Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jaka Logar

Pretvorba delno strukturiranih podatkov spletnega novičarskega portala v RDF obliko

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

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

Št. naloge: 01940 / 2013
Datum: 3.9.2013

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

Kandidat: **JAKA LOGAR**

Naslov: **PRETVORBA DELNO STRUKTURIRANIH PODATKOV SPLETNEGA NOVIČARSKEGA PORTALA V RDF OBLIKO**
TRANSFORMATION OF SEMI-STRUCTURED DATA FROM ONLINE NEWS SITE TO RDF

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Na področju povezanih podatkov, ki je podprto s tehnologijami semantičnega spleta, je na voljo veliko strukturiranih virov iz različnih problemskih domen. Te domene predvsem predstavljajo splošne zbirke, kot je npr. Wikipedija, geolokacijske informacije ipd. Pri razvoju inteligentnih aplikacij, ki takšne strukturirane podatke izkoriščajo pa je pomembno, da so ti objavljeni podatki čim bolj strukturirani in predstavljeni v kontekstu, kar pomeni, da niso zgolj objavljeni, ampak je vsak element naslovljiv z URI naslovom in opisan z metapodatki obstoječe sheme. V okviru diplomske naloge naj študent pripravi prototip informacijske rešitve, ki delno strukturirane podatke spletnega novičarskega portala pretvori standardizirano RDF obliko. Pri podatkih naj se osredotoči na objavljene novice, komentarje, uporabnike ter medsebojne povezave med omenjenimi koncepti. Pokazati je potrebno prednosti takšne predstavitve podatkov in sicer z integracijo z ostalimi strukturiranimi viri, ki nam pri uporabi mehanizma sklepanja omogočajo izvajati poizvedbe po več različnih virih, integracija pa se izvede samodejno preko opredeljene sheme podatkov.

Mentor:

doc. dr. Dejan Lavbič

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jaka Logar, z vpisno številko **63070134**, sem avtor diplomskega dela z naslovom:

Pretvorba delno strukturiranih podatkov spletnega novičarskega portala v RDF obliko

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 29. novembra 2013 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Dejanu Lavbiču za vso pomoč in nasvete pri izdelavi te diplomske naloge. Zahvala gre tudi vsem družinskim članom in prijateljem, ki so me tekom študija spodbujali in podpirali.

Kazalo

Seznam uporabljenih kratic in simbolov

AJAX (Asynchronous JavaScript and XML) - Asinhroni JavaScript in XML
API (Application programming interface) - Vmesnik za programiranje aplikacij
DBpedia - Zbirka podatkov predstavljenih s trojčki RDF, ki shranjuje informacije iz spletne strani Wikipedije
C# - Microsoftov programski jezik
CSV (Comma-separated values) - Datotečni zapis vrednosti ločenih z vejico
DI (Dependency injection) - Odvisnost injiciranja storitev
DOM (Document Object Model) - Standard, ki je neodvisen od jezika in platforme in opisuje objektni model za predstavitev XHTML in XML dokumentov
HTML (Hypertext Markup Language) - Označevalni jezik za izdelavo spletnih strani
HTTP (HyperText Transfer Protocol) - Glavna metoda za prenos informacij na spletu
IP (Internet Protocol) - Internetni protokol
Java - Objektno usmerjeni, prenosljivi programski jezik
Java JRE (Java Runtime Environment) - Izvajalno okolje Java
JSON (JavaScript Object Notation) - Preprost format za izmenjavo podatkov
LINQ (Language-Integrated Query) - Del Microsoftovega .NET ogrodja in

omogoča poizvedbe v .NET programskih jezikih

MVC (Model–View–Controller) - Programska arhitektura, ki ločuje predstavitev informacije od uporabniške interakcije z njo

OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) - Odprt protokol za pridobivanje metapodatkov

OWLIM-Lite - Shramba za podatke RDF razvita v programskem jeziku Java

Python - Tolmačeni programski jezik

RTV Slo - Prvi interaktivni multimedijski portal

RDF (Resource Description Framework) - Osnovni format za zapis podatkov na semantičnem spletu

RDF shema - Uporablja se za opis grafa, ki je sestavljen iz trojčkov

RDF trojček - Izraz (stavek) sestavljen iz subjekta, predikata in objekta

REST (Representational State Transfer) - Prenos reprezentativnega stanja

RSS (Rich Site Summary) - Družina XML datotečnih oblik za spletno zlaganje, ki ga uporabljajo spletne strani

SOAP (Simple Object Access Protocol) - Protokol za prenos podatkov po svetovnem spletu v obliki zapisa XML

SPARQL (Simple Protocol and RDF Query Language) - Poizvedovalni jezik po podatkih RDF

SQL (Structured Query Language) - Strukturirani povpraševalni jezik za delo s podatkovnimi bazami

Tomcat - Odprto kodni spletni strežnik

Ubuntu - Odprto kodni operacijski sistem

URI (Uniform resource identifier) - Niz v standardizirani obliki, ki omogoča enolično poimenovanje virov

URL (Uniform Resource Locator) - Naslov spletnih strani v svetovnem spletu

VMware - Programska oprema za ustvarjanje in poganjanje navideznih računalnikov na gostujočemu sistemu

W3C (World Wide Web Consortium) - Mednarodni inštitut, kjer člani or-

ganizacije in javnost sodelujejo ter skupaj razvijajo standarde za splet
Windows Service - Računalniški program, ki teče v ozadju operacijskega sistema
XHTML (Extensible HyperText Markup Language) - Označevalni jezik, ki ima enak namen kot HTML, vendar je usklajen s sintakso XML
XML (Extensible Markup Language) - Razširljiv označevalni jezik
XPath (XML Path Language) - Poizvedovalni jezik v dokumentih XML

Povzetek

Dandanes vsi obiskujemo spletne strani, na katerih iščemo različne informacije. Slabost večine spletnih strani je, da so zapisane v naravnem jeziku in razumljive zgolj človeškemu uporabniku. Včasih si želimo, da bi te informacije razumel in znal uporabljati tudi računalnik (statistične obdelave podatkov, iskanje zakonitosti v podatkih ...).

V ta namen smo razvili aplikacijo, ki podatke objavljene na spletni strani RTV Slovenije pretvarja v strukturirano obliko razumljivo računalniku. Osredotočili smo se predvsem na objavljene novice, pripadajoče komentarje uporabnikov in uporabniške profile. Pridobljene podatke shranjujemo v trojčkih RDF, saj je to standardizirana oblika za shranjevanje tovrstnih podatkov. Za shranjevanje in pridobivanje podatkov smo uporabili poizvedovalni jezik SPARQL. Velika prednost shranjevanja v taki obliki je tudi možnost povezovanja pridobljenih podatkov s podatki iz drugih virov, ki so prav tako shranjeni v obliki trojčkov RDF. Največji tak primer je DBpedia.

Kot praktičen primer smo iz pridobljenih podatkov izdelali nekaj vizualizacij, ki jih ne bi mogli pripraviti, če podatki ne bi bili v strukturirani obliki, saj bi bila samodejna integracija neizvedljiva.

Ključne besede: RDF, SPARQL, strukturirana oblika, pridobivanje podatkov, RTV Slovenija

Abstract

Nowadays everyone is using web pages to gather some information. But these web pages are written in natural language and understandable only to human users. We would like to achieve that computers would also understand all of the information provided. Examples: statistical processing of data, data mining.

We have developed an application for scraping data from RTV Slovenija web page and saving it in a structured form. Data saved that way are also understandable to computers. Our main goal was to save posts, comments on these posts and user profiles. We are saving the data in RDF triples because it is the standard form for saving those kinds of data. We have used the SPARQL query language for saving and querying data. The main advantage of saving data in RDF triples is the possibility of connecting our data store with other data stores which also have data saved in RDF triples, for instance DBpedia.

Saving data in structured form allowed us to make some visualization examples. We could not make visualization if data weren't saved in structured form.

Keywords: RDF, SPARQL, structured form, scraping data, RTV Slovenija

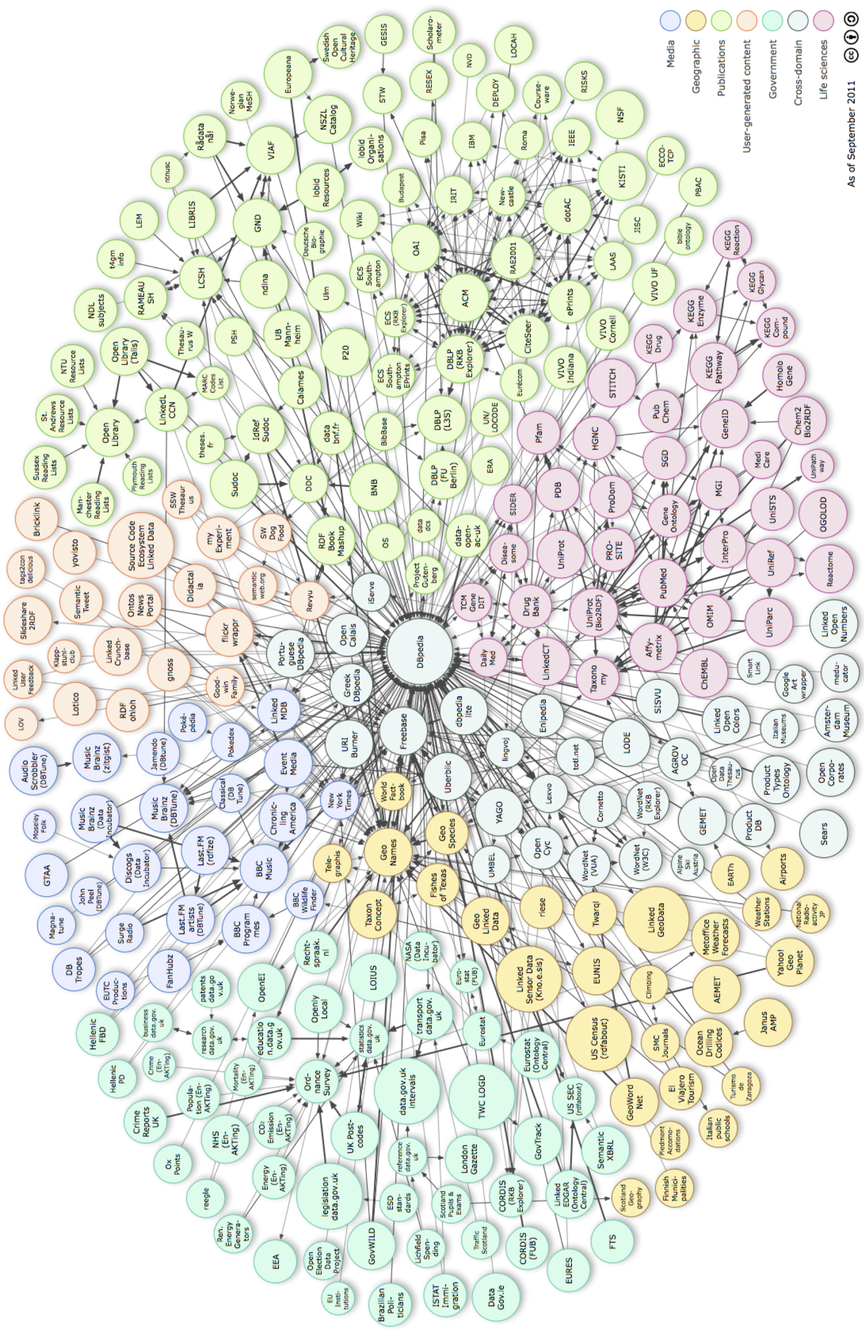
Poglavje 1

Uvod

Sodobne računalniške aplikacije v večini primerov za delovanje uporabljajo lastne baze podatkov. Včasih pa bi inteligentne računalniške aplikacije potrebovale tudi podatke iz drugih virov, saj bi se lahko z združevanjem podatkov iz več virov kvaliteta aplikacije bistveno izboljšala. Da pa lahko povezujemo podatke iz različnih virov med seboj, morajo ti podatki strukturirani in razumljivi računalniškim aplikacijam.

Tipično so taki podatki shranjeni v trojčkih RDF. To je tudi standardiziran zapis in primeren za nadaljnje integracije. Ker pa je zaenkrat takih podatkovnih shramb še zelo malo, smo se v okviru diplomske naloge odločili zgraditi shrambo strukturiranih podatkov, ki bi bili na voljo drugim razvijalcem in aplikacijam. Nekatere obstoječe shrambe in povezovanje teh shramb je prikazanih tudi na sliki 1.1, ki je povzeta iz [25]. Cilj naše naloge je torej ustvariti novo shrambo trojčkov in jo smiselno umestiti v ta graf. To pomeni tudi, da bomo našo shrambo trojčkov povezali z že kakšno od obstoječih shramb. Pridobljene podatke želimo tudi vizualizirati in tako najlažje prikazali prednosti takega načina shranjevanja podatkov.

Za vir podatkov smo izbrali spletno stran RTV Slovenije, ker je ena izmed največjih spletnih strani v Sloveniji in vsebuje ogromno informacij. Tu pa se že pojavi prvi problem, saj nimamo nobenega preprostega načina za pridobitev želenih podatkov. Edina rešitev, ki se pojavi, je luščenje podatkov iz



Slika 1.1: LOD diagram iz septembra 2011

same spletne strani. Slabost takega pristopa je velika odvisnost implementacije aplikacije s strukturo spletne strani. V kolikor bi se v bodoče spremenila struktura spletne strani, bi bilo potrebno tudi našo aplikacijo popravljati. Vendar na izbiro nimamo boljše rešitve in smo se odločili za tak pristop.

Diplomska naloga je razdeljena na 5 delov. V prvem delu bomo povedali malo splošno o RDF in poizvedovalnem jeziku SPARQL, ki se tu najpogosteje uporablja. V drugem delu bomo pregledali obstoječe rešitve in uporabljene pristope. Veliko je na tem področju že narejenega, vendar gre vseeno še za relativno novo področje hranjenja in predstavitve podatkov. V naslednjem poglavju bomo pregledali tehnologije in pristope, ki smo jih uporabili pri izdelavi diplomske naloge. Četrti del bo predstavljal samo implementacijo rešitve in predstavitev težav, s katerimi smo se med izdelavo srečali. V zadnjem delu bomo na pridobljenih podatkih prikazali nekaj primerov vizualizacij. Zgrajeno shrambo podatkov želimo tudi povezati z eno od že obstoječih shramb. Rezultat te povezave bomo poizkusili tudi prikazati v vizualni obliki.

Poglavje 2

RDF oblika zapisa in primerjava z drugimi načini

Namen RDF je predstaviti znanje na internetu, kjer so podatki predstavljeni s trojčkom :

- subjekt (vir - običajno predstavljen z URI [26])
- predikat (tip podatka in relacija med subjektom in objektom - lastnost)
- objekt (vrednost)

Preprost primer: Nebo je modre barve [5].

- subjekt - **nebo**
- predikat - **je barve**
- objekt - **modre**

2.1 Splošno o RDF

Začetki podatkovnega modela RDF segajo v leto 1997, vendar so prve specifikacije tega modela šele iz leta 1999 [3, 5]. Takrat je bil predstavljen, kot

pristop za predstavitev meta podatkov (podatkov o podatkih). S posodobitvijo specifikacije RDF leta 2004 se je šele razvil v obliko, ki jo uporabljamo danes. Vidimo, da je to relativno novo področje v računalništvu. Kljub temu je že postal W3C standard za predstavitev znanja na internetu.

RDF nam predstavlja osnovno metodo za razbitje znanja na koščke. V teh koščkih so zajeta tudi semantična pravila in njihov pomen. In ravno to je ključno, kar omogoča drugim aplikacijam razumevanje in nedvoumno uporabo teh informacij.

Naslednja glavna prednost RDF je tudi, da se dobro obnese pri porazdeljenih informacijah. Trojčki RDF so namreč lahko shranjeni na različnih lokacijah in imajo različne avtorje. Vseeno jih lahko brez problema povežemo med seboj in uporabljamo, kar zagotavlja veliko fleksibilnost.

2.1.1 Definicija RDF

RDF lahko definiramo s tremi pravili [4]:

1. Dejstva so predstavljena s trojčkom - subjekt, predikat, objekt.
2. Subjekti, predikati in objekti predstavljajo imena za entitete v resničnem svetu, ki so lahko resnične ali abstraktne. Imena so lahko:
   (a) Globalna - predstavljajo enako entiteto v vseh dokumentih RDF
   (b) Lokalna - omejena na trenutni dokument RDF
3. Objekti so lahko tudi enostavne vrednosti (literali).

2.1.2 Kako predstaviti RDF

XML

Najbolj tipična predstavitev entitet RDF je v obliki XML dokumentov, ki je tudi edina standardizirana oblika predstavitve po W3C. Taka predstavitev je v splošnem znana tudi kot RDF/XML. Na primeru 2.1 je prikazan zapis entitete v obliki XML.

