Algoritam transformacije ontologije u strukturu taksonomije za evidencijsko zaključivanj

Galba, Tomislav

Doctoral thesis / Disertacija

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:731662

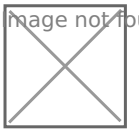
Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek







FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Tomislav Galba

Algoritam transformacije ontologije u strukturu taksonomije za evidencijsko zaključivanje

DOKTORSKI RAD

Osijek, 2016.



FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Tomislav Galba

Algoritam transformacije ontologije u strukturu taksonomije za evidencijsko zaključivanje

DOKTORSKI RAD

Mentor: doc. dr. sc. Alfonzo Baumgartner
Sumentor: dr.sc. Krešimir Šolić, znanstveni suradnik

Osijek, 2016.



FACULTY OF ELECTRICAL ENGINEERING, COMPUTER SCIENCE
AND INFORMATION TECHNOLOGY IN OSIJEK

Tomislav Galba

Ontology transformation into taxonomic structure algorithm for evidential reasoning

DOCTORAL THESIS

Supervisor: Assistant Professor Alfonzo Baumgartner, PhD
Supervisor: Krešimir Šolić, PhD

Osijek, 2016

Doktorski rad izrađen je na Sveučilištu u Osijeku Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, zavodu za programsko inženjerstvo

Mentor: doc. dr. sc. Alfonzo Baumgartner
Sumentor: dr.sc. Krešimir Šolić, znanstveni suradnik

Doktorski rad ima: 164 stranice

Doktorski rad br.: 58

Sažetak

Modeliranje podataka predstavlja proces stvaranja modela podataka kojim se simbolički opisuju hijerarhijski odnosi između stvari i događaja u nekom sustavu ili procesu. Ontologije kao napredniji način modeliranja podataka na raspolaganju imaju veći broj jezičnih konstrukata u odnosu na uobičajene načine modeliranja. Uz navedeno, ontologije na raspolaganju imaju i zaključivač koji na temelju značenja daje logičke zaključke koji utječu na hijerarhijsku strukturu i raspored objekata (instanci) klasa. S druge strane, evidencijsko zaključivanje predstavlja najnoviju metodu višekriterijskog odlučivanja temeljenu na Dempster-Shafer teoriji koja se odnosi na donošenje određenih odluka s prisutnim, a često i konfliktnim, kriterijima. Kako bi se evidencijsko zaključivanje moglo primijeniti nad OWL ontologijama potrebno je napraviti određene prilagodbe s obzirom da se OWL ontologija strukturno razlikuje od zahtjeva za primjenu evidencijskog zaključivanja.

U ovoj disertaciji predstavljen je model prilagodbe najčešće korištene OWL ontologije u taksonomsku strukturu temeljen na HermiT zaključivaču. Predloženi model dijeli se na tri glavna dijela. Prvi dio odnosi se na primjenu HermiT zaključivača nad ulaznom ontologijom, a kao rezultat dobija se hijerarhijska struktura koja predstavlja ontologiju pri čemu se u obzir uzimaju svi korišteni aksiomi OWL ontologije. Drugi dio odnosi se na primjenu algoritma za prilagodbu ontologije u taksonomsku strukturu gdje dobivena taksonomija zadovoljava pravila općeg stabla, odnosno, nad takvom taksonomijom moguće je primjeniti evidencijsko zaključivanje. Treći dio odnosi se na pripremu za proces primjene evidencijskog zaključivanja.

Najbitniji dio modela prilagodbe je predloženi algoritam prilagodbe koji na temelju skupa pravila i podpravila rješava specifične situacije koje se mogu naći u ontološkoj strukturi u svrhu dobivanja taksonomske strukture. Kako bi se provjerila ispravnost prilagođene strukture, predložena je metoda evaluacije koja se temelji na provjeri definirana tri svojstva, a to su svojstvo klasa, svojstvo veza i svojstvo povezanosti.

Primjena predloženog modela prilagodbe, algoritma i metode evaluacije prikazana je na skupu ulaznih ontologija iz dostupnih repozitorija. Dobiveni rezultati prikazuju vremena izvršavanja zasebno za HermiT zaključivač i predloženi algoritam prilagodbe, te vrijednosti definirana tri svojstva evaluacije prilagodbe. Iz rezultata je vidljivo kako bi se takav način prilagodbe mogao iskoristiti na već postojećim modelima prikazanih u OWL ontologiji, a koji opisuju neki objekt ocjenjivanja u svrhu primjene evidencijskog zaključivanja.

Ključne riječi: evidencijsko zaključivanje, HermiT zaključivač, modeliranje podataka, OWL ontologija, taksonomija, transformacija

Abstract

Data modelling represents the process of creating data model which symbolically describes hierarchical relationships between things and events within a system or process. Ontologies as more advanced data modelling approach have greater number of available language constructs compared to usual data modelling methods. Also, ontologies use reasoner which generates logical conclusions based on meaning that affect hierarchical structure and instance membership. On the other hand, evidential reasoning represents the latest multiple criteria decision method based on Dempster-Shafer theory related to the decision making process with present, but often conflicted, criteria. In order to apply evidential reasoning over OWL ontologies it is necessary to apply some adjustments considering that OWL ontology is structurally different from demands that impose evidential reasoning.

This dissertation presents adjustment model for commonly used OWL ontology into taxonomic structure based on HermiT reasoner. The proposed model is divided into three main parts. The first part is related to application of HermiT reasoner over input ontology resulting in hierarchical structure that represents original ontology regarding all used OWL ontology axioms. The second part is related to the application of ontology adjustment algorithm into taxonomic structure where resulting taxonomy satisfies general tree rules, i.e. evidential reasoning can be applied over such taxonomy. The third part is related to preparation for evidential reasoning application.

The most significant part of the adjustment model is proposed adjustment algorithm that addresses specific situations existing in ontology structure based on set of rules and sub rules in order to get taxonomic structure. The evaluation method based on satisfying of three predefined properties (class property, relation property and connectivity property) in order to verify the adjusted structure correctness is proposed.

The application of proposed adjustment model, adjustment algorithm and evaluation method is performed over the input set of ontologies from available online repositories. The results show executing times separately for HermiT reasoner and proposed adjustment algorithm and also the values of three predefined evaluation properties. The results show that proposed adjustment method can be used over existing data models represented in OWL ontology that describe some object of evaluation in order to apply evidential reasoning.

Keywords: data modelling, evidential reasoning, HermiT reasoner, OWL ontology, taxonomy, transformation

Sadržaj

1. Uvod 1
1.1. Cilj istraživanja 2
1.2. Pregled disertacije po poglavljima 4

2. Semantički web 7
2.1. Tehnologije semantičkog web-a 11
2.1.1. Jedinstveni identifikator resursa 11
2.1.2. XML 13
2.1.3. RDF i RDFS 15
2.1.4. Ontologije 19

3. Jezik za predstavljanje znanja OWL 48
3.1. Podjela OWL jezika 55
3.2. Definiranje klasa 60
3.3. OWL 2 66
3.4. Podjela OWL 2 jezika 68

4. Algoritam evidencijskog zaključivanja 70
4.1. Postupak procjene stanja 71

5. Model prilagodbe ontološke strukture u taksonomsku strukturu temeljen na HermiT zaključivaču 77
5.1. Pregled postojećeg područja istraživanja 80
5.2. Izrada modela 84

6. Algoritam transformacije ontologije i optimizacija dobivene taksonomije u skladu sa zahtjevima evidencijskog zaključivanja 101
6.1. Ovisnost pravila transformacije 112
6.2. Primjena algoritma za transformaciju 115

7. Kriterij vrednovanja rezultata transformacije ontologije u taksonomsku strukturu 124

8. Zaključak 133
8.1. Metodologija istraživanja 133
8.2. Model prilagodbe ontološke strukture u taksonomsku strukturu 134
8.3. Ograničenja algoritma za transformaciju 136
8.4. Znanstveni doprinosi istraživanja i budući rad 137

