UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Urban Puhar

Semantična integracija podatkov za podporo iskanja kadrov

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

mentor: doc. dr. Rok Rupnik

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01760/2011
Datum: 05.04.2011

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

Kandidat: **URBAN PUHAR**

Naslov: **SEMANTIČNA INTEGRACIJA PODATKOV ZA PODPORO ISKANJA KADROVSKIH PODATKOV**

SEMANTICAL DATA INTEGRATION TO SUPPORT THE SEARCH OF HUMAN RESURCE DATA

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Za problem iskanja kadrov z ustreznimi znanji, izkušnjami in kompetencami uporabite tehnologije semantičnega spletna in ontologije. Zasnujte bazo znanja, ki bo temeljila na arhitekturi ontologij, v kateri bodo poleg osrednje ontologije, ki opisuje razmerja med koncepti področja kadrovskih podatkov, tudi ontologije virov eLance, LinkedIn in drugih.

Mentor:
doc. dr. Rok Rupnik

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a _____,

z vpisno številko _____,

sem avtor/-ica diplomskega dela z naslovom:

_____

_____

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  _____

  in somentorstvom (naziv, ime in priimek)

  _____

- 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 _____ Podpis avtorja/-ice: _____

Zahvala je namenjena mentorju doc. dr. Roku Rupniku in asistentu dr. Dejanu Lavbiču za strokovno pomoč in usmerjanje pri izdelavi diplomske naloge ter staršem in Tini za podporo in potrpežljivost v času pisanja.

Kazalo

Seznam slik

Seznam tabel

KRATICE, OKRAJŠAVE, SIMBOLI

Semantic Web	**Semantični splet** je razširitev trenutnega spleta, kjer imajo informacije točno določen pomen, tako da je predvsem izboljšano sodelovanje med ljudmi in računalniki. Semantični splet prinaša ogrodje, ki omogoča ponovno uporabo podatkov med aplikacijami in tudi podjetji.
XML eXtensible Markup Language	Razširljiv označevalni jezik.
HTML HyperText Markup Language	Označevalni jezik za določitev strukture dokumentov, ki se prenašajo po spletu in pregledujejo s spletnimi brskalniki.
Tag, tagging	**Oznaka**, zaznamek, **označevanje**, »etiketiranje«.
URL Universal Resource Identifier	Spletni naslov.
RDF	Jezik za opisovanje in modeliranje informacij, vsebovanih v spletnih virih.
WWWC ali W3C World Wide Web Consortium	Konzorcij, mednarodna organizacija za standardizacijo svetovnega spleta.
Shema RDF	Preprost jezik za opisovanje razredov virov in relacij med njimi znotraj RDF modela.
xQuery	Funkcijski jezik za iskanje po podatkovnih strukturah v XML
OWL Ontology Web Language	Jezik za zapis ontologij.
API	Aplikacijski programski vmesnik.
Agent	Agent je računalniški sistem, ki je nameščen v neko okolje in je zmožen avtonomnih akcij v tem okolju z namenom, da doseže načrtovane cilje.
MVC Model View Controller	Arhitektura, sestavljena iz treh ločenih delov (model, prikaz in usmerjevalnik) za grajenje spletnih strani.
IDE integrated development environment	Aplikacija, ki zagotavlja celovito okolje programerjem za razvoj programske opreme.
RDFa Resource Description Framework – in – attributes	Jezik RDF za vpenjanje v XHTML dokumente

XHTML eXtensible HyperText Markup Language	Družina jezikov XML, ki razširjajo jezik HTML

POVZETEK

Namen diplomskega dela je raziskati pojem semantike kot študij pomena, v povezavi s svetovnim spletom. V prvi fazi gre za podrobno analizo trenutnega stanja, kjer sem se osredotočil na že uveljavljene formate za označevanje spletnih strani z vstavljanjem, računalniku razumljivih metapodatkov o pomenu njihove vsebine. Pregledal sem primere uporabe in kakšno dodano vrednost prinašajo. Druga faza je namenjena preučevanju ideje o semantičnem spletu kot tehnična rešitev, proti kateremu naj bi današnji splet konvergiral.

S konkretnim primerom uporabe, ki sloni na podatkih pridobljenih iz realnih virov, sem predstavil njune razlike. Za domeno sem si izbral iskanje ljudi ali skupine ljudi na borzi dela, v kateri je zbranih več internetnih strani, ki služijo kot viri informacij. Primer uporabe, ki sem si ga zamislil je umeščen v projekt, kjer se v določeni fazi ugotovi, da za njegovo dokončanje v zastavljenem času, ni dovolj ljudi. Tako moramo poiskati dodatne ljudi z ustreznim znanjem in ostalimi želenimi kriteriji.

Rezultat je sistem, ki je sestavljen iz dveh večjih delov. V prvem delu, ki ustreza prvi fazi, podsistem črpa informacije iz različnih virov s tehniko luščenja oz. ločevanjem podatkov od predstavitve.

Drugi del, ki je rezultat druge faze, je predstavljen kot baza znanja, ki temelji na medsebojno povezanih ontologijah različnih virov. Ker je vodilo, da se čim bolj nazorno predstavi razlike, sem bazo znanja, povezal še s spletnim uporabniškim vmesnikom za izvajanje povpraševanj.

Ključne besede: semantični splet, ontologija, človeški viri, metapodatki

ABSTRACT

The purpose of this thesis is to explore the idea of semantics being the studies of meaning in relation to the world wide web. The first phase involves a detailed analysis of the current situation, where I focused on the already established forms of web page marking with insertion of machine-understandable metadata about their meaning. The second phase is intended to examine the idea of semantic web as a technical solution, towards which today's web should converge.

Consequently, I presented the differences between the two with a concrete example of usage based on the data gathered from various real sources. The domain of this study included searching for people or groups of people at the labour exchange, which includes many web pages that serve as sources of information. The above said concrete example is placed within the project, which at a certain point reveals that in order for it to be finished in time, there are not enough people. Consequently, we need to find more people with suitable knowledge and other desired criteria.

The result is a system composed of two major parts. Within the first part, corresponding to the first phase, the subsystem gathers information from different sources by scrapping or separating data from presentation.

The second part, as the result of phase two, is presented as a knowledge directory based upon the interconnected ontologies of different sources. On account of my aim – highlighting the differences as much as possible – I have linked the said knowledge database with a web user interface for executing queries.

Keywords: semantic web, ontology, human resources, metadata

1 UVOD

V času spleta 1.0 (Web 1.0) je bil cilj spletnih strani le podajanje informacij uporabniku, kot je to še vedno pri televiziji. S prihodom spleta 2.0 (Web 2.0) so postale spletne strani dinamične. Poleg ponujanja informacij se sedaj sprejema tudi podatke uporabnikov in se na podlagi le-teh proizvaja in vrača rezultate. Spletni iskalniki uporabljajo ključne besede spletnih strani, za prikaz rezultatov iskalnega niza. To pomeni, da je vsebina indeksirana na podlagi ključnih besed ali značk, in ne na podlagi pomena.

Pojavlja se drug problem. Količina informacij, ki nam je na voljo, se povečuje eksponentno. Vedno več virov je dostopnih, vedno več jih nastaja in se razvija. Vsak dan postajata vidljivost informacije in njeno iskanje težje in kompleksnejše. Velikokrat prva stran rezultatov, ki nam jo spletni iskalnik vrne, ne zadostuje našim potrebam. Uporabniki za iskanje zahtevajo vse bolj zaletene, specifične in naravne poizvedbe. Nujno je, da se računalniki dvignejo na višji nivo razumevanja. Znati morajo interpretirati in povezati iskalni niz uporabnika z informacijami na različnih virih.

Nastopi ideja o semantičnem spletu. Semantika je študij pomena. Semantični splet je mreža virov informacij, povezanih skupaj tako, da jih računalnik na globalni ravni lahko procesira. Gre za učinkovit način predstavitve podatkov na spletu, oz. za globalno povezano podatkovno bazo.

Podatki so trenutno skriti v HTML dokumentih in so v določenih kontekstih uporabni, vendar ne vedno in povsod. Problem z večino podatkov na svetovnem spletu v trenutni obliki je, da je težko uporabna v velikem obsegu: zato, ker ni globalnega sistema za objavo podatkov na način, da bi jih lahko vsakdo obdelal (računalnik ali človek). Kot primer si lahko predstavljamo informacije glede športnih dogodkov, vremenskih informacij, letalskih urnikov in televizijskih sporedov. Vse informacije so predstavljene na različnih straneh v HTML obliki. Problem je, da je v določenih kontekstih zelo težko uporabiti te podatke na način, kot si ga želimo.

Semantični splet je, kot na velika tehnična rešitev (in več kot to). Gre za splet podatkov, ki omogoča računalnikom, da razumejo semantiko ali pomen informacij na svetovnem spletu. Razširja mrežo, človeku razumljivih spletnih strani, z vstavljanjem računalniku razumljivih metapodatkov o pomenu vsebine in kako so med seboj povezane, kar omogoča spletnim agentom inteligentnejši dostop do spleta in opravljanje nalog v imenu uporabnika [17].

V diplomskem delu želim s primerom predstaviti vrzel med trenutnim spletom in idejo o semantičnem spletu. Na eni strani so viri kot spletne strani, ki hranijo podatke zajete v HTML zapisu. Na drugi strani je ideja o semantičnem spletu, ki zahteva ločitev podatkov od njihove predstavitve in omogoča njihovo medsebojno povezanost prek več virov.

1.1 Uporabljen pristop

Rešitev je sistem, ki črpa informacije iz različnih virov in jih nato zapiše v naprej definirane, viru ustrezne ontologije, ki skupaj tvorijo bazo znanja. Ker je vodilo, da se čim bolj nazorno predstavi razlike, sem z bazo povezal še spletni uporabniški vmesnik za izvajanje povpraševanj in celoten sistem umestil v izbrano domeno.

Preden sem začel z izgradnjo sistema, sem analiziral obstoječe rešitve in pregledal tehnologije. To je podrobno opisano v 2. poglavju. Raziskal sem poizkuse semantičnega

označevanja, kot so, Microdata, Microformats in RDFa, in jih primerjal med seboj. Podrobno sem preučil idejo semantičnega spleta in izbral razvojno okolje, v katerem sem nato implementiral rešitev.

V 3. poglavju sledi podrobna predstavitev rešitve. Rešitev je sestavljena iz dveh večjih delov. Prvi del vključuje analizo ureditve podatkov na spletnih straneh in njihovo ločevanje od predstavitvenega dela ter njihovo urejanje za vnos v strukturo ontologij. Ta služi, kot ponazoritev problema računalniške interpretacije vsebine. V drugi del spada izgradnja ontologij za posamezne vire, njihovo povezovanje in vnos podatkov pridobljenih iz spletnih strani. Semantično urejeni podatki so nato na voljo uporabnikom, ki nad njimi izvajajo želena povpraševanja. Skušal sem se približati implementaciji koščka semantičnega spleta in tako predstaviti njegovo moč in potencial, ki ga ideja prinaša.

2 RAZISKOVALNA PODROČJA

2.1 Semantični splet

V prihodnjih letih bomo priča napredku zmožnosti sistemov pri dostopu, procesiranju in uporabi informacij [2]. Odražal se bo predvsem na treh področjih, med sabo povezanih v tako imenovani *semantični splet*. To so splet podatkov, splet storitev in splet ponudnikov identitete. [11]

Količina informacij in spletnih storitev eksponentno narašča, kar je razlog, da se vsak dan človek težje dokoplje do potrebnih informacij. Naučiti se je potrebno, kako računalnikom povedati, kaj si nekdo sploh želi. Zakaj računalniki takoj ne vedo, katera stran, fotografija ali objava na enem izmed socialnih omrežji je prava?

Zato, ker je to nemogoče. Računalniki ne razumejo, nimajo dostopa do večine potrebnih virov, manjka jim semantično razumevanje in zdrava pamet, s katero bi povezovali informacije med seboj.

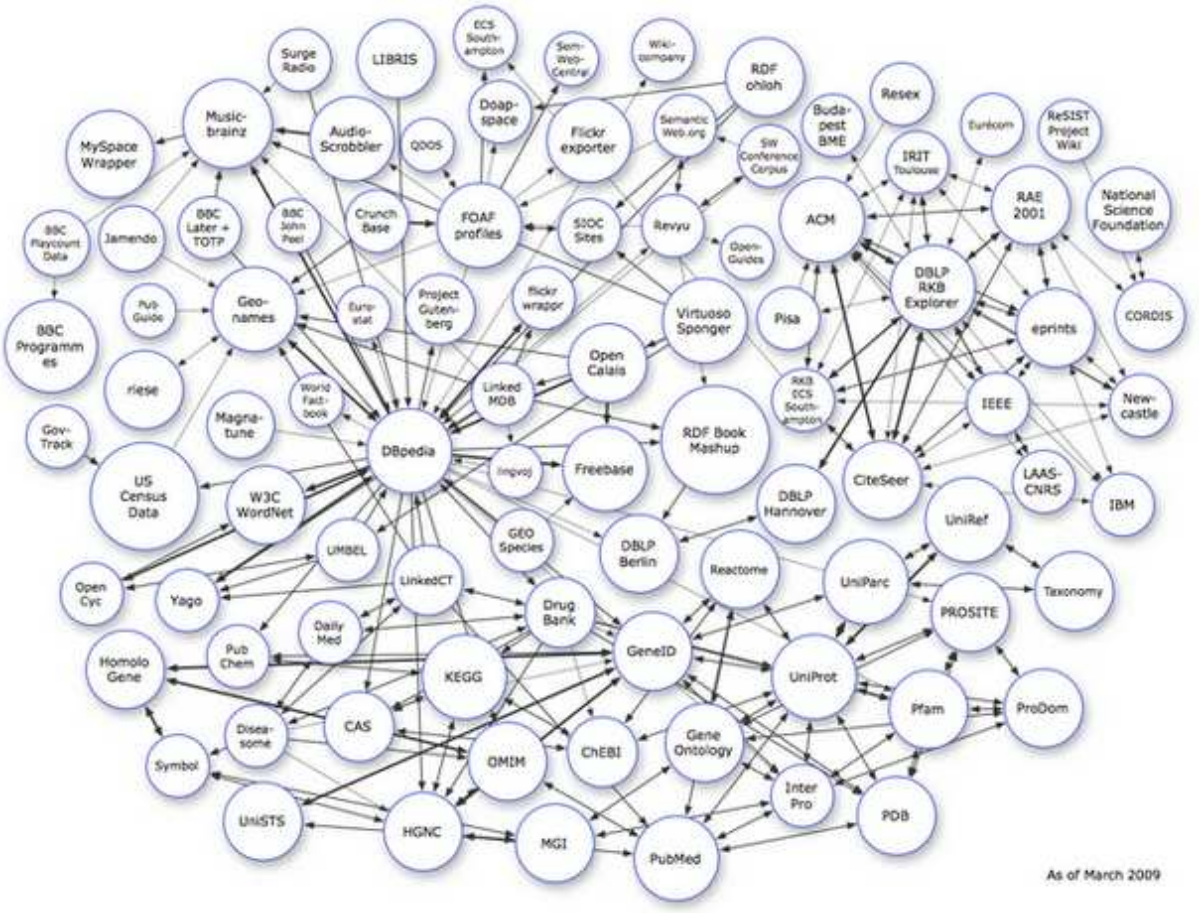
Bistveno je, da računalniki pridobijo nove ravni razumevanja. Namesto statističnega preverjanja, koliko se iskalni izraz ujema z virom informacije (spletni iskalniki) mora biti informacija na obeh straneh predstavljena na dovolj formalen način, da jo lahko računalniki interpretirajo. Potrebne so baze znanja. Primeri takih baz zajemajo:

- enciklopedijo, ki vsebuje znanje, potrebno za določitev semantičnega pomena in konteksta določenega izraza (npr. razumeti, da je Berlin mesto in ne priimek, koliko ljudi živi v njem in kje se nahaja),
- rumene strani spletnih storitev za pridobivanje vedno spreminjajočih se in kompleksnejših informacij (npr. pot od Ljubljane do Berlina z avtom, ali trenutna temperatura v Berlinu v stopinjah Celzija),
- podatkovno bazo za iskanje informacij o profilu (z dovoljenjem uporabnika), ki bi lahko izboljšale personalizacijo in priporočila želene informacije.

2.1.1 Splet podatkov

Ideja spleta podatkov izvira iz koncepta semantičnega spleta. Ljudje so skušali rešiti problem nezmožnosti razumevanja računalnikov vsebine internetnih strani. V začetku je bila oz. je še vedno ideja, da bi se internetne strani označevalo z množico meta podatkov. Prek teh podatkov bi računalnik vedel, kaj je vsebina strani in to postavil v določen kontekst. Ideja ni uspela v želenem obsegu, ker je bilo označevanje vsebine za ljudi, ki niso imeli tehničnega predznanja, pretežko. Sčasoma so se pojavili pristopi, ki poenostavljajo označevanje strani, vendar to še ni želena rešitev za koncept semantičnega spleta.

Ideja o spletu podatkov temelji na golih podatkih, ločenih od predstavitvenega dela. Splet podatkov je rezultat prej omenjenih omejitev in obstoj številnih strukturiranih zbirk podatkov (golih, neformatiranih), ki so porazdeljeni po celotnem spletu in vsebujejo različne informacije. Zbirke so v lasti podjetij, ki jih skušajo narediti dostopne (plačljivo ali zastonj). Tipične so zbirke, ki vsebujejo najrazličnejša znanja o določeni domeni: knjige, glasba, enciklopedični podatki, podatki o podjetjih, itd. Če bi bile te zbirke medsebojno povezane (kot so to npr internetne strani), bi to omogočalo računalnikom neomejeno, neovirano in neodvisno sprehajanje po urejenih podatkovnih zbirkah. Rezultat je ogromna, prosto dostopna baza znanja, ki bi bila temelj novi generaciji aplikacij in storitev.



Slika 1: Množica povezanih podatkovnih zbirk, Linking Open Data

Eden izmed obetavnih projektov je W3C Linking Open Data projekt.

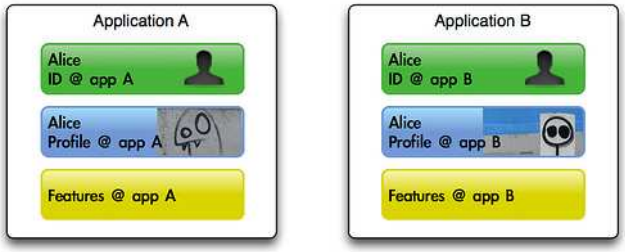
Slika 1 prikazuje množico med seboj povezanih podatkovnih zbirk, ki so vključene v ta projekt. Podatkovna zbirka je shranjena v obliki ontologije. Ontologije so formalna predstavitev množice konceptov z nekega področja in razmerij med njimi. Da se jih razširiti, ponovno uporabiti in med seboj povezati. Zato so primerne za predstavitev baze znanja shranjene v podatkovnih zbirkah.

Linking Open Data projekt šteje trenutno več kot 3 bilijona RDF trojic. RDF trojica je osnovni konstrukt ontologije, ki vsebuje informacijo. Sestavljena je iz treh gradnikov: objekt (osebek), atribut (predikat) in vrednost (predmet). Vsi gradniki skupaj predstavljajo trditev.

Trenutno lahko dostopamo do vseh podatkovnih zbirk brezplačno in jih lahko uporabljamo neomejeno.

2.1.2 Splet identitet

Splet podatkov brez omejitev dostopa je primeren za enciklopedičen tip informacij. Če bi hoteli dostopati do osebnih podatkov oziroma podatkov, ki so last nekoga, bi se stvari hitro zakomplicirale. Zato tu nastopi splet identitet.



Slika 2: Vsaka spletna aplikacija s svojo digitalno identiteto.

Ne dolgo nazaj je bilo stanje tako, da se je za vsako spletno aplikacijo, na primer socialno omrežje, katere funkcionalnosti smo hoteli uporabljati, bilo potrebno registrirati oziroma ustvariti novo digitalno identiteto. Kar posledično pomeni, da mora vsak ponudnik implementirati svoj sistem za upravljanje z identitetami.