Primer 2.1: Primer zapisa kategorije v XML obliki

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
    xmlns:sioc="http://rdfs.org/sioc/ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf−schema#"
    xmlns:news=''http://opendata.lavbic.net/news/''
    xmlns=''http://jakalogar.si/mmc/''>

    <sioc:Category rdf:about="http://jakalogar.si/mmc/
            categories/rtvslo_nogomet">
        <rdfs:label>''nogomet''</rdfs:label>
        <news:subCategoryOf rdf:resource="http://jakalogar.si/mmc/
        categories/rtvslo_sport"/>
        <rdfs:seeAlso rdf:resource=''http://www.rtvslo.si/sport/nogomet/''/>
    </sioc:Category>
</rdf:RDF>
```

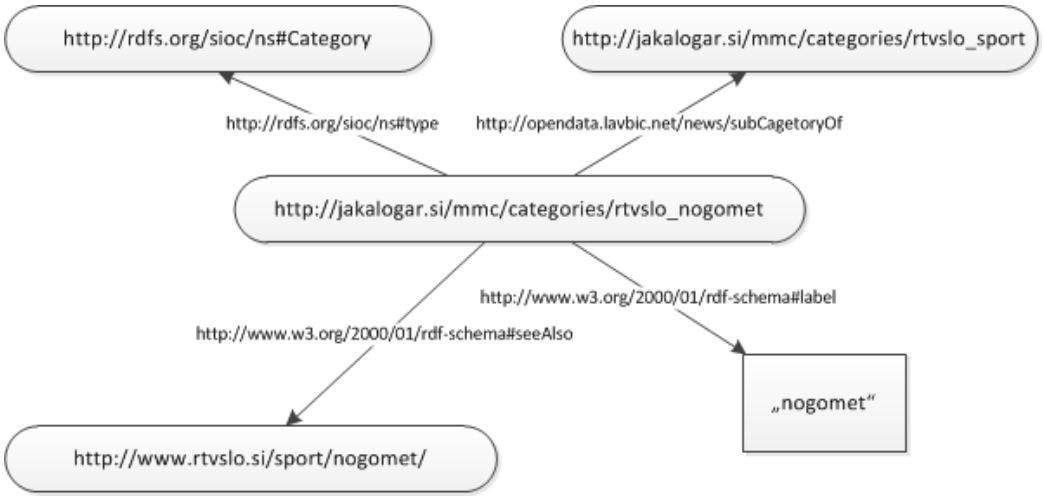
N3 (Notation3) [6]

To je zapis za predstavitev entitet RDF, ki ne temelji na XML formatu. Zaradi tega je precej bolj berljiv in s tem namenom je bil tudi zasnovan. Na primeru 2.2 je prikazan zapis entitete v obliki N3. Gre za enak primer, kot v prejšnji točki, samo da je tokrat uporabljen zapis v obliki N3.

Primer 2.2: Primer zapisa kategorije v N3 obliki

```
@prefix :  <http://jakalogar.si/mmc/> .
@prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
@prefix news: <http://opendata.lavbic.net/news/> .

:rtvslo_nogomet a      sioc:Category .
:rtvslo_nogomet rdfs:label   ''nogomet'' .
```



Slika 2.1: Predstavitev RDF entitete z grafom

```
:rtvslo_nogomet news:subCategoryOf <http://jakalogar.si/mmc/
    categories/rtvslo_sport> .
:rtvslo_nogomet rdfs:seeAlso   <http://www.rtvslo.si/sport/nogomet/> .
```

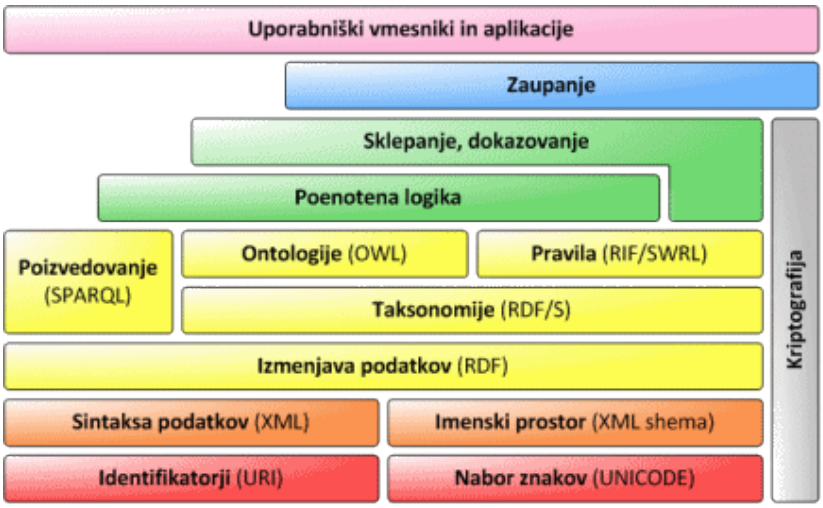
Graf

Graf je še tretja možnost, kako predstaviti RDF trojčke. Na sliki 2.1 je prikazan primer iz prejšnjih dveh točk še v grafični obliki.

2.1.3 Zakaj uporabljati RDF?

Pojavlja se tudi vprašanje, zakaj uporabljati ravno RDF in ne kaj drugega [4]:

- Podatke z različnih virov želimo povezati med sabo brez dodatnega dela.
- Podatke želimo ponuditi tudi drugim za nadaljnjo uporabo.
- Podatke lahko hranimo razpršeno.



Slika 2.2: Arhitektura semantičnega spleta

- Podatki so shranjeni v obliki, ki omogoča lahko manipulacijo (pregled, iskanje).

2.2 Primerjava z XML

Podatke lahko predstavimo tudi v drugih oblikah, poleg zapisa v RDF. Najbližje temu zapisu je XML format zapisa, kar predstavlja tudi slika 2.2, saj je RDF takoj nad XML načinom zapisa. Tudi ta zapis omogoča hierarhično strukturo podatkov. Začetna slabost tega zapisa pa je, da mora biti struktura zapisa vnaprej določena, ki jo mora aplikacija poznati in znati uporabljati. Prav tako obstaja problem razširljivosti. Zapis v XML formatu ima običajno fiksno strukturo in ga je težko razširiti. Pri RDF načinu zapisa teh problemov nimamo. Prav tako obstaja pri XML zapisu velik problem povezovanja entitet med seboj.

Poglejmo si primer, ki je dostopen na [7]:
V XML datoteki imamo seznam produktov z njihovimi imeni. Aplikacija želi omogočati komentarje na posamezne produkte, vendar nastane problem, kako povezati komentarje in produkte med seboj. Edini način je po imenu,

vendar obstaja možnost, da imajo produkti različnih proizvajalcev enako ime. Rešitev je, da uvedemo unikatne identifikatorje. Torej potrebujemo sistem za ustvarjanje unikatnih identifikatorjev. Z uporabo identifikatorjev lahko zagotovimo, da komentar pripada pravemu produktu. Še vedno pa to dvoje ni avtomatsko povezano in potrebujemo aplikacijo za dosego tega.

Uporaba RDF je zaradi tega precej boljša, saj zahteva, da so vsi identifikatorji globalno unikatni in vsak identifikator pomeni nekaj. Vse razen literalov so identifikatorji.

Zaradi takih težav XML ni najbolj primeren za razpršene in razširljive informacije, razen če je XML zapis predstavljen kot RDF.

2.3 Primerjava z relacijskimi bazami

RDF je v osnovi zelo podoben relacijskim bazam, vendar precej bolj fleksibilen. Ko pride do spremembe sheme baze, lahko nastane precejšen problem pri relacijskih bazah, kar vključuje tudi večje spremembe na aplikacijski ravni. Pri RDF se samo trojčki podatkov ustrezno uredijo in je postopek zaključen. Do velikih težav pride tudi, če imamo podatke v več ločenih podatkovnih bazah. Za poizvedovalni jezik SPARQL povezovanje na več shramb podatkov istočasno ne predstavlja težave.

Precejšnja razlika je tudi v konsistentnosti poizvedovalnega jezika SPARQL v primerjavi z SQL. Vsaka implementacija jezika SQL se namreč malenkostno razlikuje od standarda SQL. Zaradi tega je potrebno sintakso poizvedbe prirediti glede na implementacijo podatkovne baze. Pri poizvedbi z jezikom SPARQL nam za to ni potrebno nič skrbeti. Vsako poizvedbo namreč lahko izvedemo na katerikoli končni točki SPARQL, neodvisno od sheme. Poizvedbe SPARQL se prenašajo po HTTP protokolu in to ponovno predstavlja prednost, saj je veliko lažje nastaviti predpomnenje čez protokol HTTP, kot pa na sami relacijski bazi.

Pri uporabi shrambe podatkov RDF namesto relacijskih podatkovnih baz se je vedno potrebno odločiti glede na potrebe in zahteve, saj imata oba

načina shranjevanja svoje prednosti in slabosti.

2.4 Poizvedovalni jezik SPARQL

Poizvedovalni jezik SPARQL (SPARQL Protocol and RDF Query Language) se uporablja za poizvedbe po podatkih shranjenih v formatu RDF. Čeprav je še relativno nov jezik, je že priznan kot ena izmed ključnih tehnologij semantičnega spleta. SPARQL 1.0 je leta 2008 postal uradno priporočilo W3C organizacije, SPARQL 1.1 pa leta 2013 [9].

Velika prednost poizvedovalnega jezika SPARQL je zmožnost povezovanja na različne vire podatkov. Ti podatki so lahko shranjeni v RDF formatu ali jih nek vmesni člen predstavi v obliki RDF. Poizvedbe omogočajo tudi agregacije, negacije, podpoizvedbe, za rezultat pa dobimo množice ali pa graf RDF.

2.4.1 Zgradba poizvedovalnega jezika SPARQL

Primer 2.3: Novice s komentarji

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
2  PREFIX sioc: <http://rdfs.org/sioc/ns#>
3  PREFIX dct: <http://purl.org/dc/terms/>
4  SELECT DISTINCT ?post ?title
5  WHERE {
6      ?post rdf:type sioc:Post .
7      ?post dct:title ?title .
8      ?post sioc:has_reply ?comment .
9      ?comment dct:created ?created .
10     FILTER(?created >= "2013−11−01 00:00:00Z")
11 }
```

Preproste poizvedbe so tipično sestavljene iz treh delov: predpon, rezultata in omejitev. V predponi je pojasnjeno, kaj predstavlja. V primeru 2.3

imamo tri predpone:
`rdf` predstavlja `http://www.w3.org/1999/02/22-rdf-syntax-ns#`,
`sioc` predstavlja `http://rdfs.org/sioc/ns#` in
`dct` predstavlja `http://purl.org/dc/terms/`.

V stavek `SELECT` dodamo vrednosti ali spremenljivke, ki jih želimo v rezultatu. V trenutnem primeru želimo vse unikatne pare [objekt novice, naslov novice], ki zadoščajo pogoju (`?post` in `?title` sta spremenljivki). V poizvedovalnem jeziku SPARQL imajo vse spremenljivke v imenu predpono '?' ali '$'.

V `WHERE` delu poizvedbe dodamo vse omejitve, ki jih želimo doseči. V primeru 2.3 želimo v spremenljivko `?post` shraniti samo objekte, ki so izpeljani iz razreda `sioc:Site`. Hkrati mora imeti ta objekt tudi lastnost `dct:title`, kar predstavlja naslov novice. Novica pa mora imeti tudi povezavo na vsaj en komentar, ta povezava pa je shranjena v lastnosti `sioc:has_reply`. Pri vsakem od teh komentarjev pa želimo, da ima shranjen datum in čas objave in ta datum mora biti od 1.11.2013 dalje. Celotna poizvedba tako vrne unikatne pare povezava na novico in naslov novice, če ima novica vsaj en komentar, ki je bil objavljen od 1.11.2013 dalje.

2.4.2 Oblike poizvedb

V primeru poizvedb, ki samo berejo podatke iz shrambe, jezik SPARQL določa štiri načine poizvedb, ki služijo različnim namenom:

- Poizvedba SELECT je uporabljena za pridobitev surovih podatkov iz končne točke SPARQL v tabelarični obliki.

- Poizvedba CONSTRUCT je uporabljena za pridobivanje podatkov iz končne točke SPARQL, pri čemer podatke preoblikuje v veljaven RDF zapis.

- Poizvedba ASK vrne preprost JA / NE odgovor na poizvedbo.

- Poizvedba DESCRIBE je uporabljena za pridobitev grafa iz končne točke SPARQL. Vsebino grafa določi končna točka na podlagi mnenja vzdrževalca shrambe.

Vsaka od teh poizvedb zahteva tudi stavek WHERE za podajanje omejitev, opcijski je samo pri poizvedbi DESCRIBE.

Poglavje 3

Sorodne rešitve

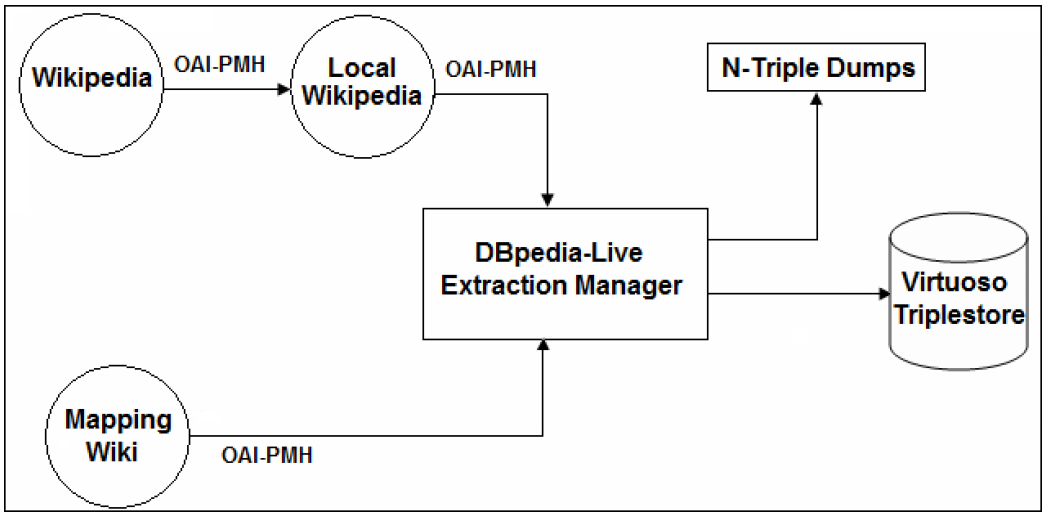
Kljub temu, da je pristop povezanih podatkov še relativno novo področje, obstaja že precejšne število shramb trojčkov RDF. Na sliki 1.1 iz septembra 2011 jih je 299, sedaj jih je že mnogo več.

3.1 DBpedia

DBpedia [10] je zagotovo največja zbirka podatkov predstavljenih s trojčki RDF, kar prikazuje tudi slika 1.1. Na sliki opazimo tudi pomembnost DBpedie drugim zbirkam podatkov, saj se ogromno zbirk povezuje na DBpedio. Namen projekta DBpedia je shranjevanje informacij iz spletne strani Wikipedie v strukturirano obliko. Celoten projekt je spisan v programskih jezikih Scala in Java. Za hrambo podatkov skrbijo strežniki Virtuoso Universal Server [15], ki omogočajo hrambo trojčkov RDF in poizvedovanje po teh podatkih s poizvedovalnim jezikom SPARQL.

Informacije na spletni strani wikipedije se stalno spreminjajo in posodabljajo. To hkrati pomeni, da informacije na DBpediji hitro postanejo zastarele. Zaradi tega ni dovolj, se izbrana vsebina iz wikipedije samo enkrat shrani, ampak je potrebno neprestano posodabljanje.

Okviren prikaz delovanja DBpedije je prikazan na sliki 3.1. Vključuje tri ključne komponente:



Slika 3.1: Shema delovanja DBpedije

- **lokalno wikipedijo**, ki je sinhronizirana z wikipedijo. Za sinhronizacijo skrbi Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [14], ki neprestano pretaka posodobitve.

- **wikipedijo preslikav**, ki je potrebna predvsem za to, da se v DBpediji poleg posodobljene strani posodobijo tudi s to stranjo povezane strani, ki niso bile neposredno posodobljene.

- **sistem za pridobivanje podatkov** črpa podatke iz lokalne wikipedije in wikipedije preslikav s pomočjo OAI-PMH [14]. Podatke obdela in tvori trojčke RDF, katere shrani v shrambo trojčkov [15] in prepiše stare trojčke.

Projekt sta začeli Univerza v Berlinu in Univerza v Leipzigu v sodelovanju s podjetjem OpenLink Software. Prva verzija projekta je bila izdana leta 2007 pod licenco GNU General Public License. Celoten projekt z izvorno kodo je tako javno dostopen.

Nekaj dejstev o DBpediji veljavnih v času pisanja:

- Podpira 119 jezikov.

- Celotna zbirka podatkov vsebuje 2,46 milijarde RDF trojčkov, ki opisujejo skoraj 25 milijonov stvari.
  - Samo angleška lokalizacija opisuje 4 milijone stvari in vsebuje skoraj 500 milijonov RDF trojčkov.
  - Slovenska lokalizacija opisuje 132 tisoč stvari [11].