Literatura 138

Popis skraćenica 151

Životopis 156

Poglavlje 1

Uvod

Sve veći broj korisnika Interneta kao glavnog medija za razmjenu i spremanje podataka rezultira stvaranjem velike količine podataka koja ima tendenciju svakodnevnog rasta. Pretraživanje informacija za nekim ciljanim pojmom predstavlja sve veći izazov za običnog korisnika zbog same strukture trenutnog web-a. Navedeni problem i njemu slični bude interes znanstvenika za stvaranjem novih funkcionalnosti i optimizacijom postojeće infrastrukture. U zadnjih nekoliko godina semantički web koji se nameće kao nadolazeća nova generacija Interneta budi sve veći interes. Glavni cilj semantičkog web-a nije zamjena postojećeg web-a već proširenje u smjeru razvoja tehnologija koje će olakšati pronalazak informacija dodavanjem značenja umjesto korištenja ključnih riječi. Nažalost, zbog tehničkih, ali i problema koji su više organizacijske prirode, semantički web ostaje za sada samo projekt unutar nekoliko skupina entuzijasta i kao predmet studiranja na sveučilištima diljem svijeta. Međutim, ontologije kao osnovni element semantičkog web-a zbog svojih mogućnosti reprezentacije znanja ostaju u upotrebi do danas, a koriste se u mnoštvu različitih domena primjene kao što su poljoprivreda, avijacija, biologija, kemija, civilno inženjerstvo, računalne znanosti, elektronika, lingvistika, matematika i drugih. Premda u znanstvenoj literaturi postoji nekoliko različitih definicija ontologija, za definiciju se uglavnom uzima da je ontologija formalna specifikacija dijeljene konceptualizacije neke domene, odnosno, ontologija predstavlja skup koncepata neke domene sa definiranim međusobnim relacijama. Kada bi se uspoređivale ontologije kao način modeliranja podataka s uobičajenim modeliranjem podataka gdje se kao primjer može navesti UML (Unified Modeling Language) jezik, onda bi se uzele sljedeće točke za usporedbu [1]:

- Područje primjene – odnosi se na određeno područje koje se želi pokriti pojedinom metodom. Uobičajeno modeliranje koristi se kako bi se ispunile specifične potrebe ovisno o zadatku i primjeni dok se kod korištenja ontologija gleda kako bi se pokrila što veća materija, bez prevelikog oslanjanja na specifične zadatke.
- Pokrivenost znanja – nadovezujući se na područje primjene, može se izvesti zaključak da ontologije nasuprot uobičajenog modeliranja podataka daju puno širu reprezentaciju

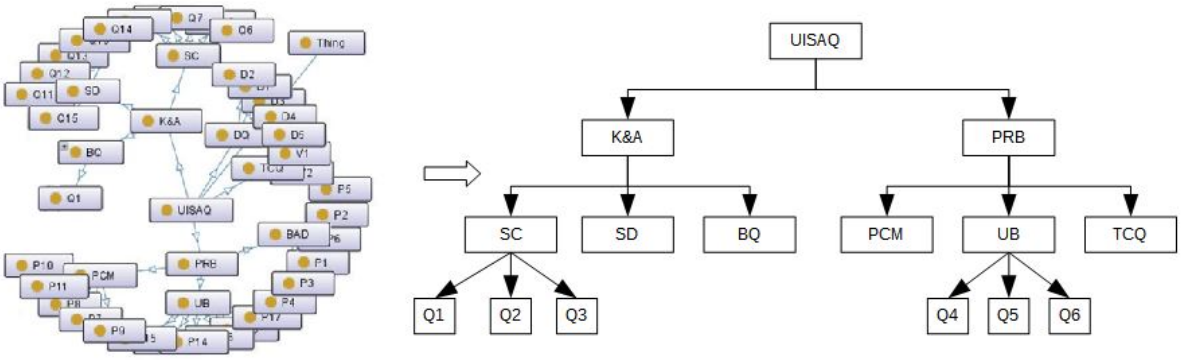
koncepata i relacija obrađivane materije.

- Ekspresivnost – ontologije raspolažu znatno većim skupom jezičnih konstrukata nasuprot uobičajenih načina modeliranja. Uz to, na raspolaganju stoji i zaključivač koji omogućava dobivanje rezultata na temelju značenja podataka unutar ontologije.
- Razina djelovanja – odnosi se na razinu apstrakcije koja je kod uobičajenog modeliranja znatno manja. Drugim riječima, prilikom opisivanja problema ontologijom, uzima se u obzir veći broj mogućih situacija koje bi se u budućnosti mogle pojaviti kao potreba.

U samim počecima, opisivanje sličnih domena informacija korištenjem različith ontoloških jezika koji su sintaksno i semantički različiti dovelo je do problema interoperabilnosti. Ti problemi rješavaju se uvođenjem standardnog jezika OWL (Ontology Web Language) 2004. godine koji predstavlja deskriptivni jezik za opisivanje ontologija na web-u. Kao najnovije razvijeni standard u to vrijeme, postaje jedan od najpopularnijih ontoloških jezika zbog izrazite ekspresivnosti i formalne semantike. Kao najpopularnije, OWL ontologije uzimaju se kao glavni objekt istraživanja u sklopu ove disertacije. Kako bi se različite implementacije sustava za reprezentaciju znanja mogle koristiti kao OWL implementacija ili obrnuto, potrebno je na neki način prilagoditi strukturu. Za tu svrhu postoje različite metode u literaturi, a spominju se transformacija (eng. transformation), translacija (eng. translation), stapanje (eng. merging), uparivanje (eng. mapping), integracija (eng. integration) i poravnanje (eng. alignment). Upravo mogućnost iskorištavanja OWL ontologija u druge svrhe korištenjem nekih od navedenih metoda dovodi do ciljeva istraživanja ove disertacije.

1.1 Cilj istraživanja

Metode višekriterijskog odlučivanja odnose se na donošenje određenih odluka prema parametrima koji se izračunavaju nad skupom nekih podataka (ocjene atributa). Primjena su različiti ekspertni sustavi, baze podataka, procjene rizika i slično. Evidencijsko zaključivanje predstavlja najvnoviju metodu višekriterijskog odlučivanja temeljenu na Dempster-Shafer teoriji u kojoj se zadani atributi opisuju distribuiranim ocjenama uključujući stupanj uvjerenja [2], [3]. Kako bi se evidencijsko zaključivanje uspješno provelo, prvo je potrebno sastaviti taksonomsku strukturu koja opisuje sve atribute, zajedno s onim atributima koji se ocjenjuju.
Proučavanjem literature iz područja modeliranja podataka koristeći ontologije i područja multikriterijskog odlučivanja, odnosno, primjene algoritma za evidencijsko zaključivanje, uočena je mogućnost iskorištavanja najboljeg od te dvije metode. Kako bi se to postiglo, potrebno je ontološku strukturu koja je predstavljena grafom prilagoditi u strukturu nižeg reda, odnosno u taksonomiju kako bi se na njoj mogao primijeniti algoritam za evidencijsko zaključivanje.



Slika 1.1: Prilagodba ontološke strukture

Potrebno je napomenuti da iako taksonomije predstavljaju jednostavnu strukturu podataka koja se često poistovjećuje sa stablima kao strukturom podataka, ne mora značiti da one nužno i zadovoljavaju pravila stabla. Prema tome, graf struktura kojom je predstavljena ontologija također bez ikakvih izmjena se može predstaviti kao taksonomija. Međutim, na strukturi koja ne zadovoljava pravila općeg stabla (svaki čvor može imati maksimalno jednog roditelja) ne može se primjeniti evidencijsko zaključivanje. Prema tome, temeljni cilj istraživanja ove disertacije je razviti metodu koja posjeduje sljedeća svojstva:

- Mogućnost iskorištavanja svih definicija svojstava i aksioma zadanih u ontologiji prilikom prilagodbe sa što manjim gubicima.
- Mogućnost potpuno automatske prilagodbe bez intervencije korisnika.
- Mogućnost prilagodbe i optimizacije prema zahtjevima primjene evidencijskog zaključivanja.
- Mogućnost evaluacije dobivenih rezultata.