Problemi, ki se pri tem pojavijo so:

- uporabniki morajo ustvariti več digitalnih identitet,
- uporabniki morajo posodabljati profile pri vsakem ponudniku storitev,
- zasebnost, lastništvo uporabnikovih podatkov.

Pojavljajo se seveda nove rešitve. Ena izmed teh se imenuje SSO (single sign-on), kar pomeni enkratna identifikacija za več spletnih strani oziroma aplikacij. Novi ponudniki storitev lahko tako svoj sistem za digitalno identiteto prepustijo drugim, že obstoječim ponudnikom. Primer je Facebook, ki ostalim spletnim stranem omogoča, da se lahko vpišejo v sistem prek njihovega vmesnika z njihovo identiteto. Tudi tukaj so se stvari do neke mere standardizirale. Ena izmed rešitev je OpenId.

2.1.3 Kaj je Open ID?

OpenID je odprto, decentralizirano, brezplačno ogrodje namenjeno digitalnim identitetam. Uporabnikom omogoča prijavo na različne spletne strani z istim OpenID uporabniškim imenom in geslom. V svetu obstaja že kar nekaj OpenID ponudnikov. Med katerimi so bolj poznani Google, Yahoo!, MySpace in Facebook. Tako lahko uporabnik izbere ponudnika, ki mu najbolj ustreza in si tam ustvari digitalno identiteto, s katero se predstavi pri ponudnikih ostalih storitev. Uporabnik lahko določa, do katerih osebnih podatkov ima posamezna spletna aplikacija dostop.

Omenja se še standard OAuth. OAuth (Open Authorization) je odprt standard za avtorizacijo. Uporabnikom omogoča, da lahko podatke (na primer slike, videe, imenik) shranjene na enem spletnem mestu delijo z drugimi, ne da bi zaupali svoje uporabniško ime in geslo oz. kakršno koli informacijo, ki se nanaša na dostop do osebnih podatkov. V prihodnosti bodo verjetno obstajali ponudniki, katerih primarna funkcija bo upravljanje z digitalnimi identitetami.

2.1.4 Splet internetnih storitev

Čeprav je standardizacija sistemov s storitveno usmerjeno arhitekturo (SOA) potekala več let, še vedno ni jasne opredelitve, kaj sploh pomeni storitev na konceptualni ravni. Vmesnik (interface), ki je format za to, kaj gre ven in kaj pride noter, je pogosto opisan formalno. Kaj storitev dejansko počne, pa semantično gledano ni. Obstajajo seveda drugačni pristopi (OWL-S, WSMO, WSDL-S), vendar so ti še vedno na akademski ravni.

Danes obstaja veliko vrst storitev z različnim nivojem kompleksnosti, ki sledijo različnim standardom. Njihovo število se povečuje. Za iskanje ustrezne storitve so na voljo katalogi, kot je *ProgrammableWeb.com* in iskalniki spletnih storitev, kot je *seekData.com*. Vendar so ti namenjeni »samo« ljudem. Kot zanimivost, obstajajo tudi storitve za naloge, ki jih računalniki ne znajo rešiti, za katerimi stojijo ljudje, ki odgovarjajo na zahtevna vprašanja (Amazon Mechanical Turk).

Problem že obstoječih storitev je ta, da niso najdene s strani računalniških sistemov, ker ti ne znajo interpretirati njihovega opisa, ki je namenjen »samo« ljudem in je tako premalo formalen. Torej, kaj bi lahko dosegli, če bi bile storitve semantično opisane?

- iskanje storitev: sistem, zadolžen za iskanje, bi sam pregledal seznam in izbral ustrezno storitev za podan problem,
- pogajanje: ko bi bila storitev izbrana, bi sistem nadaljeval s pregledovanjem pogojev uporabe in njeno ceno oz. cenik. Na podlagi teh informacij in podanega problema, bi lahko sistem spet sam izbral ustrezno opcijo uporabe storitve,
- zamenjava ob odpovedi: če bi se na storitvi pojavila napaka ali okvara, bi lahko nadzorni sistem takoj poiskal novo ustrezno enakovredno storitev,
- orkestracija storitev: sistem bi podano nalogo tudi razdelil na pod-naloge in te dodelil posameznim storitvam. Tako bi lahko dosegli celo vzporedno izvajanje in s tem pohitritev izvajanja celotne naloge.

2.2 Pridobivanje podatkov

V prejšnjem poglavju je bila opisana smer razvoja idealnega semantičnega spleta. Ena izmed prednosti, ki jih nosi semantični splet, je tudi ta, da so podatki ločeni od predstavitve, kar omogoča lažji dostop do njih.

Za pridobivanje podatkov pri omenjeni domeni je bilo potrebno pristopiti nekoliko drugače. Spletne strani, iz katerih sem črpal podatke, jih žal še ne ponujajo surovih, ločenih od predstavitve (v tem primeru HTML) in urejenih v ontologije. Podatke je bilo potrebno izluščiti iz HTML strani. Uporabil sem kombinacijo dveh pristopov: *xQuery*, ki sloni na *xPath*-u, in regularne izraze. Z xQuery-jem sem najprej izluščil del teksta iz HTML struktur, nato pa iz tega teksta izluščil še potrebno informacijo s pomočjo regularnih izrazov.

2.2.1 xPath

xPath (XML Path Language), specificiran s strani W3C mednarodnega inštituta, je jezik poizvedb, ki se uporablja za naslavljanje delov znotraj XML dokumenta. Temelji na drevesni strukturi in omogoča naslavljanje vozlišč na podlagi različnih kriterijev. xPath-ovi izrazi v dokumentu XML določajo vozlišča na osnovi njihovega tipa, imena in vrednosti ter relacije vozlišča do drugih vozlišč v dokumentu.

Primer XML dokumenta (Tabela 1):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
</book>
<book category="CHILDREN">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
</book>
<book category="WEB">
        <title lang="en">XQuery Kick Start</title>
        <author>James McGovern</author>
        <author>Per Bothner</author>
        <author>Kurt Cagle</author>
        <author>James Linn</author>
        <author>Vaidyanathan Nagarajan</author>
        <year>2003</year>
        <price>49.99</price>
</book>
<book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book></bookstore>
```

Tabela 1: Vsebina XML dokumenta

Elementi xPatha, s katerimi naslavljamo vozlišča. Zapis naslavljanja in rezultati, se nanašajo na zgornjo tabelo (Tabela 1).

Izbira vozlišč:

Zapis poizvedbe: */bookstore/book/title*

V tem primeru izberemo oz. naslovimo vsa vozlišča z imenom *title* (naslov).

Rezultat: Everyday Italian, Harry Potter, XQuery Kick Start, Learning XML

Predikati:

Zapis poizvedbe: /bookstore/book[1]/title

V tem primeru izberemo oz. naslovimo prvo vozlišče z imenom *title* (naslov).

Rezultat: Everyday Italian

Operatorji:

Zapis poizvedbe: / bookstore/book[price>35]/price

V tem primeru izberemo oz. naslovimo vozlišče z imenom *price* (naslov), ki ima vrednost več kot 35.

Rezultat: 49.99, 39.95

Zapis poizvedbe:/bookstore/book[price>35]/title

V tem primeru izberemo oz. naslovimo vozlišče z imenom *title* (naslov), ki ima vrednost cene več kot 35.

Rezultat: XQuery Kick Start, Learning XML

2.2.2 xQuery

xQuery je funkcijski programski jezik za iskanje po podatkovnih strukturah v XML dokumentu. Lahko ga primerjamo z jezikom SQL, ki služi za povpraševanje po podatkovni bazi. Prav tako, kot xPath je xQuery standard, ki ga je razvil mednarodni inštitut W3C. Sloni na xPath-u, s katerim naslavlja dele XML dokumenta. Poleg tega pa uporablja tudi SQL-u podobne izraze imenovane FLWOR. FLOWR je skupina petih izrazov, katerih prve črke sestavljajo njegovo ime. To so: FOR, LET, WHERE, ORDER BY in RETURN.

Lahko ga uporabljamo za primere, kot so:

- generiranje poročil,
- preslikavo XML podatkov v HTML obliko,
- iskanje informacij znotraj internetnih strani.

Primer uporabe:

Kot vir podatkov bom uporabil podatkovno strukturo iz prejšnjega xPath primera (Tabela 1).

- Primer 1:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

V tem primeru izberemo vse *naslove* knjig, ki imajo *price* (ceno) višjo kot 30. Isti rezultat bi dosegli s pomočjo xPath izraza: */bookstore/book[price>30]/title*

- Primer 2:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Zgornji izraz vrne isti rezultat, kot prejšnji, vendar s to razliko, da so sedaj naslovi razporejeni po abecednem redu.

- Primer 3:

```
<body>
<h1>Bookstore</h1>
<ul>
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li class="{data($x/@category)}">{data($x/title)}</li>
}
</ul>
</body>
</html>
```

Gre za primer izraza, kjer se vidi, da se lahko s pomočjo xQuery-ja preslikamo podatke iz ene oblike v drugo. V zgornjem primeru povprašujemo po podatkih iz XML dokumenta, katere nato oblikujemo z HTML jezikom.

Rezultat:

```
<html>
<body>
<h1>Bookstore</h1>
<ul>
<li class="COOKING">Everyday Italian</li>
<li class="CHILDREN">Harry Potter</li>
<li class="WEB">Learning XML</li>
<li class="WEB">XQuery Kick Start</li>
</ul>
</body>
</html>
```

2.2.3 Regularni izrazi

Z xPath-om in xQuery-em lahko dostopamo do posameznih vozlišč oz. skupine vozlišč in tako dobimo njihovo vsebino. Vsebina ni vedno goli, iskani podatek. Vsebuje lahko še različne znakovne nize, ki nas ne zanimajo in nam podatek včasih celo onesnažijo. Na primer, iz vozlišča *Location* dobimo naslednjo vsebino:

Location: Rocester, New York | United State

Vidimo, da niz vsebuje ime mesta, ime regije in ime države. Če želimo iz tega niza izluščiti posamezna imena, moramo uporabiti regularne izraze.

Regularni izraz (regular expressions), kratko tudi regex ali regexp je način opisa sestave podniza, ki izvaja iskanje v določenem nizu. Napisan je v formalnem jeziku, ki ga interpretira procesor regularnih izrazov.

Naslednji primeri ponazarjajo, kakšne nize lahko pričakujemo s podanimi izrazi:

- izraz kot zaporedje črk *car* lahko določa besede *car*, *cartoon* ali *bicarbonate*,
- izraz kot zaporedje črk *car* z možnostjo vmesnih črk določa *Icelander* ali *chandler*,
- izraz kot zaporedje črk *car* s predhodnimi besedami določa nize, kot so *blue car* ali *red car*.

Seveda so lahko izrazi tudi kompleksnejši.

Regularne izraze se uporablja v različnih urejevalnikih besedil in razvojnih okoljih za iskanje ali kodificiranje nizov.

Sintaksa

Regularni izraz, pogosto imenovan tudi vzorec, je izraz, ki opisuje nabor nizov. Na primer, set treh nizov *Handel*, *Händel* in *Haendel* je opisan z vzorcem *H(ä|ae?)ndel*. Vzorec lahko poleg nabora črk vsebuje tudi posebne znake, ki dodatno specificirajo množico nizov. Z njihovo kombinacijo lahko dosežemo kompleksnejše izraze.

Obstaja veliko posebnih znakov. Med najpogostejše spadajo:

- boolean »*or*«: pokončna črta »|« označuje logični operator *ali*. Z njo ločimo dva vzorca, ki vsak zase določa svojo množico nizov. Rezultat pa je unija teh dveh množic. Vzorec za *gray* in *grey* je:$gray|grey$
- grupiranje: za grupiranje se uporablja oklepaj. Uporablja pa se za določanje obsega in vrstnega reda ostalih operatorjev. Prejšnji primer bi lahko z uporabo oklepajev zapisali kot: $gr(a|e)y$
- kvantifikacija: kvantifikator po črki ali grupi (množici) določa, kolikokrat se lahko predhodni element ponovi. Med najpogostejše spadajo:
  - *»?«* določa, da se lahko element pojavi enkrat ali nikoli. Vzorec $colou?r$, določa besedi $color$ ali $colour$,
  - *»*«* določa, da se lahko element pojavi večkrat ali nikoli. Vzorec $ab*c$, določa ac, abc,$abbc$,$abbbc$, itn.,
  - *»+«* določa, da se lahko element pojavi večkrat ali najmanj enkrat: $ab+c$ določaabc, $abbc$, $abbbc$, itn. Vendar ne določa niza ac.,
- začetek niza »\A«,
- prazen niz »\b«.

Velja omeniti še ubežni znak »\«, ki se ga uporablja za vnos posebnega znaka v niz, kjer hočemo, da se ta znak interpretira kot zaporedje običajnih znakov ali črk. Če bi hoteli iskat nize, ki vsebujejo *»*:)«*, bi morali vzorec zapisati kot *»*:\)«*, kajti »*« se uporablja kot kvantifikator.

2.3 Označevanje vsebine (Microformats, RDFa in Microdata)

Za koncept semantičnega spleta je potrebno vsebino internetnih strani označiti, in sicer tako, da se tudi računalniki zavedajo pomena vsebine in različnih delov internetne strani. Šele tako se lahko npr. izvaja zahtevnejše internetne poizvedbe. Trenutno so nam na voljo trije standardi: Microformats, RDFa in Microdata. Navzven so si dokaj podobni. Njihov cilj je vnos semantične informacije v kodo (X)HMTL dokumenta, tako da vsebino interpretirata človek in tudi računalnik. Za označevanje dokumenta uporabljata atribute HMTL in XHTML dokumenta.

2.3.1 Microformats

Microformats so zelo razširjena in priljubljena množica formatov za vpenjanje metapodatkov v HTML zapis. Združljiv je s HTML 4.01 in XHTML 1.0. Njihova implementacija je zelo enostavna, saj že v osnovi sledijo filozofiji *najprej ljudje nato računalniki*. Podprti so s strani skupine blog agregatorja Tehnorati, ki so zaradi agregiranja razvili različne formate za označevanje vsebine bloga. Primeri takih formatov so:

- *vCard* format za predstavitev informacije o osebi, podjetju, organizacij. Vsebuje atribute kot na primer *fn*-polno ime, *n* – ime, *adr*-naslov, itd,
- *hCaledndar* format za predstavitev časovnega dogodka: vsebuje atribute *dtstart*-datum, *summary*-opis, *location*-lokacija,
- *hResume* je format za predstavitev življenjepisa, vsebuje atribute, kot so: *contact*-kontakt, *education* – izobrazba, *experience* – izkušnje.

XHTML in HTML standardi dovoljujejo dodajanje metapodatkov v kodo prek določenih atributov in označevalnih značk. Te atribute uporablja Microformats za dodajanje semantike v kodo dokumenta. Uporablja tri atribute, in sicer:

- *class*,
- *rel*,
- *rev* (samo v enem primeru, v vseh ostalih je prepovedano).

Primer uporabe:

Primer zapisa informacije o osebi v HTML obliki, brez semantičnega dela.

```
<div>
<div>Joe Doe</div>
<div>The Example Company</div>
<div>604-555-1234</div>
<a href="http://example.com/">http://example.com/</a>
</div>
```

Tabela 2: HTML zapis

hCard format pa slednji zapis (glej Tabela 2) zgleda tako:

```
<div class="vcard">
<div class="fn">Joe Doe</div>
<div class="org">The Example Company</div>
<div class="tel">604-555-1234</div>
<a class="url" href="http://example.com/">http://example.com/</a>
</div>
```

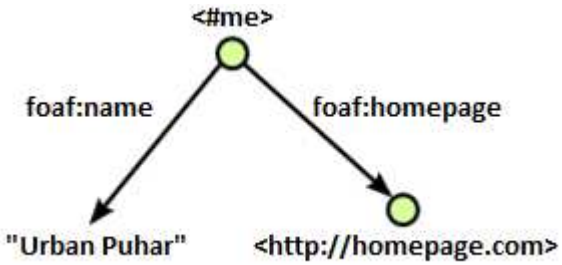
Tabela 3: HTML zapis oplemeniten s Microformats semantiko

Za razliko od prejšnjega primera (Tabela 2), so tukaj vsi podatki označeni z atributi formata *hCard* in tako točno se ve, na kaj se posamezni podatek nanaša. *fn* je uporabljen za polno ime, *org* za ime organizacije, podjetja, *tel* za telefonsko številko in *url* za naslov spletne strani osebe. Obstajajo dodatni atributi v tem formatu s katerimi bi lahko podrobneje opisali osebo. Tako strukturirano informacijo si računalnik interpretira oz. se zaveda njene vsebine. Programi, recimo spletni brskalnik, lahko tako izvlečejo iz spletne strani informacijo (v našem primeru vizitke) in jo posredujejo ostalim aplikacijam, kot je recimo imenik.

2.3.2 RDFa

Drugi format in način označevanja vsebine je RDFa. Za razumevanje RDFa formata je potrebno poznavanje RDF. RDF je format za zapisovanje informacije. Zapisan je v obliki usmerjenega grafa, katerega osnovni gradniki so: *objekt* (osebek), *atribut* (predikat) in *vrednost* (predmet). Vsi skupaj predstavljajo trditev.

Primer RDF:



Slika 3: RDF usmerjen graf

Graf (Slika 3) predstavlja osebo, njeno ime, spletno stran in definira povezave med njimi. V tem primeru je uporabljen razširjen format za predstavitev osebe, FOAF [3].

Primer zapisan v XML obliki:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/ .
<#me><foaf:name> "Urban Puhar" .
<#me><foaf:homepage><http://homepage.com>.
```

Zapis v oklepajih »<>«, je zapisan kot URL naslov. Ker so naslovi dokaj dolgi, se zaradi preglednosti oz. enostavnosti uporabe v kodi uporablja okrajšave. Strokovno so poimenovani *name space* (imenski prostor).

```
<p xmlns:foaf="http://xmlns.com/foaf/0.1/" about="#me">
```

Tako je *foaf:homepage* okrajšava za *http://xmlns.com/foaf/0.1/homepage*.

Povezave v grafu so predikati, to sta *foaf:name* in *foaf:homepage*. Predmet je lahko niz znakov ali pa drugi URL naslov. V tem primeru gre za povezavo na osebno stran.

Znak *»a«* na koncu kratice RDF se navezuje na to, kako so atributi v XHTML uporabljeni za vpenjanje RDF (glej Tabela 4):

```
<p xmlns:foaf="http://xmlns.com/foaf/0.1/" about="#me">
        I'm <span property="foaf:name">Urban Puhar</span> at
        <a rel="foaf:homepage" href="http://homepage.com/">homepage.com</a>.
</p>
```

Tabela 4: Primer vpenjanja RDFa v XHTML

Atributi HTML, ki jih uporablja standard RDFa za vpenjanje, so:

- *about* in *src* – URI ali CURIE, ki določajo vir metapodatkov,
- *rel* in *rev* – določajo povezavo z drugimi viri,
- *href* in *resource* – določajo povezavo s sorodnimi viri,
- *property* – določa lastnost vsebine elementa

2.3.3 Microdata (HTML5)

V zadnjem času se je pojavil še tretji način za semantično označevanje internetnih vsebin imenovan Microdata. Microdata je del specifikacij za najnovejši HTML standard, HTML5. Po svoji strukturi je zelo podoben Microformats, vendar s to razliko, da lahko dodajamo svoje, »po meri« narejene slovarje. Poleg tega uporablja nove atribute HTML5 (Microformats uporablja atribut *class*) namenjene semantičnemu označevanju. To sta *item* in pa *itemprop*. Njegova struktura ustreza DOM API specifikaciji, za dostopanje do podatkov urejenih v drevesno strukturo. Tako strukturo razumejo brskalniki in tudi JavaScript.

Primer zapisa HTML5 z Microdata semantiko:

```
<p item>
        <a itemprop="about" href="#me"></a>
         I'm <span itemprop="http://xmlns.com/foaf/0.1/name">
         Urban Puhar</span> at
        <a itemprop="http://xmlns.com/foaf/0.1/homepage"
        href=" http://homepage.com/"> homepage.com </a>.
</p>
```

2.3.4 Primerjava

V prejšnjih primerih smo videli tri različne pristope k semantičnem označevanju. Nedvomno so vsi trije standardi pripomogli k boljši semantiki spleta, čeprav se med seboj dokaj razlikujejo. Težko je označiti enega, kot najboljšega in najprimernejšega. Vsak ima seveda svoje prednosti in slabosti.