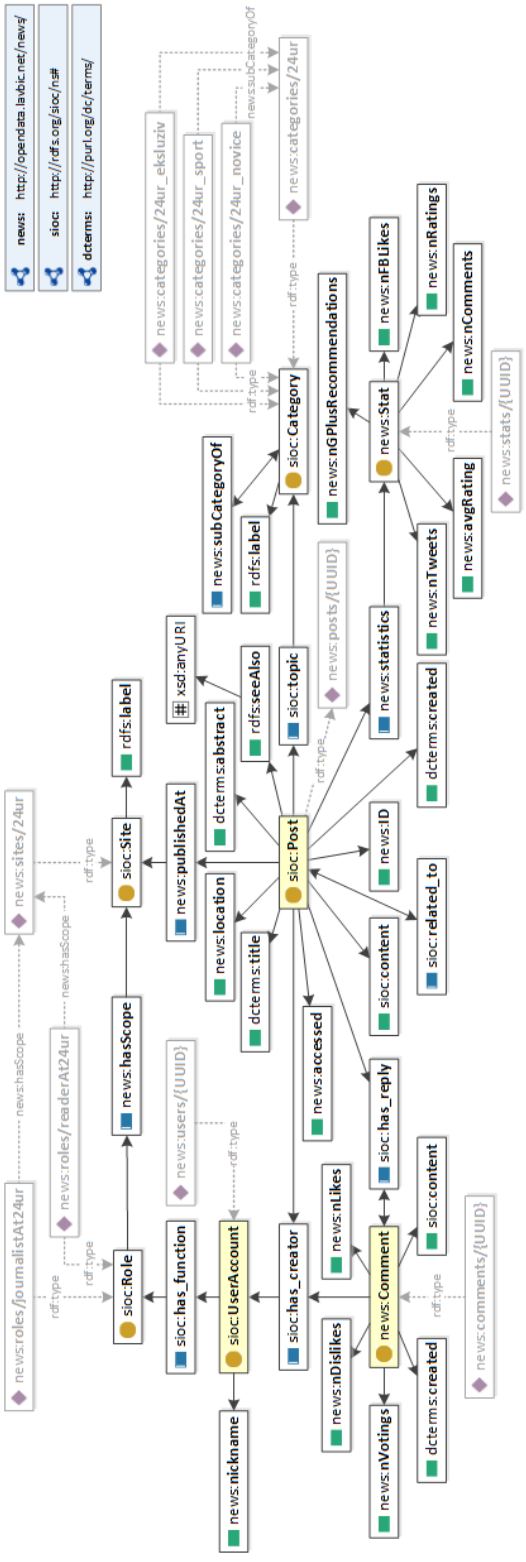
3.2 Opendata news

Opendata news [24] je projekt, ki se je izvajal v okviru Laboratorija za podatkovne tehnologije na Fakulteti za računalništvo in informatiko. Gre za zelo specifično implementacijo, ki podatke črpa iz spletne strani 24ur [2]. V osnovi je tako zelo podobna naši rešitvi, le da smo se mi osredotočili na spletno stran RTV Slo [1]. Razlika je pa tudi v jeziku implementacije, saj je za razliko od naše rešitve ta projekt spisan v programskem jeziku Java. Za hranjenje podatkov pa skrbi OWLIM-Lite paket na strežniku Tomcat, tako kot pri naši rešitvi. Shema shrambe podatkov je prikazana na sliki 3.2.

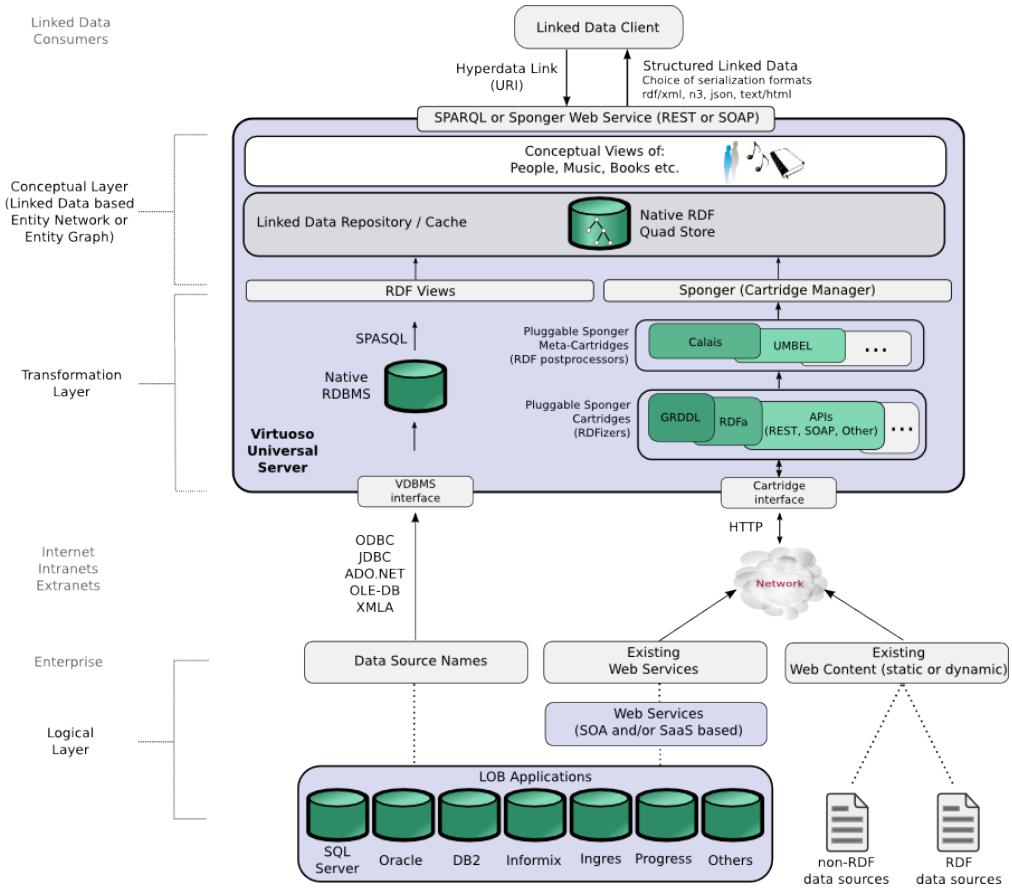
3.3 Virtuoso Sponger

Virtuoso Sponger [32] predstavlja splošno rešitev za pretvarjanje podatkov v strukturirano obliko. V splošnem gre za vmesno komponento okoli podatkovne shrambe Virtuoso Universal Server [15], ki jo uporablja tudi DBpedia. Virtuoso Sponger je pomemben predvsem zato, ker zna ustvariti povezane podatke (tipično v obliki RDF) iz nestrukturiranih virov. Ker gre za splošno rešitev, morajo biti podatki že do neke mere pripravljeni za tako obliko shranjevanja. To velja za vse splošne rešitve, ki so na trgu. Se pa rešitve razlikujejo po podprtih vhodnih in izhodnih formatih.

Rešitev ponuja tudi lahek dostop do že shranjenih podatkov preko storitev REST in SOAP. Ker pa se vse skupaj prenaša preko protokola HTTP, omogoča tudi nastavljivo storitev za predpomnjenje.



Slika 3.2: Shema shrambe Opendata news



Slika 3.3: Delovanje Virtuoso Spongerja

Celotna zgradba Virtuoso Spongerja je prikazana na sliki 3.3. Vidimo, da strežnik Virtuoso lahko pridobi podatke iz različnih virov. Omogoča neposredno povezavo na različne ponudnike relacijskih baz, lahko pa povezava poteka tudi preko spletnih storitev. Drugi način je pa pridobivanje podatkov iz že obstoječih spletnih vsebin, ki so lahko že v RDF obliki zapisa, ni pa nujno. Če še niso v formatu RDF, morajo biti vseeno vsaj delno strukturirani. Podprtih je veliko vhodnih formatov, med njimi so tudi: CSV, Microsoft Word 2003 XML Document, Microsoft XML Spreadsheet 2003, RSS, XHTML.

Virtuoso Sponger za pretvarjanje podatkov v format RDF uporablja sistem "kartuš", kar je ključna funkcionalnost celotnega sistema. Vsaka kartuša skrbi za pretvorbo podatkov iz enega ali več virov. Tipično to poteka tako, da kartuša posreduje vir nekemu tretjemu spletnemu servisu, ki za ta vir vrne podatke v formatu RDF. Nekaj primerov kartuš, ki skrbijo za pretvorbo: Google Plus, Google Search, Amazon, Facebook.

Vsi vrnjeni podatki v formatu RDF se nato shranijo v podatkovno shrambo, od koder so na voljo na nadaljnjo uporabo.

3.4 Ostale rešitve

Obstaja še veliko podobnih rešitev, kot je Virtuoso Sponger [32], vendar so tipično bolj preproste.

Tak primer je recimo projekt Semantic Fire [35], ki je spisan v programskemu jeziku Python in je uporaben za ustvarjanje trojčkov RDF iz spletnih strani. Za ločevanje podatkov med seboj uporablja XPath, kar smo uporabili tudi v naši aplikaciji. Nastavitve celotnega projekta so zajete v datoteki yaml, katere primer je prikazan tudi na primeru 3.1.

Primer 3.1: Primer nastavitvene datoteke za projekt Semantic Fire

```
1 url: http://digg.com/
2 require_id: Yes
3 dataitems:
4   - predicate: dc:title
```

```
5       xpath: id('title')/a
6     - predicate: dc:creator
7       xpath: //a[@rel='dc:creator']
8     - predicate: foaf:primaryTopic
9       xpath: id('title')/a[@href]/@href
10  databags:
11    - predicate: sioc:has_container
12      xpath: //*[@id="p-main"]
13      databag_class: sioc:Thread
14      subpredicate: sioc:container_of
15      datarow_class: sioc:Post
16      p-list: ['dc:creator', 'dc:created', 'sioc:content', 'rev:rating']
17      skip_end_rows: 2
18      row_separator: li
```

Naslednji projekt, ki je zelo podoben projektu Virtuoso Sponger, je pa projekt Apache Any23 [34]. Podpira veliko vhodnih formatov datotek, med njimi tudi format XHTML. In ravno pri XHTML formatu (standarden format za spletne strani) morajo biti podatki že vsebinsko opredeljeni. En od načinov, ki ga lahko uporabimo, je Schema.org [33]. Zamisel je v temu, da prikazanim podatkom na spletni strani dodamo metapodatke, ki pojasnjujejo njihov pomen. Ta način izkorišča tudi iskalnik Google, ki zna na podlagi tega zraven rezultatov prikazati tudi oceno, slike ipd. Primer uporabe Schema.org je prikazan na primeru 3.2.

Primer 3.2: Primer uporabe Schema.org

```
1  <div itemscope itemtype="http://schema.org/Book">
2    <img itemprop="image" src="catcher-in-the-rye-book-cover.jpg" />
3    <span itemprop="name">The Catcher in the Rye</span> -
4    <a itemprop="author" href="/author/jd_salinger.html">J.D. Salinger</a>
5
6    <div itemprop="aggregateRating"
7          itemscope itemtype="http://schema.org/AggregateRating">
```

```
8       <span itemprop="ratingValue">4</span> stars −
9       <span itemprop="reviewCount">3077</span> reviews
10    </div>
11  </div>
```

Poglavje 4

Uporabljene tehnologije in pristopi

Celotna aplikacija diplomske naloge je okvirno sestavljena iz 3 delov:

- pridobivanje podatkov
- shranjevanje podatkov v trojčkih RDF
- shramba trojčkov RDF

4.1 Pridobivanje podatkov

Za pridobivanje podatkov s spletne strani RTV Slo [1] smo najprej potrebovali vstopno točko, s katere smo pridobivali informacije. Običajno imajo spletne strani, kot je RTV Slo zbran arhiv vseh novic. To je bila naša vstopna točka za pridobivanje podatkov, saj so na enem mestu zbrane vse novice. Potrebno se je samo sprehoditi po vseh straneh seznama novic.

Celoten proces pridobivanja podatkov smo ločili na 3 dele:

- Pridobivanje okvirnih podatkov o novicah iz arhiva novic
- Pridobivanje vseh podatkov o posamezni novici

- Posodabljanje novic

Vsak od teh delov je neodvisen, zato se lahko izvajajo vzporedno. Tako izvajanje je tudi najbolj smiselno, saj je izvajanje vsakega koraka aplikacije neskončno. Neprestano se dodajajo nove novice v arhiv, ki jih je potrebno obiskati. S shranitvijo vseh podatkov o novici lahko novico začasno označimo za obdelano. Trajno obdelana ni nikoli, saj se lahko vsebina vsake novice kasneje še spremeni in bi bili naši podatki zastarani, če jo ne bi nikoli več posodobili. Večja verjetnost, kot sprememba vsebine novice, so spremembe v komentarjih. Tudi na zelo starih novicah prihaja do kakšnega novega komentarja, zato je potrebno periodično pregledovati in posodabljati vse novice.

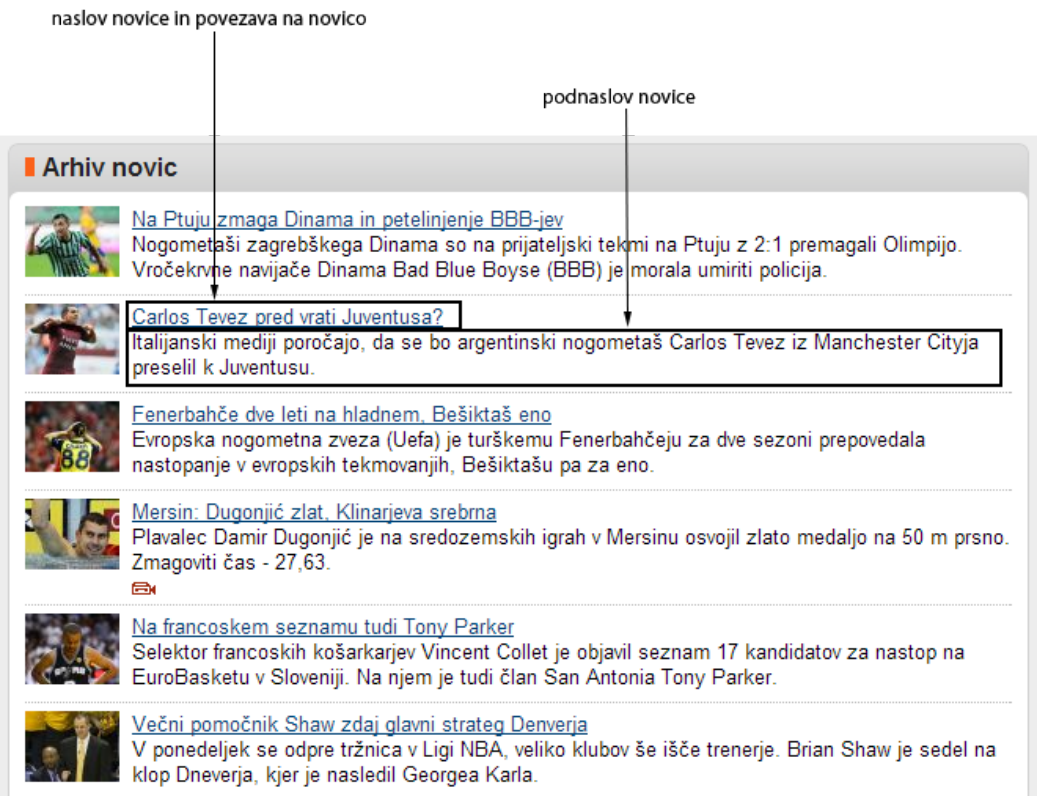
XML Path Language (XPath)

XPath [28, 29] je poizvedovalni jezik v dokumentih XML. Drevesna struktura dokumenta XML omogoča usmerjanje po drevesu glede na izbrane kriterije. Rezultat poizvedbe so lahko vozlišča drevesa, ali pa atomarne vrednosti.

Prva verzija XPath jezika je bila izdana leta 1999 pod okriljem W3C organizacije. Trenutno je v uporabi verzija 2.0, ki je bila izdana leta 2007.

Ker ima HTML dokument tudi drevesno strukturo, lahko prednosti XPath izkoriščamo tudi na takih dokumentih. Pri naši implementaciji je ključen, saj vse podatke pridobivamo na tak način. Velika slabost takega strganja podatkov iz obstoječih HTML strani se pokaže, če se spremeni struktura spletne strani. V takem primeru moramo aplikacijo prirediti novemu izgledu strani.

Primer XPath: `//div[@class='newscomments']/div[@class='com']`
Zgornji XPath uporabljamo za pridobiti vseh komentarjev na strani posamezne novice. Na trenutni HTML strani poišče vse div elemente na katerikoli globini, ki imajo class='newscomments', en novo nižje v hierarhiji pa imajo še en div, ki ima class='com'. Rezultat poizvedbe so div elementi, ki ustrezajo zgornjemu kriteriju.



Slika 4.1: Novice v na strani arhiva novic

4.1.1 Podatki iz arhiva novic

S sprehodom po arhivu novic pridobimo povezavo do vsake novice in naslov novice. Primer arhiva novic je prikazan na sliki 4.1. Naslov novice preberemo z naslova povezave na novico. Opazimo tudi, da je v arhivu tudi podnaslov vsake novice, vendar smo med analizo arhiva odkrili, da se lahko podnaslov v arhivu razlikuje od podnaslova na sami novici. Zaradi tega, podnaslova ne shranjujemo med prehodom čez arhiv.