Zadovoljavanjem prve točke iskoristit će se reprezentacija znanja predstavljena ontologijom prilagođavanjem u drugu jednostavniju strukturu kao što je taksonomija. U sklopu ove disertacije planira se ostvariti priprema ontologije koja će se zasnivati na zaključivaču HermiT. HermiT zaključivač predstavlja javno dostupan OWL zaključivač, te je uključen u standardne biblioteke Protégé alata za razvoj ontologija.
Zadovoljavanjem druge točke, omogućit će se automatska prilagodba ontologije u taksonomiju. U sklopu ove disertacije planira se definirati model prilagodbe i algoritam za prilagodbu koji će se temeljiti na određenom skupu pravila.
Zadovoljavanjem treće točke omogućit će se optimizacija taksonomske strukture kao rezultata prilagodbe. U sklopu ove disertacije planira se ostvariti algoritam koji će takvu strukturu dodatno optimizirati kako ne bi došlo do pojave nepotrebnih čvorova ili instanci, a samim time i višestrukog ocjenjivanja istog čvora prilikom primjene evidencijskog zaključivanja.
Zadovoljavanjem četvrte točke osigurat će se ispravnost prilagođene strukture s obzirom da ukupna ocjena evidencijskog zaključivanja treba biti što točnija kako bi se mogle napraviti is-

pravne odluke vezane za ocjenjivani sustav. Uz to, osigurat će se i da se svi elementi izvorne ontologije nalaze i u rezultirajućoj taksonomiji. U sklopu ove disertacije planiraju se definirati svojstva potrebna za određivanje ispravnosti prilagodbe.
Konačno, zadovoljavanjem svih navedenih točaka omogućava se korištenje svih prednosti opisivanja neke domene korištenjem ontologije dok se u isto vrijeme pojedini dijelovi ili cijeli sustav korištenjem predloženih metoda može jednostavno ocijeniti primjenom evidencijskog zaključivanja.

1.2 Pregled disertacije po poglavljima

Disertacija je podijeljena na sedam poglavlja koja su poredana tako da osim uvoda, prva tri poglavlja čine uvod u tematiku, dok zadnja tri opisuju problem i način rješavanja problema.
Drugo poglavlje opisuje semantički web kao ideju o trećoj generaciji web-a. Iako su ontologije postojale puno prije pojave ideje o semantičkom web-u, stvarna uporaba ontologija u računalnoj znanosti započinje idejom o implementaciji semantičkog web-a. Iako semantički web nije do danas zaživio zbog različitih ograničenja koji se najviše odnose na administrativne probleme, ontologije su ostale u uporabi i imaju vrlo široku primjenu. Upravo zbog toga, u drugom poglavlju prikazane su i opisane tehnologije i problemi s do tada postojećim tehnologijama reprezentacije znanja u svrhu razvoja ontoloških jezika kakvi postoje danas. Pri tome se misli na URI (Uniform Resource Identifier) kao način referenciranja resursa, XML (eXtensible Markup Language) jezik za razmjenu podataka između računala te RDF (Resource Description Framework) i RDFS (Resource Description Framework Schema) jezika kao nasljednika RDF jezika za predstavljanja informacija o web resursima. Na kraju drugog poglavlja opisane su ontologije kao osnovna komponenta semantičkog web-a uz dodatne detalje kao što su:

- Formalnost i klasifikacija ontologije prema kojoj se ontologije mogu podijeliti na opće ontologije, ontologije domene, ontologije zadatka i aplikacijske ontologije.
- Dijelovi ontologija koje se odnose na gradivne elemente svake ontologije pri čemu se misli na klase, instance, aksiome i relacije.
- Predlošci koji služe za prikaz, modeliranje i ugradnju znanja ontologije, a koji se odnose na logiku prvog reda i deskriptivnu logiku. Uz predikatnu logiku, opisane su osnove logike i propozicijska logika kao prethodnik predikatne logike prvog reda. Bitno je napomenuti da je najviše pažnje pridodano opisivanju deskriptivne logike kao teorijske osnove u implementaciji različitih sustava i predstavljanja znanja.
- Klasifikacija ontoloških jezika gdje je prikazana većina ontoloških jezika podijeljenih u tri glavne grupe (logički, temeljeni na okvirima, temeljeni na grafovima).

Treće poglavlje nastavlja se na drugo s detaljnjim opisivanjem OWL (Web Ontology Language) jezika koji pripada jezicima temeljenim na deskriptivnoj logici standardiziran 2004. godine kao

deskriptivni jezik za opisivanje ontologija na web-u. Opisani su:

- Utjecaji na razvoj kao i prednosti nad drugim već postojećim jezicima.
- Struktura OWL dokumenta koja se odnosi na zaglavlje dokumenta, informacije o verziji i uvezenim elementima druge ontologije, klasama, svojstvima i instancama klasa te završnim oznakama dokumenta.
- Podjela OWL jezika koja se odnosi na glavne tri podjele OWL jezika (OWL Lite, OWL DL i OWL Full). Uz podjele OWL jezika prikazane su i tablice apstraktne sintakse uz pripadajuću DL sintaksu za najkorištenije podjele OWL Lite i OWL DL.
- Definiranje klasa koje se odnosi na opisivanje dva glavna tipa klasa u smislu definiranja roditeljskih klasa ili klasa nasljednika (primitivne klase) i definiranja klasa koristeći neke od dodatnih klasnih izraza kao štu su enumeracija, restrikcija svojstva ili booleove kombinacije (kompleksne klase). Također, opisani su i načini definiranja klasa s obzirom na pripadnost instanci, a odnose se na korištenje samo nužnih uvjeta ili nužnih i dovoljnih uvjeta. Bitno je napomenuti da je najviše pažnje dano upravo opisivanju strukture ontologije s obzirom da je rad sa strukturom ontologije jedan od glavnih ciljeva ove disertacije.
- OWL 2 jezik koji predstavlja nastavak na OWL jezik. Opisani su problemi zbog kojih je napravljena revizija OWL jezika kao i podjela OWL 2 jezika na tri glavna dijela (OWL 2 EL, OWL 2 QL i OWL 2 RL).

Četvrto poglavlje opisuje korišteni algoritam evidencijskog zaključivanja koji se postavlja kao primjena za dobiveni rezultat algoritma za transformaciju ontologije u strukturu taksonomije kao glavnog cilja ove disertacije. Uz navođenje pripadnosti evidencijskog zaključivanja u metode višekriterijskog odlučivanja opisan je i sam postupak procjene stanja nekog sustava. Uz postupak je naveden i primjer kojim se potvrđuje način funkcioniranja same metode evidencijskog zaključivanja.
Peto poglavlje odnosi se na model prilagodbe ontološke strukture u taksonomsku strukturu gdje je opisano sljedeće:

- Metode rješavanja interoperabilnosti što se odnosi na transformaciju, translaciju, stapanje, uparivanje te integraciju i poravnanje ontologije.
- Usporedba svake od metoda rješavanja interoperabilnosti u smilsu ulazne i izlazne strukture.
- Pregled područja istraživanja gdje su temeljem pregleda literature navedeni i kratko opisani svi pronađeni radovi koji se bave transformacijom OWL ontologija.
- Izrada modela temeljena na ograničenjima postojećih metoda transformacije.

Šesto poglavlje opisuje novorazvijeni algoritam transformacije temeljen na modelu transformacije i optimizaciju rezultirajuće taksonomije prema zahtjevima evidencijskog zaključivanja. U poglavlju su opisani:

- Glavni koraci algoritma za transformaciju koji se dijele na stvaranje popisa klasa ulazne strukture, prilagodbu graf strukture podataka (OWL ontologija) i evaluaciju rezultata transformacije. Transformacija graf strukture podataka najbitniji je dio algoritma transformacije i sastoji se od niza pravila, a i podpravila koja su detaljno objašnjena.
- Ovisnost pravila transformacije.
- Primjena algoritma za transformaciju gdje je naveden skup ulaznih testnih ontologija različitih karakteristika koje su detaljno opisane pripadajućim metrikama. Također, navedena su i komentirana vremena izvođenja kako algoritma za transformaciju tako i HermiT zaključivača na kojem se temelji model za transformaciju.

