[@class="last"]/@href",
   "paginationLast":"//div[@class="headbar"]/div[@class
      ="top fr"]/ul[1]/li[6]/a[@class="last"]/@href",
   "result":"/html/body/div[3]/div[2]/div[3]/div[4]/div
      [2]/div[2]/div",
   "fields":[{"lokacija":"h2/a[1]"},
         {"slika":"a/img[1]"},
         {"kratek_opis":"div/div[2]/div[1]"},
         {"podrobnosti":"a[1]"},
         {"povrsina":"div/div[3]/span[1]"},
         {"cena":"div/div[3]/span[2][1]"},
         {"leto":"*//span[contains (. , 'Leto')]"},
         {"nadstropje":"*//span[contains (. , '
            Nadstropje')]"},
         {"posredovanje":"*//span[contains (. , '
            Prodaja')]"}]
  }
}
```

Jedro delovanja prototipa pri seznamu zadetkov na Sliki 7.4 so trije koraki, in sicer identifikacija seznama zadetkov, identifikacija podatkov posameznega zadetka in identifikacija več strani zadetkov. Algoritme teh treh korakov smo opisali v naslednjih razdelkih.

Algoritem za iskanje enakih HTML poddreves v HTML drevesu

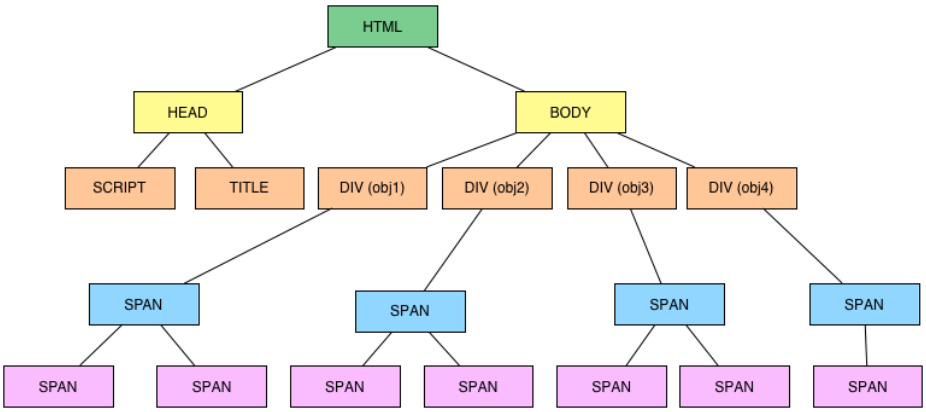
Vsako spletno stran, ki je pripravljena v HTML jeziku, si lahko predstavljamo kot drevesno strukturo. To je lahko zelo uporabno, saj je na teh drevesnih

strukturah mogoče poganjati določene algoritme, kot je npr. iskanje podobnosti med drevesi, iskanje elementov v drevesu ipd.

Spodaj smo podali programsko kodo preproste HTML spletne strani, na podlagi katere smo prikazali še predstavitev drevesne strukture HTML značk.

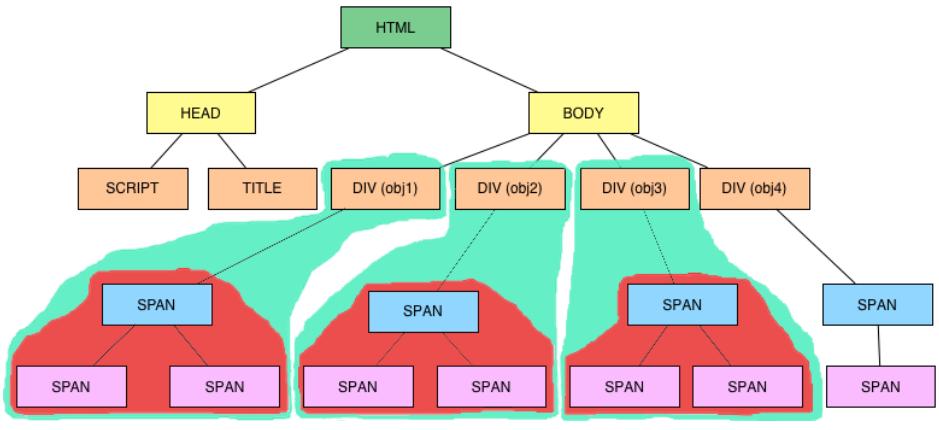
```
<html>
  <head>
    <script src="http://ajax.googleapis.com/ajax/libs/
       jquery/1.11.3/jquery.min.js"></script>
      <title>Spletna stran</title>
  </head>
  <body>
    <div class="obj1">
    <span>
      <span>Nadstropje: P</span>
      <span>Leto: 1970</span>
    w</span>
    </div>
  <div class="obj2">
    <span>
      <span>Nadstropje: 3</span>
    <span>Leto: 1999</span>
    </span>
  </div>
  <div class="obj3">
      <span>
      <span>Nadstropje: 3</span>
      <span>Leto: 1999</span>
    </span>
  </div>
    <div class="obj4">
      <span>
      <span>Nadstropje: 3</span>
    </span>
    </div>
  </body>
</html>
```

HTML struktura spletne strani je preprosta in vsebuje le nekaj HTML značk, njen namen pa je, da jo prikažemo v obliki drevesne strukture HTML značk. Slika 7.5 prikazuje primer drevesne strukture.



Slika 7.5: Predstavitev HTML strani v obliki HTML drevesa.

Na Sliki 7.6 lahko vidimo, da lahko celotno HTML drevo tvori več manjših poddreves, z rdečo in turkizno zeleno pa smo označili nekatere primere poddreves.



Slika 7.6: Z rdečo in turkizno zeleno označena enaka HTML poddrevesa.

Pri zasnovi algoritma za iskanje enakih HTML poddreves smo morali na začetku definirati metriko, na podlagi katere smo primerjali enakost HTML

poddreves. Vsako drevo (in tudi poddrevo) je sestavljeno iz enega ali več vozlišč, v našem primeru je to vozlišče HTML značka. Na podlagi tega smo za metriko uporabili zaporedni niz HTML značk izbranega poddrevesa.

Pri prehodu čez celotno drevo smo si pomagali z algoritmom za iskanje v širino (ang. Breadth-first search). Ta se uporablja za preiskovanje podatkovnih struktur dreves in grafov. Algoritem iskanja v širino začne preiskovanje v korenskem vozlišču (v našem primeru je to HTML značka html), potem se pomakne nivo nižje in preišče vsa vozlišča od leve proti desni na tem nivoju in tako nadaljuje po nivojih navzdol, dokler ne preišče vseh vozlišč. Na primeru iz slike 7.6 bi imelo turkizno zeleno drevo HTML strukturo niza "DIVSPANSPANSPAN", rdeče drevo pa "SPANSPANSPAN".

Algoritem iskanja v širino smo dopolnili tako, da smo za vsako vozlišče preverili njena sosednja vozlišča na tem nivoju in primerjali HTML strukturo niza tega vozlišča s HTML strukturami nizov sosednjih vozlišč. V primeru, da je imelo katero sosednje vozlišče enako HTML strukturo niza, smo vozlišče dodali v množico rezultatov. Prav tako smo sproti preverjali, če je bilo v to množico rezultatov že dodano vozlišče z enako HTML strukturo niza, in če je bilo, tega vozlišča nismo dodali v množico rezultatov, saj bi tako dobili podvojene rezultate. S tem postopkom smo dobili množico rezultatov vozlišč, urejenih od višjega nivoja proti nižjemu. Ta vozlišča iz množice rezultatov smo po vrsti posamično ponudili uporabniku in ga vprašali, če se strinja z izbranimi ponovljenimi zadetki. V primeru, da se uporabnik ni strinjal, smo mu ponudili naslednje vozlišče iz množice rezultatov.

Spodaj je podana programska koda algoritma samodejnega iskanja enakih HTML poddreves v programskem jeziku JavaScript.

```
function isAlreadyInArray( HTMLStructure, Array ){

  for(var i = 0; i < Array.length; i++){
    if( HTMLStructure == Array[i]['HTMLStructure'] ){
      return true;
    }
  }
  return false;
};

function findRepeatedElms() {

  var level        = $('html');
  var HTMLStructure    = "";
  var levelCounter  = 1;
  var repeatedElms  = [];

  while( level.children().length ) {
    var levelElms = [];
    level = level.children().each(function(_, el) {