Povezave na novice so pri prehodu čez arhiv najbolj pomembni podatki, saj vse ostale podatke lahko dobimo kasneje. Primer povezave na novico: `http://www.rtvslo.si/sport/nogomet/carlos-tevez-pred-vrati-juventusa/311851` Vsaka povezava je sestavljena tako, da so najprej hierarhično dodane kategorije, v kateri je novica, ime novice in na koncu še

identifikator novice. Ker povezava na novico vsebuje tudi kategorijo, si ta podatek tudi shranimo v tem koraku.

Ko izluščimo te podatke za eno stran arhiva, jih tudi takoj shranimo. S tem poskrbimo, da ne izkoriščamo preveč pomnilnika, poleg tega pa poskrbimo tudi za bolj zanesljivo delovanje. Če bi imeli na primer ogromno podatkov samo pripravljenih v pomnilniku in bi se izvajanje aplikacije nepričakovano prekinilo, bi lahko na tak način izgubili ogromno nedokončanega dela. Tega ne želimo, zato podatke sproti tudi shranjujemo v shrambo trojčkov.

Kot smo že omenili, je arhiv novice razdeljen na več podstrani. Izvajanje poteka toliko časa, dokler obstaja naslednja stran v arhivu. Ko pridemo do konca arhiva, so povezave vseh novic v shrambi podatkov in se prvi korak začasno zaključi.

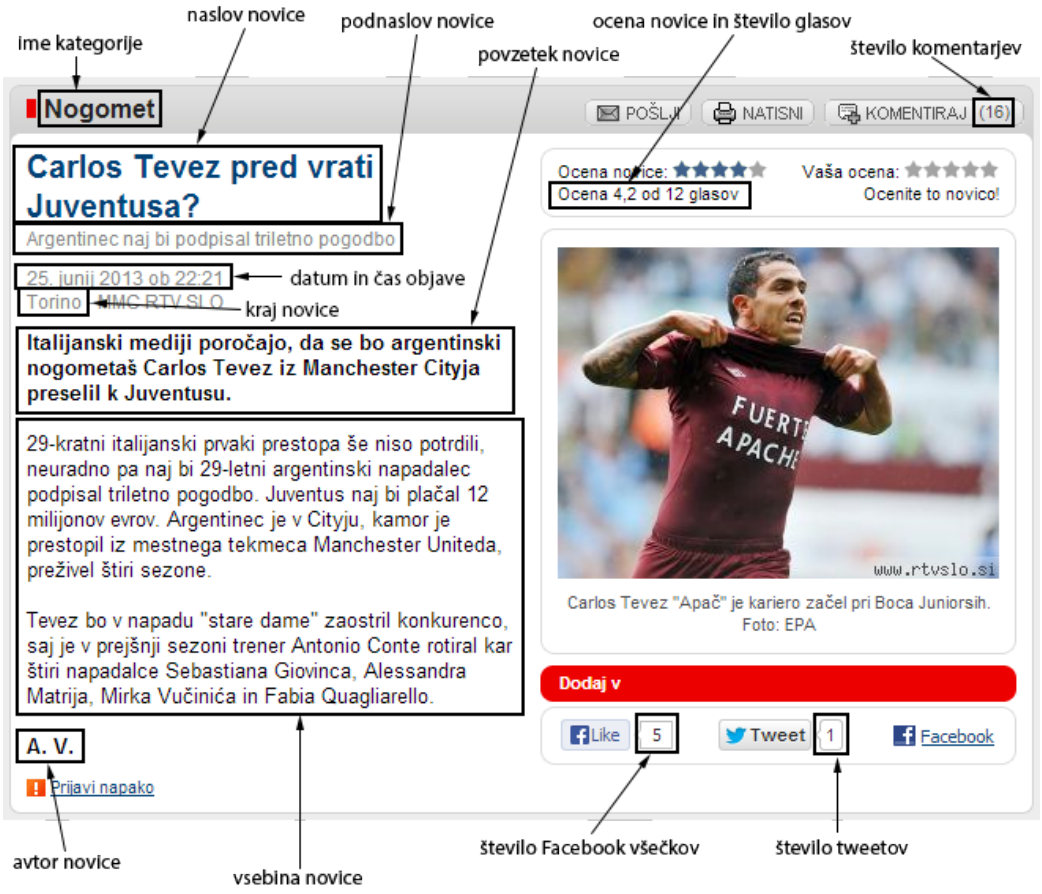
4.1.2 Podatki o posamezni novici

V drugem koraku pridobivanja podatkov poiščemo v shrambi trojčkov najstarejšo še ne obdelano novico. Ker smo imeli povezavo do novice že shranjeno, jo lahko sedaj obiščemo in izluščimo vse ključne podatke o novici, ki so prikazani tudi na sliki 4.2.

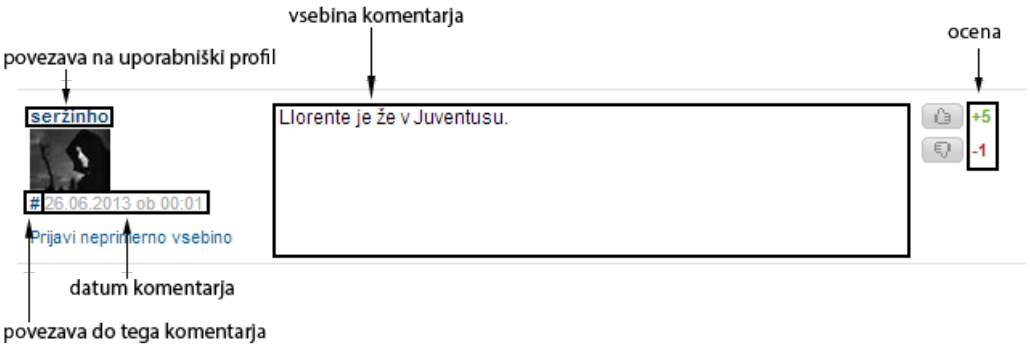
Vsebina novice

Na vrhu je najprej zapisano uporabniku prijazno ime kategorije, v kateri se nahaja ta novica. To ime tudi shranimo na kategorijo, saj imamo do sedaj na posamezni kategoriji shranjeno samo povezavo do kategorije.

Na desni strani novice je zapisano število komentarjev na to novico, zato nam ni potrebno prešteti komentarjev. Vseeno je pa potrebno vzeti v zakup, da lahko pride do odstopanja. Medtem ko preberemo podatek o številu komentarjev in izluščimo podatke na komentarjih, lahko pride do kakšnega novega komentarja. Zaradi tega je mogoče, da dejansko izluščimo podatke o več komentarjih, kot jih dejansko predvidevamo. Na desni strani je tudi podatek o povprečni oceni novice in številu vseh glasovalcev. Kot je prikazano



Slika 4.2: Primer novice



Slika 4.3: Primer komentarja na novico

na sliki 4.2, sta povsem spodaj na desni strani prikazana tudi: število všečkov na socialnem obrežju Facebook in število tweetov na socialnem omrežju Tweeter. To pa tudi zaključuje statistiko posamezne novice.

Vsi glavni podatki novice so prikazani na levi strani na sliki 4.2. Naslov imamo sicer že shranjen, vendar ga posodobimo, saj se naslov novice lahko spremeni. Za tem sledijo podnaslov, datum in čas objave, ter kraj objave. Pred samo vsebino novice je vedno v odebeljenem tisku tudi povzetek novice. Temu sledi vsebina novice in na dnu avtor novice, ki pa ni vedno prikazan. Novica ima običajno tudi več slik, vendar se v okviru diplomske naloge na njih nismo osredotočili in jih ignoriramo.

Komentarji

K novici spadajo tudi komentarji uporabnikov, zato na tem mestu izluščimo tudi vse komentarje na obdelujočo novico. Primer komentarja je prikazan na sliki 4.3. Če je komentarjev veliko, so razdeljeni na več strani. Vsaka stran vsebuje 20 komentarjev. Zaradi tega se mora aplikacija sprehoditi čez vse strani komentarjev in pri vsakem komentarju izluščiti vse podatke. Tudi tukaj podatke shranjujemo za vsako stran posebej, podobno kot pri arhivu novic.

Vsak komentar ima najprej levo zgoraj izpisan uporabniški ime avtorja komentarja in povezavo na njegovo profilno stran. Pod tem podatkom je slika

uporabnika, ki je nikjer ne shranjujemo. So pa za nas pomembni podatki pod sliko. Vsak komentar ima svoj identifikator in povezavo na samo ta komentar. Desno od te povezave pa lahko izluščimo datum in čas objave komentarja. Glavni podatek vsakega komentarja je pa vsekakor vsebina komentarja ter tudi ocena komentarja. Ko izluščimo vse te podatke, je komentar načeloma cel obdelan. Ker pa ima vsak komentar tudi avtorja in povezavo na njegov profil, smo izluščili tudi podatke o vsakem uporabniku.

Uporabniški profil

Vsak komentar novice ima avtorja in povezavo na njegov profil, zato smo se odločili to izkoristiti. Z vsakega profila izluščimo vse podatke, ki so javno dostopni. To se od uporabnika do uporabnika zelo razlikuje, vendar vsaj kakšen podatek lahko vedno izluščimo.

Na sliki 4.4 je primer profila, ki ima javno dostopne skoraj vse podatke. Desno zgoraj je vedno povprečna ocena vsakega uporabnika in število vseh ocenjevalcev. Levo od polja s povprečno oceno se vedno nahaja tudi slika uporabnika, vendar slik tudi tukaj ne shranjujemo.

Pod sliko so pa naslednji podatki, ki jih shranjujemo:

- ime uporabnika, ki je prikazano na avtorju komentarja (vedno prikazano)
- naslov elektronske pošte
- spol
- rojstni datum
- število objav na blogu
- število objavljenih slik
- število objavljenih videov
- število objav na forumih

Slika 4.4: Primer uporabniškega profila

- število objavljenih komentarjev
- datum registracije uporabnika (vedno prikazano)
- opis uporabnika

4.1.3 Posodabljanje novic

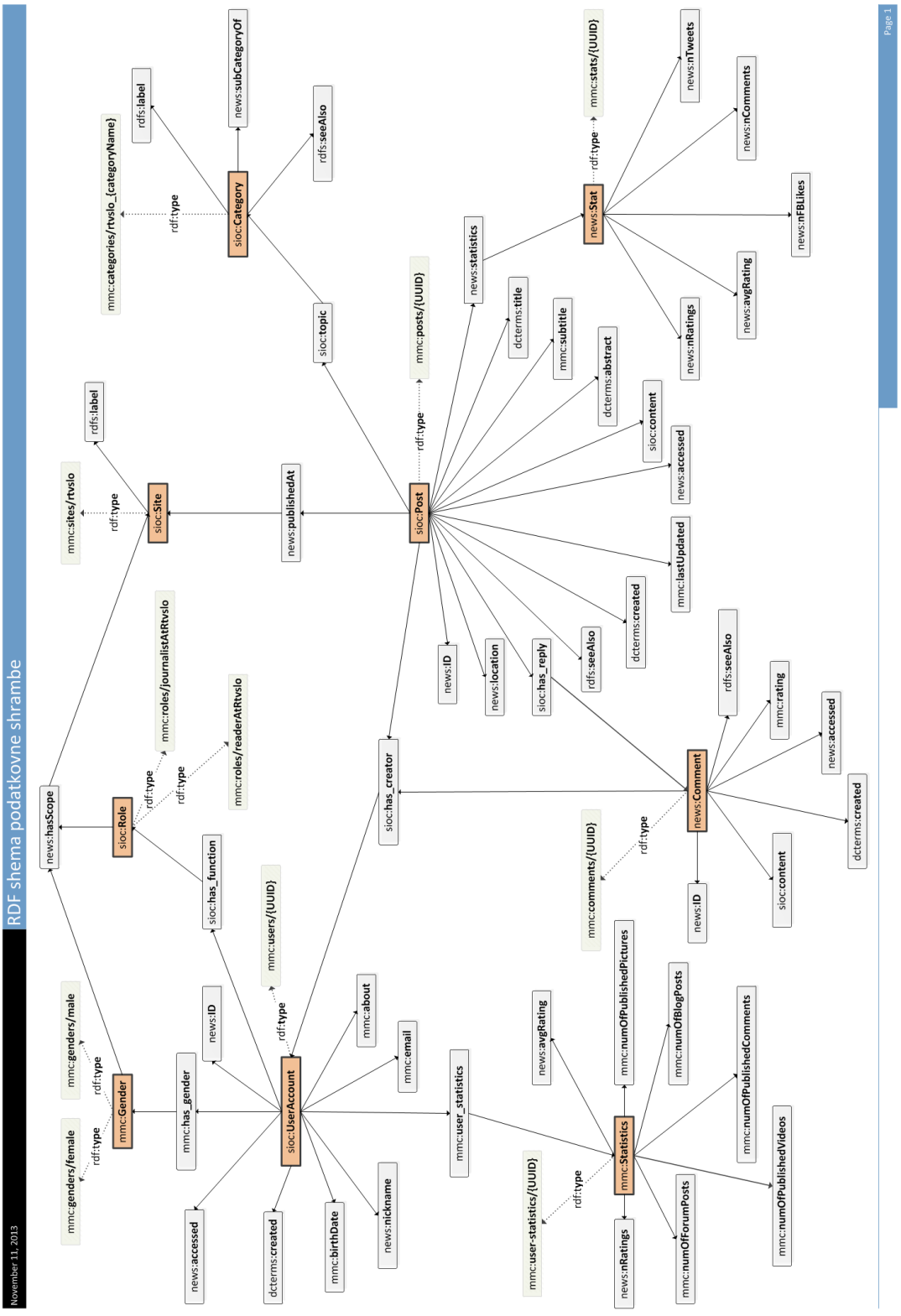
Tretji korak je v osnovi zelo podoben drugem koraku. Razlika je samo v tem, da sedaj ne izbiramo med še ne obdelanimi novicami, ampak izberemo že obdelano novico, ki najdlje ni bila posodobljena. Ves nadaljnji postopek je enaki drugemu koraku.

4.2 Shranjevanje podatkov

Ker izluščeni in pripravljeni podatki v pomnilniku ne pomenijo veliko, smo pripravili shemo shrambe trojčkov RDF za stalno shranjevanje, ki je v obliki grafa prikazana na sliki 4.5. Samo strukturo shrambe RDF smo definirali z vrsto trojčkov RDF, ki smo jih takoj ob zagonu aplikacije dodali v prazno shrambo.

Struktura naše shrambe RDF povezuje vse skupaj 9 razredov. Vsak izmed teh razredov ima edinstven vzorec URI naslova, iz katerega lahko takoj vidimo, za katero vrsto razreda gre. Primer: za URI `mmc:sites/rtvslo` vemo, da je instanca objekta iz razreda `sioc:Site`.

1. **Spletna stran** je objekt instance razreda `sioc:Site`, ki bi prišla bolj do izraza, če bi v naši shrambi hranili podatke iz več virov. Ker smo se v okviru diplomske naloge osredotočili samo na spletno stran RTV Slo [1], imamo samo en objekt tega razreda, katerega subjekt je enak `mmc:sites/rtvslo`. Razred `sioc:Site` vključuje tudi lastnost `rdfs:label`, ki pojasnjuje objekt v uporabniku prijazni obliki.
2. **Uporabniška vloga** je namenjena kategorizirati uporabnike med bralce in novinarje. Objekta sta instanci razreda `sioc:Role`. Kot smo



Slika 4.5: Shema shrambe RDF trojčkov

omenili, imamo samo dve uporabniški vlogi: bralec s subjektom objekta `mmc:roles/readerAtRtvslo` in novinar s subjektom `mmc:roles/journalistAtRtvslo`. Obe instanci imata dodano tudi lastnost `news:hasScope`, ki pojasnjuje, h kateri spletni strani spadata. V okviru te diplomske naloge je ta podatek vedno enak, saj smo pod drobnogled vzeli samo eno spletno stran.

3. **Spol** je prav tako namenjen ločevanju uporabnikov, vendar nimajo vsi tega podatka javno dostopnega. Podatek o spolu shranjujemo samo za tiste uporabnike, za katere smo pridobili njihov spol. Na voljo sta dva subjekta za spol in sicer `mmc:genders/male` za moškega in `mmc:genders/female` za žensko. Obe možnosti sta ustvarjeni, kot instanci razreda `mmc:Gender`. Tudi razred `mmc:Gender` ima lastnost `news:hasScope`, ki ima enak pomen, kot pri prejšnji točki.

4. **Kategorija** hrani podatke o vseh kategorijah, v katerih so novice. Objekti so instance razreda `sioc:Category`. Vsak objekt vsebuje lastnost `rdfs:label` za pojasnitev objekta v uporabniku prijazni obliki. Ker do vsake kategorije vodi tudi URL povezava, smo dodali to povezavo v lastnost `rdfs:seeAlso`, katera je posebej namenjena temu. Kategorije so hierarhično urejene in to hierarhijo smo ohranili znotraj lastnosti `news:subCategoryOf`, ki povezuje dve kategoriji. Subjekt vsake kategorije je ustvarjen po principu `mmc:categories/rtvslo_{imeKategorije}`, kjer je imeKategorije zamenjano z dejanskim imenom.
   Primer: `mmc:categories/rtvslo_nogomet`.