Sedmo poglavlje predstavlja zadnje poglavlje u kojem su opisani:

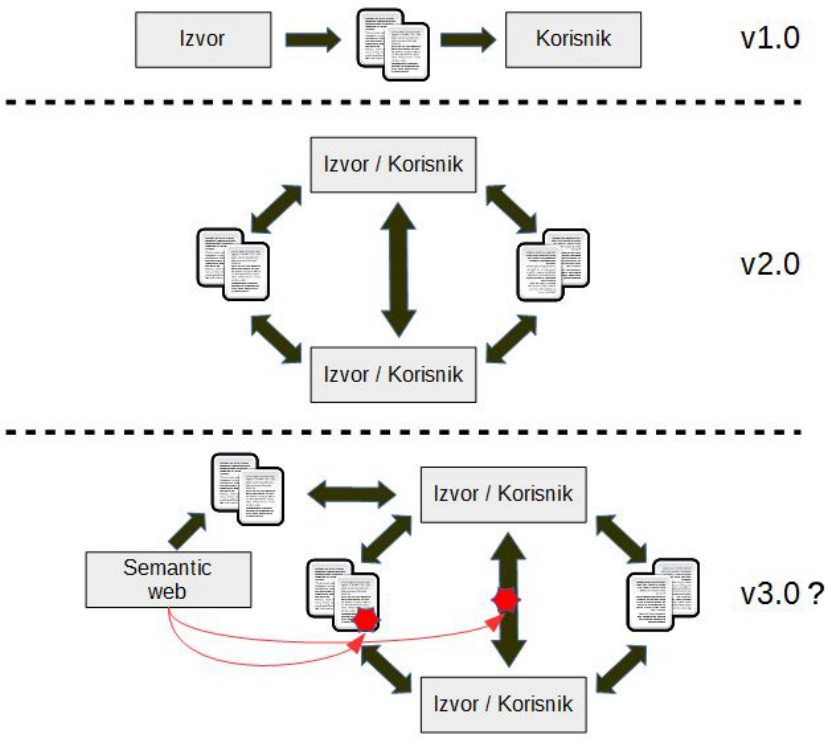
- Razlozi evaluacije ontologije.
- Postojeće metode evaluacije ontologija.
- Metoda evaluacije rezultirajuće taksonomije dobivene algoritmom transformacije s izvornom ontologijom koja je podijeljena u tri glavna dijela (svojstvo klasa, svojstvo veza i svojstvo povezanosti).
- Prikaz rezultata transformacije nad skupom ulaznih onotologija opisanih u šestom poglavlju.

Na kraju slijede zaključak i literatura.

Poglavlje 2

Semantički web

Web kao informacijski medij i rad na njegovom razvoju omogućio je pristup velikoj količini informacija milijunima ljudi [4]. Također, uz pristup informacijama, omogućava svakoj osobi objavu vlastitih dokumenata u obliku web stranica, slika i slično [5]. WWW (World Wide Web) dijeli se na dvije generacije kao što je prikazano na Slici 2.1.



Slika 2.1: Podjela WWW po verzijama

U prvoj generaciji, pojavljuju se statički definirane HTML (HyperText Markup Language) stranice, kao i nekoliko osnovnih usluga poput elektroničke pošte, foruma i popularnog čavrljanja (eng. chat). Slobodno se može reći kako je razlog tomu u to vrijeme bilo vrlo ograničeno sklopovlje kao i relativno niska brzina internet veze. Drugu generaciju obilježava širokopojasni

Internet kao jeftino i brzo sklopovlje što rezultira pojavom dinamički definiranih web stranica. Također, pojavljuju se napredniji servisi kao što je videokonferencija, društvene mreže, podkasting i drugi. Ono što je ostalo zajedničko i prvoj i drugoj generaciji je stvaranje stranica upotrebom strukturiranih opisnih jezika kao što su HTML i XML. Upotrebom tih dviju tehnologija moguće je opisati stranice u vidu korištenja meta-oznaka tako da ih računalo može analizirati.
WWW v2.0, odnosno, trenutni web može se definirati kao [6]:

- Digitalnu biblioteku u obliku skupine web stranica povezanih hipervezama.
- Aplikacijska platforma kojoj se pristupa te se prihvaćaju rezultati putem web stranica.
- Multimedijska platforma dostupna bilo gdje u svijetu.
- Poslovna platforma kao što je e-bay, Alibaba, PayPal i drugi.

Međutim, sve te prednosti dolaze i s određenim problemima koji su s vremenom sve veći. Kao jedan od većih problema odnosi se na samu iskoristivost dostupnih informacija. Premda se točan broj teško može odrediti, prema nekim izvorima trenutno na web-u postoji više od 11 milijardi stranica što prema trenutnom broju stanovnika znači gotovo 2 stranice po stanovniku. Problemi koji nastaju prilikom pretraživanja su [6]:

- Jedan dokument (stranica) kao rezultat.
- Manjak semantike, pretraživanje se odnosi samo na ključne riječi.
- Nebitni podaci kao rezultat pretrage (uzrok manjka semantike).

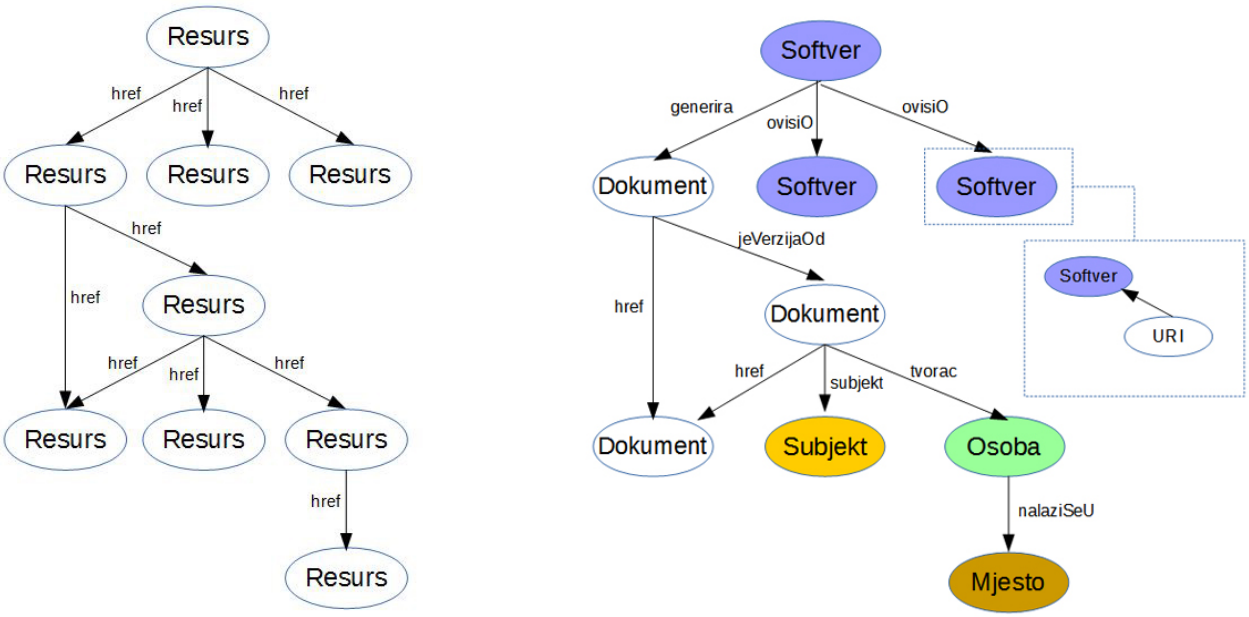
Iz navedenog, vidljivo je da semantika predstavlja veliki problem. Generalno govoreći, semantika je grana lingvistike koja se bavi isključivo proučavanjem riječi i njihovih oblika kao glavnog načina za označavanje predmeta i pojava.
Ranije u tekstu spomenuto je da se stranice opisuju korištenjem meta-oznaka kako bi bile računalno obradive. Trenutno takav način ide u korist tražilicama poput Google tražilice [7], međutim, računalo ne razumije samo značenje onoga što je zapisano već samo sintaksu [8]. Za primjer se može uzeti riječ ”OWL”. Računalo nije u mogućnosti razložiti riječ na sva značenja, odnosno, nije u mogućnosti odraditi pretragu na taj način da vrati rezultat s obzirom samo na informacijsku potrebu korisnika.
S obzirom na navedeno i ostale moguće probleme koji nisu pokriveni u ovom radu, istraživači iz industrije i akademske zajednice rade na implementaciji treće generaciji web-a (Slika 2.1) u kojem bi se stranicama dodavalo eksplicitno značenje kako bi informacije koje se nalaze na tim stranicama bile razumljive strojevima što bi osiguralo mogućnost automatskog procesiranja istih [5], [9], [10], [11].
Treća generacija web-a odnosi se na Semantički web [12],[13] koji dolazi kao jedan od najbitnijih projekata pod okriljem W3C (World Wide Web Consortium). W3C predstavlja međunarodno tijelo za standardizaciju web-a i ima primarnu ulogu u standardizaciji i realizaciji

semantičkog web-a [14].