      $(el).find('*').each(function(){
        HTMLStructure += $(this).prop('tagName');
      })
      var Obj = { level: levelCounter, HTMLStructure:
         HTMLStructure, element: $(el) };
      levelElms.push(Obj);
      HTMLStructure = "";
    })

    for( var i = 0; i < levelElms.length-1; i++ ){
      for( var j = i+1; j < levelElms.length; j++ ){
        if( levelElms[i].HTMLStructure == levelElms[j].
```

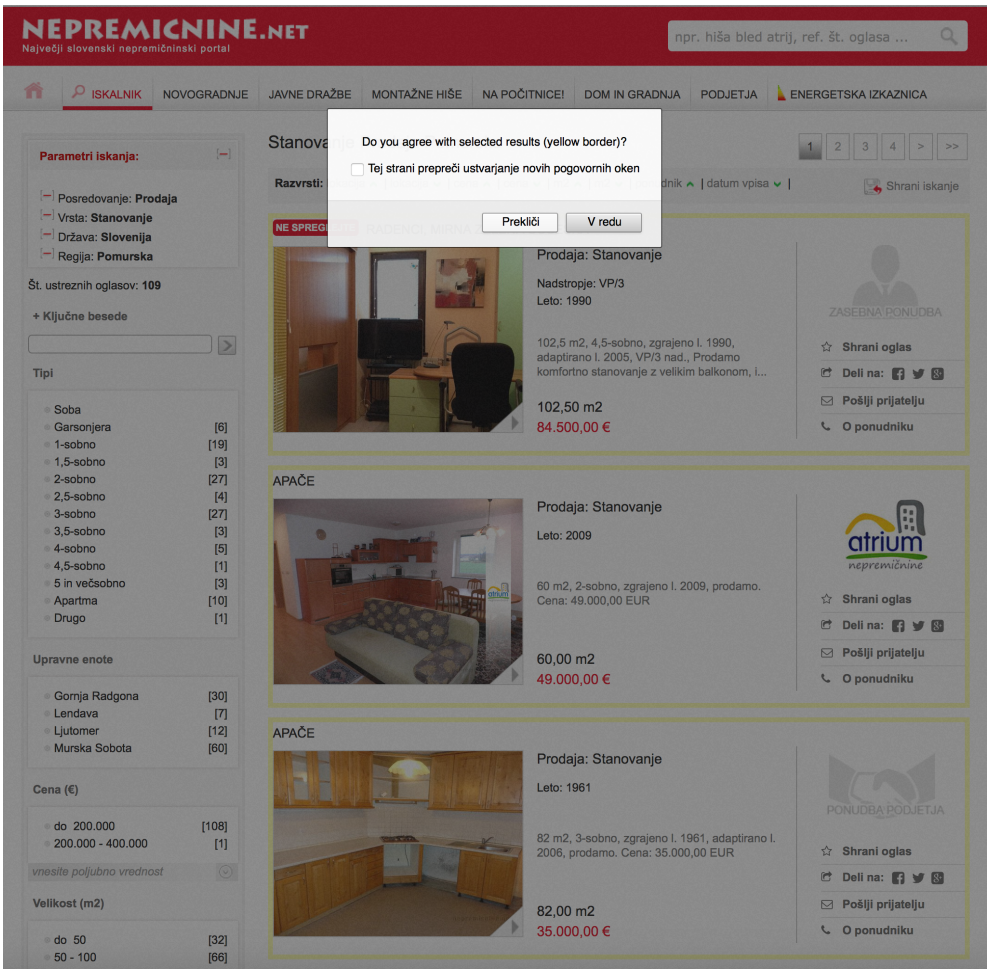
```
            HTMLStructure && levelElms[i].HTMLStructure
            != "" ){
           if( !isAlreadyInArray( levelElms[i].
              HTMLStructure, repeatedElms ) ){
            repeatedElms.push( levelElms[i] );
           }
         }
       }
     }
     levelCounter = levelCounter + 1;
   }

   return repeatedElms;
};
```

Celotni algoritem je sestavljen iz dveh funkcij. Prva je pomožna funkcija, ki preveri, če je podana HTML struktura že v množici rezultatov. Druga funkcija je glavna funkcija, kjer smo implementirali algoritem iskanja v širino in z njegovo pomočjo, ter pomočjo prve pomožne funkcije, vrnili najdena iskana ponovljena HTML poddrevesa. Ta ponovljena HTML poddrevesa so vzorci ponovljenih elementov na spletni strani.

Na sliki 7.7 lahko vidimo, kako nam prototip inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov ponudi izbiro ponovljenih elementov na spletni strani. Te ponovljene elemente obrobi z rumeno barvo.

Algoritem za iskanje HTML elementov znotraj identificiranih ponovljenih vzorcev zadetkov

Po uspešno identificiranih ponovljenih zadetkih smo s pomočjo algoritma poiskali HTML elemente znotraj posameznega zadetka. Izmed vseh elementov smo izbrali samo tiste, ki vsebujejo samo tekst. S tem smo izločili elemente, ki vsebujejo sinove, prav tako pa tudi elemente, ki ne vsebujejo teksta. Kot



Slika 7.7: Najdeni ponovljeni zadetki, ki imajo nastavljeno rumeno obrobo.

rezultat smo dobili elemente s tekstom na najnižjem nivoju. Pri implementaciji algoritma smo si pomagali s knjižnico jQuery [38], ki nam je z že vnaprej definiranimi funkcijami olajšala delo. Implementacijo algoritma lahko najdemo v spodnjem odseku programske kode.

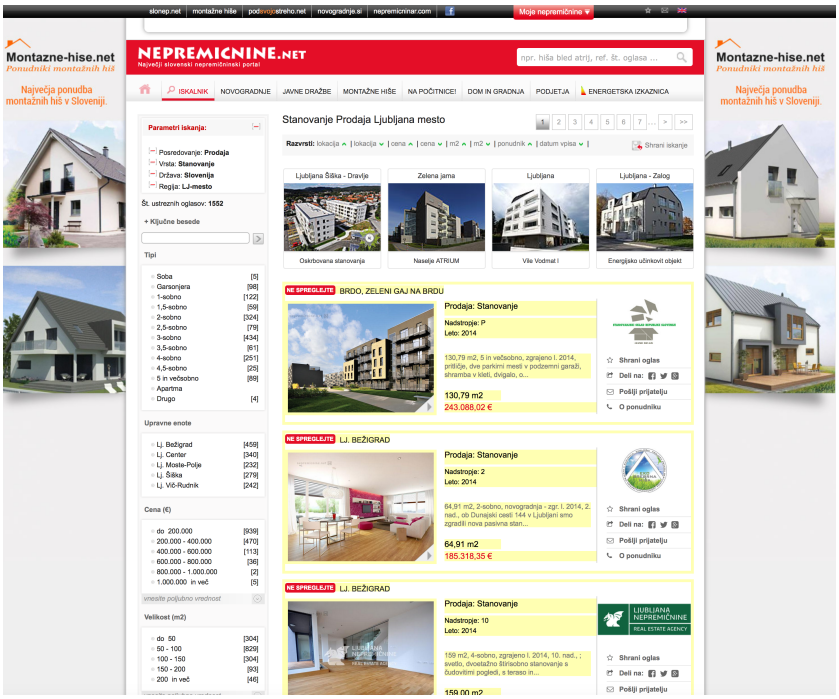
```
function findElementsWhichContainText(
   selectedResultWithMostElements ) {

  var elmsWithText  = [];
  var $this           = selectedResultWithMostElements;

  $this.find("*").each(function() {

    if ($(this).text().trim().length) {

      var match = /\r|\n/.exec($(this).text());
      if( !match ){
        elmsWithText.push($(this).get(0));
      }
    }
  })
  return elmsWithText;
};
```

Da smo lahko našli le željene elemente, smo si pomagali z regularnim izrazom. Te najdene elemente, ki vsebujejo samo tekst, smo tudi označili z rumeno barvo, kar lahko vidimo na sliki 7.8. Na podlagi teh pridobljenih elementov smo lahko poiskali njihove XPath izraze znotraj najdenih zadetkov.

Algoritem za samodejno identifikacijo številčenja strani

Med iskanjem ponovljenih vzorcev zadetkov spletne strani nam prototip inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov že samodejno poišče vzorec številčenja strani.

Po pregledu več spletnih strani smo ugotovili obliko, katera enolično predstavlja vzorec številčenja strani. Pri tej obliki v večini primerov najdemo naslednje gradnike: povezave do strani s številko, povezavo do naslednje strani,



Slika 7.8: Najdeni in označeni elementi znotraj zadetkov, ki imajo nastavljeno rumeno ozadje.

povezavo na začetno stran (če se že ne nahajamo na njej), povezavo na zadnjo stran (če se že ne nahajamo na njej).

Na podlagi teh ugotovljenih dejstev smo lahko sestavili algoritem. Ugotovili smo, da za uspešno pridobivanje podatkov iz več strani zadetkov potrebujemo XPath izraz naslednje strani, prav tako pa XPath izraz zadnje strani. Tako smo sestavili algoritem, ki nam najde HTML element, kateri vsebuje povezavo na naslednjo stran in HTML element, ki vsebuje povezavo na zadnjo stran. Za ta dva HTML elementa smo kasneje ustvarili XPath izraza, katera smo uporabili pri pridobivanju delno strukturiranih podatkov rezultatov na več straneh. Spodaj smo podali programsko kodo algoritma v programskem jeziku JavaScript.