5. **Statistika uporabniškega profila** so objekti ustvarjeni iz razreda `mmc:Statistics`. Subjekt vsakega objekta je generiran po ključu `mmc:user-statistics/{UUID}`, kjer UUID [27] predstavlja univerzalen edinstven identifikator. Gre za 128-bitno število, ki je predstavljeno z 32 heksadecimalnimi števili, med katerimi so tudi vezaji po principu 8-4-4-4-12. Primer takega identifikatorja: 550e8400-e29b-41d4-a716-

446655440000

Vsak uporabnik ima samo eno statistiko profila, vendar smo to ločili v svoj razred zaradi preglednosti. Vključuje lastnosti:

- povprečna ocena profila - predikat: `news:avgRating`
- število vseh ocen profila - predikat: `news:nRating`
- število objav na forumu - predikat: `mmc:numOfForumPosts`
- število objavljenih slik - predikat: `mmc:numOfPublishedPictures`
- število objavljenih komentarjev - predikat: `mmc:numOfPublishedComments`
- število objav na blogu - predikat: `mmc:numOfBlogPosts`
- število objavljenih posnetkov - predikat: `mmc:numOfPublishedVideos`

6. **Uporabniški profil** vsebuje vse podatke o uporabniku, ki jih lahko pridobimo s profilov uporabnikov. Vsak objekt je ustvarjen iz razreda `sioc:UserAccount`. Subjekt vsakega izmed teh objektov je ustvarjen po ključu `mmc:users/{UUID}`. Vsak uporabnik ima edinstveno uporabniško ime, po čemer tudi ločimo uporabnike med sabo.

   O uporabnikih hranimo:

   - edinstveno uporabniško ime - predikat: `news:ID`
   - informacijo o spolu, ki se povezuje na objekte iz razreda `mmc:Gender` - predikat: `mmc:has_gender`
   - informacijo o času dostopa do profila - predikat: `news:accessed`
   - datum registracije - predikat: `dcterms:created`
   - datum rojstva - predikat: `mmc:birthDate`
   - vzdevek uporabnika - predikat: `news:nickname`
   - naslov elektronske pošte uporabnika - predikat: `mmc:email`

- opis uporabnika - predikat: `mmc:about`
- vloga uporabnika - predikat: `sioc:has_function`
- povezava na objekt s statistiko uporabniškega profila - predikat: `mmc:user_statistics`

7. **Komentar** hrani informacijo o komentarju na novico. Objekti so ustvarjeni iz razreda `news:Comment`, katerih subjekti so ustvarjeni po ključu `mmc:comments/{UUID}`.

   Pripadajoče lastnosti:

   - identifikator komentarja - predikat: `news:ID`
   - vsebina komentarja - predikat: `sioc:content`
   - povezava na komentar - predikat: `rdfs:seeAlso`
   - ocena - predikat: `mmc:rating`
   - datum objave komentarja - predikat: `dcterms:created`
   - povezava na objekt o avtorju komentarja - predikat: `sioc:has_creator`
   - časovni zaznamek , kdaj smo pridobili podatke o komentarju - predikat: `news:accessed`

8. **Statistika novice** je razred, ki hrani informacije o statistiki novice. Vsaka novica ima samo eno statistiko. Objekti so ustvarjeni iz razreda `news:Stat`, katerih subjekti so ustvarjeni po ključu `mmc:stats/{UUID}`.

   Hrani pa naslednje informacije o novici:

   - število komentarjev - predikat: `news:nComments`
   - povprečna ocena novice - predikat: `news:avgRating`
   - število ocen novice - predikat: `news:nRatings`
   - število všečkov na socialnem omrežji Facebook - predikat: `news:nFBLikes`

- število tweetov na socialnem omrežju Tweeter - predikar: `news:nTweets`

9. **Novica** je razred, ki hrani vse informacije o vsaki novici objavljeni na spletni strani RTV Slo [1]. To je najpomembnejši razred v naši shemi, saj povezuje vse ostale razrede v celotni sistem. Vsebuje tudi največ lastnosti izmed vseh razredov v shemi. Objekti novice so izpeljani iz razreda `sioc:Post`, čigar subjekti so ustvarjeni po ključu `mmc:posts/{UUID}`.

   Pripadajoče lastnosti razreda:

   - identifikator novice - predikat: `news:ID`
   - kraj novice - predikat: `news:location`
   - povezava na to novico na strani RTV Slo - predikat: `rdfs:seeAlso`
   - naslov novice - predikat: `dcterms:title`
   - podnaslov novice - predikat: `mmc:subtitle`
   - povzetek novice - predikat: `dcterms:abstract`
   - vsebina novice - predikat: `sioc:content`
   - datum in čas, ko je bila novica objavljena - predikat: `dcterms:created`
   - datum in čas zadnje posodobitve novice - predikat: `mmc:lastUpdated`
   - časovni zaznamek, ko smo pridobili podatke o novici - predikat: `news:accessed`
   - povezava na objekte komentarjev na novico - predikat: `sioc:has_reply`
   - povezana na objekt s statistiko novice - predikat: `news:statistics`
   - povezava na objekt o avtorju novice - predikat: `sioc:has_creator`
   - povezava na objekt o kategoriji novice - predikat: `sioc:topic`

- povezava na objekt o spletni strani, kjer je ta novica objavljena - predikat: `news:publishedAt`

4.3 Shramba RDF trojčkov

Za potrebe seminarske naloge smo ustvarili navidezni računalnik VMware na računalnik, kjer smo tudi razvijali programski del seminarske naloge. Nanj smo namestili operacijski sistem Ubuntu 12.10. Hitro se je pokazalo, da Ubuntu preveč obremenjuje gostujoči sistem, vendar smo vseeno želeli imeti strežniški del ločen. Precej boljšo odzivnost smo dosegli, ko smo izklopili grafični uporabniški vmesnik na operacijskem sistemu Ubuntu. Da smo se iz gostujočega sistema lahko povezali na strežnik, smo izklopili še požarni zid in nastavili statični IP naslov.

S pripravljenim operacijskim sistemom, ki nam bo služil za hrambo podatkov, še nismo dosegli željenega cilja. Potrebujemo tudi aplikacijsko podporo za hrambo podatkov. Odločili smo se za odprto kodni strežnik Apache Tomcat 7 [16], ki temelji na Javi. Sama Java JRE je že prednameščena na operacijskem sistemu Ubuntu. Podporo za hrambo in povpraševanje po RDF trojčkih smo dosegli z namestitvijo OWLIM-Lite paketa na strežnik Tomcat 7. OWLIM-Lite vključuje tudi grafični uporabniški vmesnik za pregled shranjenih trojčkov, brisanje trojčkov in ustvarjanje poizvedb po trojčkih.

Poglavje 5

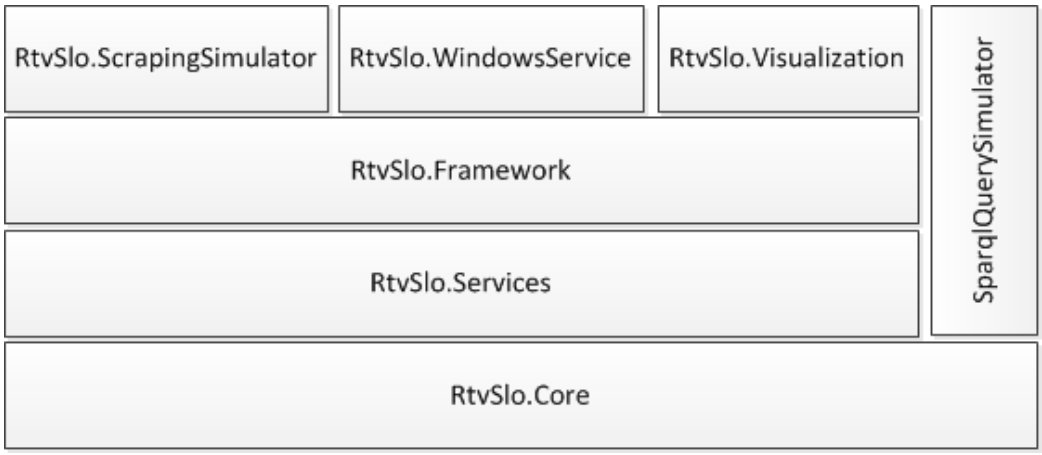
Implementacija rešitve

V prejšnjem poglavju smo opisali uporabljene pristope pri izvedbi diplomske naloge. Sedaj je potrebno vse skupaj podpreti še z implementacijo rešitve. Izbrali smo programski jezik C# in ogrodje .NET 4. Ostali specifični rešitvi, ki smo ju opisali, sta spisani v odprtokodnem programskem jeziku Java, vendar smo se mi odločili za Microsoftov programski jezik C# in razvojno okolje Microsoft Visual Studio 2010.

Na začetku je bilo potrebno celoten projekt smiselno razdeliti na manjše dele, ki smo jih kasneje povezali v celoto. Aplikacijo smo zasnovali kot Windows Service, ki bi se izvajala skupaj z ostalimi servisi na operacijskem sistemu Windows. Za potrebe testiranja pa smo komponente aplikacije združili v preprosto konzolno aplikacijo, kar je bilo v fazi razvoja povsem zadovoljivo. Za dodajanje vseh dodatnih knjižnic v aplikacijo smo uporabili upravitelja paketov NuGet [21], ki skrbi za uporabo knjižnic v razvojnem okolju Microsoft Visual Studio. Poleg knjižnic za shranjevanje in pridobivanje podatkov smo skozi celotno aplikacijo uporabljali tudi knjižnico `log4net`, ki skrbi za beleženje napak in ostalih informacij.

Postavitev in odvisnost posameznih projektov je prikazana na sliki 5.1. Višje postavljeni projekti so odvisni od nižje postavljenih, v vodoravni smeri pa ni odvisnosti med projekti.

Celotna implementacija tako obsega sedem projektov:



Slika 5.1: Postavitev celotne aplikacije

- **RtvSlo.Core** vključuje ključne elemente aplikacije. Ta projekt je vključen v vseh ostalih projektih, saj je ključen za delovanje. Po hierarhiji je na najnižjem nivoju. Vsi ostali projekti so odvisni od tega projekta. Vsebuje vse entitete, ki smo jih ustvarili in druge pomožne metode.

- **RtvSlo.Services** vključuje vse implementirane servise in vmesnike. Naša aplikacija vsebuje samo dva servisa. Prvi skrbi za luščenje podatkov s HTML strani in pretvarjanje v strukturirane objekte, drugi pa za upravljanje s shrambo trojčkov RDF.

- **RtvSlo.Framework** predstavlja nekakšno ogrodje nad RtvSlo.Core in RtvSlo.Services in skrbi za odvisnost injiciranja (DI) servisov, za kar smo uporabili Castle Windsor [22]. Te nastavitve je smiselno imeti ločene, saj bi morali v nasprotnem primeru to implementirati v vsakem hierarhično višjem projektu posebej, kar pa ni najbolj praktično.

- **RtvSlo.Simulator** smo uporabljali za testiranje in razhroščevanje aplikacije. Vse teče samo v eni niti, zaradi tega je potrebno izvajati vsak korak posebej ali pa jih združiti v en sam korak, kjer izvajamo celoten proces naenkrat.

- **RtvSlo.WindowsService** je projekt, ki ustvari Windows Service in je po hierarhiji enakovreden projektu RtvSlo.Simulator. Razlika je le v tem, da tukaj rezultat ni konzolna aplikacija, ampak Windows Service, ki vsak korak aplikacije izvaja v svoji niti in preprečuje nepričakovano ustavitev celotne aplikacije. Če kateri od korakov nima dela, se ta nit začasno ustavi in ponovno zažene po eni uri od ustavitve.

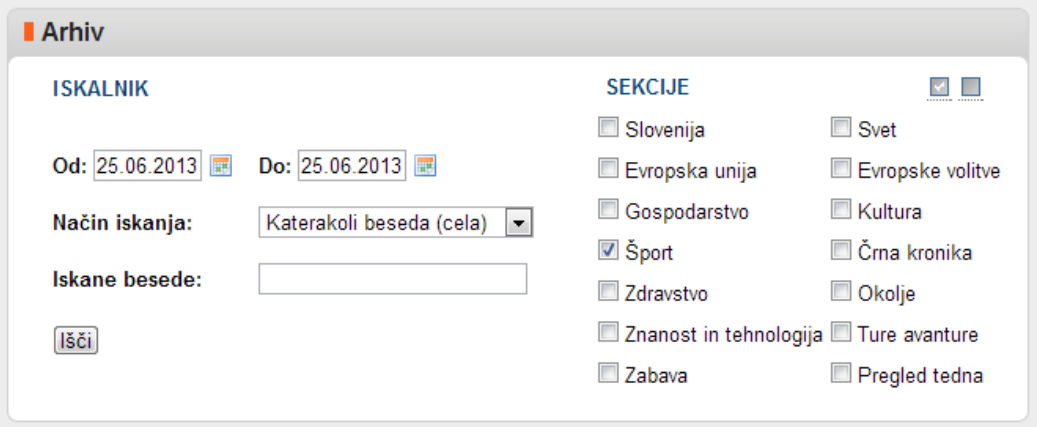
- **RtvSlo.Visualization** je spletni projekt, ki nam je v pomoč za grafično predstavitev nekaterih rezultatov in zaključkov. Sam projekt je zasnovan na arhitekturi MVC [30], uporabili smo pa ASP.NET MVC4 [23].

- **SparqlQuerySumulator** je povsem ločen pomožni projekt, ki ga uporabljamo za testiranje poizvedb po RDF trojčkih v shrambi trojčkov. Možnost imamo pisati poizvedbe SPARQL. Rezultat poizvedbe izpišemo, kot serializiran objekt v XML formatu.

Vse nastavitve aplikacije smo shranili v ločeno nastavitveno datoteko, ki vsebuje nastavitve za pridobivanje podatkov, shranjevanje podatkov, ter nastavitve za knjižnico `log4net`.

Da se lahko začne aplikacija izvajati, potrebuje povezavo do arhiva novic, kar predstavlja našo začetno točko. To povezavo smo shranili v nastavitveno datoteko. Na sliki 5.2 vidimo, da ima stran arhiva tudi nekaj filtrov, ki jih lahko nastavimo. V nastavitveno datoteko smo tako shranili tudi datum objave novic in kategorije novic, ki nas zanimajo. Ker se komentarji novice nalagajo asinhrono z Ajaxom, smo potrebovali tudi koren povezave, s katere se nalagajo komentarji.

Za luščenje podatkov s HTML strani smo uporabili knjižnico `HtmlAgilityPack` [18]. Knjižnica nam omogoča gradnjo HTML DOM drevesa, po katerem se lahko sprehajamo. Zadnja stabilna verzija je 1.4.6 iz 10 julija 2012 in podpira tudi LINQ [20], kar smo precej izkoriščali in prihranili veliko dela.

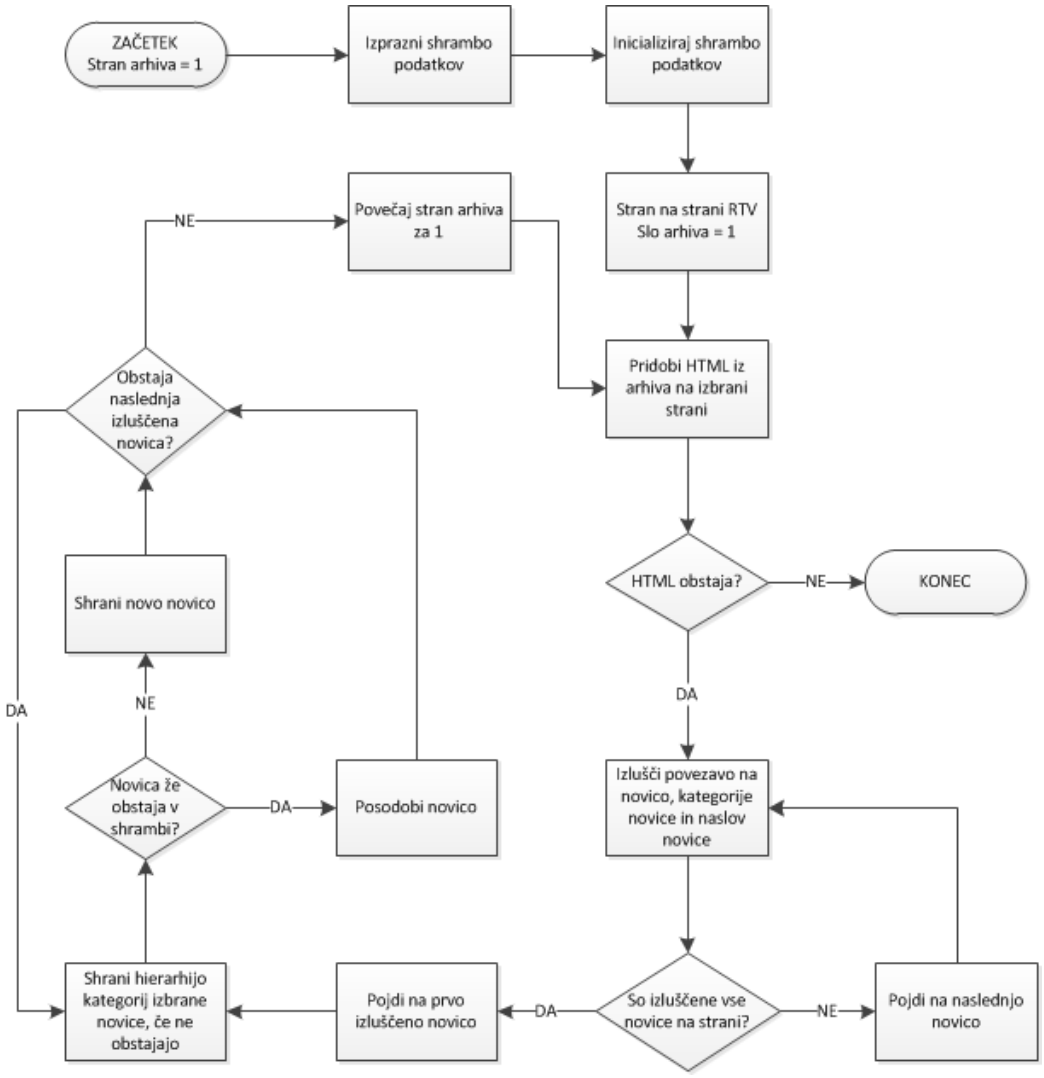


Slika 5.2: Filtri na strani arhiva novic

Za potrebe shranjevanja smo v nastavitveno datoteko shranili tudi povezavo do shrambe podatkov, kamor bo aplikacija zapisovala vse podatke. Uporabili smo odprto kodno knjižnico `dotNetRDF` [19]. V času pisanja je trenutna verzija knjižnice 1.0.0 s 6. maja 2013. Knjižnica vključuje API za povezavo na različne shrambe podatkov. Podpira tudi shrambe temelječe na Sesame, zato pri povezovanju nismo imeli težav. Poleg povezave na shrambo smo knjižnico uporabili še za izvrševanje poizvedb SPARQL (pridobitev trojčkov), dodajanje novega grafa v shrambo, dodajanje novih trojčkov v graf in brisanje trojčkov iz grafa.