Cilj semantičkog web-a je proširenje postojećeg web-a [12] na način da se uvedu novi standardi i tehnologije sa svrhom definiranja značenja informacija na web-u definiranjem relacija između elemenata i njihovih svojstava čime bi se riješio problem višeznačnosti. Usporedba trenutnog web-a i semantičkog web-a kao ideje prikazana je na Slici 2.2. Takvo proširenje dovelo bi do napretka na nekoliko područja[15]:

- Pretraživanje informacija tako da se dobiju bolji i prošireniji rezultati (umjesto utvrđivanja sličnosti i ranga mrežnih stranica, gleda se relacija između elemenata i njihovih svojstava).
- Automatizacija.
- Bolja interakcija između računala i čovjeka.

Pod automatizacijom se prvenstveno misli na bolju povezanost pretraživanja u smislu da se rezultati kombiniraju iz više izvora u jedan "dokument" kao i na automatizaciju usluga iz različitih područja kao što su digitalne knjižnice, zdravstvo i ostalo.



(a) Trenutni web.

(b) Semantički web.

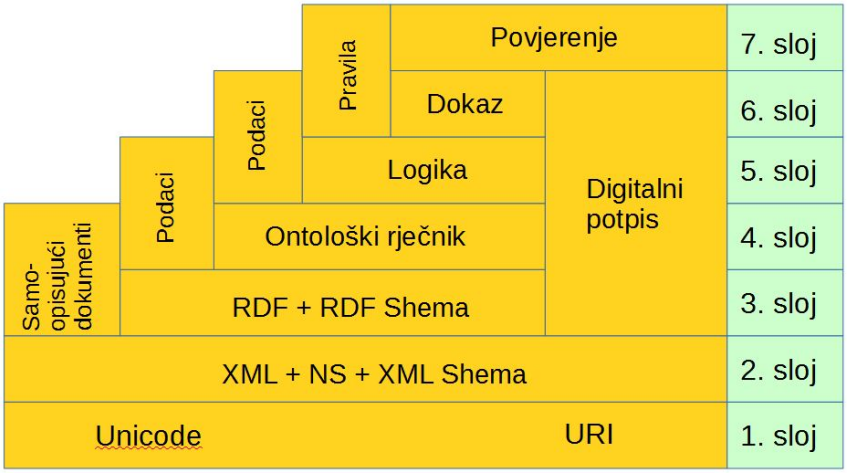
Slika 2.2: Usporedba trenutnog web-a i ideje semantičkog web-a.

Semantički web temelji se na sljedećim principima [4]:

- Jednostavnost.
- Modularnost.
- Tolerancija.
- Decentralizacija.

Decker,S. [16] definira uvjete koji trebaju biti zadovoljeni kako bi se informacije na web-u mogle dijeliti u strojno razumljivom obliku [15]:

- Sveopća moć izražavanja.
- Sintaksna interoperabilnost.
- Semantička interoperabilnost.



Slika 2.3: Informacijska interoperabilnost [7],[16]

Na Slici 2.3 prikazani su slojevi informacijske interoperabilnosti poznato još i kao "Berners-Lee pie chart" po začetniku semantičkog web-a gdje:

- Prvi sloj podržava prepoznavanje znakova i hiperveza.
- Drugi sloj omogućava razmjenjivanje meta-oznaka.
- Treći i četvrti sloj omogućavaju razumijevanje meta-podataka aplikacijama uvođenjem ontoloških rječnika i općih okvira za opis resursa.
- Peti i šesti sloj omogućavaju zaključivanje.
- Sedmi sloj omogućava provjeru vjerodostojnosti znanja.

Međutim, kako bi se Semantički web u potpunosti uveo u svakodnevicu, potrebno je riješiti određene izazove koji se nameću na tom putu [5]:

- Provodljivost zaključivanja.
- Logička dosljednost.
- Promjenjivost informacija.

Provodljivost zaključivanja odnosi se na mogućnost kvalitetnog sažimanja rezultata prikupljanjem informacija iz većeg broja nepovezanih izvora kao rezultat nekog upita. Glavni problem su situacije kada nije moguće zaključiti koji je odgovor dovoljno dobar za prikaz korisniku kao i preveliki broj informacija koji se nalazi trenutno na web-u, što može rezultirati vrlo sporom obradom.

Logička dosljednost odnosi se na strojno rasuđivanje na semantičkom web-u koje se sastoji prvenstveno od deduktivnog zaključivanja.

Promjenjivost informacija odnosi se na brzinu izmjena informacija na stranicama. Pojavljuju se problemi predvidljivosti gdje se za dva identična upita mogu dobiti dva različita odgovora u različitim vremenskim intervalima zbog promjene određene informacije na stranici. Ontološka prilagodljivost nameće se kao jedno od rješenja za ovaj problem.

2.1 Tehnologije semantičkog web-a

Implementacija semantičkog web-a nije jednostavan proces što dokazuje i trenutno stanje razvoja koje je trenutno u stagnaciji. U [6] navode se neka od ograničenja implementacije:

- Tehnološka ograničenja - predstavljaju poteškoće u implementaciji RDF (Resource Description Framework) parsera. Trenutna rješenja su još previše spora, dok je s druge strane još uvijek teško omogućiti djelotvoran i pouzdan alat namijenjen ljudima koji se bave izradom mrežnih stranica.
- Poslovna ograničenja - ukoliko bi došlo do uvođenja semantičkog web-a cjelokupna web industrija trebala bi uvoditi radikalne promjene na poslovnoj i tehnološkoj razini što nije jednostavno ako se uzme za primjer veliko poduzeće kao što je Google.
- Socijalna ograničenja - način na koji bi semantički web trebao funkcionirati bitno se razlikuje od toga kako trenutni web funkcionira što bi moglo izazvati probleme privikavanja ljudi na novi način prezentiranja informacija kao i načina prikupljanja i procesiranja zatraženih informacija.

Osim navedenih, može se reći općenitih problema, potrebno je napomenuti da je također potrebno uvesti nove razine interoperabilnosti koje se temelje na standardima koji definiraju sintaksnu formu dokumenta. Isto tako, potrebno je uvođenje novog jezika prilagođenog semantičkom web-u koji bi služio opisivanju sadržaja mrežne stranice.
Nadovezujući se na te probleme, pri razvoju takvog jezika W3C se oslanja na već postojeće tehnologije i standarde [17], a također uvodi i neke nove jezike kao što je prikazano na Slici 2.3.

2.1.1 Jedinstveni identifikator resursa

Mogućnost da bilo tko može imenovati i opisivati stvari jedno je od glavnih načela semantičkog web-a. Međutim, kako bi bili u mogućnosti opisivati stvari potrebno je odrediti način referenciranja istih. U tu svrhu trenutni web i semantički web oslanjaju se na jedinstveni identifikator resursa, tzv. URI (Uniform Resource Identifiers).