```
function findPaginationNext() {

  var uls = $('ul');
  var paginationNext;

  for( var i = 0; i < uls.length; i++ ){
    var pages = [];
    $( uls.get(i) ).children().each(function() {
      pages.push( $( this ).text() );
    });
    if( pages[0] == "1" && pages[1] == "2" && pages[2]
       == "3" ){
      paginationNext = $(uls.get(i));
      break;
    }
  }

  paginationNext = findPaginationSubAddHref(
     paginationNext );
  paginationNext =
     findPaginationSubGeneralizePagination(
     paginationNext );

  return paginationNext;
};

function findPaginationLast() {

  var uls = $('ul');
  var paginationLast;

  for( var i = 0; i < uls.length; i++ ){
    var pages = [];
```

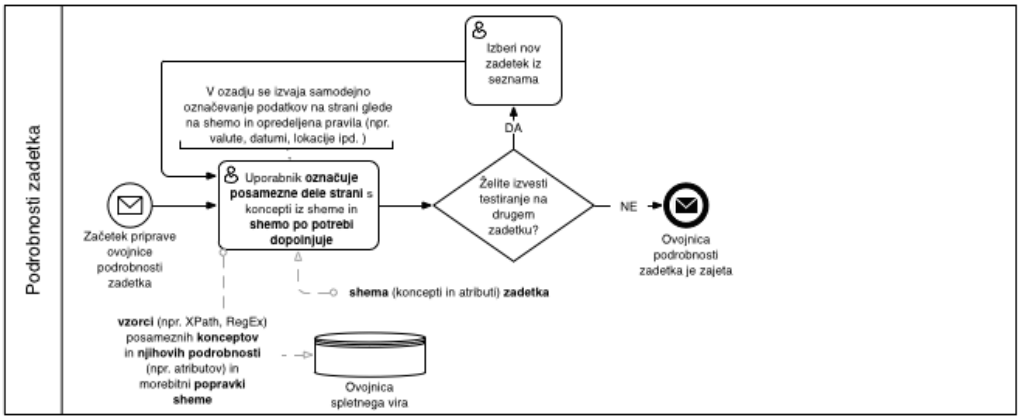
```
    $( uls.get(i) ).children().each(function() {
      pages.push( $( this ).text() );
    });
    if( pages[0] == "1" && pages[1] == "2" && pages[2]
       == "3" ){
      paginationLast = $(uls.get(i));
      break;
    }
  }

  paginationLast = findPaginationLastSubAddHref(
     paginationLast );

  return paginationLast;
}
```

Glavni algoritem smo razdelili na dve glavni funkciji. S pomočjo prve poiščemo XPath izraz elementa, ki vsebuje povezavo na naslednjo stran, z drugo pa XPath izraz elementa, ki vsebuje povezavo na zadnjo stran. Kodo funkcij bi še lahko optimizirali, saj se ponavljajo nekateri deli, vendar smo hoteli delovanje algoritma prikazati na nazornejši način. Jedro vsake funkcije predstavlja del, s katerim na spletni strani poiščemo neurejeni seznam, v katerem prvi element vsebuje tekst “1”, drugi “2” in tretji “3”. Znotraj tega elementa poiščemo željena elementa in vrnemo njuna XPath izraza, katera kasneje uporabimo na zalednem delu za pridobivanje delno strukturiranih spletnih podatkov.

7.1.5 Podrobnosti posameznega zadetka

Podrobnosti posameznega zadetka je zadnja ovojnica, ki jo lahko pripravimo s pomočjo inteligentnega pomočnika. Pri tem delu ovojnice poskušamo pridobiti XPath izraze, s pomočjo katerih lahko pridobivamo delno strukturirane spletne podatke s podrobnostmi posameznega zadetka.

Ob prvem obisku spletne strani posameznega zadetka se v ozadju izvede samodejno označevanje podatkov na spletni strani glede na shemo in opredeljena pravila. Potem lahko uporabnik sam označuje posamezne dele spletne strani s koncepti iz sheme in shemo po potrebi s tem dopolnjuje. Na sliki 7.9 vidimo prikazane korake pri gradnji ovojnice na strani posameznega zadetka.



Slika 7.9: Koraki pri gradnji ovojnice na strani posameznega zadetka.

Po končanem dopolnjevanju sheme uporabnik zapusti stran s podrobnimi podatki posameznega zadetka, takrat se s pomočjo AJAX klica na programski vmesnik zadnji del ovojnice shrani na trdi disk in tako imamo pripravljeno celotno ovojnico za pridobivanje delno strukturiranih spletnih podatkov iz izbrane spletne strani.

Primer tretjega dela ovojnice, ki ga pripravimo v obliki JSON dokumenta, smo podali spodaj.

```
{"data":
  {"domain":"www.nepremicnine.net",
   "url":"http://www.nepremicnine.net/oglasi-prodaja/
      radenci-mirna-zelena-soseska-stanovanje_5966576/"
      ,
   "fields":[{"kratek":"/html/body/div[3]/div[2]/div
      [3]/div[4]/div[2]/div[2]/div[1]"},
         {"referencna_st_":"/html/body/div[3]/div[2]/
            div[3]/div[4]/div[2]/div[2]/div[8]/strong
            [1]"},
         {"dodaten_opis":"/html/body/div[3]/div[2]/div
            [3]/div[4]/div[2]/div[2]/div[5]/p[1]"},
         {"com_tel":"//*[@class='com tel']"}]
  }
}
```

7.2 Zaledni del sistema

V prejšnjem poglavju smo predstavili vidni del sistema za uporabnika in interakcijo z njim, v tem delu pa se bomo osredotočili na del implementacije prototipa inteligentnega pomočnika, ki ni viden za uporabnika - akcije se izvajajo v ozadju. Zaledni del sistema je sestavljen iz treh glavnih komponent:

- skripte, ki shranjuje dele ovojnice na trdi disk;
- kartuše znotraj arhitekturne rešitve Virtuoso, ki kliče namestniško javansko storitev;
- namestniške javanske storitve, ki s pomočjo ogrodja HtmlUnit opravlja pridobivanje delno strukturiranih spletnih podatkov.

Vsaka od teh komponent je razvita v svojem programskem jeziku: skripta za shranjevanje ovojnice na trdi disk v PHPju, kartuša v PL/SQLu in namestniška javanska storitev v Javi.

Za vsakega od teh treh delov smo podali odsek ali celotno programsko kodo, prav tako pa tudi podrobneje opisali vsakega.

7.2.1 Skripta, ki shranjuje dele ovojnice na trdi disk

Prva komponenta, ki pride v stik s čelnim delom sistema je skripta, ki shranjuje dele ovojnice na trdi disk. To je programski vmesnik, ki smo ga večkrat omenili v prejšnjih poglavjih. Ta skripta je pripravljena v programskem jeziku PHP, ki se uporablja na zalednih delih sistemov.

Ta skripta opravlja vlogo povezovalnega člena čelnega dela sistema z zalednim. Tudi ta je v osnovi sestavljena iz treh delov, saj prejema tri vrste JSON post zahtevkov in tako smo jo tudi razdelili:

- prvi del, kjer prejmemo JSON post zahtevek s podatki, pridobljenimi s pomočjo dela ovojnice s podatki glavnega iskalnega obrazca;
- drugi del, kjer prejmemo JSON post zahtevek s podatki, pridobljenimi s pomočjo dela ovojnice z XPath izrazom zadetkov, podatkov znotraj njih in številčenja strani;
- tretji del, kjer prejmemo JSON post zahtevek s podatki, pridobljenimi s pomočjo dela ovojnice z XPath izrazi podrobnosti podatkov posameznega zadetka.

Spodaj smo podali odsek programske kode prvega dela skripte, kjer preko AJAX klica prejmemo podatke, na podlagi katerih ustvarimo prvi del ovojnice.

```
<?php

  if ( isset($_POST["dataForm"]) ) {

    $data     = $_POST["dataForm"];
    $dec_data = json_decode($data);
    $domain   = $dec_data->data->domain;
    $dir      = '/Applications/XAMPP/htdocs/NIK/
       magistrska/files/' . $domain . '.form.json';
    file_put_contents( $dir, $data );

  }
  ...
?>
```

V osnovi v zgornjem odseku programske kode prejmemo podatke preko AJAX klica, katere smo na čelnem delu sistema označili z imenom "dataForm". Te podatke s pomočjo funkcije json_decode() pretvorimo iz niza znakov v objekt, iz objekta preberemo vrednost spremenljivke domain in na podlagi te nastavimo pot in ime datoteke, v katero shranimo vse prejete podatke s pomočje funkcije file_put_contents().

Drugi in tretji del skripte imata podobno vlogo, le da namesto podatkov ovojnice lokalnega iskalnika preko AJAX klica prejmeta podatke ovojnice seznama zadetkov in podrobnosti posameznega zadetka.

7.2.2 Kartuša, ki kliče namestniško javansko storitev

Ko imamo pripravljene ovojnice poljubnih spletnih strani, lahko podamo SPARQL poizvedbo znotraj arhitekturne rešitve Virtuoso. S pomočjo poizvedbe pridobimo delno strukturirane podatke iz teh poljubnih spletnih strani.

Znotraj te SPARQL poizvedbe podamo vzorec, na podlagi katerega kom-

ponenta Virtuoso Sponger pokliče specifično kartušo. Ta specifična kartuša je PL/SQL procedura, katero smo definirali znotraj arhitekturne rešitve Virtuoso. Tudi tej kartuši smo določili, na kateri vzorec SPARQL poizvedbe naj se odziva, tako da se požene samo ob točno določenem vzorcu. Primer kartuše, ki smo jo razvili za naše potrebe pridobivanja delno strukturiranih spletnih podatkov, imamo podan spodaj.