5.1 Prvi korak

Na sliki 5.3 je prikazan diagram poteka celotnega prvega koraka, kakor smo ga implementirali. Omenili smo že, da prvi korak aplikacije vključuje luščenje in shranjevanje okvirnih podatkov o novicah. Če se je prvi korak pognal prvič, potem je potrebno najprej v shrambo podatkov shraniti tudi celotno shemo shrambe. Shemo shranimo enako, kot shranjujemo podatke, torej v obliki trojčkov RDF. Za preverjanje, če je bila inicializacija že izvedena, poiščemo trojček, ki ima objekt `sioc:Site`. Če ga najdemo, sklepamo, da je bila inicializacija že izvedena. Taka poizvedba je prikazana tudi na primeru



Slika 5.3: Diagram poteka prvega koraka

5.1.

V kolikor pa shramba še nima shranjene strukture, kakršna je prikazana na sliki 4.5, je potrebno storiti najprej to. Primer trojčkov za inicializacijo objekta `sioc:Site`:

```
mmc:sites/rtvslo
  rdf:type
    sioc:Site
mmc:sites/rtvslo
  rdfs:label
    "RTV Slovenija"
```

Vsi primeri trojčkov predstavljeni v diplomi uporabljajo naslednje predpone:

```
PREFIX dct:  <http://purl.org/dc/terms/>
PREFIX mmc:  <http://jakalogar.si/mmc/>
PREFIX news:  <http://opendata.lavbic.net/news/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sioc:  <http://rdfs.org/sioc/ns#>
```

Primer 5.1: Preverjanje, če je shema shrambe že shranjena

```
1 PREFIX sioc: <http://rdfs.org/sioc/ns#>
2 SELECT *
3 WHERE {
4     ?s ?p sioc:Site
5 }
```

Luščenje arhiva

Podatke s spletne strani dobimo tako, da ustvarimo spletni zahtevek na stran arhiva novic. Vse filtre, ki smo jih nastavili v arhivu novic, pravilno dodamo, kot parametre zahtevka in prejmemo samo filtrirane rezultate. Primer zahtevka, ki je prikazan na sliki 5.2:

`http://www.rtvslo.si/arhiv/?date_from=2013-06-25&date_to=2013-06-`

```
25&search_type=or&section=3
```

Iz prejetega HTML dokumenta je tako potrebno izluščiti podatke, ki nas zanimajo. Ker je HTML samo niz znakov in bi bilo iskanje po temu zelo naporno, samo iz tega prejetega HTML-ja zgradili DOM [31] za lažjo obdelavo. Tako lahko rezultate preprosto poiščemo s pomočjo XPath. Za ustvarjanje pravilnega niza XPath smo uporabili spletni brskalnik Google Chrome in vgrajene pripomočke za razvijalce. Omogoča vpogled v izvorno kodo HTML dokumenta, kot tudi testiranja XPath, če nam vrača pravilne rezultate.

V ustvarjenem objektu DOM smo najprej poiskali vse DIV elemente, ki vsebujejo povezavo na novico. Te podatke smo pridobili z sledečim XPath nizom:

```
//div[@id='container']//div[@id='contents']//div[@class='contents']
/div[@class='body']/div[@id='sectionlist']/div[@class='listbody']
```

V vsakem rezultatu poiščemo povezavo na novico in naslov novice. Ta podatka shranimo v pomnilnik v entiteto `Post`, ki nam služi za hranjenje podatkov o novici, ki smo jih opisali že v prejšnjem poglavju. Iz povezave na novice izluščimo tudi identifikator novice. Pogosto se namreč dogaja, da se spremeni naslov novice, kar posledično vpliva tudi na spremembo povezave na novico, identifikator pa ostaja vedno enak. Po identifikatorju tudi ločimo novice med seboj.

Vsaka povezava na novico hrani tudi hierarhično strukturo kategorij, v katerih je novica. Zaradi tega smo si ustvarili entiteto `Category`, ki vsebuje povezavo do te kategorije, ime kategorije, ime kategorije v uporabniku prijazni obliki, ter povezavo na starševsko in otroško kategorijo. To entiteto smo dodali na entiteto `Post`.

Zanima nas pa tudi informacija, če obstaja naslednja stran arhiva. Če ne obstaja, potem smo pregledali cel arhiv glede na izbrane filtre. Ko pride do tega, se prvi korak začasno zaključi. Ponovno se aktivira po eni uri in preveri, če mogoče že obstaja kakšna nova novica.

Shranjevanje novic arhiva

Za vsako stran arhiva ustvarimo seznam `Post` entitet pripravljenih za trajno shranitev.

Za vsako entiteto najprej shranimo pripadajoče kategorije. Če že obstaja najbolj specifična kategorija v shrambi, potem samo vrnemo povezavo nanjo znotraj shrambe. Drugače pa dodamo manjkajoče kategorije in vrnemo povezavo na najbolj specifično. To povezavo tudi shranimo na novico, če še ne obstaja. Obstaja lahko v primeru, da ne shranjujemo novice prvič. Obstaja pa tudi možnost, da najdemo novico v več kategorijah znotraj arhiva, zato na novico shranimo vse kategorije, v katerih je novica. Primer trojčkov za kategorijo nogomet:

```
mmc:categories/rtvslo_nogomet
  rdf:type
    sioc:Category
mmc:categories/rtvslo_nogomet
  rdfs:label
    "nogomet"
mmc:categories/rtvslo_nogomet
  rdfs:seeAlso
    http://www.rtvslo.si/sport/nogomet/
mmc:categories/rtvslo_nogomet
  news:subCategoryOf
    mmc:categories/rtvslo_sport
```

Poleg kategorije shranimo tudi identifikator novice, naslov novice in povezavo do novice. Najbolj je pomembno, da si shranimo identifikator novice, po katerem lahko iščemo novico v shrambi. S shranitvijo povezave na novico si omogočimo dostop do novice na spletni strani. To povezavo potem uporabi drugi korak aplikacije, da izlušči vse potrebne podatke.

5.2 Drugi korak

Drugi korak aplikacije shranjuje podrobne informacije o novicah. Celotni diagram poteka je prikazan na sliki 5.4.

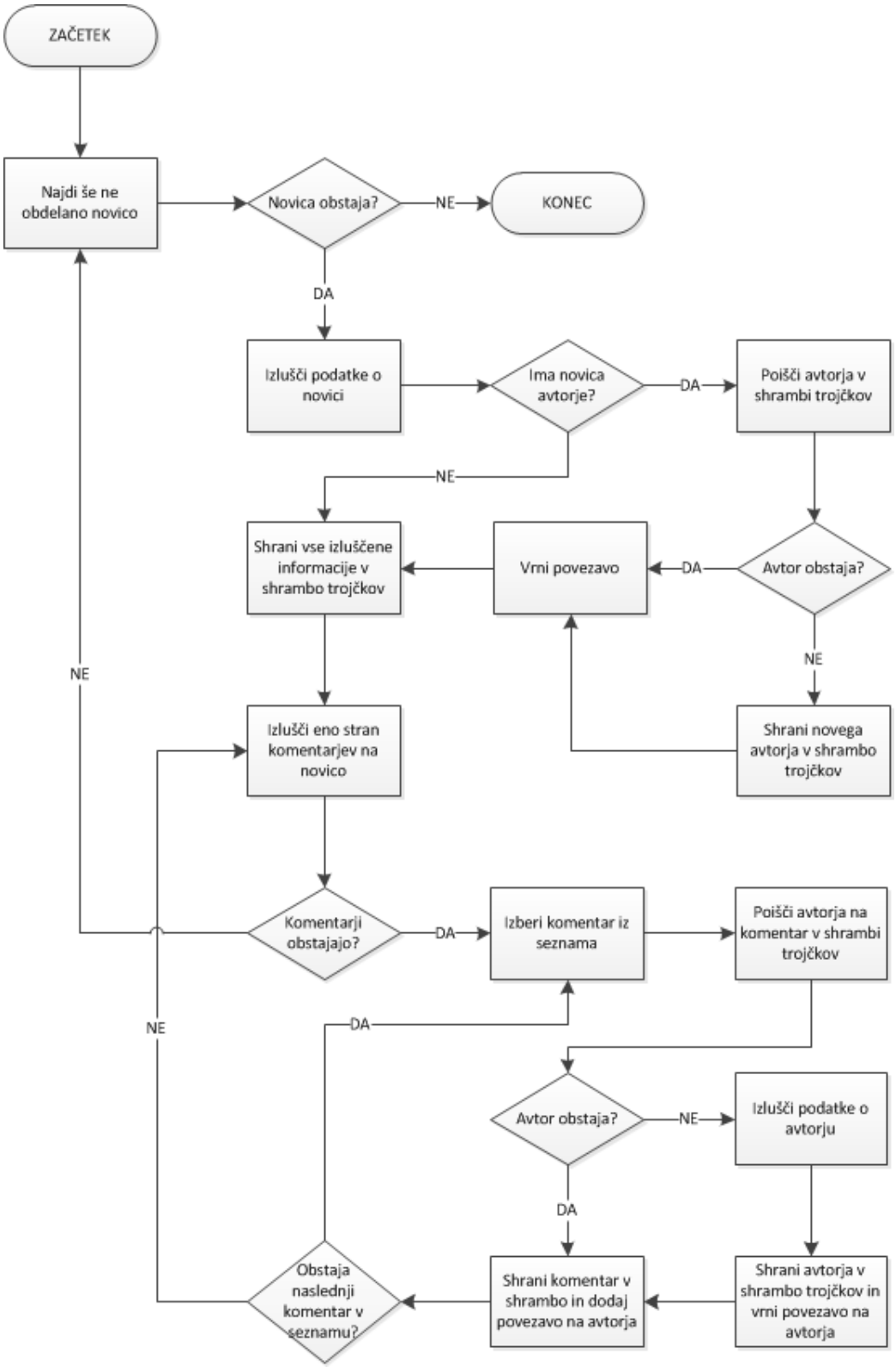
5.2.1 Novica

V shrambi podatkov najprej poiščemo še ne obdelano novico. Vsaka novica ima datum in čas objave, vendar pri prehodu čez arhiv te informacije ne shranjujemo. Zato poiščemo eno novico, ki še nima nastavljenega datuma in časa objave. Poizvedba po taki novici z jezikom SPARQL je prikazana v poizvedbi 5.2.

Primer 5.2: Iskanje še ne obdelane novice

```
1  PREFIX dct: <http://purl.org/dc/terms/>
2  PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
3  PREFIX rdfs: <http://www.w3.org/2000/01/rdf−schema#>
4  PREFIX news: <http://opendata.lavbic.net/news/>
5  PREFIX sioc: <http://rdfs.org/sioc/ns#>
6  SELECT ?url ?id
7  WHERE {
8      ?post rdf:type sioc:Post .
9      ?post rdfs:seeAlso ?url .
10     ?post news:ID ?id
11     MINUS { ?post dct:created ?date . }
12 }
13 LIMIT 1
```

Če poizvedba ne vrne nobene novice, pomeni, da imamo vse novice v celoti shranjene. Drugi korak se začasno ustavi in ponovno zažene čez eno uro.



Slika 5.4: Diagram poteka drugega koraka

Luščenje podatkov o novici

Ker imamo shranjeno povezavo do novice, izvedemo sedaj spletni zahtevek na stran novice in pridobimo pripadajoči HTML. Iz tega HTML-ja ustvarimo DOM objekt, po katerem lahko iščemo. Pridobljeni HTML ima dodanega tudi ogromno balasta, ki nas ne zanima, zato najprej poiščemo odsek, ki vsebuje podatke o novici. Uporabili smo sledeči XPath:

```
//div[@id='container']//div[@class='contents']/div[@class='body']
/div[@id='newsbody']
```

Preden začnemo luščiti podatke o novici, si shranimo tudi časovni zaznamek, kdaj smo obiskali novico. Pri luščenju naslova in podnaslova nismo imeni nikakršnih težav. Malo več problemov smo imeli pri luščenju datuma in časa objave novice in zadnje posodobitve novice. Ta del namreč nima vedno enake strukture. Včasih je prisotna informacija, kdaj je bila novica zadnjič posodobljena, ne pa vedno. Za preverjanje tega smo preprosto pogledali, če obstajajo tri vrstice v HTML. V prvi vrstici je informacija o datumu in času objave novice, zadnja vrstica nosi informacijo o kraju novice. Druga vrstica pa ima informacijo o datumu in času zadnjega posega v novico, če obstaja. Primer:

```
<div class="info≫16. februar 2013 ob 07:22,<br>zadnji poseg: 16.
februar 2013 ob 15:16<br>Schladming - MMC RTV SLO</div>
```

Pri luščenju vsebine novice smo tudi imeli kar nekaj težav zaradi neenotnosti strukture vsebine. Najprej opazimo, da je povzetek novice umeščen kar v samo vsebino. Ker je povzetek vedno v odebeljenem tisku in na začetku vsebine, smo ga na podlagi tega uspešno ločili od ostale vsebine. Pogosto so vsebini novice dodane tudi slike, kar smo se odločili ignorirati. Predvsem pri športnih novicah so pogosto dodani tudi tabelarični prikazi, kar smo tudi ignorirali.

Običajno je vsebina razdeljena po več HTML P elementih, včasih je pa tudi znotraj DIV elementa, ki ima dodane P elemente in zunaj tega DIV-a so še tudi P elementi. Da smo zavzeli vse možnosti in ignorirali vse ostalo, smo konstruirali pogoj:

`P ali DIV element, ki ima naslednika in ta naslednik ni enak DIV elementu`. V rezultatu dobimo tudi povzetek vsebine, kar naknadno odstranimo. Preostali tekst znotraj vseh elementov je vsebina novice.

Na dnu novice je običajno zapisan tudi avtor novice. Včasih je na novici tudi več avtorjev ločenih z vejico. To poskušamo čim bolj uspešno zaznati in ločiti avtorje med sabo.

Luščenje statistike novice

V statistiki novice smo zajeli povprečno oceno novice, število komentarjev, število Facebook všečkov in število tvitov na Tweeterju. Pridobitev povprečne ocene novice in števila ocenjevalcev nam ni predstavljalo nobenega problema. Potrebno je samo prebrati število in pretvoriti v ustrezen format. Enako velja za podatek o številu vseh komentarjev na novico.

Smo pa naleteli na problem pri številu Facebook všečkov in številu tvitov. Tekom implementacije diplomske naloge so na strani RTV Slo spremenili implementacijo socialnih vtičnikov. Sprva so vtičnike prikazovali znotraj iframe. Pri tej implementaciji smo prebrali URL, s katerega se naloži stran v iframe, naredili spletni zahtevek na ta URL in pridobili HTML. V tem HTML smo prebrali število všečkov ali tvitov in to shranili.

Kasneje so to implementacijo spremenili in sedaj se vtičniki naložijo asinhrono z Ajaxom. Posledično smo morali tudi mi temu prilagoditi implementacijo. Preverili smo, kakšen spletni zahtevki se ustvarijo za pridobitev vtičnikov in simulirali te zahtevke.

Facebook naloži vtičnik iz spletnega naslova:
`http://www.facebook.com/plugins/like.php?href={url_novice}&layout=button_count`, kjer je {url_novice} enak url-ju do novice, za katero želimo število Facebook všečkov. Ta url je zapisan v obliki, da ga lahko dodamo kot parameter spletnega zahtevka. Rezultat zahtevka za zgornji spletni naslov nam vrne dokument HTML, v katerem je vtičnik in število Facebook všečkov. Iz tega dokumenta samo še izluščimo število všečkov.