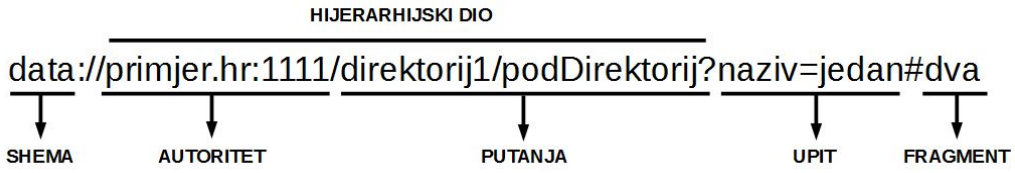
URI predstavlja skup znakova za identifikaciju nekog resursa koji se opisuje kao:

- Jedinstveni (eng. uniform) - definira jedinstvenu semantičku interpretaciju, drugim riječima, omogućava ponovnu upotrebu istih identifikatora u različitim kontekstima.
- Resurs (eng. resource) - specifikacija koja opisuje što sve može biti resurs. Drugim riječima, resurs može biti nekakav digitalni dokument, slika, mrežni servis i drugo, a isto tako resurs može biti i kombinacija drugih resursa.
- Identifikator (eng. identifier) - sadržava informacije koje su bitne kako bi se razlikovalo ono što se identificira od svih drugih stvari u području identifikacije.

URI se dijeli na apsolutni i relativni [18]. Apsolutni URI može se zapisati na dva načina:

- Dio koji navodi shemu, piše se bez početnog ”/” znaka (urn:isbn:1111111yx). Takav način zapisa ne zahtijeva daljnje parsiranje.
- Dio koji navodi shemu počinje znakom ”/” (hijerarhijski URI), s tim da se shema može i ne mora navesti. U slučaju da se shema navodi tada se govori o apsolutnom URI-u, dok se u suprotnom slučaju govori o relativnom URI-u.

Hijerarhijski zapis URI-ja može se dodatno parsirati, a kako izgleda takav zapis prikazano je na Slici 2.4 [19].



Slika 2.4: Primjer jedinstvenog identifikatora resursa

Shema - URI započinje nazivom sheme koja predstavlja specifikaciju restrikcija koje se odnose na sintaksu i semantiku identifikatora. Sastoji se od niza znakova koji započinje slovom, a ostatak se može sastojati također od slova ili od kombinacije slova, brojeva i određenih znakova.

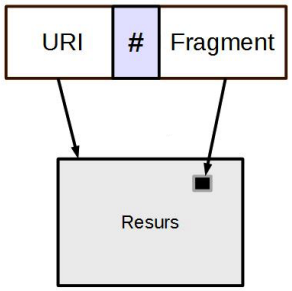
shema = ALPHA * (ALPHA / DIGIT / ‘+‘ / ‘-‘ / ‘.‘)

Postoje predefinirane URI sheme čije se korištenje može vidjeti na web-u (’esl’, ’http’, ’mid’, ’tag’ i drugi).

Autoritet - predstavlja DNS (Domain Name System) ili IP adresu. Također, može sadržavati broj pristupa (eng. port) ukoliko se razlikuje od predefinirane vrijednosti pristupa za URI. Sastoji se od dva dijela, gdje se prvi dio odnosi na autentifikaciju koja sadrži korisničko ime i zaporku i uređaja koji je definiran registriranim nazivom ili IP adresom (IPv4 ili IPv6).

authority = [userinfo "@"] host [":" port]

Putanja - Sadrži hijerarhijski organizirane podatke koji služe identifikaciji resursa, a parsira ih računalo kako bi se utvrdio pristup. Kraj putanje označava se znakovima ”?” ili ”#”.
Upit - Sadrži ne hijerarhijski organizirane podatke u svrhu identifikacije resursa definiranih shemom.
Fragment - Ako je fragment dostupan, služi kako bi se identificirao dio resursa specificiran apsolutnim URI-em. Fragment dio započinje znakom ”#”. Identifikacija dijela resursa fragmentom prikazana je na Slici 2.5 [18].



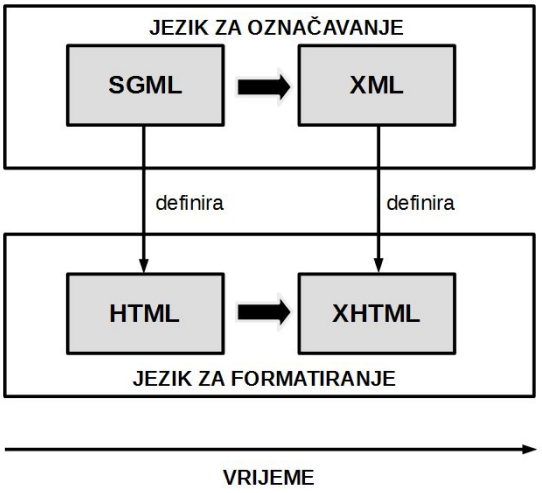
Slika 2.5: Identifikacija dijela resursa fragmentom

2.1.2 XML

XML predstavlja standardni jezik za razmjenu podataka te je ujedno i prva tehnologija na kojoj se temelji razvoj semantičkog web-a. Njegov razvoj započinje razvojem jednog drugog jezika 70-ih godina prošlog stoljeća, a radi se o SGML (Standardized Generalised Markup Language) jeziku [16], [20], [21]. Glavna svrha SGML jezika bila je stvoriti rječnik koji bi služio za označavanje dokumenata određenim strukturnim oznakama koje bi bile strojno čitljive.
Međutim, postoje određeni problemi vezani za SGML standard, a odnose se na [18]:

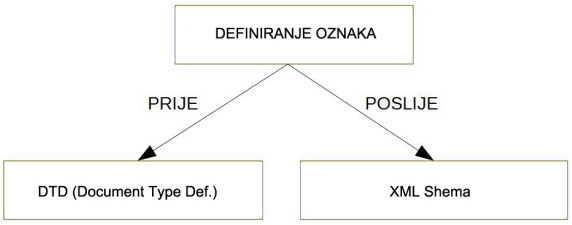
- Prevelik i širok standard za opis dokumenata, a samim time i prezahtjevan za implementaciju.
- Nedostatak alata otvorenog koda, odnosno, skupi alati za obradu. Isto vrijedi i za samu specifikaciju jezika.

Upravo zbog navedenih problema, 1996. godine pod nadzorom W3C organizacije dolazi do razvoja XML-a kao zamjene i pojednostavljene forme SGML-a prikazano na Slici 2.6 [18].



Slika 2.6: Evolucija jezika za označavanje

Glavni ciljevi pri razvoju XML-a bili su razviti jezik koji nije složen i lako se razumije, podržava skup različitih aplikacija, kompatibilan s SGML-om, čija izrada uzima minimalno vremena te za koji je moguće vrlo jednostavno napisati algoritme za parsiranje. Svaki XML dokument građen je kao stablo i sastoji se od niza čvorova i njihovih podčvorova sa jasno istaknutim korijenskim čvorom. Svaki čvor, odnosno element u stablu, ima svoj naziv, svojstvo i sadržaj. Ne postoji jednoznačno predstavljanje neke strukture XML-om, odnosno, ista stvar se može predstaviti na više načina (sintaksna nejednakost) [8], [22].



Slika 2.7: Definiranje oznaka

Struktura XML dokumenta definira se kao što je prikazano na Slici 2.7, DTD (Document Type Definition) ili XML shemom [18], [23], [24], [25].
DTD je razvijen kako bi se omogućila računalna analiza sintaksne točnosti XML dokumenata. Uz to, DTD također provjerava samo strukturu stabla, atribute, te da li se određeni element zapravo treba nalaziti u određenom XML dokumentu. S druge strane, neki od nedostataka DTD-a su izostanak imenika, provjera tipova podataka i sintaksna provjera vrijednosti elemenata.
XML Shema razvijena je kao nastavak na DTD i predstavlja shemu koja je proširenija i formalnija u smislu što omogućava definiranje više tipova podataka unutar oznaka i atributa, kao i točniju specifikaciju pravila koja se ne odnose samo na strukturu XML dokumenta već i na sadržane podatke. Na taj način moguće je definirati oznaku tako da se npr. provjerava je li spremljena cjelobrojna vrijednost veća ili manja od nule.