Če gledamo iz vidika končnega uporabnika, ki je zadolžen za oblikovanje vsebine, je definitivno najbolj primeren Microformats, najmanj pa RDFa. Slovarji so zbrani na enem mestu (na njihovi uradni strani) in so preprosti za uporabo. Pri RDFa je potrebno že poznati XML in *name space* (imenski prostor). Lahko se problem premosti z naprednejšimi sistemi za upravljanje vsebine (CMS).

Naslednja razlika je v načinu vpenjanja v dokument HTML. Tu prednjačita Microformats in Microdata. RDFa ima to omejitev, da se sklada samo z dokumentom, ki ustreza XHML standardu. Gre za veliko omejitev, ker je večina spletnih strani napisanih v jeziku HTML.

Nekateri formati so torej lažji za implementacijo kot drugi. Po eni strani je to prednost, vendar po drugi, zaradi enostavnosti, trpi razširljivost in povezljivost med podatki. Razširljivost v smislu oblikovanja slovarjev, ki so potrebni za označevanje. Microformats je s tega vidika najmanj razširljiv, saj je oblikovanje, definiranje in odločanje o slovarjih prepuščeno eni sami skupini. Povezljivost pa kot povezava med podatki na različnih straneh oziroma med podatkovnimi izvori. Ta omogoča enostavno sprehajanje računalnikov med podatki. S tega

vidika prednjači RDFa, ki vse svoje atribute označuje z *name space* in s tem pripenja podatek o naslovu vira podatka.

Prednost RDFa formata je obenem ta, da so podatki ločeni od predstavitve. Microformats recimo uporablja HTML atribut *class*, ki pa je že uporabljen s strani CSS, kar pripomore k večji kompleksnosti zapisa. Poleg tega zahteva, da imajo vsi elementi določenega formata enega skupnega prednika. Kar je v določenih primerih nemogoče izvesti.

RDFa je globoko zakoreninjen v skupnosti Semantičnega Spleta, zato se sklada z vsemi načeli semantičnega spleta in je po mojem mnenju tudi najbolj primeren za uresničevanje vizije o semantičnem spletu.

2.4 Ontologije

Na področju računalništva in informatike pojem ontologija pomeni predstavitev znanja znotraj določene domene kot množico konceptov in razmerij med njimi. Z njo definiramo in določimo entitete, ki pripadajo določeni domeni. Domeno tudi opisuje.

V teoriji se uporablja definicija »formalna, eksplicitna specifikacija skupne konceptualizacije«. Ontologija določa skupni slovar, ki se uporablja za opis domene. To so vrste objektov in/ali konceptov, ki obstajajo in njihove lastnosti in razmerja.

Ontologije so strukturna ogrodja za organiziranje informacij, ki se uporabljajo v umetni inteligenci, semantičnem spletu, sistemskem inženiringu, programskem inženiringu, biomedicinski informatiki, bibliotekárstvu in informacijski arhitekturi kot oblika predstavitve znanja o svetu ali delu njega.

V kontekstu računalništva in informatike ontologija določa množico reprezentativnih primitivov, s katerimi se modelira domeno znanja. Reprezentativni primitivi so tipično razredi, atributi in povezave. Definicije reprezentativnih primitivov vsebujejo informacijo o njihovem pomenu in omejitvah z logično konsistentno uporabo. V kontekstu podatkovnih sistemov, se lahko ontologijo razume, kot nivo abstrakcije podatkovnih modelov, analogno s hierarhičnimi in relacijskimi modeli, ampak z namenom modeliranja znanja o posameznih entitetah, njihovih atributih in njihovih razmerjih do ostalih entitet. Ontologije so tipično določene v jezikih, ki omogočajo abstrakcijo od podatkovnih struktur in implementacijskih strategij. V praksi so jeziki ontologij bližje v izrazni moči prvo nivojski logiki (*first-order logic*) kot pa jeziki za modeliranje podatkovnih baz. Ontologije so zato na semantičnem nivoju, medtem ko so podatkovne sheme podatkov na logični oz. fizični ravni. Zaradi njihove neodvisnosti od nižjih nivojev podatkovnih modelov, ontologije omogočajo integracijo heterogenih podatkovnih baz, interoperabilnost (medsebojna obratovalnost) med različnimi sistemi in specificirajo vmesnike do neodvisnih, na znanju temelječih storitev. Na skladu tehnologij[1] standardov semantičnega spleta so ontologije imenovane eksplicitna plast. Trenutno obstaja več standardnih jezikov in več različnih komercialnih in odprtokodnih orodij za grajenje in delo z ontologijami [6].

2.4.1 Zgodovina

Pojem ontologija izhaja iz področja filozofije, ki se ukvarja z raziskovanjem obstoja. V filozofiji se govori o ontologiji kot teorija narave obstoja. Na področju računalništva in

[1] technology stack

informatike je ontologija tehnični termin, ki označuje artefakt, ki omogoča modeliranje znanja o neki domeni, namišljeni ali realni.

Termin so posvojili tudi na področju umetne inteligence, kjer so prepoznali uporabnost ontologije, ki temelji na matematični logiki za ustvarjanje ontologij, ki bi lahko služile kot računalniški model, ki omogoča določene načine avtomatskega sklepanja. V 80' se je v umetni inteligenci termin uporabljal že v dveh pomenih. Prvič, kot teorija modeliranega sveta in drugič, kot komponenta sistemov znanja. Nekateri raziskovalci, ki so se zgledovali po filozofskih ontologijah, so videli računalniške ontologije kot neke vrste aplicirano filozofijo. V zgodnjih 90' so prizadevanja za vzpostavitev interoperabilnih standardov izvajanja obrodila sklad tehnologij, ki definira ontologijo kot enega izmed standardnih slojev sistemov znanja. Pogosto citirana internetna stran in publikacija [7], ki je povezana s temi prizadevanji je tako zaslužna za preudarno definicijo ontologije kot tehnični termin v računalništvu. Publikacija definira ontologijo kot »eksplicitna specifikacija konceptualizacije«, kar v bistvu pomeni »objekti, koncepti in ostale entitete, ki domnevno obstajajo na nekem področju zanimanja in povezave, ki veljajo med njimi«. Medtem, ko sta pojma specifikacija in konceptualizacija povzročila veliko razprav, bistvene točke definicije ontologije ostajajo:

- ontologija definira (specificira) koncepte, povezave in ostale relacije, ki so pomembne za oblikovanje domene,
- specifikacija uporablja obliko definicij reprezentativnega slovarja (razredi, relacije itd), ki ponuja pomene slovarja in za formalne omejitve z njihovo koherentno uporabo.

Očitek definiciji je, da je preširoka, ker omogoča vrsto specifikacij, od enostavnih slovarjev do teorij logike, ki ležijo v predikatnih računih [18]. Sicer pa to velja tudi za podatkovne modele različnih kompleksnosti, npr: relacijska baza, ki je sestavljena iz tabele in stolpca, je še vedno instanca relacijskega podatkovnega modela. Pragmatično gledano je ontologija orodje in produkt inženiringa in je tako definirana z njeno uporabo. Pomembna s te perspektive je uporaba ontologij za zagotavljanje reprezentativnih mehanizmov, s katerimi bi umestili domenske modele v baze znanja, izvrševali poizvedbe prek storitev baz znanj in prikazovali rezultate takih storitev. Npr. programski vmesnik iskalne storitve bi lahko ponujal tekstovni slovar terminov, s katerimi bi tvorili povpraševanja, kar bi delovalo kot ontologija. Po drugi strani današnji W3C Semantic Web standardi predlagajo specifičen formalizem za zapisovanje ontologij (OWL) v različnih variantah, ki se razlikuje v izrazni moči [13]. Ontologija je specifikacija abstraktnega podatkovnega modela (domenska konceptualizacija), ki je neodvisna od določene oblike. [12]

2.4.2 OWL - Jezik za zapis ontologij

OWL (Ontology Web Language) je zgrajen na RDF podlagi, ki je zapisana v XML obliki. RDF in OWL ponujata možnost kreiranja razredov, lastnosti in instanc. Razredi (ali koncepti) so splošne kategorije, ki jih lahko hierarhično uredimo. Vsak razred vsebuje množico instanc, ki sodijo skupaj, ker si delijo določene lastnosti. Instance so specifični objekti, katerim razred določa tip. Lastnosti so atributi instanc, opredeljene na splošno in določajo bodisi vrednost podatkov, bodisi povezavo do drugih instanc. Da bi si predstavljali, kako se te tri elemente uporablja, poglejmo sledeč primer.

Primer:

Določili bomo razred, ki se bo imenoval *Oseba* in nekaj lastnosti, kot so *ime*, *»rojstni dan«* in *prijatelj* (ki povezuje dve osebi, ki sta prijatelja). RDF sintaksa zgleda tako:

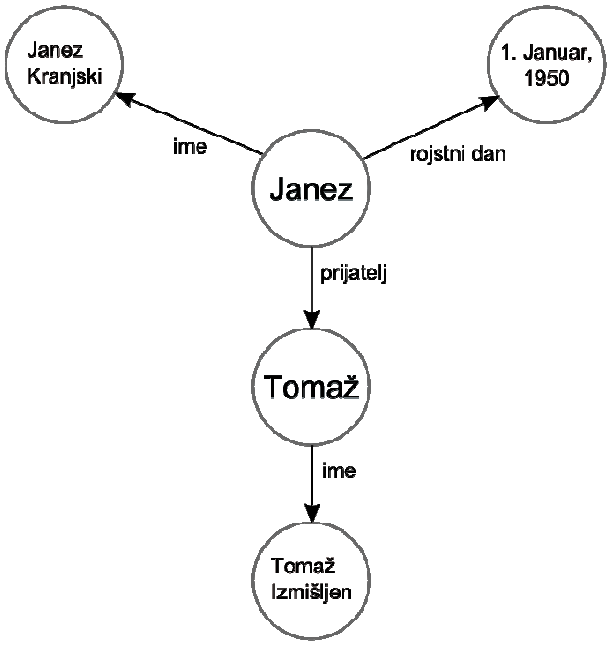
```
<Razred ID="Oseba"/>
<Lastnost ID="ime"/>
<Lastnost ID="rojstni dan"/>
<Lastnost ID="prijatelj"/>
```

(Zaradi lažje predstave v tem primeru nisem uporabil imenskih prostorov)

Ko je razred definiran, ga lahko uporabimo za opis instance našega razreda *Oseba*. V tem primeru bomo uporabili podatke o osebi *Janez Kranjski*, rojen *1. januarja leta 1950*, ki je prijatelj *Tomaža Izmišljenega*.

```
<Oseba ID="Janez">
        <name>Janez Kranjski</name>
        <rojstni datum>1 Januar, 1950</ rojstni datum >
        <prijatelj izvor="#Tomaž"/>
</Oseba>
<Oseba ID="Tomaž">
        <ime>Tomaž Izmišljen</ime>
</Oseba>
```

Tukaj je nekaj zanimivih značilnosti, ki jih lahko izpostavimo. Prvič, ko opišemo Janezovega prijatelja Tomaža, uporabimo povezavo do vira *#Tomaž*, namesto da podamo ime prijatelja *Tomaž Izmišljeni*, kot smo npr. podali Janezov rojstni datum. To omogoča, da imamo instanco definirano nekje drugje. Zmožnost povezovanja objektov je lastnost RDF grafa. Grafično predstavitev RDF zapisa, se lahko vidi na spodnji sliki (Slika 4).



Slika 4: RDF usmerjen graf, ki povezuje dva prijatelja

Druga značilnost je predpostavka, da lastnosti, definirane, pripadajo samo razredu *Oseba* in ne ostalim razredom, ki jih bomo mogoče definirali. Nočemo dodati lastnost *prijatelj* razredu *Stol*, ki bi ga eventualno definirali kasneje. V tej fazi so to samo domneve. Da bi zagotovili takšno omejitev, se lahko uporabimo lastnosti jezika OWL, kot so domena in doseg atributov, ki omejujejo povezave:

- domena nam omogoča, da omejimo razrede, na katerih lahko uporabimo lastnost.
- doseg določa razred, ki mu instanca mora pripadati, da se jo lahko uporabi kot vrednost v določeni povezavi.

Primer razširimo z dosegom in domeno:

```
<Razred ID="Oseba"/>
        <Lastnost ID="name">
                <domena izvor="#Oseba"/>
        </Lastnost>
        <Lastnost ID="rojstni datum">
                <domena izvor="#Oseba"/>
        </Lastnost>
        <Lastnost ID="prijatelj">
                <domena izvor="#Oseba"/>
        <doseg izvor="#Oseba"/>
</Lastnost>
```

OWL precej razširja funkcionalnost RDF in RDFS, medtem ko njegovo jedro, ohranjanje kompatibilnosti z osnovno spletno arhitekturo, odprto, ne lastniško in porazdeljeno po številnih sistemih, omogoča uporabnikom, da si izmenjujejo podatke (ontologije) in dopušča razširljivost. V družino OWL spadajo trije jeziki, ki se razlikujejo glede na izrazno moč, ki jo nudijo.

To so [16][4]:

- *OWL Lite* je bila prvotno namenjena tistim uporabnikom, ki so potrebovali predvsem klasifikacijo hierarhije in preproste omejitve. Npr.: čeprav podpira kardinalnost povezave, dovoljuje samo kardinalnost vrednosti 0 ali 1. S kompleksnimi kombinacijami gradnikov jezika OWL Lite je možno sestaviti tudi večino konstruktov jezika OWL DL. Razvoj orodij za OWL Lite pa je ravno tako kompleksen kot razvoj OWL DL orodij, zato OWL Lite ni tako razširjen.
- *OWL DL* vključuje vse konstrukte jezika OWL, vendar so nekateri lahko uporabljeni samo pod določenimi pogoji (na primer razred je lahko podrazred večjih razredov, ne more pa biti nek razred primerek drugega razreda). OWL je bil načrtovan za podporo področju obstoječe poslovne opisne logike in ima ustrezne računske lastnosti.
- *OWL Full* temelji na drugačni semantiki kot jezika OWL Lite in OWL DL in je bil zasnovan za ohranitev skladnosti z RDF shemo. Namenjen je uporabnikom, ki želijo največjo mogočo izraznost in sintaktično svobodo jezika RDF, vendar brez zagotavljanja celovitosti izračunavanja. V OWL Full se razred obravnava kot zbirko primerkov, obenem pa kot primerek sam po sebi. To v jeziku OWL DL ni dovoljeno. OWL Full omogoča dodajanje pomena ontologijam, že definiranim v jeziku RDF ali obeh ostalih OWL jezikih. Zaradi velike izraznosti je malo verjetno, da bi kakšno orodje uspelo zagotoviti podporo celotnemu mehanizmu sklepanja jezika OWL.

Obstaja več zmogljivosti, ki jih ima OWL glede na RDF in RDFS. Prva je zmožnost ustvarjanja več lokalnih omejitev dosega. V RDF in RDFS je možno dodati en doseg za eno lastnost. V večini primerov bi želeli imeti doseg, ki se spreminja glede na to, v kateri domeni je razred.

Npr. predstavljajmo si, da bi dodali lastnost *je* (je, kot jesti) na razred *Oseba*. Hoteli bi, da je doseg lastnosti *je* omejen na instance razreda *Hrana*. To načeloma velja za večino oseb. Vendar predpostavimo, da definiramo še dodaten razred *Vegetarijanec*. Vemo, da vegetarijanci jedo hrano, vendar je ta množica bolj ozka od splošne množice *Hrana*. Samo z uporabo RDF in RDFS jezika ni mogoče izraziti, da vegetarijanci ne jedo mesa. Z OWL jezikom pa bi to lahko opisali takole: Doseg in domeno lastnosti *je* pustimo nespremenjeno in v razred *Vegetarijanec* (podrazred razreda *Oseba*) dodamo omejitev, da so njegovi člani samo tiste instance, ki imajo lastnost *je* vezano na razred *Vegetarijanske jedi*. Takšne omejitve dosega lastnosti omogočajo uporabnikom, da definirajo bolj izrazne ontologije in računalnikom lažje logično sklepanje. OWL vsebuje poleg opisanih, tudi funkcionalnosti kot so: unija, presek, komplement in disjunktnost. Če nadaljujemo s prejšnjim primerom, definiramo še podmnožici množice *Oseba*, *Mrtva oseba* in *Živa oseba*. Glede na prejšnje definicije in omejitve, bi lahko ustvarili objekt, ki bi bil instanca obeh razredov, kar ni realno. Z RDF jezikom bi to ne bilo mogoče izraziti. Z uporabo OWL se lahko trdi, da sta dva razreda disjunktne množice. Tako se doseže, da objekt ne more biti instanca dveh razredov, ki sta disjunktna.

Nah omenim še kardinalnost povezav, ki se nanaša na omejitev povezave. Objekt mora vsebovati minimalno, maksimalno ali točno določeno število istih povezav, če želi biti instanca razreda. Kot primer lahko definiramo razred *Moštvo*, ki mora imeti natanko 11 članov. Če hočemo moštvo *Olimpija* (objekt), ki zadostuje pogojem množice *Moštvo*, mora ta imeti natanko 11 povezav *je v moštvu* na instance razreda *Igralec*.

2.5 Razvojna orodja

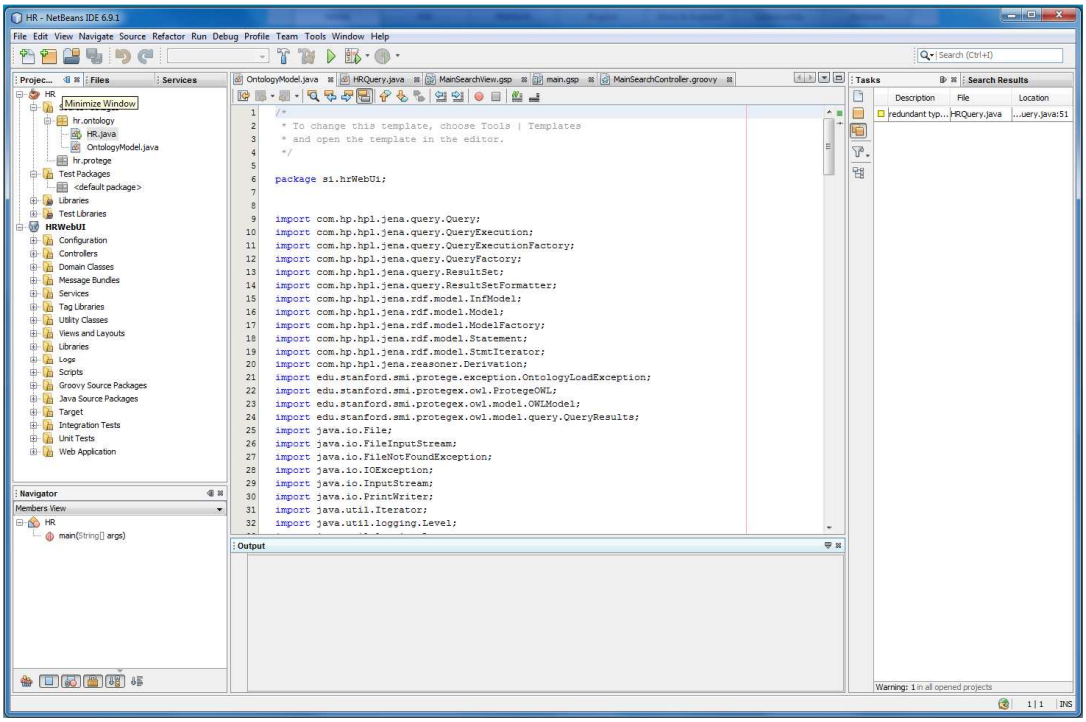
2.5.1 NetBeans

NetBeans je integrirano razvojno okolje. Omogoča razvoj s programskimi jeziki, kot so: Java, JavaScript, PHP, Python, Ruby, Groovy, C, C++, Scala, Clojuree. V celoti je napisan v programskem jeziku Java in ga tako lahko poganjamo na sistemih, kjer imamo nameščeno Java Virtual Machine (Window, Linux, Mac Os in Solaris). Razvoj poteka pod okriljem Sun Microsystems, ki tudi skrbi za programski jezik Java. Njegova uporaba je brezplačna. Najdemo ga na strani (9). Na voljo je več različnih paketov, ki se med seboj razlikujejo predvsem po programskem okolju, v katerem želimo razvijati. V času pisanja diplomske naloge je trenutna verzija 6.9.1.