```
create procedure DB.DBA.RDF_PROXY_CLASS (in graph_iri
   varchar, in new_origin_uri varchar,  in dest varchar
   ,    inout _ret_body any, inout aq any, inout ps any
   , inout _key any, inout opts any)
{
  declare ret nvarchar;

  java_load_class ('Form', file_to_string ('Form.class
     '));
  java_load_class ('ResultList', file_to_string ('
     ResultList.class'));
  java_load_class ('ResultDetails', file_to_string ('
     ResultDetails.class'));
  java_load_class ('Proxy', file_to_string ('Proxy.
     class'));

  ret := java_call_method ( 'Proxy', NULL, 'parse', '
     Ljava/lang/String;');

  DB.DBA.TTLP (cast (ret as varchar), '', 'http://
     localhost/customwebsites', 32);

  return 1;
}
```

Kartuša, ki smo jo razvili za potrebe pridobivanja delno strukturiranih

spletnih podatkov, je preprosta in vsebuje le nekaj vrstic programske kode.

V prvi vrstici deklariramo spremenljivko ret, v katero kasneje shranimo trojčke, pridobljene s pomočjo namestniške javanske storitve. V naslednjih štirih vrsticah, kjer je programska koda, naložimo z metodo java_load_class() že prevedene javanske razrede. Nato s pomočjo metode java_call_method() pokličemo metodo parse, ki se nahaja v prevedenem javanskem razredu Proxy.class. Ta metoda nam vrne trojčke, pridobljene iz spletnih strani za katere smo ustvarili ovojnice. Te trojčke shranimo v spremenljivko ret, vrednost katere pa kasneje shranimo še v graf, za možno kasnejšo uporabo.

Priprave kartuše bi se lahko lotili tudi na drugačen način. Lahko bi jo pripravili tako, kot so že vnaprej pripravljene nekatere kartuše in pri katerih je del kode za pridobivanje podatkov sprogramiran v jedru kartuš oz. procedur v programskem jeziku PL/SQL. Mi smo se odločili za drugačen pristop in kartušo uporabili le za klic namestniške javanske storitve, saj lahko znotraj okolja Java uporabimo ogrodje HtmlUnit, ki nam precej olajša delo glede pridobivanja delno strukturiranih spletnih podatkov.

7.2.3 Namestniška javanska storitev

Zadnja komponenta zalednega dela, katero smo že nekajkrat omenili v prejšnjem poglavju, je namestniška javanska storitev. Ta komponenta opravlja zelo pomemben del zalednega dela, saj ravno ta pridobiva delno strukturirane spletne podatke, kar je bistvena funkcionalnost prototipa inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov.

Za razvoj namestniških storitev znotraj arhitekturne rešitve Virtuoso smo imeli na izbiro naslednje tri programske jezike: Java, C++ in Python. Izbrali smo okolje Java, ker lahko uporabimo ogrodje HtmlUnit.

Namestniško javansko storitev smo razdelili v štiri dele. Glede na to delitev smo tudi razdelili vso programsko kodo v štiri različne javanske razrede in sicer: Form, ResultList, ResultDetails in Proxy. Ta delitev logično sovpada z deli ovojnice, ki smo jih pripravili. Vsakega od teh razredov smo podrobneje opisali in razložili njegov namen.

Javanski razred Form

Prvi del namestniške javanske storitve, ki smo ga razvili je bil razred Form. Znotraj tega razreda smo prebrali vrednosti vhodnih polj obrazca iz prvega dela ovojnice, nato pa na podlagi teh vrednosti sestavili začetni spletni naslov od koder smo lahko začeli pridobivati delno strukturirane spletne podatke.

Javanski razred ResultList

V drugem delu namestniške javanske storitve smo prebrali vse vrednosti XPath izrazov, ki smo jih shranili pri ustvarjanju drugega dela ovojnice. Na podlagi teh XPath izrazov smo pridobili podatke posameznega zadetka iz zadnje strani seznama zadetkov. Nato smo se sprehodili od prve do predzadnje strani seznama zadetkov in pridobili delno strukturirane podatke posameznega zadetka. Pri pridobivanju spletnih podatkov posameznega zadetka, smo pridobili še podrobne podatke posameznega zadetka, kjer smo si pomagali z metodami iz Javanskega razreda ResultDetails.

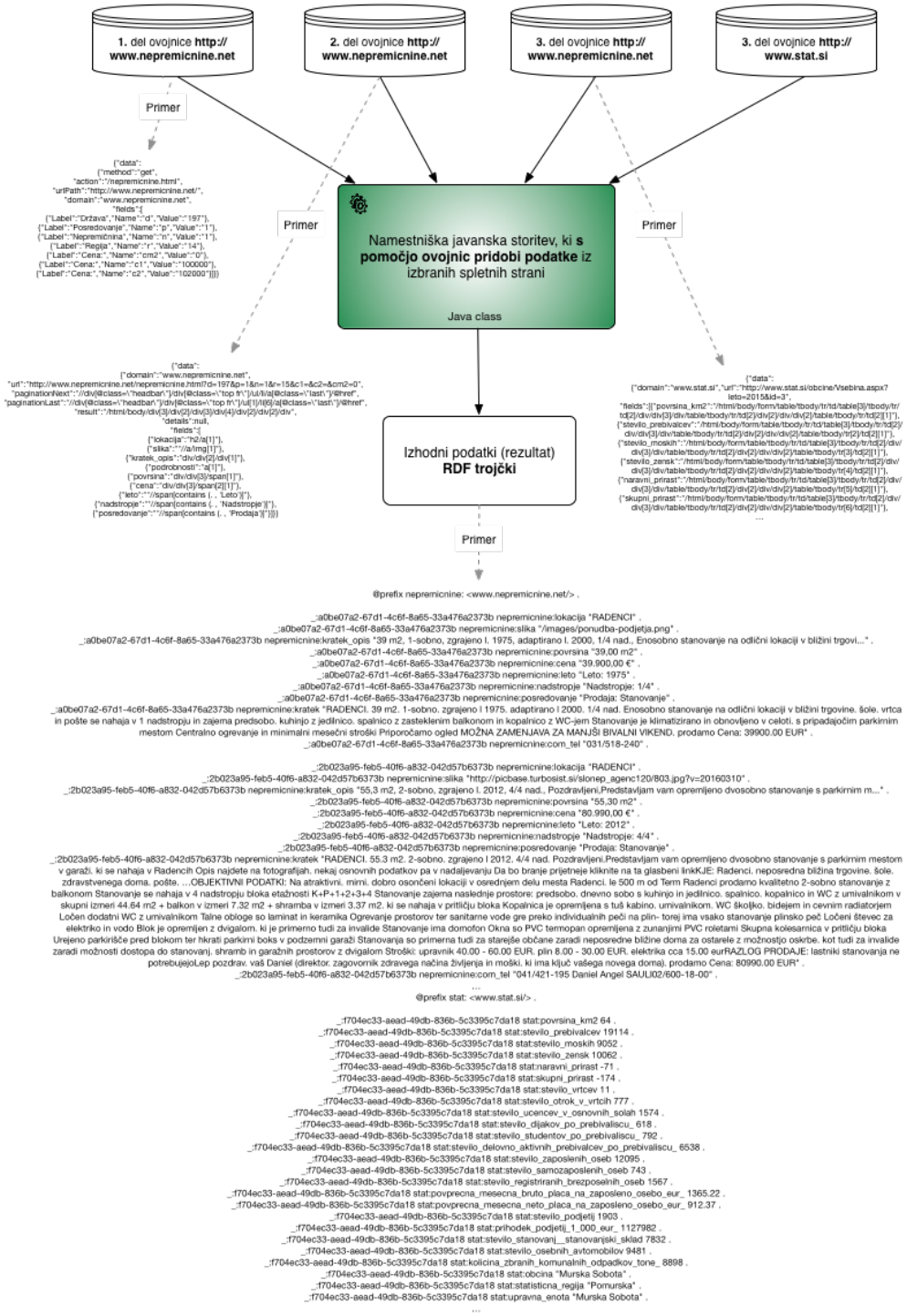
Javanski razred ResultDetails

Tretji del namestniške javanske storitve je razred ResultDetails, znotraj katerega smo prebrali vse vrednosti XPath izrazov tretjega dela ovojnice. Na podlagi teh XPath izrazov smo pridobili podrobne podatke posameznega zadetka. Metode tega razreda so bile klicane znotraj razreda ResultList.

Javanski razred Proxy

Zadnji del je razred Proxy, ki vsebuje metodo parse, katero kličemo iz kartuše. Znotraj te metode združimo klice metod iz ostalih treh razredov in kot rezultat vrnemo niz trojčkov, pridobljenih iz različnih spletnih virov.

Na Sliki 7.10 lahko vidimo, da namestniška javanska storitev najprej prebere dele ovojnice spletnih strani http://www.nepremicnine.net in http://stat.si, potem pa s pomočjo teh delov ovojnic izlušči podatke in jih vrne v RDF obliki trojčkov.



Slika 7.10: Primer vhodnih in izhodnih podatkov namestniške javanske storitve.

Poglavje 8

Vrednotenje predlagane metode in vizualizacija pridobljenih delno strukturiranih spletnih podatkov

V tem poglavju smo najprej predstavili, kako smo izvajali teste predlagane metode, kateri so pokazatelji prednosti naše predlagane metode v primerjavi z že obstoječimi metodami.

Izbrali smo si množico testnih spletnih strani, na katerih smo testirali prototip orodja inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov.

Da smo lahko testiranje opravili, smo si postavili metrike, na podlagi katerih smo lahko predstavili rezultate testov. Glavne metrike, na podlagi katerih smo izvajali teste, so bile: pridobivanje delno strukturiranih spletnih podatkov na dinamičnih spletnih straneh, zahtevano tehnično znanje poslovnega uporabnika pri uporabi prototipa orodja, čas, ki ga naša metoda porabi, da pridobi željene spletne podatke, pravilnost delovanja algoritma za iskanje ponovljenih vzorcev zadetkov, zaščitenost spletnih strani.

Na koncu smo spletne podatke, pretvorjene v štiri- oz. petzvezdično

obliko, še vizualizirali in s tem prikazali dodano vrednost predlagane metode.

8.1 Testna množica spletnih strani

Za testno množico smo poskušali izbrati čim bolj raznolike spletne strani, da smo lahko testirali, kako se obnaša prototip orodja inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov. Te izbrane strani vsebujejo podatke, ki so vsebinsko povezani, zato smo na podlagi pridobljenih podatkov iz teh strani lahko logično sklepali.

Odločili smo se, da bomo primerjali podatke dveh poljubnih občin v Republiki Sloveniji. Na podlagi tega smo si izbrali naslednje tri spletne strani, iz katerih smo pridobili podatke:

- http://www.nepremicnine.net,
- http://www.stat.si,
- http://wiki.dbpedia.org.

Iz prve spletne strani smo pridobili vse podatke o nepremičninah, ki se prodajajo, kupujejo, dajejo v najem ali najemajo v posamezni občini.

Iz druge statistične podatke posamezne občine.

Iz tretje, ki pa je nekoliko drugače formatirana in vsebuje dostop do podatkov že v obliki podatkovnega grafa RDF pa še dodatne podatke, katerih na preostalih dveh spletnih straneh ne moremo najti.

8.2 Testiranje predlagane metode na podlagi podanih metrik

8.2.1 Dinamične spletne strani

Kot smo omenili že v prejšnjih poglavjih je ena glavnih težav, ki jo imajo že obstoječi pristopi, pridobivanje delno strukturiranih podatkov iz dinamičnih

spletnih strani. Na teh spletnih straneh lahko srečamo velik delež programske kode, napisan v programskem jeziku JavaScript, kateri omogoča manipulacijo spletnih gradnikov med samim brskanjem po spletni strani.

Tako lahko lastniki spletnih strani s pomočjo programskega jezika JavaScript ovirajo bote ali druge spletne storitve pri pridobivanju podatkov iz njihovih strani.

Problem dinamičnih spletnih strani smo rešili tako, da smo znotraj našega prototipa inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov vklopili izvajanje programskega jezika JavaScript, ki se izvaja na strani odjemalca. To smo izvedli znotraj uporabljenega ogrodja HtmlUnit, z odsekom programske kode, katero smo podali spodaj.