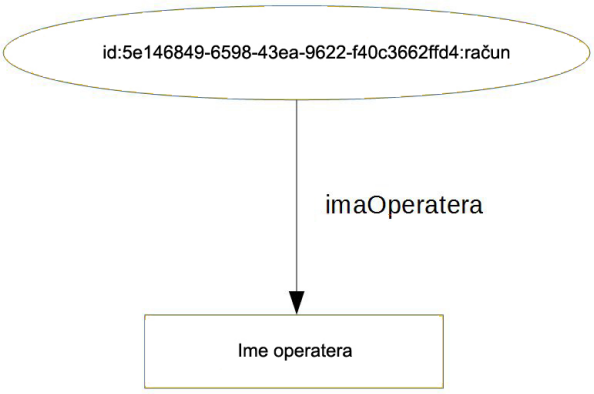
Od početka korištenja do danas XML je našao uporabu u različitim područjima kao što su [26]:

- Razvoj standardnih metoda razmjene i prikaza podataka između različitih uređaja.
- E-poslovanje - prijenos transakcija između dva poduzeća ili poduzeća i korisnika (banke).
- Razvoj mrežnih stranica - olakšava prilagodbu i izmjenu stranica.
- Automatizacija web procesa - s obzirom na tipski definirani sadržaj unutar dokumenta, olakšava pretraživanje istih, kao i dohvaćanje kvalitetnijih rezultata prilikom pretraživanja.

Međutim, prema Dekleru, XML ne zadovoljava sva tri pravila koja su navedena ranije u radu (str. 10), a nedostaci se najviše odnose na preveliku fokusiranost XML-a na sintaksu, odnosno, XML je zamišljen kako bi prenosio format dokumenta umjesto njegovog značenja [16].
RDF rješava neke nedostatke XML-a i o njemu će se govoriti u sljedećem odlomku.

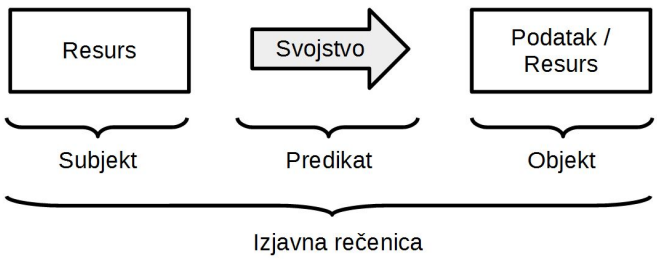
2.1.3 RDF i RDFS

RDF [11], [27], [28], [29], [30], [18] je standard kojeg razvija W3C kao osnovu semantičkog web-a, a nastao je kao rezultat istraživanja za jednostavnim jezikom za predstavljanje informacija o web resursima. RDF nije jezik već model podataka s mogućnošću predstavljanja podataka na eksplicitniji način za razliku od onoga što je moguće u XML-u, a sve u svrhu opisivanja strojno čitljive semantike.



Slika 2.8: Jednostavna RDF shema

Kao što je prikazano na Slici 2.8, RDF se sastoji od usmjerenih grafova koji se sastoje od čvorova povezanih s označenim lukovima. Osnovna konstrukcija RDF-a je izjavna rečenica koja se sastoji od tri dijela, tzv. tripleta, kojima se opisuju semantičke veze između resursa.
RDF izjavne rečenice predstavljaju jedan od bitnijih napredaka u odnosu na XML u smislu općenitosti, što omogućava opisivanje bilo koje domene. Također, RDF izjavne rečenice mogu se raspodijeliti po poslužiteljima kao što je to slučaj kod HTML sadržaja, te se mogu razmjenjivati između heterogenih aplikacija bez gubitka značenja. Na taj način, odnosno, kombinacijom distribuiranja, može se stvoriti velika virtualna baza znanja.



Slika 2.9: RDF izjavna rečenica [18]

Na Slici 2.9 prikazana je struktura izjavne rečenice RDF-a gdje se objekt / subjekt, objekt / vrijednost smješta na čvor usmjerenog grafa, a predikat / atribut na luk:

- Subjekt / objekt je stvar na koju se u rečenici ukazuje i predstavlja glavni gradivni blok u RDF-u (resurs). Resurs može biti bilo što, pa se tako resursi dijele na virtualne i fizičke, gdje virtualni predstavljaju resurse koji su dostupni na mreži npr. mrežna stranica, slika, servis i slično, dok fizički predstavljaju npr. knjigu, restoran, hranu i druge slične stvari.

www.feritos.hr/Dekan # PeroPeric

- Predikat / atribut je svojstvo koje je pripisano subjektu, odnosno, resursu. Predstavlja specijalni tip resursa koji ima svrhu opisivanja atributa nekog resursa kao i relacije između dva resursa. S obzirom da se i tu radi o jednoj vrsti resursa, osim što opisuje druge resurse, može opisati i sam sebe.

doktorat:nazivPoglavlja rdf:type rdf:Property doktorat:nazivPoglavlja rdfs:label ”Resource Description Framework”

- Objekt / vrijednost predstavlja vrijednost koja je dodijeljena subjektu. U sebi može sadržavati neki od dostupnih jezičnih identifikatora kao i dodatne identifikatore tipa podatka.

”11.00kn” ”Kneza Trpimira 2b” ”Restoran Restoranko” xsd:integer 777

Sva tri sastavna dijela RDF izjavne rečenice međusobno su povezana i predstavljaju resurs, a identificiraju se pomoću URI-ja. RDF, kao i XML, ima svoje nedostatke vezane za implementaciju semantičkog web-a. Jedan od problema je taj da je RDF model podataka koji je potrebno prikazati kao jezik za što u većini slučajeva služi XML prema specifikaciji W3C-a. Ta pretvorba sintaksno gledajući nije "elegantna". Jednostavnije pretvorbe uključuju N3 i podskup N-Triples, kao i TURTLE (Terse RDF Triple Language), ali niti u jednom slučaju nije moguće izvesti pretvorbu RDF-a na nezavisan način što je i cilj za uspješnu semantičku interoperabilnost [16].
Kao drugi problemi navode se [16], [18]:

- Problem višeznačnosti.
- Skrivena istoznačnost.
- Logička bezizražajnost.
- Slabo ontološko prilagođavanje.

RDF shema [27], [28], [29], [31], [18] razvijena je kao poboljšanje RDF-a kako bi se omogućila kvalitetnija podrška za definiranje i klasifikaciju ontologije (klase, svojstva, tipovi podataka, instance (individue)) pristupom koji je vrlo sličan objektnom programiranju. Drugim riječima, RDFS (Resource Description Framework Schema) proširuje RDF podršku za stvaranje jednostavnih izjavnih rečenica dodajući semantiku specifičnim resursima.

Klase predstavljaju način opisivanja koncepata kao kategorija resursa ovisno o domeni. Bilo koji resurs može se predstaviti klasom ili skupinom klasa. Bitno je napomenuti da su klase također resursi koji se predstavljaju upotrebom URI-ja. Svaka klasa može sadržavati pridružene resurse koji se nazivaju instancama.

Tablica 2.1: Predefinirane klase [18]

Klasa	Opis
rdfs:Resource	Korijenska klasa svih resursa
rdf:Property	Klasa koja predstavlja sva svojstva
rdfs:Class	Klasa koja predstavlja sve klase
rdfs:Literal	Klasa koja predstavlja skup vrijednosti za predstavljanje tipova podataka.
rdf:XMLLiteral	Podklasa klase Literal i predstavlja tipove podataka predefiniranih unutar RDF-a
rdfs:Datatype	Klasa za identifikaciju tipa podatka.