Podobno velja tudi za število tweetov na Tweeterju. Razlika je pa v tem,

da se število tweetov znotraj vtičnika ponovno naloži z zahtevkom Ajax. Zaradi tega smo mi izvedli samo zahtevek na naslov, ki nam vrne dejansko število tweetov in je enak:
`http://cdn.api.twitter.com/1/urls/count.json?url={url_novice}`.
{url_novice} je predstavljen enako, kot pri vtičniku za Facebook. Rezultat zahtevka je objekt JSON, ki vsebuje število tweetov, ki nas zanimajo.

Shranjevanje novice

Pred shranjevanjem trojčkov o novici najprej preverimo, če avtorji novice že obstajajo v naši shrambi. Če ne obstajajo, potem najprej shranimo vse nove avtorje. Od podatkov imamo samo ime avtorja ali njegove inicialke. Nastavimo mu pa tudi vlogo `mmc:roles/journalistAtRtvslo` in shranimo, ter si zapomnimo povezavo na avtorja. Če avtor že obstaja v bazi, potem si samo zapomnimo njegovo povezavo. Iskanje avtorja novice izvedemo s poizvedbo 5.3, kjer je 'name' zamenjan z dejanskim imenom avtorja. V primeru več rezultatov vzamemo prvega.

Primer 5.3: Iskanje avtorja novice

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX news: <http://opendata.lavbic.net/news/>
3 PREFIX sioc: <http://rdfs.org/sioc/ns#>
4 SELECT ?guidUrl
5 WHERE {
6     ?guidUrl rdf:type sioc:UserAccount
7     ; news:nickname "name"
8 }
```

Primer 5.4: Pridobitev vseh trojčkov novice

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX news: <http://opendata.lavbic.net/news/>
3 PREFIX sioc: <http://rdfs.org/sioc/ns#>
4 SELECT ?guidUrl ?predicate ?object
```

```
5 WHERE {
6     ?guidUrl rdf:type sioc:Post
7     ; news:ID "id"
8     ; ?predicate ?object
9 }
```

Avtorji so urejeni, zato lahko shranimo vse pridobljene podatke o novici. Najprej poiščemo vse obstoječe trojčke novice s poizvedbo 5.4, kjer je 'id' zamenjan z dejanskim identifikatorjem novice. Iz pridobljenih trojčkov odstranimo star naslov in shranimo trenutno izluščenega. Shranimo tudi ostale podatke, ki smo jih izluščili.

Primer nekaterih trojčkov posamezne novice:

```
mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  rdf:type
    sioc:Post

mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  rdfs:seeAlso
    http://www.rtvslo.si/sport/nogomet/carlos-tevez-pred-vrati
                                                -juventusa/311851

mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  news:ID
    "311851"

mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  dct:title
    "Carlos Tevez pred vrati Juventusa?"

mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  sioc:topic
    mmc:categories/rtvslo_nogomet

mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  news:publishedAt
    mmc:sites/rtvslo
```

```
mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  dct:created
    "2013-06-25 20:21:00Z"
mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  mmc:subtitle
    "Argentinec naj bi podpisal triletno pogodbo"
mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  news:statistics
    mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
mmc:posts/40323adb-b1d5-4e6b-8fba-2d056daf096a
  sioc:has_reply
    mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
```

Prvi primer prikazuje inicializacijo objekta iz razreda `sioc:Post`. Vsak objekt novice hrani tudi povezave na objekte komentarjev, ki pripadajo tej novici, kar ponazarja zadnji primer trojčka.

Skupaj z novico shranimo tudi izluščeno statistiko novice. Kot je prikazano na sliki 4.5, imamo statistiko novice shranjeno v novem objektu, na novici pa hranimo povezavo na ta objekt. Ta povezava je prikazana na predzadnjem primeru.

Vsi trojčki za statistiko novice so prikazani v spodnjih primerih, vključno z inicializacijo objekta, kar prikazuje prvi primer:

```
mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
  rdf:type
    news:Stat
mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
  news:nComments
    "16"
mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
  news:avgRating
    "4,2"
```

```
mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
  news:nRatings
    "12"
mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
  news:nFBLikes
    "5"
mmc:stats/e88580ea-20b2-4119-bc32-245e8628f905
  news:nTweets
    "1"
```

5.2.2 Komentar

Ko shranimo vse podatke o sami novici, so na vrsti vsi komentarji na novico.

Luščenje podatkov

Komentarji se na strani novice naložijo asinhrono z Ajaxom. Zaradi tega smo za pridobitev podatkov ročno simulirali Ajax zahtevek, ki nam je vrnil HTML s komentarji na novico.

Primer take povezave:
`http://www.rtvslo.si/index.php?&c_mod=news&op=comments&func=ajax&id=311851&page=0&hash=0&sort=asc`. V tem zahtevku spreminjamo parameter `page`, ker so komentarji lahko ločeni na več strani. Vsaka stran vsebuje 20 komentarjev. Parameter za številko strani povečujemo toliko časa, dokler prejemamo rezultat. Ko dobimo za rezultat prazno stran, vemo da je bila pred tem zadnja stran komentarjev. Naslednji parameter, ki ga spreminjamo, je pa `id`, ki vsebuje identifikator novice. Ostalih parametrov ni potrebno spreminjati in so ves čas enaki, kot na primeru.

Vsak izluščen komentar shranimo v objekt `Comment`, za celotno stran komentarjev tako ustvarimo seznam teh objektov. Primer komentarja je prikazan na sliki 4.3. Za ločitev komentarjev od ostalega HTML-ja smo uporabili sledeči XPath:

`//div[@class='newscomments']/div[@class='com']`.

Objekt `Comment` ima naslednje lastnosti:

- **Url** uporabimo za hranjene povezave na točno ta komentar.
- **DateCreated** hrani datum in čas, ko je bil komentar ustvarjen.
- **AccessedDate** v vseh objektih uporabljamo za shranjevanje časovnega zaznamka, ko smo shranili podatke v shrambo.
- **Id** hrani identifikator komentarja, ki ga izluščimo iz povezave na komentar. Primer povezave: `http://www.rtvslo.si/novice/komentarji/4310870`, kjer je identifikator 4310870.
- **PostId** hrani identifikator novice, na katero se nanaša komentar.
- **UserId** hrani identifikator avtorja komentarja.
- **PostGuidUrl** vsebuje povezavo na novico v naši shrambi trojčkov.
- **UserUrl** hrani povezavo na uporabnikovo stran. Povezavo potrebujemo, da lahko izluščimo podatke o uporabniku.
- **UserGuidUrl** uporabimo za shranitev povezave do trojčkov o uporabniku, ko uporabnika shranimo.
- **Content** hrani celotno vsebino komentarja. Vse HTML elemente v vsebini ignoriramo, shranimo samo tekst.
- **Rating** hrani povprečno oceno komentarja. Kot smo že omenili, je implementacija na spletni strani RTV Slo sprva prikazovala samo skupno oceno med pozitivnimi in negativnimi ocenami. V zadnji implementaciji pa so to spremenili in prikazujejo pozitivne in negativne ocene ločeno. Mi nismo spreminjali implementacije, zato obe oceni združimo v skupno oceno.

Shranjevanje komentarja

Ko izluščimo eno stran komentarjev in napolnimo seznam `Comment` objektov, so komentarji pripravljeni na shranjevanje. Za vsak komentar v seznamu moramo najprej pogledati, če že obstaja njegov avtor v naši shrambi. Če obstaja, potem si samo zapomnimo povezavo na njega, drugače ga moramo še shraniti. Luščenje in shranjevanje uporabnika je opisano v naslednji točki.

Preden shranimo nov komentar, preverimo, če mogoče že obstaja komentar z enakim identifikatorjem. Če ga najdemo, potem v tem koraku shranjevanje preskočimo in zaključimo postopek. V nasprotnem primeru shranimo nov komentar v shrambo trojčkov. Obvezno je potrebno dodati tudi trojček za povezavo na novico, na katero se nanaša ta komentar.

Primeri nekaterih trojčkov komentarja:

```
mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
  rdf:type
    news:Comment

mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
  rdfs:seeAlso
    http://www.rtvslo.si/novice/komentarji/4310870

mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
  news:ID
    "4310870"

mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
  sioc:content
    "Llorente je že v Juventusu."

mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
  sioc:has_creator
    mmc:users/519988b6-c2ea-4d40-a43d-50fee175ca3f

mmc:comments/35f49015-0d16-4768-8a59-cd9f895e1747
  mmc:rating
    "4"
```

5.2.3 Uporabniški profil

Preden shranimo podatke o komentarju, moramo imeti povezavo na avtorja komentarja. Če avtorja komentarja še nimamo shranjenega v shrambi, potem te povezave ne moremo pridobiti, dokler ne ustvarimo objekta o avtorju komentarja v shrambi RDF trojčkov.

Luščenje podatkov

Vsak komentar ima dodano povezavo na profil avtorja komentarja. Na to povezavo ustvarimo spletni zahtevek, ki nam vrne HTML profilne strani uporabnika. Iz prejetega HTML-ja ustvarimo DOM objekt, kot v prejšnjih primerih, po katerem se lahko sprehajamo in iščemo.

Pri luščenju uporabniških podatkov smo imeli nekaj težav, saj nimajo vsi uporabniški profili enake HTML strukture. Pri analiziranju profilov smo našli 3 različne strukture HTML-ja: kompaktno, srednjo in razpršeno. Zaradi tega je bilo potrebno uporabiti tudi svoj XPath za vsako od teh struktur. Ker je HTML znotraj vsake strukture potem enak, je bila potrebna ločitev samo na tem mestu, naprej je za vse tri strukture enako.

Vse podatke o uporabniku smo si v pomnilnik shranili v objekt `User` z naslednjimi lastnostmi:

- **Url** uporabimo za hranjene povezave do tega uporabniškega profila.
- **DateCreated** hrani datum, ko se je uporabnik registriral na spletno stran RTV Slo.
- **AccessedDate** tudi v tem primeru hrani časovni zaznamek, kdaj smo shranili trojčke uporabniškega profila v shrambo podatkov.
- **Id** hrani identifikator uporabnika, ki je enak uporabniškemu imenu uporabnika.
- **Function** uporabimo za shranjevanje funkcije uporabnika, bodisi bralec bodisi novinar.

- **Name** hrani ime uporabnika, ki lahko vsebuje presledke in šumnike.
- **Email** je polje, v katerega shranimo elektronski naslov uporabnika, če je le ta javno dostopen.
- **Gender** hrani spol uporabnika, če ga je uporabnik nastavil.
- **Birthdate** hrani datum rojstva uporabnika, če je le ta javno dostopen.
- **Rating** uporabljamo za shranjevanje povprečne ocene uporabniškega profila.
- **NumOfRatings** hrani število vseh oddanih ocen uporabniškega profila.
- **ForumPosts** hrani število vseh objav uporabnika na forumu.
- **BlogPosts** hrani število vseh objav na blogu.
- **PublishedPictures** hrani podatek o številu vseh objavljenih slik uporabnika.
- **PublishedComments** hrani podatek o številu vseh objavljenih komentarjev uporabnika.
- **Description** je polje za hrambo izluščenega opisa uporabnika.

Shranjevanje uporabnika

Preden shranimo izluščene podatke o uporabniku, preverimo, če mogoče v naši shrambi že obstaja uporabnik s tem uporabniškim imenom. Če ga najdemo, potem ne shranimo novega uporabnika ampak vrnemo samo povezavo na že shranjenega uporabnika v shrambi. Pri shranjevanju novega uporabnika shranimo samo trojčke za podatke, ki smo jih uspeli izluščiti.

Primeri nekaterih trojčkov o avtorju:

```
mmc:users/defc6740-a974-4ba9-96ae-b1f34e79ee5b
  rdf:type
```

```
    sioc:UserAccount
mmc:users/defc6740-a974-4ba9-96ae-b1f34e79ee5b
  sioc:has_function
    mmc:roles/readerAtRtvslo
mmc:users/defc6740-a974-4ba9-96ae-b1f34e79ee5b
  news:nickname
    "veselo-na-delo"
mmc:users/defc6740-a974-4ba9-96ae-b1f34e79ee5b
  rdfs:seeAlso
    http://www.rtvslo.si/profil/veselo-na-delo
```

5.3 Tretji korak

Bistvo tretjega koraka je posodabljanje že obdelanih novic. Algoritem najprej izbere primerno novico za posodobitev. Primerne so vse novice, ki so bile nazadnje posodobljene več kot 14 dni nazaj. Med vsemi primernimi izbere tako, ki najdlje ni bila posodobljena. Če bi vedno samo poiskali novico, ki najdlje ni bila posodobljena, bi lahko prišli v situacijo, kjer bi novico najprej prvič shranili, takoj zatem bi jo pa že posodobili. Temu scenariju se želimo izogniti, zato najprej filtriramo. V primeru, da ni nobene novice za posodobiti, se tretji korak ustavi in ponovno preveri čez eno uro.

Ko algoritem najde prvo primerno novico, ki jo je potrebno posodobiti, najprej izlušči podatke o tej novici iz spletne strani. Ker podatki o novici že obstajajo v shrambi trojčkov, poiščemo trojčke o tej novici. Posodobitev izvedemo samo na tistih trojčkih, ki so spremenjeni, tako da izbrišemo star trojček in ga nadomestimo z novim. V splošnem je tretji korak zelo podoben drugemu, le da tukaj brišemo že obstoječe trojčke.

5.4 Diskusija

Sedaj, ko je celotna implementacija izvedena, lahko že podamo nekatere ugotovitve.

Samo hitrost aplikacije nismo testirali, ker je preveč zunanjih faktorjev, ki lahko vplivajo na izvajanje. Najbolj na hitrosti izvajanja vpliva hitrost pridobivanja vsebine iz spletnega strežnika. Ker pa ne želimo prepogosto pošiljati spletnih zahtev na strežnik, smo zaradi tega precej zmanjšali hitrost aplikacije. Sama hitrost internetne povezave na hitrost skoraj ne vpliva, ker pridobivamo zgolj tekstovne datoteke, brez ostalih dodatkov, potrebnih za pravilen prikaz v spletnem brskalniku.

Izvajanje aplikacije je dovolj robustno, da bi lahko sčasoma pridobila vse informacije. Vsak korak se izvaja ločeno in tudi če bi prišlo do napake v katerem od korakov, to ne vpliva na izvajanje ostalih korakov. Do napak pri izvajanju ne pride niti, če na primer med izvajanjem ugasnemo računalnik. Drugi in tretji korak bosta nadaljevala svoje delo, ko se spet aktivirata. Prvi korak pa bo ponovno izvedel celoten prehod čez arhiv, saj ne moremo določiti, kje se je izvajanje zaključilo in kaj se je spremenilo med tem časom.

Če bi želeli povečati hitrost aplikacije in število zahtev na spletni strežnik ne bi bila ovira, bi lahko pognali tudi več instanc drugega in tretjega koraka. Paziti bi bilo potrebno samo, da posamezne instance enakega koraka ne bi istočasno obdelovale iste novice. V takem primeru bi lahko prišlo do napak v shranjenih podatkih.

Poglavje 6

Vizualizacija rezultatov

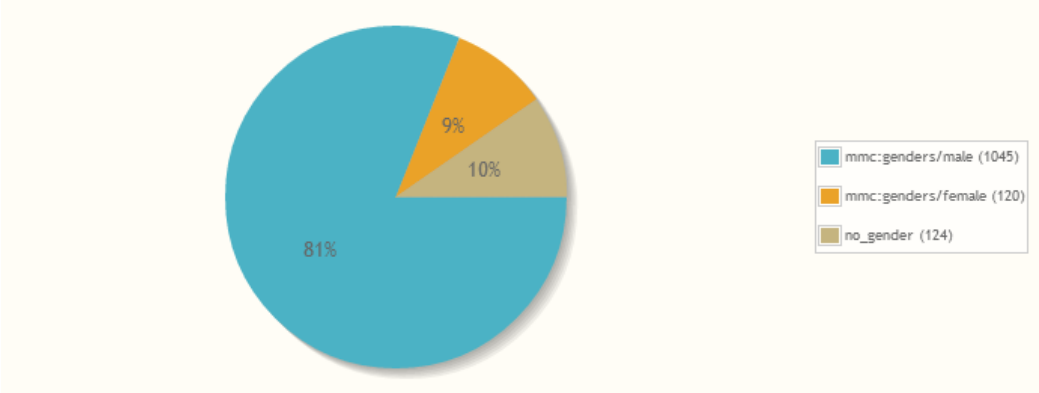
Celotno implementacijo je smiselno podkrepiti tudi z vizualizacijo implementirane rešitve. Že v uvodu smo dejali, da želimo našo shrambo podatkov povezati s katero od že obstoječih shramb in to bomo tudi poskušali doseči. Najlažje je tako povezavo predstaviti v grafični obliki.

Spodaj je predstavljenih nekaj rezultatov poizvedb v grafični obliki. Vse te poizvedbe potrebujejo podatke najprej v strukturirani obliki, saj drugače ne moremo priti do takih zaključkov. Vse vizualizacije so samo primer, kaj vse je mogoče doseči, ko imamo podatke pripravljene v strukturirani obliki. Naša shramba ima trenutno namreč še premalo podatkov, da bi lahko prišli do kakšnih zaključkov. Trenutno se v njej nahaja samo 48 kategorij, 779 novic, 5159 komentarjev in 1480 uporabnikov, vključno z avtorji novic. Med testiranjem tudi nismo izbrali vseh vrhnjih kategorij v arhivu, vendar samo 6 od možnih 14 kategorij. Zaradi tega precej kategorij še manjka v shrambi.