```
WebClient webClient = new WebClient(BrowserVersion.
    CHROME);
webClient.getOptions().setJavaScriptEnabled(true);
webClient.getOptions().setThrowExceptionOnScriptError(
    false);
```

Ko vklopimo izvajanje programskega jezika JavaScript znotraj ogrodja HtmlUnit, nam ta najprej prenese osnovno spletno stran v delovni pomnilnik, se sprehodi skozi stran in prenese vse vključene JavaScripte skripte na spletni strani ter vse te JavaScript skripe izvede in nam vrne spletno stran z dinamično ustvarjeno vsebino.

Primer testiranja smo izvedli na preprosti spletni strani, ki vsebuje tabelo z delno dinamično ustvarjeno vsebino. V primeru, ko smo v spletnem brskalniku na strani odjemalca izklopili izvajanje JavaScripta, smo dobili spletno stran s tabelo, prikazano na Sliki 8.1.

Če smo pa vklopili izvajanje JavaScripta, pa smo dobili dopolnjeno tabelo z dinamično ustvarjeno vsebino (dvakrat podatek JavaScript Loaded), kar lahko vidimo na Sliki 8.2.

Delovanje prototipa inteligetnega pomočnika na dinamični spletni strani smo testirali tako, da smo s pomočjo prototipa orodja inteligentnega pomočnika

Test [1][1]	
Test [2][1]	Test [2][2]
Test [3][1]	Test [3][2]
Test [4][1]	Test [4][2]
Test [5][1]	
Test [6][1]	Test [6][2]

Slika 8.1: Primer preproste spletne strani z izklopljenim izvajanjem JavaScripta na strani odjemalca.

Test [1][1]	JavaScript Loaded
Test [2][1]	Test [2][2]
Test [3][1]	Test [3][2]
Test [4][1]	Test [4][2]
Test [5][1]	JavaScript Loaded
Test [6][1]	Test [6][2]

Slika 8.2: Primer preproste spletne strani z vklopljenim izvajanjem JavaScripta na strani odjemalca.

pridobili podatke iz te spletne strani, kjer je delno dinamični ustvarjena vsebina. Primere teh trojčkov smo prikazali v spodnjem odseku kode. Na enak način smo pridobili dinamično ustvarjeno vsebino iz izbrane testne množice spletnih strani.

```
@prefix localhost: <http://localhost.si/> .