NetBeans IDE Download Bundles							
Supported technologies *	Java SE	JavaFX	Java	Ruby	C/C++	PHP	All
NetBeans Platform SDK	●	●	●				●
Java SE	●	●	●				●
JavaFX		●					●
Java Web and EE			●				●
Java ME			●				●
Java Card™ 3 Connected			●				●
Ruby				●			●
C/C++					●		●
Groovy			●				●
PHP						●	●
Bundled servers							
GlassFish Server Open Source Edition 3.0.1			●	●			●
Apache Tomcat 6.0.26			●				●

Slika 5: NetBeans različice

Za potrebo diplomske naloge sem si namestil celotno verzijo »All«, ki vsebuje vse pakete, ker sem razvijal rešitev v programskem okolju Java in Groovy.



Slika 6: Uporabniški vmesnik programskega orodja NetBeans IDE

2.5.2 Protégé IDE

Protégé programsko orodje je zastonj odprtokodna platforma, ki nudi programski nabor za razvoj domenskih modelov in aplikacij, ki temeljijo na znanju[2], z ontologijami. V svojem jedru Protégé implementira bogato množico, in akcij, ki podpirajo kreiranje, vizualiziranje in manipulacijo ontologij v različnih predstavitvenih formatih. Protégé lahko prilagodimo, kot domeni prijazno podporo za kreiranje modelov znanja in vnos podatkov [9].

[2] knowledge-based applications

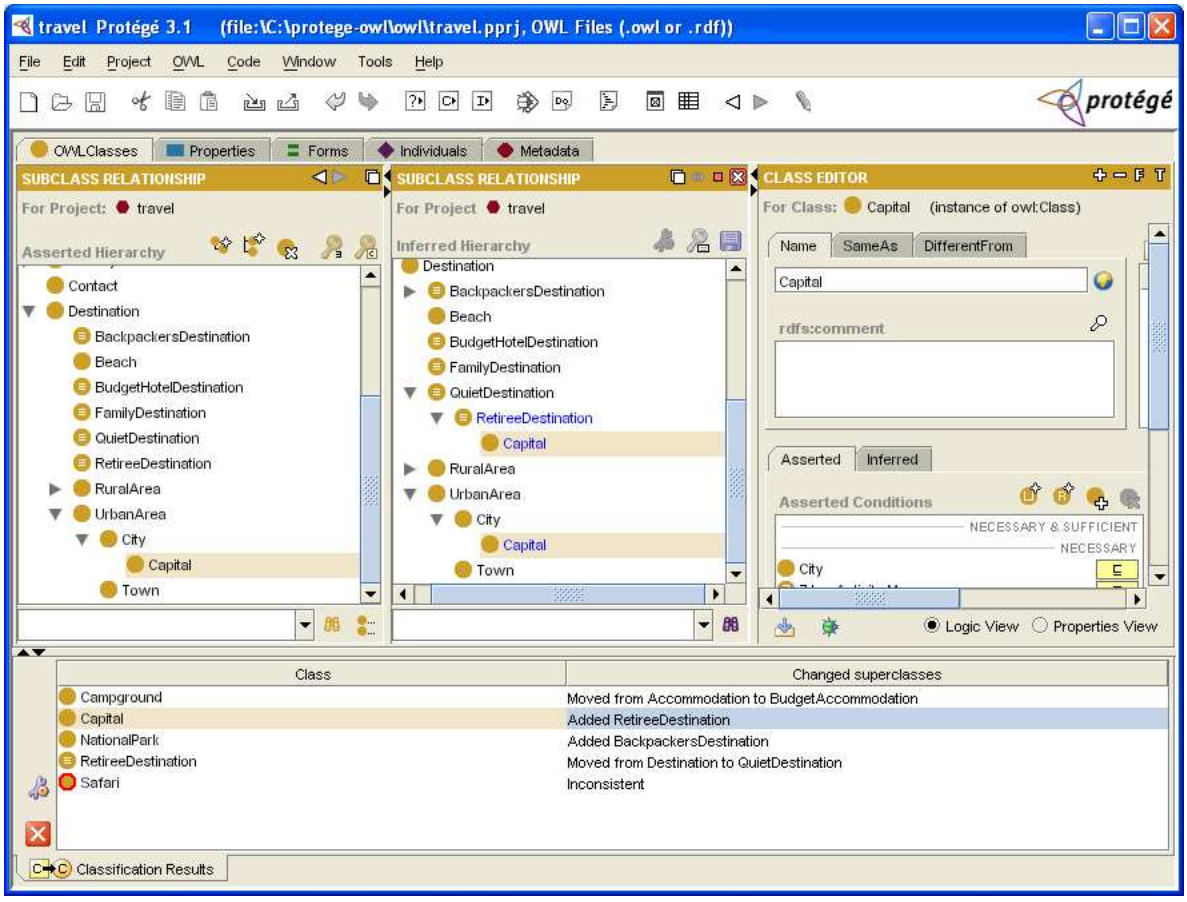
Protégé platforma podpira dva pristopa k modeliranju ontologij:

- ***Protégé-Frames urejevalnik***

Protégé-Frames ponuja samostojen uporabniški vmesnik in strežnik baze znanja za podporo uporabnikom pri gradnji, shranjevanju domenskih ontologij in pri urejanju obrazcev za vnos podatkov. Protégé-Frames implementira model znanja, ki je kompatibilen z »Open Knowledge Base Connectivity protocol (OKBC)«. V tem modelu je ontologija sestavljena iz množice razredov, organiziranih v hierarhijo za predstavitev domenskih konceptov, množico lastnosti in povezav za posamezen razred in množico objektov, instanc razredov, ki vsebujejo lastnosti in povezave.

Lastnosti Protégé-Frames urejevalnika:

- široka množica prilagodljivih elementov uporabniškega vmesnika, ki omogočajo uporabnikom vnos podatkov v domeni prijazni obliki,
- arhitektura vtičev, ki omogoča razširitve z elementi, kot so: grafične komponente (grafi in tabele), različni mediji (zvok, slike in video), različni formati shranjevanja (RDF; XML, HTML) in dodatna orodja za upravljanje (upravljanje z ontologijami, vizualiziranje ontologij, itd),
- osnovan je na Java programskem vmesniku, ki omogoča vtičem in ostalim aplikacijam dostop, uporabo in prikaz ontologij, ki so bile narejene z Protégé-Frames urejevalnikom.



Slika 7: Protégé-Frames urejevalnik

- ***Protégé-OWL vmesnik***

Protégé-OWL urejevalnik je razširitev Protégé-ja, ki podpira delo z jezikom OWL (Web Ontology Language). OWL je najbolj razširjen standard za jezike ontologij, odobren s strani W3Cja za promocijo vizije o semantičnem spletu.

The Protégé-OWL urejevalnik omogoča:

- nalaganje in shranjevanje OWL in RDF ontologij,
- urejanje in vizualiziranje razredov, lastnosti in SWRL pravil,
- definiranje karakteristik logičnih razredov kot OWL izrazov,
- izvajanje stroja za sklepanje[3], kot so klasifikatorji opisne logike,
- urejanje OWL instanc za internetno semantično označevanje.

Fleksibilna Protégé-OWL arhitektura omogoča preprosto konfiguracijo in razširitev tega orodja. Protégé-OWL je tesno povezan z Jena in vsebuje odprto kodni Java programski vmesnik za razvoj komponent uporabniškega vmesnika ali poljubnih semantično spletnih storitev.

2.6 Programska orodja

2.6.1 Java

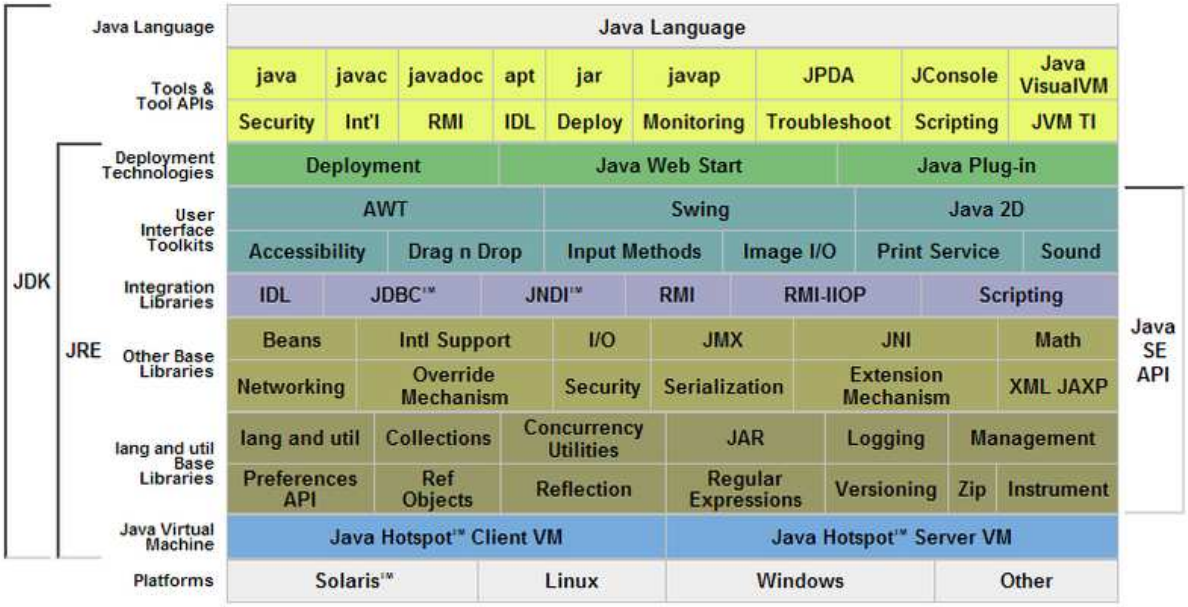
Glavna komponenta za delo z ontologijami Protégé API ponuja svoj programski vmesnik napisan v jeziku Java. Zato je bil tudi večji del mojega programiranja opravljenega v tem jeziku. Poleg tega pa je tudi osnova programskega okolja Groovy, kjer se koda prevede v *Java bytecode*

Java je programski jezik, ki ga je prvotno začel razvijati James Gosling, takrat zaposlen v podjetju Sun Microsystems (pred kratkim ga je kupilo podjetje Oracle) in bil nato izdan leta 1995 kot glavna komponenta Sun Microsystem Java platforme. Sintaksa v večini izhaja iz jezika C in C++, vendar ima preprostejšo obliko objektnega modela in ima manj klicev metod nižjih plasti. Aplikacije napisane v Javi se prevedejo v *bytecode*, ta se nato poganja na katerikoli *Java Virtual Machine*[4] *(JVM)*, ne glede na arhitekturo računalnika. To pomeni, da se programi lahko izvajajo na kateremkoli računalniku, s katerimkoli operacijskem sistemom, če le ima nameščen Java Navidezni Stroj (JVM). Poleg računalnikov se Java kot distribucija *Java ME* uporablja tudi na mobilnih napravah. Zaradi teh idej in vizij je njen glavni moto »*Napiši enkrat, poženi kjerkoli*«. Trenutno je Java ena izmed najbolj priljubljenih programskih jezikov, ki se pogosto uporablja vse od aplikacij za delovne postaje do spletnih aplikacij. [14]

[3] reasoner
[4] Java navidezni stroj

Sun Mycrosystem podpira štiri različne izdaje Jave glede na različno okolje aplikacij:

- java Card za pametne kartice,
- Mmicro Edition (Java ME) za naprave z omejenimi resursi (telefoni),
- standard Edition (Java SE) za delovne postaje (namizni računalniki),
- enterprise Edition (Java EE) za porazdeljene sisteme in internetne storitve.



Slika 8:Arhitektura Java platforme Standard Edition

2.6.2 Protégé API

Protégé-OWL API je odprto kodna knjižnica za OWL in RDF zapis ontologij. Programski vmesnik ponuja razrede in metode za nalaganje in shranjevanje OWL datotek, za povpraševanje in manipuliranje OWL podatkovnih modelov in za izvajanje sklepanja na podlagi opisne logike[5].

Programski vmesnik je načrtovan za uporabo v dveh kontekstih:

- za razvoj komponent, ki se izvajajo znotraj uporabniškega vmesnika Protégé-OWL urejevalnika,
- za razvoj samosvojih, neodvisnih aplikacij[6] (kot so Swing aplikacije, Servleti ali Eclipse vtičniki).

Protégé je fleksibilna in nastavljiva platforma za modelno usmerjen[7] razvoj aplikacij in komponent. Temelji na odprti arhitekturi, ki omogoča programerjem integracijo vtičev, ki se lahko pojavljajo kot svoji zavihki, posebne komponente za uporabniški vmesnik ali izvajajo katerokoli drugo nalogo na trenutnem modelu.

Protégé-OWL programski vmesnik ponuja številne enote za urejanje in pregledovanje OWL modelov in tako lahko služi kot privlačno izhodišče za hiter razvoj aplikacij. Osredotočen je

[5] Description Logic
[6] Stand-alone applications
[7] Model driven

na zbirko Java vmesnikov, ki omogočajo dostop OWL modelov in njihovih elementov, kot so: razredi, lastnosti in instance.

Najbolj pomemben vmesnik modela je *OWLModel*, ki ponuja dostop do vsebnika najvišjega nivoja resursov v ontologiji. *OWLModel* se lahko uporabi za ustvarjanje, poizvedovanje in brisanje resursov različnih tipov in uporabo objektov, ki jih vrne *OWLModel*, za izvajanje specifičnih operacij. Npr., spodnja koda (Tabela 5) ustvari nov razred imenovan *Svet*, tipa *OWLNamedClass* (ki ustreza *owl:Class* v OWL jeziku) in nato izpišemo naslov (URI) razreda:

```
OWLModel owlModel = ProtegeOWL.createJenaOWLModel();
OWLNamedClass worldClass = owlModel.createOWLNamedClass("Svet");
System.out.println("Class URI: " + worldClass.getURI());
```

Tabela 5: Kreiranje novega razreda *Svet*

Razred *ProtégéOWL* ponuja nekaj priročnih statičnih metod za kreiranj OWL modelov (*OWLModel*). Npr. , že obstoječo ontologijo lahko naložimo tudi iz spletnegavira:

```
String uri = "http://www.co-ode.org/ontologies/pizza/2007/02/12/pizza.owl";
OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
```

Tabela 6: Nalaganje ontologije iz spletnega vira

OWL in RDF resursi so globalno identificirani prek njihovih URI naslovov (imenski prostor), kot na primer:

http://www.owl-ontologies.com/potovanje.owl#Cilj.

Tabela 7: Imenski prostor, kot identifikacija vira razreda Cilj

Ker pa so URI dolgi in pogosto nerodni za uporabo, se v ProtégéOWL resursi ontologij naslavljajo in identificirajo prek njihovih imen. Ime je skrajšana oblika, sestavljena iz lokalnega imena in predpone, ki je opcijska in neobvezna. Predpone so tipično opredeljene v ontologiji za skrajševanje imen uvoženih ontologij. Na primer, namesto zapisa *http://www.w3.org/2002/07/owl#Class*, lahko krajše zapišemo *owl:Class*, ker je *owl* predpona za imenski prostor *http://www.w3.org/2002/07/owl#*. Podobno, kot pri prejšnjem primeru (Tabela 6), lahko ontologija, ki uvozi ontologijo *potovanje.owl*, dostopa do uvoženih razredov in lastnosti prek predpone *potovanje*, na primer: *potovanje:Destinacija*. Če smo v notranjosti privzetega imenskega prostora, je predpona prazna oziroma jo ne uporabimo.

Razvijalci lahko upravljajo z predponami imenskih prostorov prek objekta *NamespaceManager*. Za dostop do *NamespaceManager* trenutnega owl modela (*OWLModel*), lahko uporabimo *OWLModel.getNamespaceManager*. Ob predpostavki, da imamo naloženo ontologijo potovanje, kot privzeti imenski prostor. Takrat se lahko dostopa do resursov v OWLModelu z uporabo naslednjih klicev: [10]

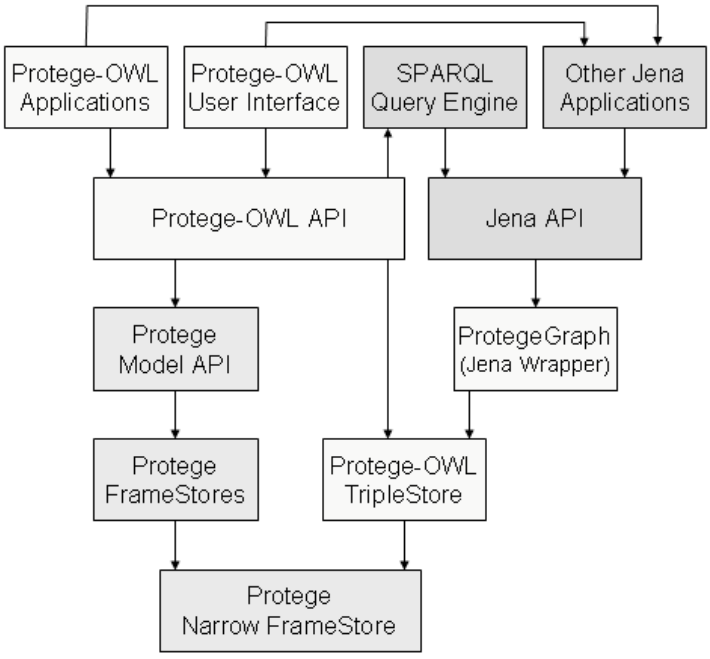
```
OWLNamedClass destinationClass = owlModel.getOWLNamedClass("Cilj");
OWLObjectProperty hasContactProperty = owlModel.getOWLObjectProperty("imaKontakt");
OWLDatatypeProperty hasZipCodeProperty =
        owlModel.getOWLDatatypeProperty("imaPoštnoŠtevilko");
OWLIndividual sydney = owlModel.getOWLIndividual("Sydney");
```

Tabela 8: Dostop do razreda, lastnosti in instance

2.6.3 Jena

Jena je najbolj pogosto uporabljen Java programski vmesnik (API) za RDF in OWL. Ponuja funkcionalnosti za predstavitev, razčlenjevanje, povpraševanje in vizualizacijo ontologij. Protégé-OWL je zelo tesno povezan z Jena programskim vmesnikom. Npr. Jena ARP razčlenjevalnik kode se uporablja v Protégé-OWL razčlenjevalniku kode. Prav tako Protégé-OWL uporablja storitve Jena, kot so validacija razredov in upravljanje s podatkovnimi tipi. Mogoče je tudi pretvoriti Protégé OWLModel v Jena OntModel, ker se tako dobi statični posnetek modela med časom izvajanja programa. Vendar se model po vsaki spremembi vsakič znova sestavi.

Od avgusta 2005 pa je Protégé-OWL še bolj tesno povezan z Jena. Nova povezava omogoča programerjem uporabo Jena funkcionalnosti med izvajanjem programa brez počasne procedure za ponovno sestavljanje modela. Nova arhitektura je prikazana na spodnji sliki:



Slika 9: Jena API arhitektura

Ključ integracije je dejstvo, da oba sistema (Protégé-OWL in Jena) uporabljata predstavitev modela na najnižjem nivoju. Protégé vsebuje svoje ogrodje za mehnizem shranjevanja, ki je ovito v Protégé-OWL s *TripleStore* razredi. V Jena se ustrezni vmesniki imenjujejo *Graph* in *Model*. Protégé *TrippleStore* je ovit v Jena *Graph* tako, da vsak bralni dostop iz *Jena API* pravzaprav deluje na Protégé trojicah (*TripleStore*). Za spremembo trojic je tako potrebno uporabiti konvencionalen Protégé programski vmesnik. Mehanizem dovoljuje uporabo Jena metod za povpraševanje, medtem, ko se ontologije lahko spreminja v Protégé urejevalniku.