6.1 Delež uporabnikov glede na spol

Prva preprosta poizvedba, ki smo jo prikazali v grafični obliki, je razvrstitev uporabnikov po spolu. Osredotočili smo se samo na bralce novic, saj od avtorjev novic nismo nikjer pridobili informacije o spolu.

Rezultat poizvedbe je prikazan v tortni obliki na sliki 6.1. Vsak uporabnik



Slika 6.1: Delež uporabnikov glede na spol

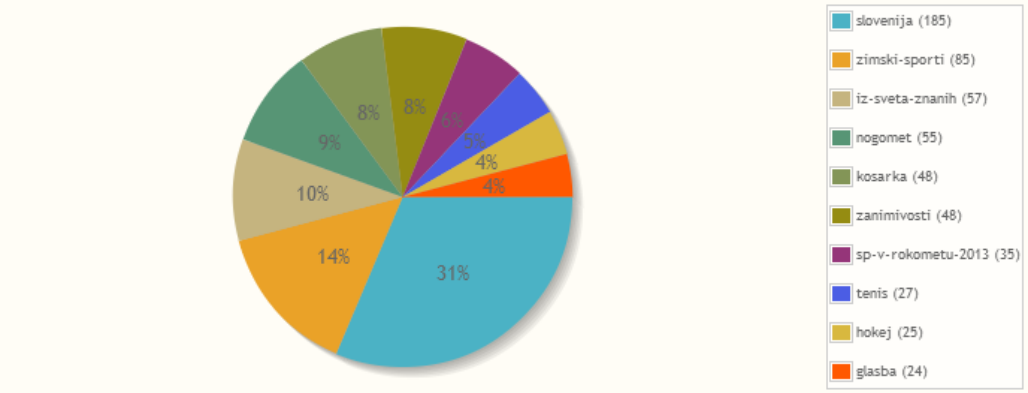
v svojem profilu lahko izbere spol, ni pa nujno. Zaradi tega so na sliki prikazane tri možnosti: moški spol, ženski spol ali pa spol ni nastavljen. Morda je nekoliko presenetljiv rezultat, da ima večina bralcev nastavljen spol. Med testiranjem luščenja podatkov smo namreč pogosto dobivali primere, ko spol ni bil nastavljen.

6.2 Število novic po posameznih kategorijah

V tortni obliki smo prikazali tudi število novic po kategorijah. Za bolj pregleden prikaz smo prikazali samo prvi deset kategorij po številu novic. Tu je potrebno upoštevati tudi, da je lahko kakšna novica v več kategorijah, čeprav se to ne dogaja pogosto. Dodali pa smo tudi opcijo datumskega omejevanja, če bi mogoče želeli samo rezultate iz določenega časovnega okvira. Primer poizvedbe je prikazan na primeru 6.1, rezultat te poizvedbe pa na sliki 6.2. V poizvedbi sta `[zacetni_datum]` in `[koncni_datum]` zamenjani z dejanskimi vrednostmi. Ker nekatere kategorije s pripadajočimi novicami še manjkajo v shrambi, je rezultat morda zavajujoč.

Primer 6.1: Število novic po posameznih kategorijah

```
1 PREFIX dct: <http://purl.org/dc/terms/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

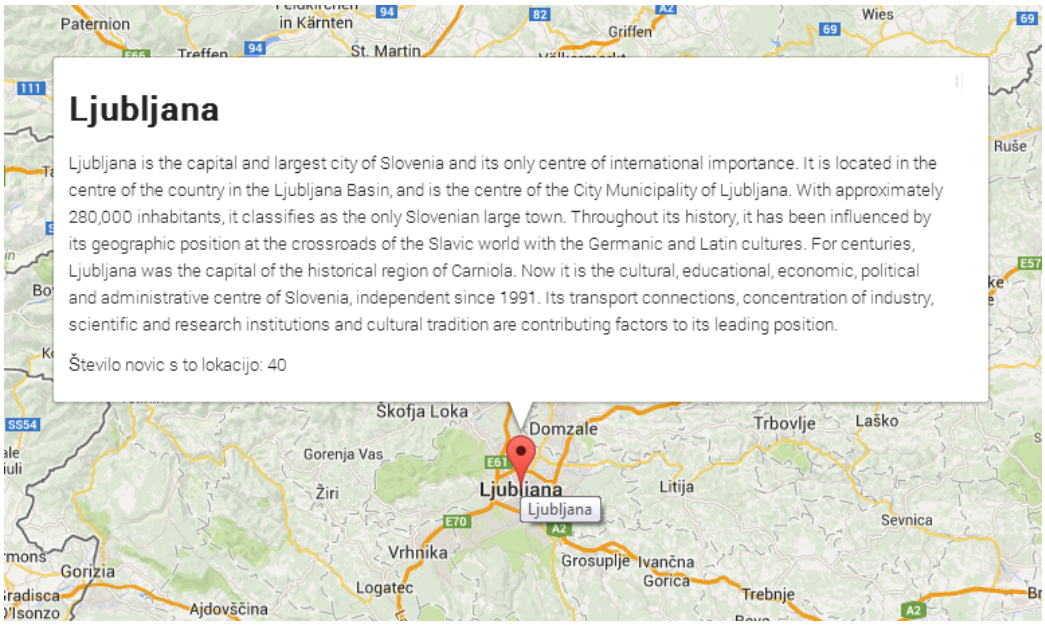


Slika 6.2: Število novic po posameznih kategorijah

```
4  PREFIX sioc: <http://rdfs.org/sioc/ns#>
5  SELECT ?category (COUNT(?post) as ?count)
6  WHERE {
7      ?post rdf:type sioc:Post .
8      ?post sioc:topic ?categoryGuid .
9      ?categoryGuid rdfs:label ?category .
10     ?post dct:created ?date .
11     FILTER(?date >= "[zacetni_datum]" && ?date <= "[koncni_datum]")
12 }
13 GROUP BY ?category
14 ORDER BY DESC(?count)
15 LIMIT 10
```

6.3 Število novic glede na lokacijo objave

Naslednji primer vizualizacije je malo bolj kompleksen, saj smo našo shrambo podatkov povezali z DBpedio [10]. Vsaka novica vsebuje tudi podatek o lokaciji objave novice. V shrambi trojčkov smo poiskali deset najpogostejših lokacij in jih prikazali na zemljevidu, kot prikazuje slika 6.3. Poleg lokacije smo dodali tudi opis lokacije. Ker teh podatkov naša shramba nima, nam pride tu v pomoč povezava na DBpedio, ki hrani te podatke. Z DBpedie



Slika 6.3: Število novic glede na lokacijo objave

tako dobimo opis lokacije in seveda koordinate lokacije, da lahko prikažemo točko na zemljevidu. Pride pa do problema, če obstaja več lokacij z enakim imenom. V tem primeru lahko pridobimo opis in koordinate napačne lokacije. Naša shramba namreč hrani samo ime lokacije, nič bolj podrobnega. Naslednja slabost, do katere lahko pride, je pa različno črkovanje imen lokacij v različnih jezikih ali pa celo različno poimenovanje. Na DBpedii namreč iščemo po angleškem naslovu lokacije, saj imamo tako največjo verjetnost za zadetek.

6.4 Novice iz izbrane slovenske regije

Zadnji primer je najbolj kompleksen, kjer pride zelo do izraza povezava na drugo shrambo podatkov in izkoriščanje tujih podatkov. Ponovno smo uporabili podatke iz DBpedije, kjer želimo prikazati samo novice, objavljene v izbrani slovenski regiji. V DBpediji najprej poiščemo vse slovenske regije, na podlagi katerih bomo izvedli filtriranje. Uporabljena poizvedba je prikazana

na primeru 6.2 in nam vrne rezultate, ki so prikazani v tabeli 6.1. Na uporabljeni poizvedbi se opazi tudi, da smo uporabili nizozemski prevod imen. To smo storili zato, ker nam je tak prevod vrnil najbolj razumljive rezultate. Slovenski prevod žal ne obstaja.

Tabela 6.1: Vse slovenske regije

Regije
Gorenjska
Jugovzhodna Slovenija
Koroška
Notranjskokraška
Obalnokraška
Osrednjeslovenska
Podravska
Pomurska
Savinjska
Spodnjeposavska
Zasavska

Primer 6.2: Vse slovenske regije

```
1  PREFIX dbpedia: <http://dbpedia.org/resource/>
2  PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
3  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4  SELECT ?label
5  WHERE {
6      SERVICE <http://dbpedia.org/sparql> {
7          ?region dbpedia-owl:type dbpedia:Statistical_regions_of_Slovenia .
8          ?region rdfs:label ?label .
9          FILTER(langMatches(lang(?label ), "NL"))
10     }
11 }
```

Primer 6.3: Novice iz izbrane regije

```
1  PREFIX dbpedia: <http://dbpedia.org/resource/>
2  PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
3  PREFIX dct: <http://purl.org/dc/terms/>
4  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
6  PREFIX news: <http://opendata.lavbic.net/news/>
7  PREFIX sioc: <http://rdfs.org/sioc/ns#>
8  SELECT DISTINCT ?post ?locationName ?seeAlso
9  WHERE {
10     ?post rdf:type sioc:Post .
11     ?post news:location ?locationName .
12     ?post rdfs:seeAlso ?seeAlso .
13     ?post dct:created ?date .
14
15     SERVICE <http://dbpedia.org/sparql> {
16         ?region dbpedia-owl:type dbpedia:Statistical_regions_of_Slovenia .
17         ?region rdfs:label "[ime_regije]"@NL .
18
19         { ?city ?x ?region } UNION { ?region ?z ?city }
20         ?city ?y ?locationName
21     }
22     FILTER(langMatches(lang(?locationName ), "EN") &&
23         ?date >= "[zacetni_datum]" && ?date <= "[koncni_datum]")
24 }
```

Ko izberemo regijo in opcijsko tudi časovni okvir, ki nas zanima, nam poizvedba na primeru 6.3 vrne vse ustrezne novice. Pri čemer so `[ime_regije]`, `[zacetni_datum]` in `[koncni_datum]` zamenjani z dejanskimi vrednostmi. V poizvedbi zelo površno gledamo, kdaj novica spada v določeno regijo. Želimo samo, da obstaja trojček, ki ima za objekt ali subjekt izbrano regijo, kar zagotovimo v vrstici 19 na primeru. V vrstici 20 pa primerjamo lokacijo objave novice, da ugotovimo, če novica spada v željeno regijo. Čeprav iščemo zelo

površno, vseeno dobimo spodbudne rezultate.

V tabeli 6.2 je prikazanih nekaj novic, ki jih dobimo, če iščemo novice iz Osrednjeslovenske regije.

Tabela 6.2: Primer novic iz izbrane regije

Lokacija	**Povezava na novico**
Ljubljana	http://www.rtvslo.si/sport/citat-za-prebrat/peter-prevc/288418
Ljubljana	http://www.rtvslo.si/slovenija/ajnur-rojen-v-brezicah-je-prvi-novorojencek-v-letu-2013/299206
Ljubljana	http://www.rtvslo.si/sport/nogomet/zahovic-zadovoljen-da-je-katanec-spet-selektor/299209
Ljubljana	http://www.rtvslo.si/slovenija/za-vzdrznost-pokojninske-blagajne-bo-treba-delati-dlje/299210
Ljubljana	http://www.rtvslo.si/slovenija/z-novim-letom-drazje-komunalne-storitve/299215
Ljubljana	http://www.rtvslo.si/slovenija/virant-v-drzavnem-holdingu-zeli-strokovnjake-in-ne-strankarskih-kadrov/299457

Poglavje 7

Zaključek

V tej diplomski nalogi smo najprej splošno opisali, kaj je RDF, za kaj se uporablja, ter njegove prednosti in slabosti. V nadaljevanju smo pregledali trenutne aktivnosti na področju shranjevanja podatkov v obliki RDF in poizvedb po teh podatkih.

Glavni namen diplomske naloge je bilo implementirati aplikacijo, ki bo shranjevala podatke iz specifičnega vira v shrambo trojčkov RDF. Na koncu smo vse skupaj podprli še s preprosto vizualizacijo doseženega dela in zmožnostih podatkov shranjenih na tak način. Za vir podatkov smo si izbrali spletno stran RTV Slovenije, iz katere smo luščili podatke o objavljenih novicah, komentarjih na te novice in avtorjih komentarjev. Vse te podatke smo shranjevali v obliki RDF v shrambo trojčkov. Aplikacija je zasnovana, kot Windows Service in je dovolj robustna, da lahko sčasoma pridobi vse podatke in jih tudi stalno osvežuje. Tako smo pripomogli, da je še en spletni vir več shranjen tudi v strukturirani obliki. Tu se odpira priložnost za računalniške aplikacije, ki bi želele izkoriščati podatke s spletnega vira RTV Slovenije, saj je to sedaj mogoče.

7.1 Nadaljnje delo

To diplomsko delo se lahko nadaljuje v več smereh. Za nadaljevanje zastavljenega dela je celotna izvorna koda aplikacije dostopna na [36].

Lahko se uporabi za shranjevanje podatkov s kakšnega drugega spletnega vira, pri čemer bi morali prilagoditi luščenje podatkov. Seveda ob predpostavki, da je tematika spletnega vira podobna in zajema podobne atribute.

Lahko bi se tudi iskale kakšne zakonitosti v pridobljenih podatkih, ko bi shramba vsebovala zadostno količino podatkov. Marsikaj se namreč lahko odkrije v veliki količini strukturiranih podatkov.

Obstaja pa tudi možnost tesne povezave s kakšno drugo podobno shrambo. Tako bi lahko tudi na podlagi povezave odkrili kakšno zanimivost v podatkih. Možnosti se odpira kar nekaj.

Literatura

[1] RTV Slo, Prvi interaktivni multimedijski portal, Dostopno na:
http://rtvslo.si

[2] 24ur, Dostopno na
http://www.24ur.com/

[3] Resource Description Framework, RDF (2004), Dostopno na:
http://www.w3.org/RDF/

[4] Resource Description Framework, RDF, Dostopno na:
http://www.rdfabout.com/

[5] Resource Description Framework, RDF (2004), Dostopno na:
http://en.wikipedia.org/wiki/Resource_Description_Framework

[6] N3 notacija za RDF, Dostopno na:
http://www.w3.org/TeamSubmission/n3/

[7] Primerjava RDF in XML, Dostopno na:
http://www.rdfabout.com/intro/#ComparingRDFwithXML

[8] Poizvedovalni jezik SPARQL (2008), Dostopno na:
http://www.w3.org/TR/rdf-sparql-query/

[9] Poizvedovalni jezik SPARQL, Dostopno na:
http://en.wikipedia.org/wiki/SPARQL

[10] DBpedia, Dostopno na:
http://dbpedia.org

[11] O DBpediji, Dostopno na:
http://wiki.dbpedia.org/About

[12] Delovanje DBpedije, Dostopno na
http://wiki.dbpedia.org/DBpediaLive

[13] DBpedia na wikipedia, Dostopno na:
http://en.wikipedia.org/wiki/DBpedia

[14] The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH), Dostopno na:
http://www.openarchives.org/pmh/

[15] Virtuoso Universal Server, Dostopno na:
http://virtuoso.openlinksw.com/

[16] Apache Tomcat, Dostopno na:
http://tomcat.apache.org/index.html

[17] Apache log4net, Dostopno na:
http://logging.apache.org/log4net/

[18] HTML Agility Pack, Dostopno na:
http://htmlagilitypack.codeplex.com/

[19] dotNetRDF, Dostopno na:
http://www.dotnetrdf.org/

[20] LINQ, Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx

[21] NuGet, Dostopno na:
http://www.nuget.org/

[22] Castle Project, Dostopno na:
`http://www.castleproject.org/`

[23] ASP.NET MVC, Dostopno na:
`http://www.asp.net/mvc/overview/what-is-mvc`

[24] Opendata news, Dostopno na:
`http://opendata.lavbic.net/news/`

[25] The Linking Open Data diagram, Dostopno na:
`http://lod-cloud.net/`

[26] Uniform resource identifier, Dostopno na:
`http://en.wikipedia.org/wiki/Uniform_resource_identifier`

[27] Universally unique identifier, Dostopno na:
`http://en.wikipedia.org/wiki/Universally_unique_identifier`

[28] XML Path Language (XPath) 2.0 (Second Edition) (2010), Dostopno na:
`http://www.w3.org/TR/xpath20/`

[29] XML Path Language (XPath), Dostopno na:
`http://en.wikipedia.org/wiki/XPath`

[30] MVC, Dostopno na:
`http://en.wikipedia.org/wiki/Modelviewcontroller`

[31] Document Object Model (DOM), Dostopno na:
`http://en.wikipedia.org/wiki/Document_Object_Model`

[32] Virtuoso Sponger, Dostopno na:
`http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger`

[33] Schema.org, Dostopno na:
`http://schema.org/`

[34] Apache Any23, Dostopno na:
http://any23.apache.org/

[35] Semantic Fire, Dostopno na:
https://code.google.com/p/semantic-fire/

[36] Izvorna koda aplikacije, Dostopno na:
https://github.com/jlogar7/RtvSloScraping