_:188226f1-4047-4fa6-ae0c-b3c39d903f95 localhost:
   Test_1_1_ "JavaScript Loaded" .
_:188226f1-4047-4fa6-ae0c-b3c39d903f95 localhost:
   Test_2_1_ "Test [2][2]" .
_:188226f1-4047-4fa6-ae0c-b3c39d903f95 localhost:
   Test_3_1_ "Test [3][2]" .
_:188226f1-4047-4fa6-ae0c-b3c39d903f95 localhost:
   Test_4_1_ "Test [4][2]" .
_:188226f1-4047-4fa6-ae0c-b3c39d903f95 localhost:
   Test_5_1_ "JavaScript Loaded" .
_:188226f1-4047-4fa6-ae0c-b3c39d903f95 localhost:
   Test_6_1_ "Test [6][2]" .
```

Iz primera trojčkov lahko vidimo, da je naš inteligentni pomočnik uspešno pridobil dvakratno vrednost JavaScript Loaded, ki je bila dinamično ustvarjena s pomočjo izvajanja JavaScripta.

8.2.2 Zahtevano tehnično znanje poslovnega uporabnika

Večina obstoječih orodij na področju pridobivanja delno strukturiranih spletnih podatkov od poslovnega uporabnika zahteva določen nivo tehničnega znanja, da lahko le-ta uspešno pridobiva željene podatke. Zato smo prototip orodja inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov poskušali implementirati na način, da bi bil čim bolj enostaven za uporabo in prijazen do poslovnega uporabnika.

Spodaj smo podali tabelo, v kateri vidimo primerjavo zahtevanih znanj za pridobivanje delno strukturiranih spletnih podatkov našega orodja, v primerjavi z že obstoječimi rešitvami.

Obrazložene vrednosti iz tabele:

Orodje	Zahtevano znanje
Fake	3
Semantic Fire	5
Apache Any23	5
IRobotSoft	3
Selenium IDE	1
Visual Web Ripper	1
Prototip inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov	1

Tabela 8.1: Zahtevana tehnična znanja obstoječih pristopov

- 1 - zahtevano osnovno znanje poznavanja dela z računalnikom,
- 3 - zahtevano osnovno znanje enega programskega jezika,
- 5 - zahtevano osnovno znanje vsaj dveh programskih jezikov.

Res je, da nekatera že obstoječa orodja ne zahtevajo tehničnega znanja poslovnega uporabnika, vendar ta orodja ali ne omogočajo neposredno pretvorbo podatkov v štiri- ali celo petzvezdično obliko podatkovnega grafa RDF ali so plačljiva ipd.

8.2.3 Čas za pridobitev delno strukturiranih spletnih podatkov

S prototipom orodja inteligetnega pomočnika smo želeli pokazati, da z njegovo pomočjo lahko pridobimo večjo količino podatkov hitreje, kot če bi te podatke pridobivali s pomočjo ročne metode kopiraj-prilepi ali katere druge metode. Ko imamo enkrat pripravljeno ovojnico spletne strani, nam te strani ni potrebno ponovno več obiskati, ampak samo naredimo poizvedbo s pomočjo poizvedovalnega jezika SPARQL.

Akcija	**Inteligentni pomočnik**	**Metoda ročno kopiraj-prilepi**
Čas za ustvarjanje ovojnice	21s	0s
Čas pridobivanja podatkov posameznega zadetka (zaledni del)	8.57s	72s
Čas pridobivanja podatkov zadetkov iz 50 strani (zaledni del)	11998s	100800s

Tabela 8.2: Časi pridobivanja delno strukturiranih spletnih podatkov

Primerjali smo čas, ki ga porabimo za pridobitev delno strukturiranih spletnih podatkov s pomočjo prototipa orodja inteligetnega pomočnika in čas, ki ga porabimo za pridobitev delno strukturiranih spletnih podatkov, če podatke ročno označimo, kopiramo in prilepimo v orodje Microsoft Excel 2013. Rezultate smo prikazali v Tabeli 8.2.

$$faktor\ pohitritve = \frac{\text{čas počasnejše metode}}{\text{čas hitrejše metode}}$$

Pri enem zadetku dobimo:

$$\frac{72s}{21s + 8.57s} = 2.435$$

Pri zadetkih iz 50 strani dobimo:

$$\frac{100800s}{21s + 11998s} = 8.387$$

Te teste smo izvedli na spletni strani http://www.nepremicnine.net, primerljiv izkupiček pa se pričakuje na podobnih straneh.

Iz rezultatov testa vidimo, da smo s prototipom orodja precej pohitrili postopek pridobivanja delno strukturiranih spletnih podatkov. Pri enem samem zadetku smo porabili manj kot polovico časa za pridobitev njegovih

Spletna stran	**Število poskusov**
http://www.nepremicnine.net	5
http://www.salomon.si	5
http://mobile.de	6
http://odpiralnicasi.com	7
http://avto.net	10
http://superge.si	16

Tabela 8.3: Število poskusov, da algoritem najde ponovljene vzorce zadetkov

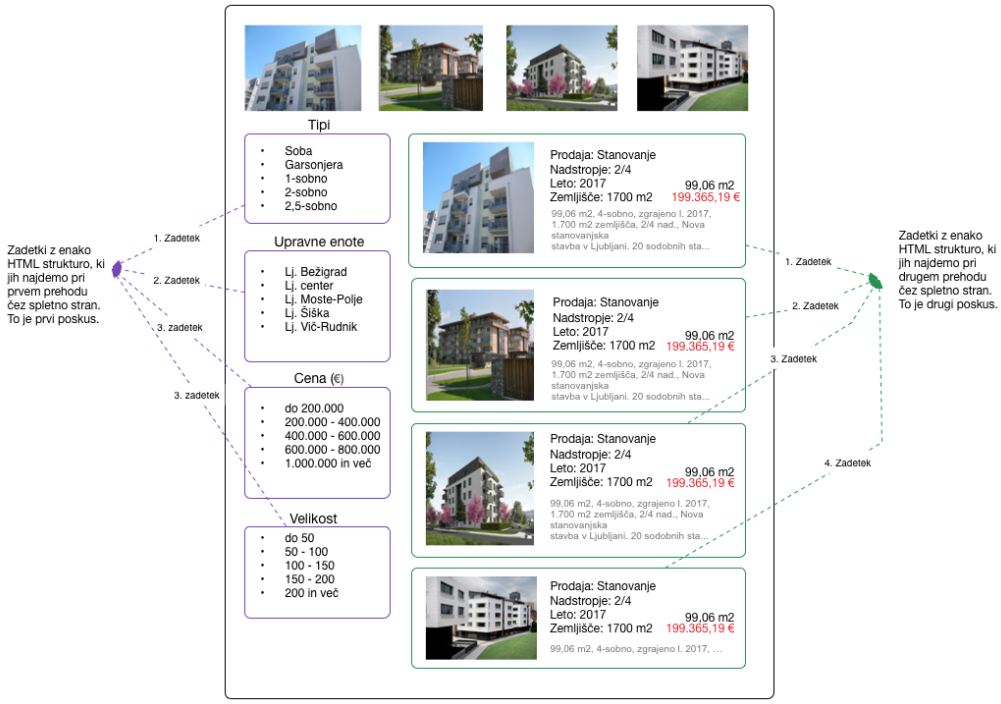
podatkov, v primerjavi z metodo ročnega kopiranja in prilepljanja podatkov, na straneh 50 zadetkov pa približno osemkrat manj časa.

Prav tako pa pri inteligetnem pomočniku dobimo izhodne podatke v obliki strukturirane oblike podatkovnega grafa RDF, v Microsoft Excelu pa le v obliki tabele.

8.2.4 Pravilnost delovanja algoritma za iskanje ponovljenih vzorcev zadetkov

Ena izmed metrik, katero smo želeli izmeriti, je pravilnost delovanja algoritma za iskanje ponovljenih vzorcev zadetkov. To je eden izmed pomembnejših algoritmov znotraj našega orodja prototipa inteligetnega pomočnika. Iz zgoraj podane testne množice vsebuje ponovljene vzorce zadetkov le spletna stran http://www.nepremicnine.net. Zato smo za testiranje te metrike poiskali še nekatere druge spletne strani, ki vsebujejo ponovljene vzorce zadetkov, da smo lahko izmerili pravilnost delovanja algoritma. Za prikaz pravilnosti delovanja algoritma smo merili, v katerem poskusu algoritem uspešno najde željene ponovljene vzorce zadetkov. Kot poskus smo v našem primeru določili vsak posamezni prehod algoritma čez množico ponovljenih vzorcev zadetkov. Primera dveh poskusov lahko vidimo na Sliki 8.3

Rezultate testiranja smo podali v Tabeli 8.3. Iz rezultatov testiranja je razvidno, da na večini spletnih strani algoritem približno v šestem poskusu



Slika 8.3: Primer dveh prehodov algoritma čez spletno stran, vsak od njiju predstavlja poskus.

najde pravi vzorec ponovljenih zadetkov. Pri množici iz tabele izstopata spletni strani http://avto.net in http://superge.si, kjer algoritem najde pravi vzorec ponovljenih zadetkov kasneje. Glavni razlog za to so različni seznami na spletni strani, ki so na spletni strani na višjem nivoju kot željeni zadetki. Težave posebej povzročajo oglasi, ki so na spletni strani, saj imajo le-ti med sabo enako HTML strukturo in jih posledično algoritem tudi prepozna kot potencialne zadetke.

Iz podatkov iz Tabele 8.3 lahko izračunamo, da mora algoritem v povprečju devetkrat iterirati skozi polje najdenih zadetkov; v deseto nam bo verjetno ponudil in označil željeni vzorec ponovljenih zadetkov na spletni strani.

8.2.5 Zaščitenost spletnih strani

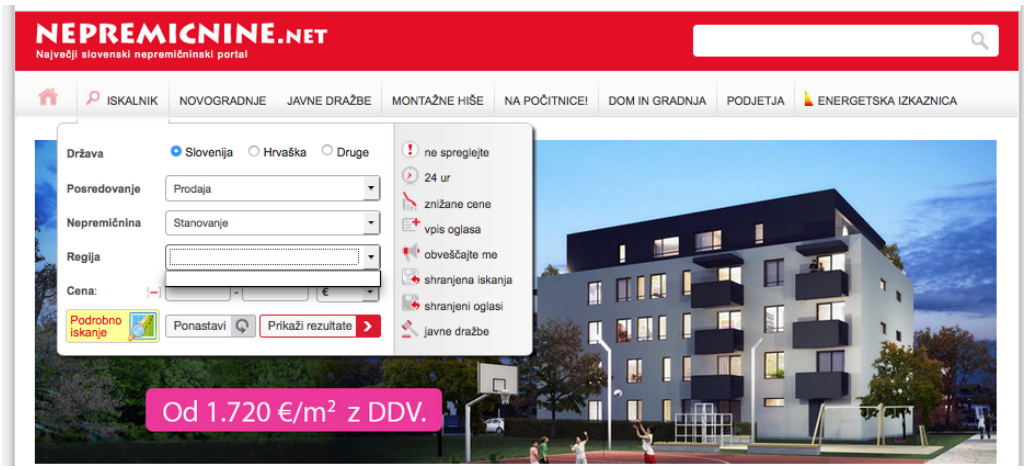
Pri testiranju zaščitenosti spletnih strani smo morali testirati, katere ovire za pridobivanje delno strukturiranih spletnih podatkov je lahko naš inteligentni pomočnik obšel.

Prva taka stvar, ki nam je povzročila težave, je bila odpiranje željene spletne strani znotraj HTML značke iframe. Pri nekaterih spletnih straneh npr. http://www.nepremicnine.net smo si poskušali pomagati z atributom sandbox, vendar tudi to ni bilo dovolj. Stran smo poskušali odpreti na naslednji način:

```
<iframe id="iFrame" src="http://www.nepremicnine.net"
   frameborder="0" sandbox="allow-forms allow-same-
   origin"></iframe>
```

Z zgornjo vrstico smo omogočili predložitev iskalnih obrazcev in vsebini iframea nastavili, da si jo interpretiramo z enakim izvorom kot naša primarna spletna stran. S tem smo uspešno odprli spletno stran znotraj iframea, brez preusmeritve na drugo stran, saj smo onemogočili JavaScript skripte, s tem da smo izpustili vrednost allow-scripts. Vendar se nam je tukaj pojavil drug problem: ker pri atributu sandbox nismo imeli nastavljene ravno te vrednosti allow-scripts, smo posledično onemogočili vse skripte znotraj iframea. S tem smo onemogočili tudi AJAX klice do uporabniškega vmesnika na spletni strani. Zaradi onemogočenih AJAX klicev se nam na prvem delu spletne strani v glavni iskalni obrazec niso naložile vse vrednosti, kar pa pomeni, da ovojnice nismo mogli uspešno ustvariti . V primeru, da bi atributu sandbox dodali vrednost allow-scripts, bi nas pa spet samodejno preusmerilo na drugo spletno stran, saj bi s tem spet ponovno omogočili vse skripte spletne strani http://www.nepremicnine.net. Na Sliki 8.4 lahko vidimo, da se nam zaradi izklopa skript ne naložijo vrednosti opcijskega izbirnega polja pri regiji.

S tem smo ugotovili, da iz nekaterih bolj zapletenih spletnih strani kot je npr. http://www.nepremicnine.net, ne bomo mogli pridobivati delno struk-



Slika 8.4: Primer, kako se nam zaradi izklopa skript znotraj HTML značke iframa preko AJAX klicev ne naložijo vrednosti opcijskega izbirnega polja pri regiji.

turiranih spletnih podatkov, tako da jih bomo odprli znotraj HTML značke iframe. Lahko pa podatke vseeno pridobivamo tako, da jih odpremo neposredno v brskalniku in vključimo našo Greasmonkey skripto, s pomočjo katere ustvarimo ovojnico za izbrano spletno stran.

Tako smo spletne strani razdelili na dva dela, na tiste bolj preproste (manj zaščitene ali nezaščitene), za katere lahko ovojnico za pridobivanje delno strukturiranih spletnih podatkov ustvarimo tako, da jih odpremo znotraj HTML značke iframe, in na tiste bolj zahtevne (zelo zaščitene), za katere ovojnico ustvarimo tako, da jih odpremo neposredno v brskalniku.

Druga stvar, katero smo opazili pri testiranju inteligentnega pomočnika na spletni strani http://www.nepremicnina.net, je ta, da ima vsakič, ko osvežimo spletno stran s seznamom ponovljenih zadetkov, vsak izmed zadetkov drugačno vrednost atributa razred. Lastniki spletnih strani so s tem skušali doseči, da nebi bilo možno pridobivanje delno strukturiranih spletnih podatkov iz njihove strani v primeru, da bi imeli zgrajene XPath izraze, ki so vezani na razreda HTML elementa. Naš algoritem za ustvarjanje XPath izrazov tukaj ni imel nobenih težav.

Nekateri lastniki spletnih strani sestavijo spletno stran semantično nepravilno, da bi bilo onemogočeno avtomatsko pridobivanje delno strukturiranih spletnih podatkov. Določeni se odločijo tudi, da znotraj enega elementa podajo dva podatka in ju postavijo - vsakega v novo vrstico z - novovrstčnim HTML elementom $< br/ >$.

Med samim testiranjem so lastniki spletne strani http://www.nepremicnine.net integrirali CAPTCHO. Ta je popolnoma onemogočila delovanje ogrodja HtmlUnit, s pomočjo katerega smo pridobivali delno strukturirane spletne podatke. Na srečo smo delno strukturirane spletne podatke pridobili že pri prvih testiranjih in jih vpisali v graf, tako da smo jih imeli shranjene za kasnejšo poizvedbo.

8.3 Vizualizacija pridobljenih podatkov

Na koncu smo podatke, pridobljene v obliki podatkovnega grafa RDF, še vizualizirali. Te smo znotraj arhitekturne rešitve Virtuoso izvozili v JSON obliko in shranili. S spodnjim SPARQL stavkom smo pridobili podatke o občini Lendava iz vseh treh spletnih strani testne množice. Podobno smo storili za drugo občino, le da smo vrednost lokacija, občina in label zamenjali z njenim imenom, da smo lahko potem podatke med seboj primerjali.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX stat: <http://www.stat.si/>
PREFIX nepremicnine: <http://www.nepremicnine.net/>
SELECT ?s ?p ?o
WHERE {
  {
    ?s ?p ?o .
    ?s nepremicnine:lokacija ?lokacija .
    FILTER regex(?lokacija, "Lendava", "i")
  } UNION {
    ?s ?p ?o .
    ?s stat:obcina ?obcina .
    FILTER regex(?obcina, "Lendava", "i")
  }
   UNION {
    ?s ?p ?o .
    ?s rdfs:label ?label .
    FILTER regex(?label, "Lendava", "i")
  }
}
```