OWLModel programski vmesnik ima metodo *getJenaModel()* za dostop do Jena predstavitve Protégé modela med izvajanjem. To se lahko npr. uporablja za programiranje vtičev.

S serviranjem povezave do modela, generiranega s strani Protégé je možno tudi ostale Jena storitev »oviti« v Protégé vtičnike. [15]

2.6.4 Groovy On Grails

Grails je odprto-kodno internetno aplikacijsko ogrodje, ki uporablja Groovy programski jezik, ki sloni na Java platformi. Njegov namen je služiti kot visoko-produktivno ogrodje, ki se drži kodiranje po dogovoru[8] paradigme, to pomeni samostojno razvijalsko okolje in skrivanje večjega dela konfiguracijskih detajlov pred razvijalci. Ob nastanku »Groovy on grails« je že obstajalo ogrodje imenovano »Ruby on Rails«. Ker se je njegov avtor pritožil, se je »Groovy on Rails« leta 2006 preimenoval v »Groovy on Grails«. Verzija 1.0 je bila izdana 18. februarja 2008. Nato so istega leta, novembra 2008 podjetje G2One (Groovy on Grails) kupili VMware. [8]

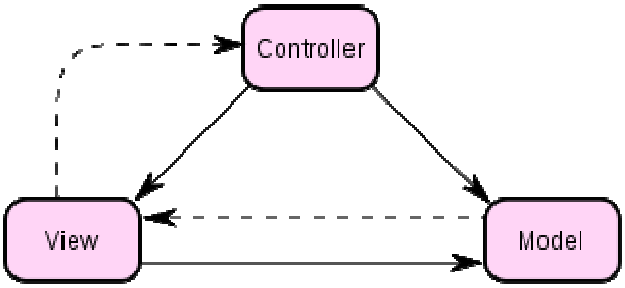
Grails se ponaša s tremi lastnostmi, s katerimi skuša povečati produktivnost v primerjavi s tradicionalnim Java internetnimi ogrodji:

- brez XML konfiguracije: Gradnja internetnih aplikacij v Javi vključuje konfiguracijo ogrodja na začetku in tudi med razvojem. Konfiguracija se po navadi shrani kot zunanji XML dokument, kar poenostavi konfiguracijo in ne vključuje konfiguracijskih procesov v samo kodo aplikacije. V začetku je bil XML dobrodošel, ker je ponujal boljšo konsistenco pri konfiguriranju aplikacij. Sčasoma, se je izkazalo, da je s pomočjo takega načina konfiguracije zelo težko vzpostaviti in vzdrževati okolje. Vpliva na produktivnost, ker se programerji preveč časa posvečajo razumevanju in konfiguriranju ogrodja med razvojem aplikacije. Grails se je tako izognil XML konfiguracijskim datotekam. Uporablja množico pravil ali dogovorov, ki se jih mora programer držati. Npr., ime razreda, ki se konča z besedo *Controller* šteje, kot krmilnik (MVC arhitektura),
- takojšnja uporaba razvojnega okolja: Java: pred začetkom uporabe je potrebno zbrati enote, ki se jih potrebuje pri razvoju, kar je zamudno. Grails vsebuje celotno razvojno orodje, ki med drugim zajema internetni strežnik za takojšnji pričetek z delom.
- funkcionalnosti na voljo prek *mixin:* Ena izmed posebnosti Grails je, da ponuja dinamične metode imenovane *mixim*. *Mixin* je metoda, ki je dodana razredu brez uporabe dedovanja ali razširjanja razredov. Te metode Grails dodaja dinamično glede na tip razreda. Npr. vsi domenski razredi vsebujejo metode za operacije ohranjevanja podatkov, kot so: shranjevanje, brisanje in iskanje.

[8] Coding by convention

2.6.5 MVC

Ogrodje Grails je bilo načrtovano v skladu z *Model–view–controller* (na kratko MVC) arhitekturo. MVC je programska arhitektura, ki je trenutno najbolj uporabljena in razširjena arhitektura v programskem inženiringu. Ločuje domensko logiko (*domain*, to je logika sistema) od uporabniškega vmesnika (*view* vnos in prikaz) z namenom lažjega vzdrževanja in testiranja vsakega dela posebej. Povezavo med uporabniškimi akcijami in programsko logiko pa krmili krmilnik (*controller*).



Slika 10: MVC arhitektura

Krmilnik (controller)

Grails uporablja krmilnike za implementacijo obnašanja internetnih strani. Spodaj (Tabela 9) je podan primer enostavnega krmilnika:

```
class BookController {
        def list = {
                [books: Book.findAll() ]
        }
}
```

Tabela 9: Priner krmilnika

Zgornji krmilnik definira akcijo seznam, ki vrača model, ki vsebuje vse knjige v podatkovni bazi.

Primer ukaza za kreiranje krmilnika:

```
grails create-controller Book
```

Tabela 10: Ukaz za kreiranje krmilnika

Zgornji ukaz (Tabela 10) ustvari razred v *grails-app/controller* mapi Grails projekta. Akcija seznam je dostopna na naslovu http://localhost:8080/book/list v lokalnem spletnem brskalniku.

Prikazi (view)

Grails podpira JSP (Java Servlet Pages) in GSP (Grails Servlet Pages). Spodnji primer prikazuje prikaz (*view*) zapisan v GSP, ki vsebuje seznam knjig v modelu.

```
<html>
<head>
<title>Our books</title>
</head>
<body>
<ul>
<g:each in="${books}">
<li>${it.title} (${it.author.name})</li>
</g:each>
</ul>
</body>
</html>
```

Tabela 11: Prikaz (view) za prikaz seznama knjig

Prikaz se nahaja v mapi *grails-app/views/book/list.gsp* Grails projekta. Ta lokacija pa je povezana/preslikana na prikaz *BookController* in akcijo seznam. Namestitev datoteke na to mesto je dovolj, da jo Grails prepozna.

Model

Domenski model v Grails je povezan s podatkovno bazo z GORM (Grails Object Relational Mapping). Domenski razredi so shranjeni v *grails-app/domain* mapi in so lahko ustvarjeni z ukazom :

```
grails create-domain-class Book
```

Tabela 12: Ukaz za kreiranje domenskega razreda *Book*

Ta ukaz zahteva ime razreda in nato ustvari ustrezno datoteko z domenskim razredom Book:

```
class Book {
        String title
        Person author
}
```

Tabela 13: Vsebina domenskega razreda

Za programerja je dovolj, da ustvari datoteko, za vse ostalo poskrbi Grails.

3 SISTEM ZA SEMANTIČNO INTEGRACIJO VIROV

3.1 Problem

Živimo v času, obdani z veliko količino informacij. Dostopne so nam vedno in povsod. Eden izmed glavnih nosilcev informacij je internet. Sestavljen je iz različnih strani, kot so: blogi, raznovrstni portali in zadnje čase kot vir socialna omrežja. Na nekatere se vračamo vsak dan, na druge po potrebi oz., ko iščemo določeno informacijo. Iskanje navadno poteka tako, da se v izbran iskalnik vpiše niz reprezentativnih besed in tako se sproži iskanje. S količino in dostopnostjo nastopi vprašanje verodostojnosti, kvalitete in ažurnosti informacije, ki nam jih iskalniki vrnejo. Pojavlja se torej problem, kako ob pravem trenutku najti pravi odgovor.

Količina informacij in spletnih storitev eksponentno narašča, kar je razlog, da se vsak dan težje dokopljemo do potrebnih informacij. Naučiti se moramo, kako računalnikom povedati, kaj sploh želimo. Zakaj računalniki takoj ne vedo, katera stran, fotografija ali objava na enem izmed socialnih omrežji nas zanima?

Odgovor je: ker je to nemogoče. Računalniki ne razumejo, nimajo dostopa do večine potrebnih virov, manjka jim semantično razumevanje in zdrava pamet, s katero bi povezovali informacije med seboj. Bistveno je, da računalniki pridobijo nove ravni razumevanja. Namesto statističnega preverjanja, koliko se iskalni izraz ujema z virom informacije (spletni iskalniki) mora biti informacija na obeh straneh predstavljena na dovolj formalen način, da jo lahko računalniki interpretirajo. Potrebne so baze znanja, na katere se lahko obrnemo.

V prihodnjih letih bomo nedvomno priča napredku v zmožnosti sistemov za dostop, procesiranje in uporabo informacij. Odražal se bo predvsem na treh področjih, med seboj povezanih v t.i. semantični splet. Gre za splet podatkov, splet storitev in splet ponudnikov identitete. [11]

V diplomski nalogi bi s praktičnim primerom predstavil razliko med idejo o semantičnem spletu in trenutnim stanjem. Na eni strani so viri, kot spletne strani, ki hranijo informacije, zajete v HTML zapisu. HTML je namenjen oblikovanju spletnih strani in ne kot jezik za semantično označevanje. Na drugi strani gre za idejo o semantičnem spletu, ki priporoča ločitev golih podatkov od njihove predstavitve. Če to posplošimo, lahko rečemo, da ljudje tako dostopajo do informaciji prek spletnih strani. Računalniki, ki ljudem servirajo informacije, pa do golih, semantično urejenih podatkov.

Na grobo je moja rešitev sestavljena iz dveh delov. Prvi del vključuje analizo ureditve podatkovna spletnih straneh in njihovo ločevanje od predstavitvenega dela. Služi tudi, kot ponazoritev problema računalniškega razumevanja vsebine. V drugi del pa spada izgradnja ontologij za posamezen vir, njihovo povezovanje in vnos podatkov pridobljenih iz spletnih strani.

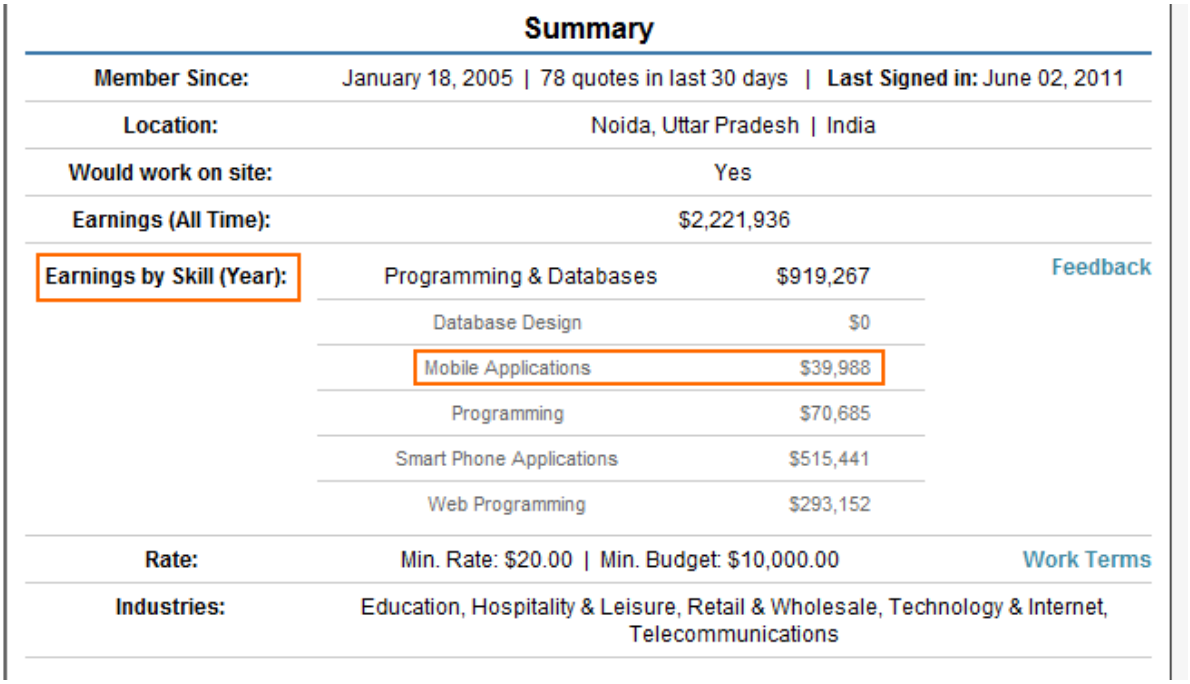
3.2 Uvod v rešitev

Končna rešitev bo implementirala idejo o semantičnem spletu. Prvi del je sestavljen iz ontologij, ki vsebujejo podatke, pridobljene iz realnih virov, trenutnih podatkov na spletnih straneh. Drugi del je namenjen izvajanju operacij nad informacijami, shranjenimi v ontologijah. Končni uporabnik navzven sistem vidi kot spletno stran, kjer lahko poganja različne poizvedbe nad množico virov.

3.2.1 Trenutne stanje podatkov, zajetih v HTML zapisu

Trenutno so vse spletne strani napisane v označevalnem jeziku HTML, na katerega pa ne smemo gledati kot na jezik za semantično označevanje, ampak kot na jezik za predstavitev podatkov uporabnikom.

Slika 11 prikazuje del spletne strani, kjer je podana informacija, koliko je v tem primeru podjetje do sedaj zaslužilo z mobilnimi aplikacijami.



Summary

Member Since:	January 18, 2005 | 78 quotes in last 30 days | **Last Signed in:** June 02, 2011		
Location:	Noida, Uttar Pradesh | India		
Would work on site:	Yes		
Earnings (All Time):	$2,221,936		
Earnings by Skill (Year):	Programming & Databases	$919,267	Feedback
	Database Design	$0	
	Mobile Applications	$39,988	
	Programming	$70,685	
	Smart Phone Applications	$515,441	
	Web Programming	$293,152	
Rate:	Min. Rate: $20.00 | Min. Budget: $10,000.00		Work Terms
Industries:	Education, Hospitality & Leisure, Retail & Wholesale, Technology & Internet, Telecommunications		

Slika 11: Del spletne podstrani (www.guru.com), kjer je prikazan kratek opis podjetja

Tabela 14 prikazuje, kako je stran zapisana v jeziku HTML.

```
<td>
        <div class="paddingTop5 paddingRight5">
        <strong>Earnings by Skill (Year):</strong>
        </div>
</td>
<td width="54%">
<table width="100%" cellpadding="0" cellspacing="0">

…….

<tr valign="top">
        <td style="border-bottom:0px" class="borderTopGray txt11px txtGray666 padding2
                paddingRight30">Mobile Applications</td>
        <td style="border-bottom:0px" class="borderTopGray txt11px txtGray666 padding2
                paddingRight30" align="right">$39,988</td>
</tr>
……
```

Tabela 14: Del HTML zapisa spletne podstrani (www.guru.com)

V obeh primerih gre za predstavitev iste informacije, vendar za dva različna odjemalca. Zgornji zapis je namenjen računalnikom, ki s pomočjo pravil sestavijo spletno stran, ta pa je nato servirana uporabnikom, ljudem. Ljudje podano informacijo razumemo in jo uvrstimo v določen kontekst. To storimo na podlagi izkušenj in znanja, ki ga imamo oz. smo ga pridobili v preteklosti. Znamo namreč povezati znesek $39,988 z »Mobile Applications« (mobilne aplikacije) in »Earnigns by Skill« (zaslužek glede na spretnost, znanje). Se pravi podjetje je zaslužilo $39,988 z mobilnimi aplikacijami. Če pa zdaj pogledamo spodnji zapis, HTML, vidimo, da vsebuje veliko atributov, oznak, ki služijo računalnikom, kot pravila za vizualni izpis informacije. Atributi nikakor ne služijo kot semantično označevanje. Ostaja torej problem, kako računalnikom razložiti, katere informacije se nahajajo določeni na strani.

3.2.2 Trenutna semantika

Skozi leta so se pojavili različni pristopi in rešitve k semantičnem označevanju podatkov zajetih v HTML dokumentu (podrobneje v poglavju 2.3). Med njimi je trenutno najbolj razširjen Microformats. Gre za množico vnaprej opredeljenih formatov za označevanje vsebine. Na primer: *vCard* (predloga za vizitko). Microformats so podprti s strani skupine blog agregatorja Tehnorati, ki so ravno zaradi agregiranja razvili različne formate za označevanje vsebine bloga (podrobneje v poglavju 2.3.1).

Ideji o semantičnem spletu je najbolj blizu RDFa. RDFa je priporočilo W3C mednarodnega inštituta, ki dodaja svojo množico atributov v XHTML. RDFa je razširitev RDF formata za modeliranje informacij. Njegovi osnovni konstrukti so objekt, atribut/povezava in vrednost. Skupaj tvorijo usmerjeno povezavo. Njegova pomanjkljivost je v tem, da je omejen na XHTML (podrobneje v poglavju 2.3.2).

Najnovejši med formati za označevanje je Microdata. Microdata je del specifikacije za najnovejši HTML standard, HTML5. V svojem bistvu je podoben formatu Microformats, s to razliko, da lahko dodajamo tudi svoje slovarje (podrobneje v poglavju 2.3.3).

To so trije načini, kako dodati semantiko v HTML zapis (primerjava v poglavju 2.3.4). Po eni strani je to dobrodošel napredek, vendar pa ni v skladu z idejo o semantičnem spletu pri kateri veljajo drugačni koncepti označevanja podatkov.

3.2.3 Pridobivanje podatkov iz HTML zapisa

V prejšnjih dveh poglavjih sem se dotaknil strukture spletnih strani in označevanja vsebine s pomočjo različnih formatov, prek katerih računalniki lažje razumejo podatke. Vendar je to še vedno premalo, da bi zaživela ideja o semantičnem spletu. Ker je del diplomske naloge pokazati moč te ideje in dobiti pravi občutek, kako semantičen splet deluje, sem za nadaljnje procesiranje uporabil že obstoječe podatke iz internetnih strani. Ločil sem jih od predstavitvenega dela in vnesel v svoj sistem. Ločevanje je potekalo prek že obstoječih tehnologij. Uporabil sem xQuery v kombinaciji z regularnimi izrazi. Tudi ta način pridobivanja podatkov za nadaljnje procesiranje ne rešuje trenutnega stanja, če pomislimo, bi se morali tako sprehoditi čez celoten splet in vsakič ponovit proceduro ločevanja in shranjevanja, kar ni mogoče. Vsaka stran zahteva poseg človeka, ki analizira strukturo in poda pravila ločevanja računalniku, da lahko le-ta nato opravi svoje delo.

3.2.4 Ideja o semantičnem spletu

Glavni namen semantičnega spleta je spodbujanje evolucije trenutnega spleta, da lahko uporabniki izkoristijo ves njegov potencial, kar jim omogoča lažje iskanje, kombiniranje in deljenje informacij z drugimi. Ljudje lahko uporabljajo splet za opravljanje nalog, kot so poiskati italijanski prevod za »stol« ali rezervirati knjigo v knjižnici. Vendar pa računalnik ne more opraviti vseh teh nalog, ne da bi dobil navodila od uporabnika. Spletne strani so namenjene ljudem in ne računalnikom. Semantični splet je vizija informacij, ki jih lahko interpretira računalnik in je tako zmožen opravljati dolgočasne in dolgotrajne naloge pri iskanju, kombiniranju in odločanju na podlagi informacij na spletu.

3.3 Podrobnosti rešitve

V tem poglavju bom podal podroben opis sistema, ki predstavlja rešitev problema. Da bi rešitev čim lepše ponazorila razliko med trenutnim stanjem in idejo o semantičnem spletu, sem jo postavil v domeno in si tudi zamislil primer uporabe. Izbral sem pametno iskanje po borzi dela, ki sem jo sestavil iz več internetnih strani in jih med sabo povezal. Na teh straneh so registrirani ljudje in podjetja, ki iščejo delo oz. nudijo različne storitve. Primer uporabe, ki sem si ga zamislil je umeščen v projekt, kjer se v določeni fazi ugotovi, da za njegovo dokončanje v zastavljenem času ni dovolj ljudi. Tako moramo poiskati dodatne ljudi z ustreznim znanjem, poleg znanja pa lahko definiramo tudi ostale kriterije, kot so npr. reference, lokacija stalnega bivališča, izkušnje itd.

Ker pa je internetnih strani, ki te informacije ponujajo več, sem si izbral le nekaj večjih, od tam pa črpal podatke in oblikoval ontologije, ki so naslednje:

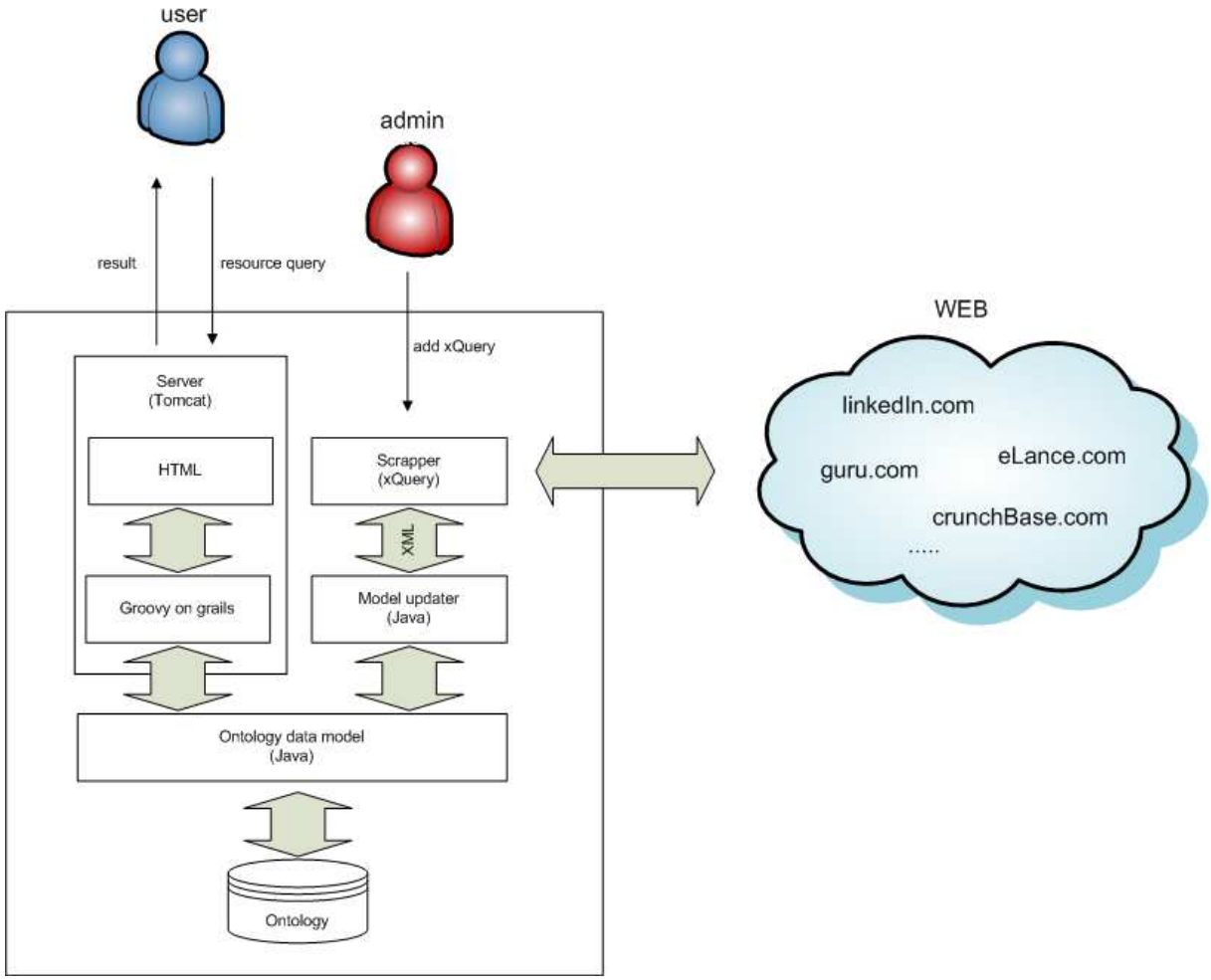
- elance.com,
- freelance.com,
- crunchbase.com,
- linkedIn.com.

3.3.1 Opis arhitekture

Stična točka med uporabnikom in sistemom je uporabniški vmesnik, ki je zasnovan kot spletna stran. Zgrajena je v okolju Groovy on Grails. Groovy je programski jezik, ki razvijalcem omogoča uporabo Java kode. Glede, na to, da so Protégé knjižnice spisane v Javi, je to eden iz med razlogov, da sem se odločil za Groovy.

Predstavitvena plast, prek katere uporabnik generira poizvedbe, kliče spodnjo plast - domenski model, ki ima implementirane vse funkcionalnosti za delo z ontologijami. Ta skrbi za vnašanje podatkov, pridobljenih iz spletnih strani, v ontologije: iz XML datotek, ki so generirane s pomočjo xQuery transformacije iz HTML oblike. Znotraj xQuery, ki uporablja xPath za naslavljanje vozlišč sem uporabil regularne izraze.

Slika 12 prikazuje grobo arhitekturo celotnega sistema. V naslednjih poglavjih bom predstavil vsako komponento posebej.



Slika 12: Arhitektura sistema

3.3.2 Spletni uporabniški vmesnik

Uporabnik dostopa do informacij o iskalcih dela prek internetnega uporabniškega vmesnika, ki ponuja različne atribute za usmerjanje iskanja. Izbira lahko med kriteriji, kot so znanje oziroma izkušnje in leta izkušenj. Poda lahko želeno področje bivanja iskalcev dela in omeji vire, iz katerih se podatki zajemajo. Na sliki (glej Slika 13) je podan primer iskanja kadra z bivališčem v Londonu, ki ima več kot dve leti izkušenj v programskem jeziku J2me (Java mobile). Kot zaključek pa se še omeji rezultate iskanja na le tri od štirih spletnih virov. Pod formo za vpis iskalnih parametrov se izpišejo rezultati. Vsi rezultati ustrezajo iskalnim kriterijem, razen zadnjega, ki le delno ustreza. Ta ima, kot znanje namesto iskanega J2me podano znanje razvoja za Android platformo. V ontologiji je namreč definirano pravilo, da sta si jezika sorodna in je zato kot rezultat veljavno tudi znanje iz android platforme. Več o tem bom opisal v poglavju o uporabljenih ontologijah.

Slika 13: Spletni uporabniški vmesnik

Spletni uporabniški vmesnik je grajen s pomočjo tehnologije Groovy on Grails. Groovy on Grails, kot vsa novejša internetna ogrodja, sloni na MVC arhitekturi. To je arhitektura, ki je sestavljena iz treh med seboj ločenih delov:

- model,
- prikaz
- usmerjevalnik.

Več o tehnologiji v poglavju 2.6.4.

Prikaz

V prikazu »MainSearchView.gsp« v tabeli (glej Tabela 15) je definiran vizualni del internetne strani.

```
<h2>Human resources search</h2>
<div id = "mainSearch">
<h3><label class="h2">Find resources that have...</label></h3>
<g:form method ="POST" action = "mainSearchView">
<table>
<tr>
<td class="description"><label>skill:</label></td>
<td><input type="text" name="skill" style="width:500px"></td>
        <td><input type="text" name="skillDuration" style="width:35px"><label class="detail">
                (>years)</label></td>
</tr>
<tr>
<td class="description"><label>location:</label></td>
<td><input type="text" name="location" style="width:100%"></td>
</tr>
<tr>
<td></td>
<td style="alighn:right" align="right"><input type=submit value="Search"></td>
</tr>
<tr>
<td></td>
<td class="description">
<g:checkBox name="cbLinkedIn" value="true" /> linkedId
<g:checkBox name="cbGuru" value="${true}" /> guru
<g:checkBox name="cbELance" value="${1}" /> eLance
<g:checkBox name="cbCrunchBase" value="${false}" /> CrunchBase
</td>
<td></td>
<td></td>
</tr>
</table>
```

Tabela 15: Koda razreda MainSearchView.gsp

Prikaz je sestavljen iz HTML elementov in dodatnih Groovy standardnih gradnikov. Primer takšnega gradnika je *g:checkbox* za delo s potrditvenimi polji. Njegova uporaba se nahaja na koncu tabele (glej Tabela 15) in sicer kot 4 potrditvena polja. Prvo izmed njih je polje za izbiro vira *linkedIn*. Gradnik vsebuje atributa ime (name = »*cbLinkedIn«*) in vrednost (value = »*true«*), ki skupaj sestavljata spremenljivko z dodeljeno vrednostjo, to spremenljivko pa za svoje procesiranje uporabi usmerjevalnik. Ob kliku na *search* (išči) se na usmerjevalniku kliče akcija, ki za svoje procesiranje uporabi vsebino vnosnih polj na strani.

Usmerjevalnik

Usmerjevalnik je tisti del arhitekture, ki skrbi za pravilno pretvarjanje akcij uporabnika v akcije, razumljive modelu.

```
class MainSearchController {

   HRQuery test = new HRQuery();

   def MainSearchView = {
      def result = []
      if (request.method == "POST"){
         String skillName = params.skill
         String location = params.location
         String minSkillDuration = params.skillDuration

         def cbLinkedIn = params.cbLinkedIn ? true : false
         def cbGuru = params.cbGuru ? true : false
         def cbELance = params.cbELance ? true : false
         def cbCrunchBase = params.cbCrunchBase ? true : false

         result = test.getPersonBySkill(skillName, location, minSkillDuration, cbLinkedIn, cbGuru,
           cbELance, cbCrunchBase);

      }
      [results:result]
   }
}
```

Tabela 16: Koda umerjevalnika MainSearchController.groovy

V tabeli (glej Tabela 16) je razvidno, kako usmerjevalnik dostopa do spremenljivk, ki so kot vnosna polja zapisana v prikazu. Spremenljivki *cbLinkedId* dodeli vrednost vnosnega polja cbLinkedId. Te spremenljivke pa nato posreduje kot atribute metodi *getPersonBySkill* objekta *test*, ki je instanca javanskega objekta *HRQuery*. Objekt *HRQuery* spada v arhitekturni del modela.

Model

Tretji del arhitekture MVC je model. Sem spada vsa logika procesiranja in poslovna pravila, ki se izvajajo v sistemu. V našem primeru je to razred *HRQuery*, ki skrbi za nalaganje in izvajanje operacij nad ontologijami.

3.3.3 Pridobivanje podatkov

Del sistema je komponenta za pridobivanje informacije o profilih posameznikov ali podjetij iz spletnih strani. Informacije so le delno semantično urejene in so v taki obliki neprimerne za uporabo v sistemu, ki sem si ga zamislil. Zaradi tega je bilo treba podatke iz HTML strani ločiti od predstavitvenega dela, jih urediti v RDF trojice, te pa nato vnesti v ontologije.

Luščenje podatkov

Luščenje podatkov iz internetnih strani je ločevanje golih podatkov od predstavitvenega dela, ki je dokaj zahteven in dolgotrajen postopek. Zahteva določeno razumevanje gole HTML kode s katero je stran zapisana in arhitekturo strani (skupina HTML dokumentov), ker so lahko podatki o osebi razpršeni po več podstraneh. V mojih primerih so bile informacije o posameznikih oz. podjetjih zapisane na eni podstrani. Zato sem za vsako vir napisal po en xQuery. Za ponazoritev slednjega sem podal par primerov, ki prikazujejo, kako je informacija zapisana v virih *eLance.com* in *guru.com*:

eLance.com: http://www.elance.com/s/scopicsoftware/about/

Del HTML kode:

```
<span class="p-summary-tagline">Scopic Software: Visual Software at its Best</span>
```

guru.com: http://www.guru.com/freelancers/Softweb-Solutions-Inc-

Del HTML kode:

```
span id="lblPageTitle" class="txt20px" style="font-weight:bold;">Softweb Solutions Inc.</span>
```

V obeh primerih gre za zapis imena podjetja. V prvem primeru je informacija zajeta v HTML gradniku *span* z imenom razreda *class="p-summary-tagline"*. Dobeseden prevod atributa razred je »povzetek-značka«. Na podlagi imena razreda in primerjavi z ostalo semantiko v HTML kodi lahko sklepamo, da bo na tem mestu vedno zapisano ime osebe ali podjetja. Problem takega pristopa je, da je potrebno »ročno« analizirati HTML zapis in najti atribute, ki določajo semantiko ustreznih podatkov. Lahko se zgodi, da upravljavec spremeni ogrodje strani in tako onemogoči pravilno branje podatkov, kar zahteva ponovno analizo strani. Postopek je relativno dolgotrajen in včasih tudi zahteven. Lahko si predstavljamo, kakšne težave bi imel šele računalnik z interpretacijo in identifikacijo želenih informacij. Po identifikaciji želenih podatkov je bilo potrebno le-te pretvoriti v format, v katerem so zapisane ontologije (OWL). Odločil sem se, da podatke zapisane v HTML obliki najprej pretvorim v svoj XML dokument nato pa jih prek domenske plasti zapišem kot ontologije. Vsak vir ima posledično svoje XML dokumente, kjer so shranjene informacije o posameznikih. Vmesnemu koraku *preslikava v svoj XML dokument*, bi se lahko izognil in podatke takoj preslikal v dokumente, kjer so shranjene ontologije. Vendar sem ocenil, da je izbran pristop varnejši in robustnejši. Do ontologij se namreč dostopa samo prek ene točke in to je Protégé programski vmesnik. To omogoča dodatno verifikacijo vsebine, kar sistem ščiti pred napakami. Poleg tega se ni treba ukvarjati z OWL sintakso.

Tabela 17 vsebuje vsebino xQuery transformacijske datoteke za internetno stran eLance.com.

```
declare namespace xhtml = 'http://www.w3.org/1999/xhtml';
<resource type = "eLance">
        <resourceType>
        {
        //xhtml:div[@class = "identity-item-first"]
        /following-sibling::xhtml:div[@class = "identity-item"]
        /xhtml:div[2]/xhtml:div[1]/text()
        }
        </resourceType>

        <resourceName>
        {
        //xhtml:div[@class = "p-summary-title"]/xhtml:h2[@class = "left"]/text()
        }
        </resourceName

        <location>
                <country>
                {
                //xhtml:div[@class = "identity-item"]/xhtml:div[contains(text()[1],"Location")]
                /following-sibling::xhtml:div[@class = "identity-item-value"]
                /xhtml:div/xhtml:div/text()[1]
                }
                </country>
                <city>
                {
                tokenize(//xhtml:div[@class = "identity-item"]/xhtml:div[contains(text()[1],"Location")]
                /following-sibling::xhtml:div[@class = "identity-item-value"]/xhtml:div/xhtml:div/text()[2],",")[2]
                }
                </city>
        </location>

        <skill>
        {
        for $x in //xhtml:table[@class="profileTable"]/xhtml:tbody/xhtml:tr
        /xhtml:td[@class="binrowheader providerProfileSkillItemRow"]
        return <li>{$x/text()}</li>
        }
        </skill>

        <hourlyRate>
        {
        tokenize(//xhtml:div[@class = "eol-layout-x3 eol-layout-first"]
        /xhtml:div[@class="p-about-l"]/xhtml:h6[@class="grey"]/text()," ")[4]}
        </hourlyRate>

        <url>
        {
```

```
        data(//xhtml:div[@class = "identity-
item"]/xhtml:input[@type="text"][@class="readonly"]/@value)
        }
        </url>
</resource>
```

Tabela 17: xQuery datoteka za luščenje in preslikavo podatkov

Rezultat te transformacije pa je XML dokument, prikazan v:

```
<entity>
        <resourceType>Company</resourceType>
        <resourceName>Gen Nex Infosoft</resourceName>
        <location>
                <country>India  </country>
                <city/>
        </location>
        <skill>
                <li>PHP4</li>
                <li>Joomla! 1.5.x Knowledge</li>
                <li>ASP.NET with SQL Server</li>
                <li>Joomla! Coding Techniques</li>
                <li>Microsoft SQL Server 2005</li>
                <li>English Basic Skills (U.S. Version)</li>
                <li>osCommerce v2.2</li>
                <li>DHTML</li>
                <li>ASP.Net 3.5 using C#</li>
                <li>MySQL</li>
                <li>Adobe Photoshop CS3</li>
                <li>PHP5</li>
                <li>Advanced PHP</li>
                <li>DotNet 2.0 using C#</li>
                <li>DotNet 2.0 using VB</li>
        </skill>

        <hourlyRate>$10</hourlyRate>
        <url>http://chandu4ugandhi.elance.com </url>
</ entity >
```

Tabela 18: XML dokument, kot rezultat transformacije

3.3.4 Posodabljanje ontologij

Prek xQuery preslikav se generirajo XML datoteke s podatki. Potrebno jih je še preslikati, urediti in dodati v ontologije. To je naloga modula *OntologyUpdater*. *OntologyUpadter* je spisan v programskem jeziku Java. Trenutno deluje kot samostojni program, ki vsebino izbrane XML datoteke preslika v ustrezno ontologijo. Celoten modul je sestavljen iz treh delov. Prvi skrbi za branje podatkov iz XML datotek. Drugi in tretji se ukvarjata z dodajanjem podatkov v ontologije. Drugi del *OntologyUpdater* pokriva skupne funkcionalnosti, kot so shranjevanje in nalaganje ontologij. Tretji del je množica virom specifičnih razčlenjevalnikov kode HTML. Dodajati nove komponente, kot razčlenjevalnike za nove vire je dokaj enostavno.

```
protected static JenaOWLModel owlModel;

public void initOwlModel(String uri){
        if (uri == null){
                throw new NullPointerException("Uri can not be null.");
        }else{
                owlModel = null;
        try {
                owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
        } catch (OntologyLoadException ex) {
                Logger.getLogger("main").log(Level.SEVERE, null, ex);
        }

}
```

Tabela 19: Metoda za nalaganje ontologije v spomin

Metoda *initOwlModel (String uri)* v tabeli (glej Tabela 19) sprejme parameter kot naslov OWL datoteke, kjer je shranjena ontologija. Nato prek statične metode *createJenaOWLModelFromURI()* razreda *ProtégéOWL* naloži OWL model v spomin, ki ga lahko prek ostalih metod vmesnika posodabljamo in nato spet shranimo na disk.

Primer klica metode na objektu *OWLModel* za shranjevanje ontologije:

```
owlModel.save(new File(url).toURI(), FileUtils.langXMLAbbrev, errors);
```

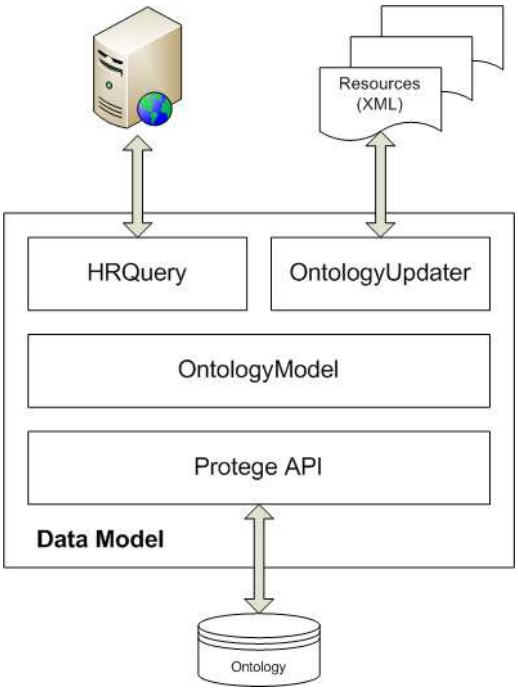
Ko je ontologija naložena v spomin in dostopna prek Protégé API, lahko po njej iščemo, dodajamo nove razrede, atribute ali entitete. Spodnja vrstica kode je primer, kako lahko dostopamo do razreda *City* (mesto) v ontologiji *Guru*, prek metode *getOWLNamedClass*, na prej ustvarjenem programskem modelu ontologije:

```
OWLNamedClass classCity = owlModel.getOWLNamedClass(„guru:City“);
```

Spodnji primer prikazuje, kako dodamo razredu *City* novo instanco, in sicer novo mesto *London*:

```
classCity.createOWLIndividual("guru:London");
```

3.3.5 Izvajanje poizvedb



Že v prejšnjih poglavjih sem se dotaknil javanskega razreda *HRQuery*. Gre za statični razred, ki ga uporablja usmerjevalnik *MainSearchController.groovy*, kot procesno logiko za vračanje rezultatov glede na uporabnikove iskalne kriterije.