Z vizualizacijo podatkov v JSON obliki smo pokazali dodano vrednost štiri- oz. petzvezdične oblike podatkovnega grafa RDF.

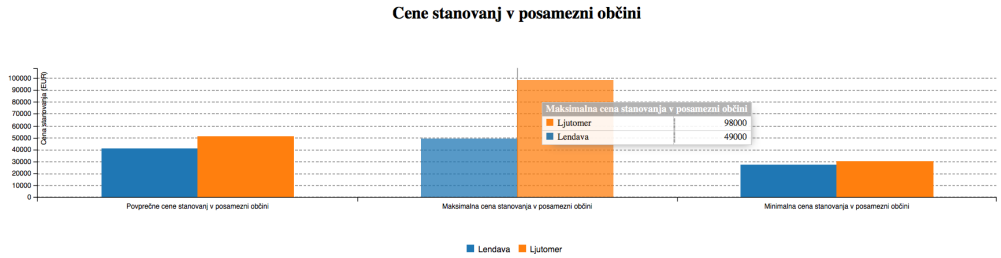
Ker ni bil primarni namen magistrskega dela vizualizacija podatkov, smo za ta namen razvili le preprosto spletno aplikacijo za prikaz podatkov. Za to smo uporabili knjižnico C3.js [20], ki bazira na knjižnici D3.js [21], katera nam služi za vizualizacijo podatkov.

Knjižnica C3.js nam omogoča, da objamemo D3.js knjižnico s programsko kodo in zgradimo grafe. Tem grafom lahko po meri nastavimo stil, obliko, ipd. Prednost knjižnice C3.js je, da omogoča prikaz vhodnih podatkov v JSON obliki.

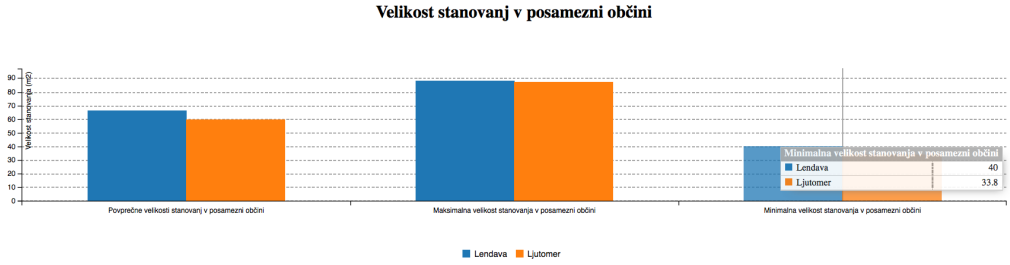
Del podatkov smo prikazali s pomočjo spletne storitve Google Maps, ka-

tera zagotavlja podrobne geografske podatke o geografskih regijah in straneh okrog sveta. Programski vmesnik Google Maps nam omogoča, da si prilagodimo zemljevide in informacije na njih.

Na spodnjih slikah lahko vidimo vizualizacijo zanimivih podatkov, ki so nam npr. v pomoč, če se odločamo za nakup nepremičnine v izbrani občini v Republiki Sloveniji.

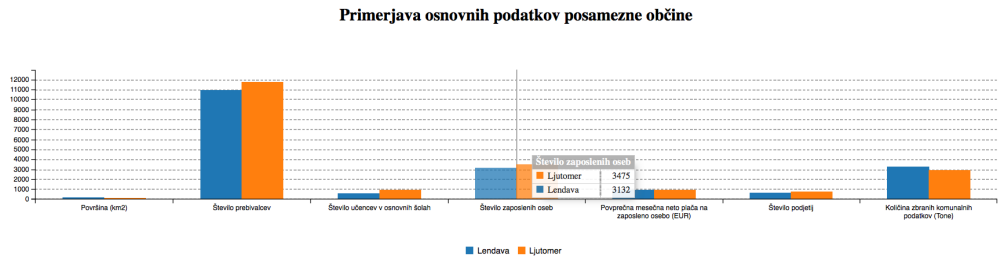


Slika 8.5: Cene stanovanj v občinah Lendava in Ljutomer, podatki pridobljeni iz spletne strani http://www.nepremicnine.net.

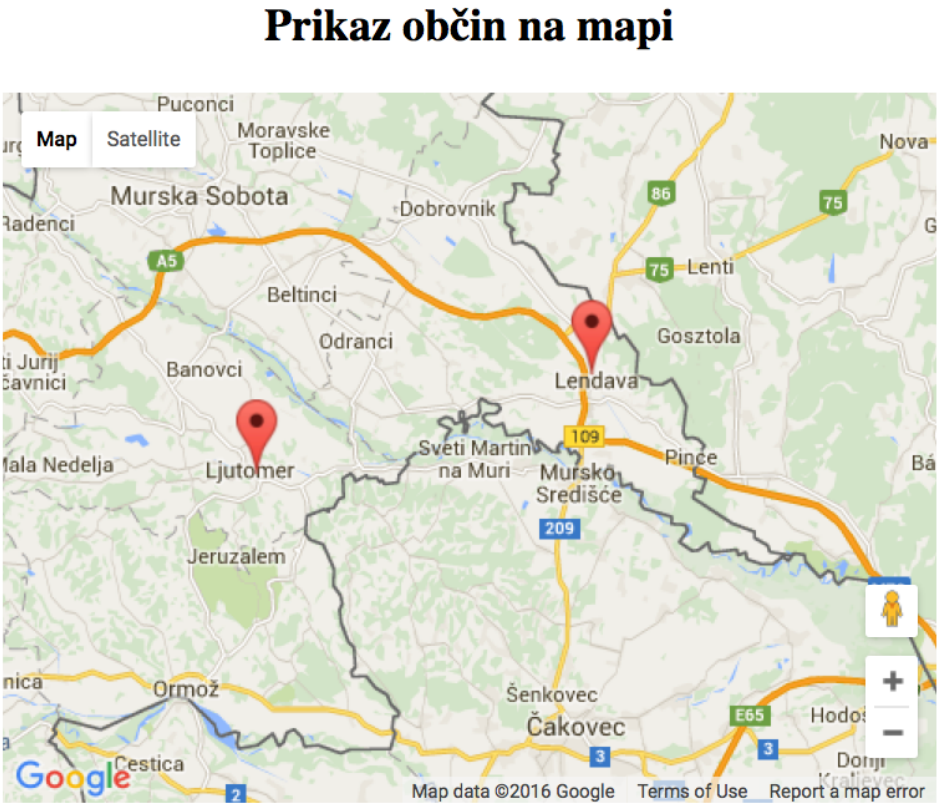


Slika 8.6: Velikost stanovanj v občinah Lendava in Ljutomer, podatki pridobljeni iz spletne strani http://www.nepremicnine.net.

Na podlagi vizualiziranih podatkov iz treh različnih virov lahko sklepamo, kakšne so cene nepremičnin, glede na velikost, si pogledamo osnovne podatke o posamezni občini, npr. koliko neto zaslužijo zaposleni, prav tako pa si na zemljevidu lahko ogledamo, kje se nahaja katera občina, blizu katere meje ipd. Iz teh podatkov se lažje odločimo za nakup nepremičnine, če npr. izbiramo med dvema občinama.



Slika 8.7: Osnovni podatki občin Lendava in Ljutomer, podatki pridobljeni iz spletne strani http://www.stat.si.



Slika 8.8: Prikaz občin Lendava in Ljutomer na zemljevidu, podatki pridobljeni iz spletne strani http://wiki.dbpedia.org.

Poglavje 9

Sklepne ugotovitve

Namen te magistrske naloge je bila izboljšava že obstoječih pristopov s področja pridobivanja delno strukturiranih spletnih podatkov. Najprej smo pregledali obstoječe pristope s tega področja. Ugotovili smo, da bi z združitvijo različnih pristopov skupaj dobili enovito rešitev, ki bi nam omogočila pridobivanje delno strukturiranih spletnih podatkov iz dinamičnih spletnih strani, te podatke pretvoriti v 4* oz. celo 5* obliko podatkovnega grafa RDF, ter poslovnemu uporabniku dala možnost, da na enostaven način pridobiva željene spletne podatke.