Razred ustvari instanco modela v spominu, na katerem poganja poizvedbe. Poizvedbe so tipa SPARQL. V programu so predstavljene, kot niz znakov[9], ki se oblikujejo glede na uporabnikove omejitve in iskalne kriterije.

```
String queryString = "SELECT DISTINCT ?firstName ?surname ?skillName ?duration ?locationName\n" +
        "WHERE {"+
        "?individual foaf:firstName ?firstName.\n" +
        "?individual foaf:surname ?surname.\n" +
        "?individual hr:hasSkill ?skill.\n" +
        "?individual hr:isResidentOf ?location.\n";

if (skill != null && !skill.equals("")) {
        queryString = queryString + "?skill hr:name \"" + skill + "\".\n";
}

if (location != null && !location.equals("")) {
        queryString = queryString + "?location hr:name \"" + location + "\".\n";

}
```

Tabela 20: Sestavljanje poizvedbe SPARQL glede na parametre

[9] String

Po oblikovanju poizvedbe, se odvije povpraševanje. Metoda v tabeli (glej Tabela 21) služi za poganjanje poizvedb na OWL podatkovnem modelu. Na Modelu za poganjanje poizvedb skrbi metoda *executeSPARQLQuery(query).* Pred poizvedbo, se preveri, če je model veljaven.

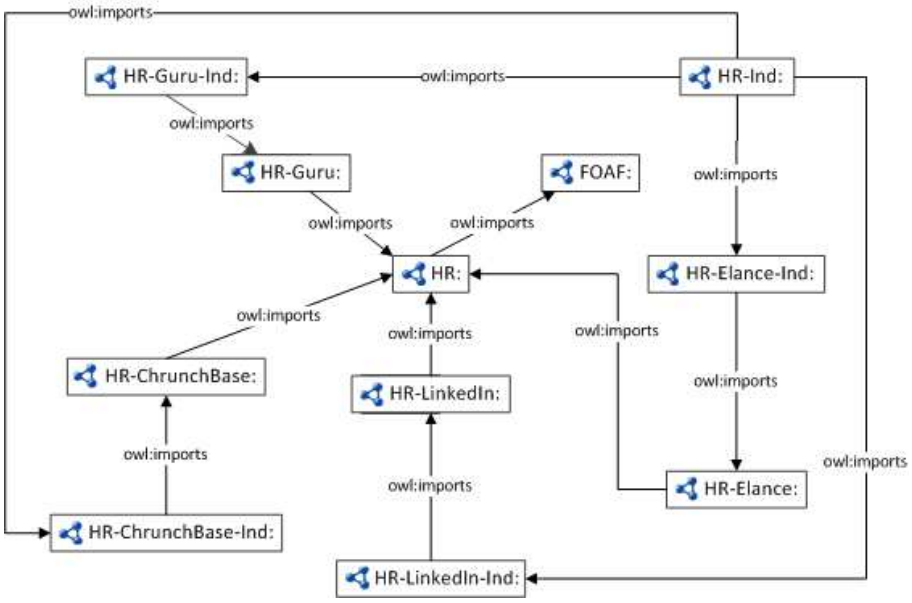
```
private static QueryResults getQueryResults(String query){
isOwlModelValid();

QueryResults queryResult = null;
try {
queryResult = owlModel.executeSPARQLQuery(query);

} catch (Exception ex) {
Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
return queryResult;

}
```

Tabela 21: Metoda, ki skrbi za izvrševanje poizvedb

3.3.6 Ontologije

Jedro diplomske naloge tvorijo ontologije. Definicija je podrobneje opisana v poglavju 2.4. V njih so urejeni vsi podatki oz. informacije. Poleg tega so zajeta vsa dodatna pravila; pravila tipa SWRL. Tako se v tem poglavju osredotočim na ontologije in njihovo uporabo v svojem sistemu.

Arhitektura povezanosti ontologij posameznih virov



Slika 14: Arhitektura povezanosti ontologij

Na sliki (glej Slika 14) je arhitektura ontologije na najvišjem nivoju, kjer je razvidna povezanost med ontologijami različnih virov. Ontologije se delijo na ontologije, ki definirajo vire in ontologije, ki hranijo instance[10]. Slednje se končajo s končnico *-ind*. Glavna komponenta je HR ontologija. Služi kot predloga oziroma pokriva skupne lastnosti vseh ostalih ontologij virov.

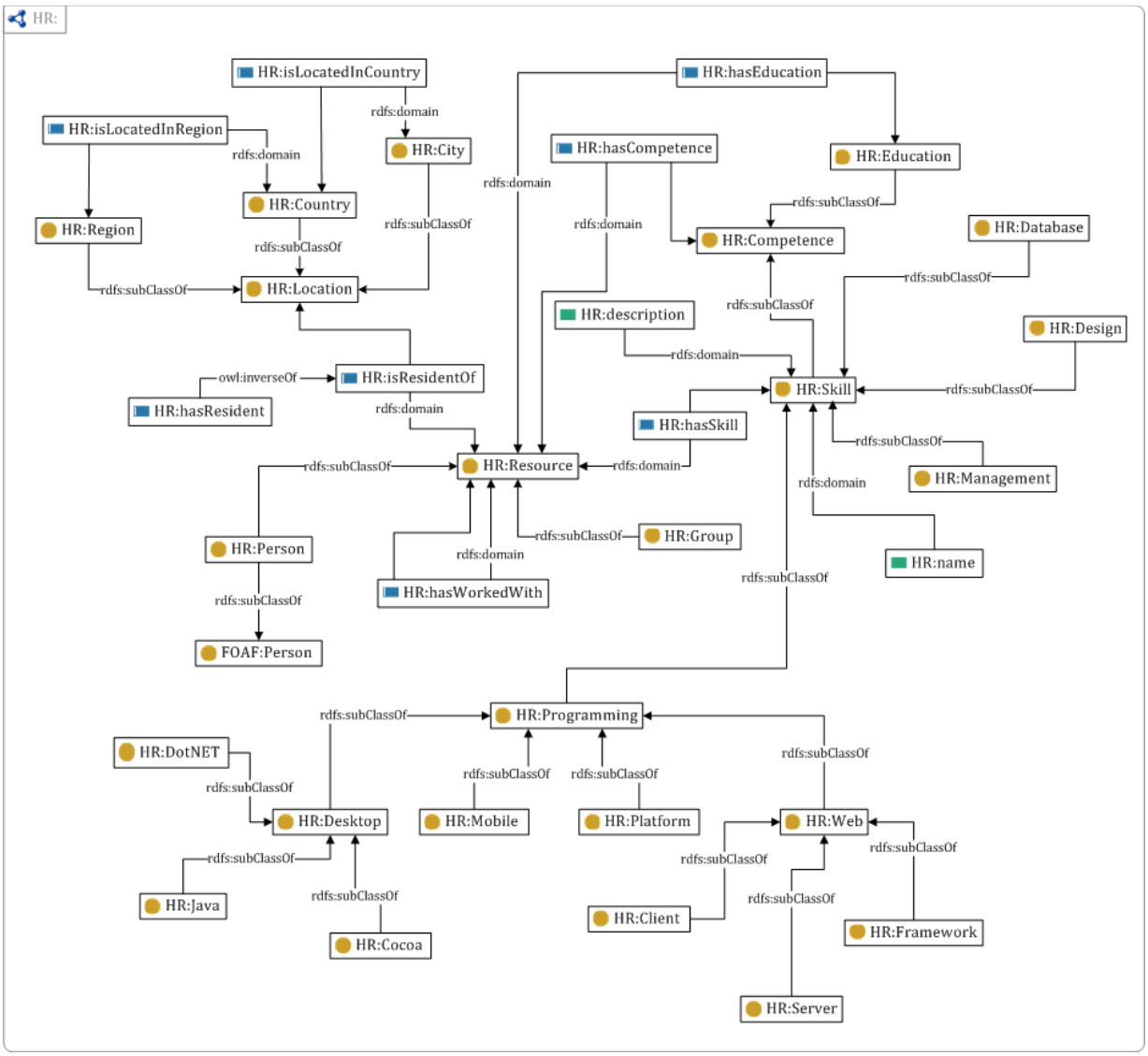
Za opis oseb in povezav med njimi obstaja na spletu projekt imenovan FOAF. Gre za odprti projekt, ki se ukvarja z izgradnjo spleta, računalnikom razumljivega, sestavljenega iz strani, ki opisujejo ljudi, povezave med njimi in njihovimi deli. Gre za že sestavljeno ontologijo, ki jo lahko uporabimo in razširimo s svojimi lastnostmi v našem projektu. Gre za še eno izmed mnogih prednosti ontologij in ideje semantičnega spleta na sploh.

Ontologije virov so razdeljene v dve skupini. Prva podaja podroben opis vira in izhaja iz ontologije HR. Njeno ime je sestavljeno, kot HR-(ime vira). Druga je enaka kot prva, vendar poseljena z instancami. Njeno ime pa je sestavljeno, kot HR-(ime vira)-Ind. Instance so lahko mesta, države, znanje, izkušnje in pa osebe ali podjetja, ki iščejo delo.

Vse ontologije, ki vsebujejo instance (-ind), so spet združene v ontologijo *HR-ind*. Vse ontologije se uvozi v ontologijo s*HR-ind*. Ker ta ontologija vsebuje instance oziroma podatke, se nad njo izvajajo poizvedbe, katere sproži uporabnik.

[10] individuals

Ontologija HR (human resources)



Slika 15: Ontologija HR (human resources)

Slika 15 prikazuje glavno ontologijo, ki vsebuje lastnosti skupne vsem ostalim ontologijam virov. Vsebuje štiri večje razrede:

- *HR:Resource* (resours): posamezniki ali organizacije,
- *HR:Skill* (vrlina): gre za praktične vrline, ki jih ima iskalec,
- *HR:Location* (lokacija): lokacija, kjer se iskalec nahaja,
- *HR:Competence* (kompetence): kompetence, kot so: izobrazba in udeležba na projektih.

Poleg teh razredov je pomembna tudi lastnost *HR:hasWorkedWith* (»je delal z«) s katero med sabo povežemo uporabnike, ki so že kdaj izpeljali kak skupni projekt oz. so bili v kakršnem koli sodelovanju. Razred *HR:Skill* se še naprej deli v podrazrede, ki zajemajo glavna področja domene iskanja.

Te so:

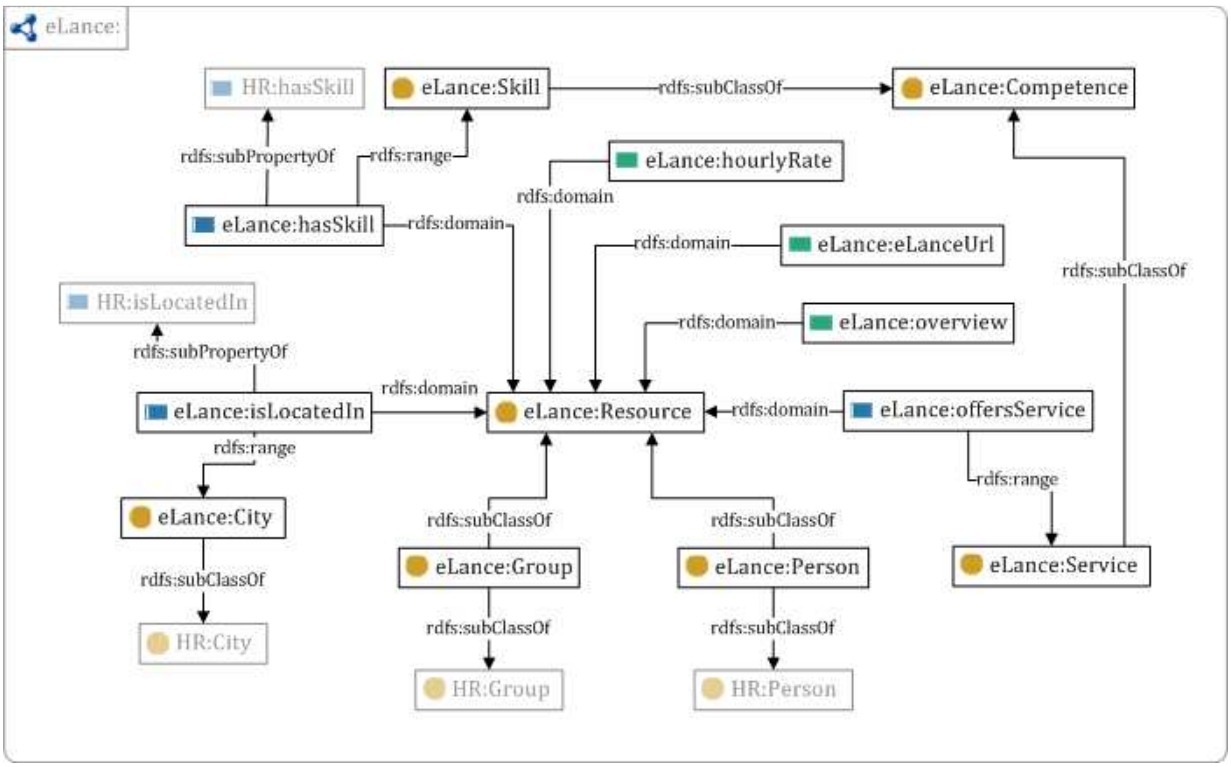
- *HR:Database* (podatkovne baze),
- *HR:Design* (dizajn),
- *HR:Management* (ravnateljevanje),
- *HR:Education* (izobrazba),
- *HR:Programming* (programiranje).

Včasih je pogoj, da je iskani kader lociran v določenem mestu, včasih pa lahko lokacijo razširimo na državo. Tako se lokacija deli na:

- *HR:Region* (Regija): npr. Skandinavija, Evropa, Amerika
- *HR:Country* (Država): npr. Anglija, Amerika, Slovenija
- *HR:City* (Mesto): npr. London, New York, Ljubljana

Vsi razredi pa so povezani z glavnim razredom *HR:Resource*, katerega instance so posamezniki ali organizacije. Gre za relativno glavno množico, saj vsebuje elemente, kateri se združujejo z instancami ostalih razredov in tako tvorijo rezultate poizvedb uporabnika sistema.

Ontologija vira eLance



Slika 16: Ontologija vira eLance

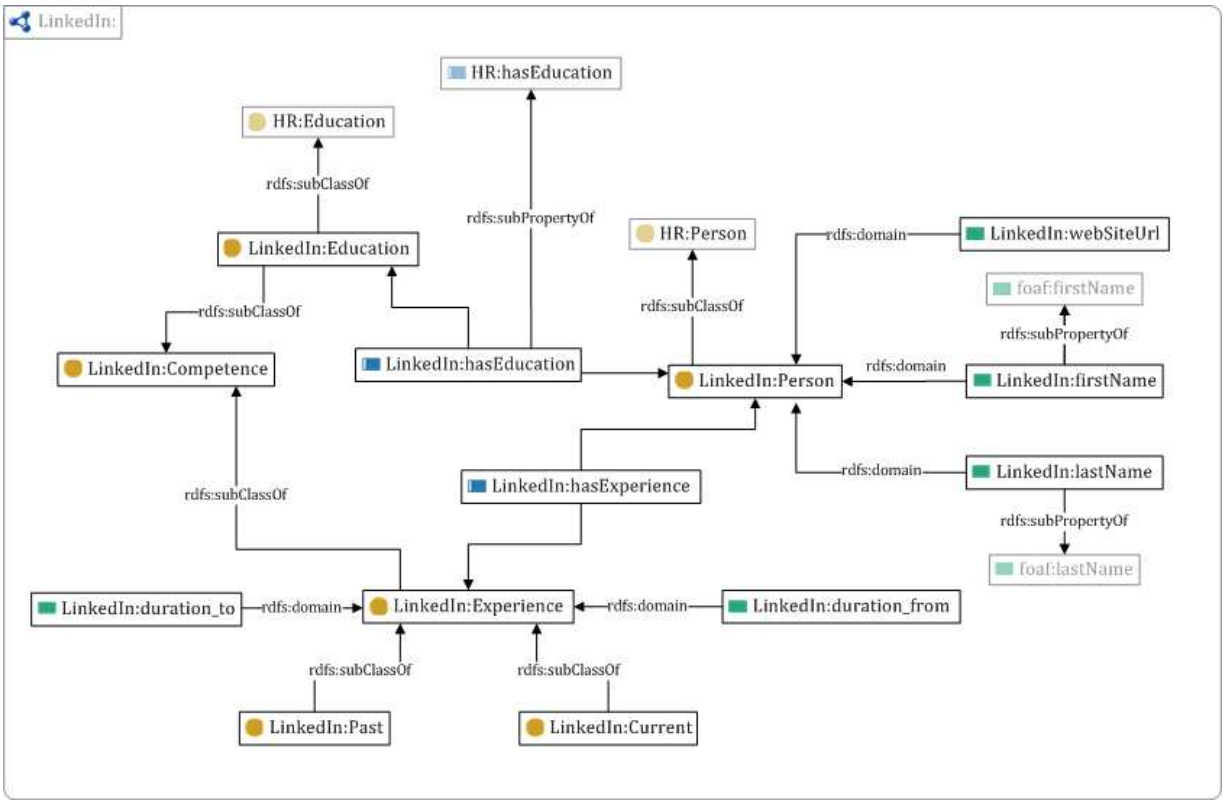
Ontologija eLance je povezana z internetnim virom *www.eLance.com*. Ontologije virov za svojo osnovo uporabljajo glavno ontologijo HR:

- *eLance:Person* je podrazred razreda *HR:Person,*
- *eLance:Group* je podrazred razreda *HR:Group,*
- *eLance:isLocatedIn* je podlastnost lastnosti *HR:isLocatedIn,*
- *eLance:hasSkill* je podlastnost lastnosti *HR:hasSkill.*

Poleg teh lastnosti, razredov in atributov pa dodaja tudi svoje nove, ki so specifični samo za ta vir:

- atribut *eLance:hourlyRate* (atribut, ki določa vrednost urne postavke),
- atribut *eLance:eLanceUrl* (atribut, ki določa stran, kjer je opis posameznika ali organizacije),
- razred *eLance:Service* (določa skupino storitev, ki jih posameznik ali organizacija ponuja),
- lastnost *eLance:offersService* (lastnost, ki povezuje posameznika ali organizacijo s storitvijo).

Ontologija vira LinkedIn



Slika 17: Ontologija vira LinkedIn

Ontologija LinkeIn je povezana z virom linkedin.com in je po svoji strukturi dokaj podobna ontologiji eLance. Podrazredi glavne ontologije *HR*:

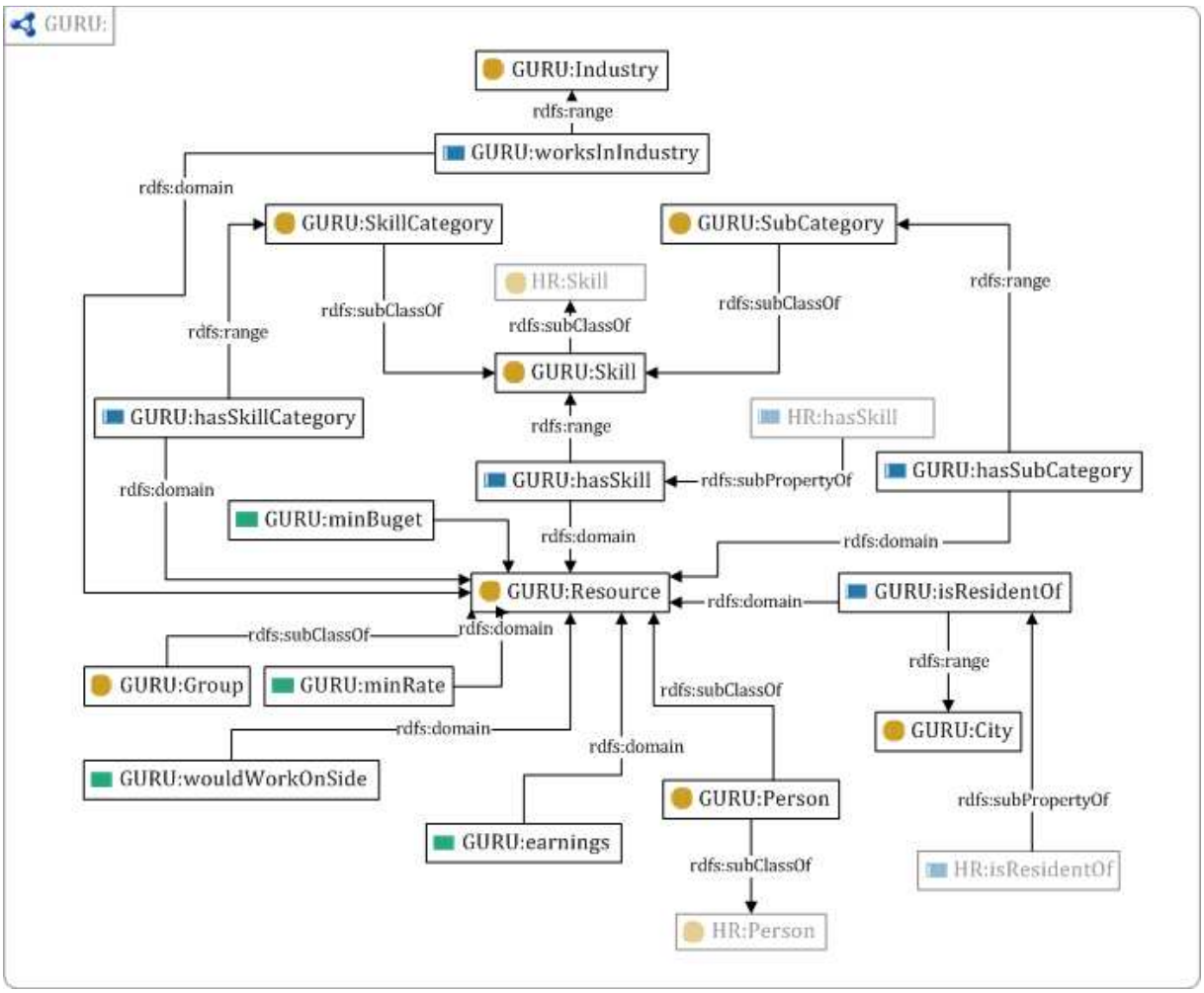
- LinkedIn:Person je podrazred razreda HR:Person,
- LinkedIn:Education je podrazred razreda HR:Education,
- LinkedIn:hasEducation je podlastnost lastnosti HR:hasEducation.

Dodaja še par novih atributov in razredov:

- LinkedIn:duration_from (atribut, ki določa kdaj je bil začetek določene izkušnje),
- *LinkedIn:duration_to* (atribut, ki določa kdaj je bil konec določene izkušnje),
- *LinkedIn:Past* (razred, ki vsebuje izkušnje, ki so trajale pred izkušnjami, ki se trenutno odvijajo),
- *LinkedIn:Current* (razred, ki vsebuje izkušnje, ki se trenutno odvijajo).

Posebnost, ki jo opazimo pri tej ontologiji je dedovanje iz ontologije *FOAF*. In sicer *LinkedIn:firstName* je podlastnost podatkovnega tipa atributa *foaf:firstName*. Tukaj bi lahko atribut *LinkedIn:firstName* spustil in uporabil kar atribut *foaf:firstName*, vendar je LinkedIn je družbena spletna stran, kjer so prijavljeni uporabniki, kjer si oblikujejo portfolio delovnih izkušenj in izobrazbe. Podobna je Facebook-u vendar s to razliko, da je LinkedIn poslovno orientirano družbeno omrežje. Uporabniki se identificirajo z imenom in priimkom. Zaradi tega sta uvedena dva nova atributa *firstName* (ime) in *lastName* (priimek).

Ontologija viraGuru



Slika 18: Ontologija vira Guru

Ontologija GURU je povezana z virom guru.com in je po svoji strukturi podobna ontologiji prejšnjih dveh.

Uvaja tri nove razrede, ki se nanašajo na izkušnje, in sicer:

- *GURU:SkillCategory:* razred, ki vsebuje kategorije izkušenj, npr. programiranje, dizajn, itd.,
- *GURU:SubCategory:* razred, ki vsebuje podkategorijo izkušnje, ki podrobneje podajajo kategorije izkušnje, npr. Java, Objective C, C#,
- *GURU:Industry* razred, ki vsebuje panoge, v katerih posameznik ali organizacija deluje.

Poleg razredov ima tudi nove atribute:

- *GURU:minBudget*: atribut, ki določa minimalna sredstva za projekt,
- *GURU:minRate*: atribut, ki določa minimalno urno postavko podjetja,
- *GURU:earnings*: atribut, ki določa, koliko je podjetje ali posameznik že zaslužilo na projektih posredovanih prek te strani.

3.3.7 Posebnosti ontologij

V podpoglavjih od 4.1 do 4.2 sem opisal strukturo ontologij vsakega vira in skupno strukturo. V tem poglavju se posvečam bolj posebnostim oz. posebnim konstruktom, kot so T-Box, A-Box, SWRL Rules in Same As.

T-Box in A-Box

Pri podrobni analizi arhitekture se opazi, da so skoraj vse ontologije podvojene oz., da ima vsaka ontologija sorodno ontologijo s končnico *_ind*. Gre za posledico prakse pri načrtovanju ontologij, kjer so instance shranjene v ločenih ontologijah. Ontologije s končnico *_ind* imajo uvoženo kot zunanjo ontologijo ontologijo vira, poleg tega pa vsebujejo še instance razredov, uvožene ontologije. V nadaljevanju poglavja skušam razložiti zakaj je to dobra praksa pri načrtovanju sistema ontologij.

Opisna logika in njena semantika tradicionalno ločita koncepte in njihove relacije od instanc, njihovih atributov in vlog, izražene kot trditve. Konceptu kot ločenemu delu pravimo tudi T-Box in predstavlja shemo ali taksonomijo domene. T-Box je strukturna komponenta, kjer je opredeljena shema in relacije.

Drugemu delu, delu instanc, pravimo A-Box in opisuje atribute instanc, vloge med instancami in ostale trditve o instancah, ki se nanašajo na njihovo pripadnost razredom v T-Box konceptih.

T-Box operacije so osnovane na sklepanju in sledenju ali preverjanju pripadnosti razredom v hierarhiji. A-Box operacije temeljijo na pravilih in vodenju preverjanja dejstev, preverjanju instanc in preverjanju konsistence. A-Bob sklepanje[11] je splošno bolj kompleksno in bolj obseženo kot T-Box.

Zgodnji semantični spletni sistemi, so bolj nagnjeni k prizadevanju ohranjevanja in upoštevanja razlik med A-Box in T-Box, medtem, ko se zadnje čase pojavljajo dvomi, da so se uporabnost in osnove za ločevanje nekako izgubila. Posebej zdaj postaja »Linked Data« postaja vse bolj razširjen in tako ista vprašanja razširljivosti in dejanske interoperabilnosti predstavljajo nove pragmatične izzive.

Bodisi samostojna podatkovna baza ali porazdeljena, imamo podatkovne zapise (strukture instanc) in logično shemo (ontologija konceptov in razmerij), s katero hočemo povezati informacijo. Gre za naravno in smiselno delitev: struktura in relacije in instance, ki poseljujejo to strukturo.

Ločitev med shemo in podatki je jasna in očitna. Medtem ko skupnost relacijske baze ni vedno vzdrževala te ločnice in RDF, semantični splet in »Linked Data« skupnost tudi ne, je ločitev smiselna, kot način (ideja) za vzdrževanje želene ločitve skrbništva[12].

Pomembnost opisne logike poleg njene vloge, kot logične podlage za semantičen splet, je zmožnost zagotavljanja ogrodja, ki omogoča ločevanje. Lahko se ustvari naravno ogrodje za usmerjanje arhitekture in načrtovanja. [1]

[11] Reasoning
[12] separation of concerns

SWRL Pravila

Naslednja posebnost, ki sem jo uporabil pri grajenju ontologij, so SWRL pravila. Semantično gledano so SWRL pravila razširitev OWL z novo vrsto pogojev (npr. če-nato stavkov). OWL ima že neke vrste pogojev, npr.:

- *SubClassOf (Person, ObjectUnionOf (Human IntelligentComputer))*
- *SubObjectPropertyOf (parentOf, ancestorOf)*

Prva vrstica predstavlja pogoj: če si *Person* (oseba), potem si ali *Human* (človek) ali *IntelligentComputer* (inteligentni računalnik). Druga vrstica pa: če imamo relacijo oče/si, potem imamo tudi relacijo prednik.

OWL pogoji so iz različnih razlogov zelo omejujoči in specializirani. Npr. *SubClassOf* pogoj ima lahko le razreda v *if (če)* ali *then (potem)* delih izraz. Npr, ni mogoče direktno mešati razredov in lastnosti:

SubClassOf(parentOf, ObjectUnionOf(Human IntelligentComputer))

Vendar imajo ti specializirani pogoji tudi svoje prednosti:

- za spremenljivko dovoljujejo neomejeno sintakso,
- bolj razkrivajo svoj namen
- pomagajo uveljavljati omejitve, ki olajšajo obdelavo in procesiranje OWL.

Slednje pa prinaša določene pomanjkljivosti, kot je izrazna moč. To je še posebej očitno, ko naletimo na pogoje lastnosti.

SWRL generalizirajo OWL pogoje na dva načina:

- omogočajo poljubne vzorce spremenljivk,
- omogoča precej neomejeno mešanje izrazov (npr. izrazi lastnosti in razredov).

SWRL je enaka OWL v tem, da zajema predpostavko o odprtem svetu[13]. Med drugim v SWRL ni negacije, kot je npr. neuspeh. Vsa ta dejstva vplivajo na to, da ima SWRL veliko izrazno moč. Veliko lastnosti, ki so vgrajene v OWL, postane ob SWRL pravilih redundantnih. Če pogledamo tranzitivnost. OWL ima specifičen konstrukt za to, da označi lastnosti, kot tranzitivno, (npr. *TransitiveObjectProperty(ancesterOf)*). V SWRL pa lahko to enostavno zapišemo s pravilom:

ancestorOf(?X, ?Z) :- ancestorOf(?X, ?Y), ancestorOf(?Y, ?Z).

Med oblikovanjem ontologije sem tudi sam naletel na omejitev, ki sem jo hotel implementirati: povezati programski jezik kot podoben ali enakovreden drugemu. Konkreten primer: definiranje programskega znanja za J2me in Android platformo. V obeh primerih gre za programski jezik Java in programiranje za mobilno platformo, vendar je J2me programiranje bolj specifično in zahtevnejše v smislu dela z delovnim spominom in procesorsko močjo. Torej: če oseba pozna J2me platformo, bo poznala, ali se bo hitro naučila tudi Android platforme. [1]

[13] open world assumption

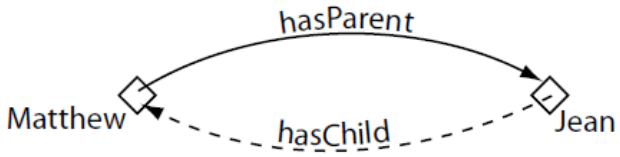
Lahko bi uporabil lastnost *sameAs*, vendar jezika nista ista in bi mogoče to v prihodnosti, z razširitvijo sistema, vodilo v napačno sklepanje. Tako sem uporabil SWRL pravilo:

hr:Person(?x) ∧ hr:hasSkill(?x, ?skill) ∧ hr:name(?skill, "J2me")
→ hr:hasSkill(?x, ?skill) ∧ hr:name(?skill, "Android")

Torej, če obstaja oseba, ki ima znanje iz J2me, lahko iz tega sklepamo, da ima ista oseba tudi znanje iz Android platforme.

Inverzna lastnost

Vsaka lastnost ima lahko tudi ustrezno inverzno lastnost. Če določena lastnost povezuje instanco A z instanco B, potem njena inverzna lastnost, povezuje instanco B z instanco A (enosmerne povezave) [9]. Npr. na sliki (Slika 19) je prikazana lastnost *hasParent* (ima starša) in inverzna lastnost *hasChild* (ima otroka). Če ima *Matthew* kot starša *Jean*, lahko zaradi inverzne lastnosti, sklepamo, da ima *Jean* kot otroka osebo *Matthew*.

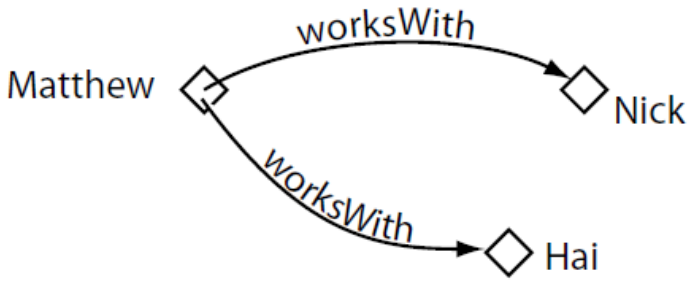


Slika 19: Inverzna lastnost

Pri gradnji ontologije sem se tudi sam posluževal inverzne lastnosti. Za določanje stalnega bivališča osebe sem uporabil *isResidentOf* (je prebivalec). Primer: *Matthew isResidentOf London,* kar pomeni, da je Matthew prebivalec Londona. Z definiranjem inverzne lastnosti *hasResident,* pa lahko računalnik sklepa, da ima *London* za prebivalca osebo *Matthew.* [1]

Kardinalnost povezav

V OWL lahko opišemo razred instanc, ki ima vsaj, največ ali točno določeno število relacij z drugimi instancami ali podatkovnimi vrednostmi. Omejitve, ki opisujejo te razrede, se imenujejo kardinalnost povezav. Za lastnost P, kardinalnost minimum določa minimalno število povezav, v katerih instanca sodeluje. Kardinalnost maksimum, pa določa maksimalno število povezav, v katerih sodeluje instanca. Lahko se uporabi kardinalnost za točno določeno število povezav. Relacije med dvema instancama štejemo kot različne le, če lahko trdimo, da se instance, ki nastopajo kot polnila relacije (na koncu usmerjene relacije), med seboj razlikujejo [9]. Npr. na sliki (glej Slika 20) je ponazorjeno, kako je *Matthew* povezan z instancami *Nick* in *Hai* prek lastnosti *worksWith* (dela z).



Slika 20: Kardinalnost povezav: Štetje povezav

Instanca *Matthew* zadostuje pogoju minimalne kardinalnosti 2 prek lastnosti *worksWith* (dela z), če sta instanci *Nick* in *Hai* različni. V tem primeru gre za dve osebi, kar ustreza pogoju.

V svoji ontologiji HR sem uporabil kardinalnost povezave za razred *City* (mesto) in lastnost *isLocatedIn* (je locirana v):

City isLocatedIn exactly 1 Cuntry.

Slednje pravilo omejuje število povezav *isLocatedIn (*je locirana v) na točno določeno število, in sicer število 1. [1]

4 ZAKLJUČEK

V diplomskem delu sem skušal prikazati razliko med idejo o semantičnem spletu in trenutni semantiki. Mislim, da sem v času pisanja dobil boljši vpogled, kaj semantični splet je in kaj na splošno semantika pomeni. S primerom uporabe sem pokazal, da se s semantično urejenimi podatki odpre nov svet aplikacij, katerih implementacija je trenutno nemogoča. Po drugi strani predstavljata kompleksnost implementacije in rahlo pomanjkanje interesa s strani uporabnikov in trga, resen problem. Predstavljati si moramo, da je potrebno označiti vse podatke na spletu, kar pa zahteva veliko denarja in energije.

Kljub problemom pa se pojavljajo nove in zanimive semantične aplikacije, ki kažejo alternativne pristope k označevanju in implementaciji ideje semantičnega spleta. Med drugimi spadata sem tudi Powerset in TripIt. Veliko strani že uporablja opisane formate za označevanje. Čeprav niso v skladu s semantičnim spletom, je njihova širša uporaba (uporablja jih namreč tudi Google za indeksiranje strani) pokazatelj, da se vendarle pojavljajo zahteve po tovrstnem označevanju.

Iz vsega zapisanega lahko sklepam, da se bo zaradi potreb uporabnikov po drugačnem, predvsem pa hitrejšem in neoviranem dostopanju do informacij, trenutni splet sčasoma spreminjal v novo obliko [5].

Pri nadaljnjem delu se bom posvečal razširitvi sistema s komponento za realno časovno posodabljanje ontologij. Trenutno se namreč sistem posodablja »ročno«, po potrebi. To predstavlja tehnični izziv, saj bi se zaradi količine podatkov in dostopanja do virov čas iskanja in obremenitev strežnika močno povečala. Ena izmed rešitev bi lahko bila delno realno časovno posodabljanje v kombinaciji s pametnimi agenti, ki bi dostopali do virov podatkov.

SEZNAM UPORABLJENIH VIROV

[1] D. Allemang, J. Hendler, Semantic Web for the Working Ontologist Modeling in RDF, RDFS and OWL, Morgan Kaufmann Publishers, Burlington, 2008, ISBN-13: 978-0123735560

[2] T. Berners-Lee, Weaving the Web: The original Design and Ultimate Destiny of the World Wide Web, Harper Collins, 2000

[3] D. Brickley, L. Miller, FOAF Vocabulary Specification 0.98. Dostopno na: http://xmlns.com/foaf/spec/.

[4] D. Brickley, R. V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 Februar 2004. Dostopna na: http://www.w3.org/TR/rdf-schema/

[5] J. Davies, Semantic Web Technologies: Trends and Research in Ontology-based Systems, Wiley, 2006, ISBN 0-470-02596-4.

[6] T. Gruber, Ontology to appear in the Encyclopedia of Database Systems, 2008. Dostopno na: http://tomgruber.org/writing/ontology-in-encyclopedia-of-dbs.pdf.

[7] T. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *International Journal Human-Computer Studies*, št. 5-6, zv. 43, 1993.

[8] C. Harrington, VMW are acquiring SpringSource which Acquired G2One, Colin Harrington blog,2009. Dostopno na: http://colinharrington.net/blog/2009/08/vmware-acquiring-springsource-which-acquired-g2one/.

[9] M. Horridge, S. Jupp, G. Moulton, A. Rector, R. Stevens, C. Wroe, A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools, University of Manchester, 2004.

[10] H. Knublauch, Protege-OWL API Programmer's Guide, Protegewiki. Dostopno na: http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide.

[11] Korth, B. Hirsch, T. Plumbaum, A Trilogy of Webs for Machines, *Technische Universitat Berlin, DAI-Labor*, Germany, 15.12.2008, Dostopno na: http://tu-berlin.academia.edu/TillPlumbaum/Papers/441420/A_Trilogy_of_Webs_for_Machines.

[12] D. Lavbič, M. Krisper, Semantika podatkov in ontologije, Uporabna informatika, št. 3, letnik XIII, sekcija Razpravel,2005.

[13] D. McGuinness, F. van Harmelen, OWL Web Ontology Language, *W3C Recommendation* 10 February 2004. Dostopno na: http://www.w3.org/TR/owl-features/.

[14] Oracle, Java. Dostopno na: http://www.oracle.com/technetwork/java/index.html.

[15] Protege, Integration of jena in protégé-owl. Dostopno na: http://protege.stanford.edu/plugins/owl/jena-integration.html.

[16] Senica, Primerjava metodologij in orodij za razvoj ontologji, diplomsko delo, 2008.

[17] N. Shadbolt, W. Hall, T. Berners-Lee, The Semantic Web Revisited, IEEE Intelligent Systems, 2007. Dostopno na: http://eprints.ecs.soton.ac.uk/12614/1/Semantic_Web_Revisted.pdf

[18] Smith, C. Welty, Ontology - towards a new synthesis, *proceedings of the International Conference on Formal Ontology in Information Systems* (FOIS2001). ACM Press, 2001.