V nadaljevanju smo podrobneje opisali konceptualni predlog te rešitve, ki združuje prednosti posameznih pristopov. S tem konceptualnim predlogom rešitve smo želeli odpraviti zgoraj navedene pomanjkljivosti že obstoječih pristopov. Pri konceptualnem predlogu smo podrobneje opisali hiter razvoj prepletene storitve aplikacije, komponento, ki uporabniku pomaga pridobivati spletne podatke med brskanjem po spletni strani. Konceptualni predlog te rešitve smo tudi umestili znotraj arhitekturne rešitve Virtuoso.

Predlagani konceptualni predlog rešitve smo tudi implementirali in ga predstavili še iz praktičnega vidika. Pri predstavitvi praktičnega dela predlagane konceptualne rešitve smo podrobneje opisali vse komponente, ki sestavljajo to rešitev. Za potrebe praktičnega delovanja rešitve smo morali razviti kar nekaj algoritmov, kateri so nam predstavljali enega izmed večjih

izzivov magistrskega dela.

Za konec smo izvedli še testiranje prototipa inteligetnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov. Te teste smo izvedli na predlagani testni množici, v katero smo postavili čim bolj različne spletne strani, dinamične, statične, zaščitene ipd. Odkrili smo, da naš podani pristop rešuje težave že obstoječih pristopov. Omogoča nam pridobivanje delno strukturiranih spletnih podatkov iz zelo dinamičnih spletnih strani, ti podatki, ki jih pridobi, so že v 4* oz. celo v 5* obliki podatkovnega grafa RDF, za pridobivanje teh podatkov pa poslovni uporabnik ne potrebuje tehničnega znanja, saj podatke prototip orodja označi samodejno ali jih pa uporabnik s pomočjo ročnega označevanja označi po potrebi in tako dopolni shemo.

Poleg same primerjave z obstoječimi rešitvami smo izvedli še nekaj dodatnih testov. S temi testi smo ugotovili, da orodje prototipa inteligetnega pomočnika precej hitreje pridobi delno strukturirane spletne podatke v - primerjavi z metodo ročno kopiraj-prilepi.

Njegova glavna slabost je, da lahko lastniki spletnih strani vendarle zaščitijo svojo spletno stran tako, da pridobivanje delno strukturiranih spletnih podatkov iz te strani sploh ni mogoče. Podobno težavo pa imajo tudi drugi pristopi, saj se dandanes vsi lastniki spletnih strani borijo proti pridobivanju podatkov iz njihovih spletnih strani.

V zadnjem delu testiranja smo še vizualizirali podatke, katere smo dobili med samim testiranjem orodja, to vizualizacijo smo izvedli s pomočjo nekaterih JavaScript knjižnic, z njo pa smo pokazali dodano vrednost predlagane in implementirane rešitve inteligentnega pomočnika za pridobivanje delno strukturiranih spletnih podatkov.

Literatura

[1] Vmesna programska komponenta arhitekturne rešitve Virtuoso. Dostopno na: `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger/` (pridobljeno 4.2.2015)

[2] Programska rešitev Semantic Fire. Dostopno na: `hhttps://code.google.com/p/semantic-fire/` (pridobljeno 8.7.2015)

[3] Programska rešitev Visual Web Ripper. Dostopno na: `http://www.visualwebripper.com/` (pridobljeno 16.5.2015)

[4] Programska rešitev Fake. Dostopno na: `http://fakeapp.com/documentation/` (pridobljeno 16.5.2015)

[5] Programska rešitev IRobotSoft Dostopno na: `http://www.irobotsoft.com/` (pridobljeno 17.5.2015)

[6] Programska rešitev Selenium IDE Dostopno na: `http://www.seleniumhq.org/projects/ide/` (pridobljeno 17.5.2015)

[7] Programske rešitev Apache Any23 Dostopno na: `https://any23.apache.org/` (pridobljeno 19.5.2015)

[8] H. Liu, Y. X. Ma, "Web Data Extraction Research Based on Wrapper and XPath Technology", v zborniku *Advanced Materials Research (Volumes 271 - 273)*, jul. 2011, str. 706 - 712.

[9] S. L. Phye, M. M. Thein, T. T. Win, M. M. S. Thwin, "Semantic Web Information Retrieval in XML By Mapping To RDF Schema", v zborniku

Networking and Information Technology (ICNIT), 2010 International Conference on, Manila, Filipini, jun. 2010, str. 500-503.

[10] D. Amalfitano, A. R. Fasolino, P. Tramontana, "Reverse Engineering Finite State Machines from Rich Internet Applications", v zborniku *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, Antwerp, Belgija, oct. 2008, str. 69-73.

[11] N. K. Tran, K. C. Pham, Q. T. Ha (2010), "XPath-Wrapper Induction for Data Extraction", v zborniku *Asian Language Processing (IALP), 2010 International Conference on*, Harbin, China, dec. 2010, str. 150-153.

[12] HtmlUnit - Welcome to HtmlUnit. Dostopno na: `http://htmlunit.sourceforge.net/` (pridobljeno 14.5.2016)

[13] Greasemonkey. Dostopno na: `http://www.greasespot.net/` (pridobljeno 14.5.2016)

[14] OpenLink Virtuoso Home Page. Dostopno na: `http://virtuoso.openlinksw.com` (pridobljeno 14.5.2016)

[15] Cumulus RDF - Semantic Web Standards - W3C. Dostopno na: `https://www.w3.org/2001/sw/wiki/CumulusRDF` (pridobljeno 14.5.2016)

[16] DBPedia. Dostopno na: `http://wiki.dbpedia.org` (pridobljeno 14.5.2016)

[17] Chrome DevTools Overview - Google Chrome - Google Developers. Dostopno na: `https://developer.chrome.com/devtools` (pridobljeno 14.5.2016)

[18] Firebug. Dostopno na: `https://addons.mozilla.org/sl/firefox/addon/firebug/` (pridobljeno 14.5.2016)

[19] Statistični urad RS. Dostopno na: `http://www.stat.si/statweb` (pridobljeno 14.5.2016)

[20] C3.js — D3-based reusable chart library. Dostopno na: `http://c3js.org` (pridobljeno 14.5.2016)

[21] D3.js - Data-Driven Documents. Dostopno na: `https://d3js.org` (pridobljeno 14.5.2016)

[22] Apache Jena - SDB - persistent triple stores using relational databases. Dostopno na: `https://jena.apache.org/documentation/sdb/` (pridobljeno 15.5.2016)

[23] AllegroGraph - Semantic Graph Database - Franz Inc.. Dostopno na: `http://franz.com/agraph/allegrograph/` (pridobljeno 15.5.2016)

[24] The Apache Cassandra Project. Dostopno na: `http://cassandra.apache.org` (pridobljeno 16.6.2016)

[25] MongoDB for GIANT Ideas — MongoDB. Dostopno na: `https://www.mongodb.com` (pridobljeno 16.6.2016)

[26] IE7pro - Download. Dostopno na: `http://ie7pro.en.softonic.com` (pridobljeno 16.6.2016)

[27] Sleipnir 2 for Windows (2.9.18) released (Sleipnir Browser Blog). Dostopno na: `http://blog.sleipnirbrowser.com/2013/07/sleipnir-2-for-windows-2-9-18-released.html` (pridobljeno 16.6.2016)

[28] Browser Automation, Data Extraction and Web Testing — iMacros. Dostopno na: `http://imacros.net` (pridobljeno 16.6.2016)

[29] GreaseKit - User Scripting for all WebKit applications - 8-p.info. Dostopno na: `http://8-p.info/greasekit/` (pridobljeno 16.6.2016)

[30] C. Tejo-Alonso, D. Berrueta, L. Polo, S. Fernández, "Metadata for Web Ontologies and Rules: Current Practices and Perspectives", v zborniku *Metadata and Semantic Research*, Izmir, Turčija, 2011, str. 56-67.

[31] Antoniou, G. in Harmelen, V. F. *A semantic Web primer* 1. izd. Cambridge, Massachusetts: The MIT Press, 2004. ISBN-262-01210-3.

[32] O. Erling, I. Mikhailov, "RDF Support in the Virtuoso DBMS", v zborniku *Networked Knowledge – Networked Media*, 2009, str. 7-24.

[33] S. Ceri, A. Bozzon, M. Brambilla, E. D. Valle, P. Fraternali, S. Quarteroni, "Multimedia search", v zborniku *Web Information Retrieval*, Milan, Italija, 2013, str. 207-221.

[34] L. Goddard, G. Byrne, "The Strongest Link: Libraries and Linked Data" v zborniku *Emerging Technologies in Academic Libraries*, Trondheim, Norveška, apr. 2010, str 7.

[35] P. Montoto, A. Pan, J. Raposo, F. Bellas, J. López, "Automated browsing in AJAX websites", v zborniku *Data & Knowledge Engineering*, A Coruña, Španija, 2011, str. 269–283.

[36] Primer slike oblaka odprtih podatkov. Dostopno na: `http://lod-cloud.net/` (pridobljeno 27.6.2016)

[37] Predstavitev hrambe trojčkov. Dostopno na: `http://www.dataversity.net/introduction-to-triplestores/` (pridobljeno 27.6.2016)

[38] jQuery. Dostopno na: `hhttps://jquery.com` (pridobljeno 24.8.2016)

[39] Programska koda prototipa orodja za pridobivanje delno strukturiranih spletnih podatkov. Dostopno na: `https://github.com/nadzic/Intelligent-assistant` (pridobljeno 6.7.2016)