U Tablici 2.1 navedene su neke predefinirane klase od kojih se mogu izdvojiti rdfs:Resource, rdfs:Class i rdf:Property. Klasa rdfs:Resource predstavlja klasu svih korištenih resursa, drugim riječima, svaka instanca je zapravo instanca klase Resource. rdfs:Class predstavlja klasu za organizaciju ostalih resursa. rdf:Property predstavlja klasu kojom se opisuju dodijeljena svojstva, drugim riječima, svako svojstvo instanca je klase rdf:Property. Neka od bitnijih svojstava prikazani su u Tablici 2.2

Tablica 2.2: Temeljna svojstva RDFS-a [18]

Temeljna svojstva	Opis
rdf:type	Definira pripadnu klasu instance
rdfs:subClassOf	Predstavlja dodatan opis određene klase
rdfs:subPropertyOf	Predstavlja dodatan opis nekog svojstva
rdfs:range	Određuje vrijednost svojstva
rdfs:domain	Određuje pripadnost svojstva za određenu instancu

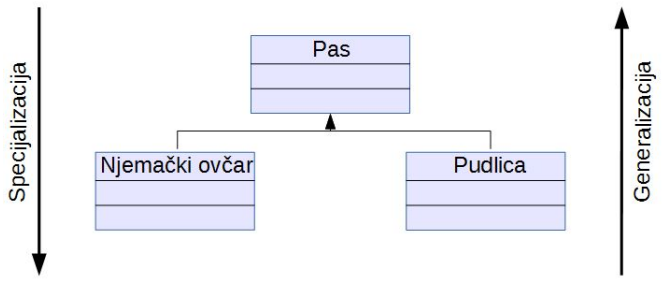
Referencirajući se na Tablicu 2.2, kako bi se odredilo da je neki resurs dio nekog drugog resursa koristi se svojstvo rdf:type kao što je to prikazano primjerom:

```
rdfs:Resource rdf:type rdfs:Class
rdfs:Class rdf:type rdfs:Class
rdf:Property rdf:type rdfs:Class
rdf:type rdf:type rdf:Property
```

Prikazanim primjerom definirano je da je resurs tip klase, klasa je tip klase, svojstvo je tip klase, ”tip” je tip svojstva. Na taj način vrlo je jednostavno definirati nekakvu proizvoljnu klasu i pripadajuću instancu:

```
:Sveuciliste rdf:type rdfs:Class
:FERIT rdf:type :Sveuciliste
:ETFOS rdf:type :Sveuciliste
```

Klase se mogu organizirati u hijerarhijsku strukturu koja može predstavljati specijalizaciju / generalizaciju određenog resursa kao što je to prikazano na Slici 2.10 [18].



Slika 2.10: Specijalizacija/Generalizacija hijerarhijske strukture

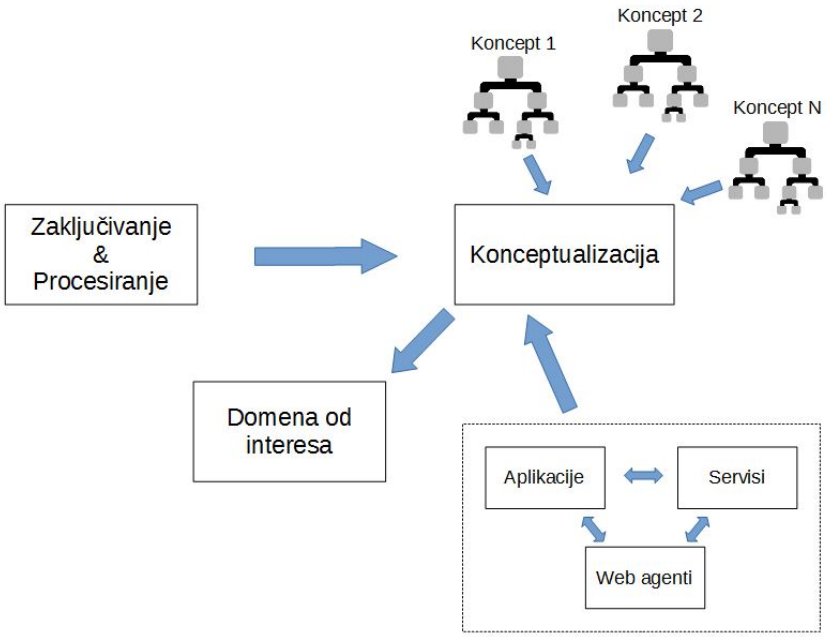
Svaka stvorena hijerarhija sastoji se od jednog korijenskog čvora (eng. superclass) i pripadajućih podčvorova, gdje jedna klasa može imati i više roditeljskih klasa. Svaka definirana instanca

jedne klase smatra se i instancom pripadajuće roditeljske klase.
Kao što je slučaj kod RDF-a i RDFS ima ograničenja ekspresivnosti zbog čega je bilo potrebno razviti napredniji jezik. Neka od tih ograničenja su [32]:

- Lokalni opseg svojstva (eng. local scope of properties) - ograničena upotreba svojstva, primjerice, nije moguće definirati svojstvo koje se odnosi samo na određene klase.
- Različitost klasa (eng. disjointness) - nije moguće definirati različitost klasa, primjerice, ako postoje klase *zvijezde* i *planeti* nije moguće definirati da su one različite.
- Booleove kombinacije (eng. boolean combinations) - nije moguće koristiti booleove kombinacije kako bi se za rezultat dobila nova klasa.
- Kardinalnost (eng. cardinality) - nije moguće ograničavanje vrijednosti svojstava, primjerice, avion ima točno dva glavna krila.

2.1.4 Ontologije

Uz navedene tehnologije XML, RDF i RDFS, potrebno je nešto više kako bi se riješili navedeni problemi vezani za te tehnologije i kako bi se značenje odredilo u potpunosti. Na taj način dolazi se do razvoja ontologija kao osnovne komponente semantičkog web-a [12], [22], [33]. Ontologije se prvi puta koriste i potječu iz grane filozofije koja se bavi proučavanjem postojanja gdje riječ ”ontos” znači biće, a ”logia” nauka [6], [34]. Odnose se na skup elemenata i njihovih međusobnih veza unutar nekog sustava [35]. Definicija ontologija prikazana je na Slici 2.11.
Premda ne postoji univerzalna definicija ontologije, u znanstvenoj literaturi može se naći više različitih definicija. Kao najčešća, uzima se [4], [8], [15], [36], [37] ”Ontologija je formalna eksplicitna specifikacija dijeljene konceptualizacije”, gdje se ”konceptualizacija” odnosi na apstraktni model, ”eksplicitno” na jasno definiranje elemenata, a ”formalna” na mogućnost računalne obrade ontologije [38], [39].
Razvoj ontologije nije trivijalan problem, stoga treba pripaziti na koji način se definiraju činjenice o nekoj domeni. Što je određena domena bolje definirana to je ontologija točnija. Samim time će i rezultati pretraživanja ontologije biti točniji s obzirom da rezultat pretrage ne ovisi o samim pojmovima unutar ontologije već i o ostalim informacijama koje su vezane za traženi pojam [40].
Ontologije mogu biti složene i jednostavne s razlikom u razini aksiomatizacije. To znači da se jednostavne ontologije sastoje od konačnog broja čvorova i međusobnih relacija te instanci dok složenije ontologije sadrže aksiome koji omogućavaju semantičku interpretaciju instanci i relacija ontologije.



Slika 2.11: Definicija ontologije

Ontologije su razvijene unutar zajednice za umjetnu inteligenciju u svrhu razvoja strojno obradive semantike stvaranjem pretpostavki o domeni, dijeljenja znanja razumijevanjem strukture informacija i podjele između softverskih agenata i ljudi te ponovne upotrebe podataka korištenjem u drugim sustavima slične ili iste domene [4], [6], [15], [41], [42].
Ponovna upotreba znanja [31] postiže se korištenjem eksplicitnih ontologija uzimajući u obzir konceptualizaciju (odabiranje odgovarajućeg referentnog modela, predstavljanje činjeničnog znanja definiranjem klasa i relacija unutar domene), rječnik (pokrivanje sintakse kao što je dodjeljivanje simbola klasama, serijalizacija konceptualizma u eksplicitnu reprezentaciju) i aksiomatizaciju (generiranje novih činjenica iz postojećeg znanja, validacija konzistencije znanja). Dijeljenje znanja između različitih domena moguće je uzimajući u obzir proširivost (mogućnost iskorištavanja i proširivanja već postojećih klasa), vidljivost (mogućnost razumijevanja od strane ljudi i strojeva) i zaključivost (osiguravanje logičkog zaključivanja te podržavanje izražajnosti i računalne složenosti).

Formalnost i klasifikacija ontologije

Premda u znanstvenoj literaturi postoji nekoliko klasifikacija ontologija, najčešća korištena je jednostavna klasifikacija tipova ontologija prema stupnju ovisnosti o zadatku [43], [44]:

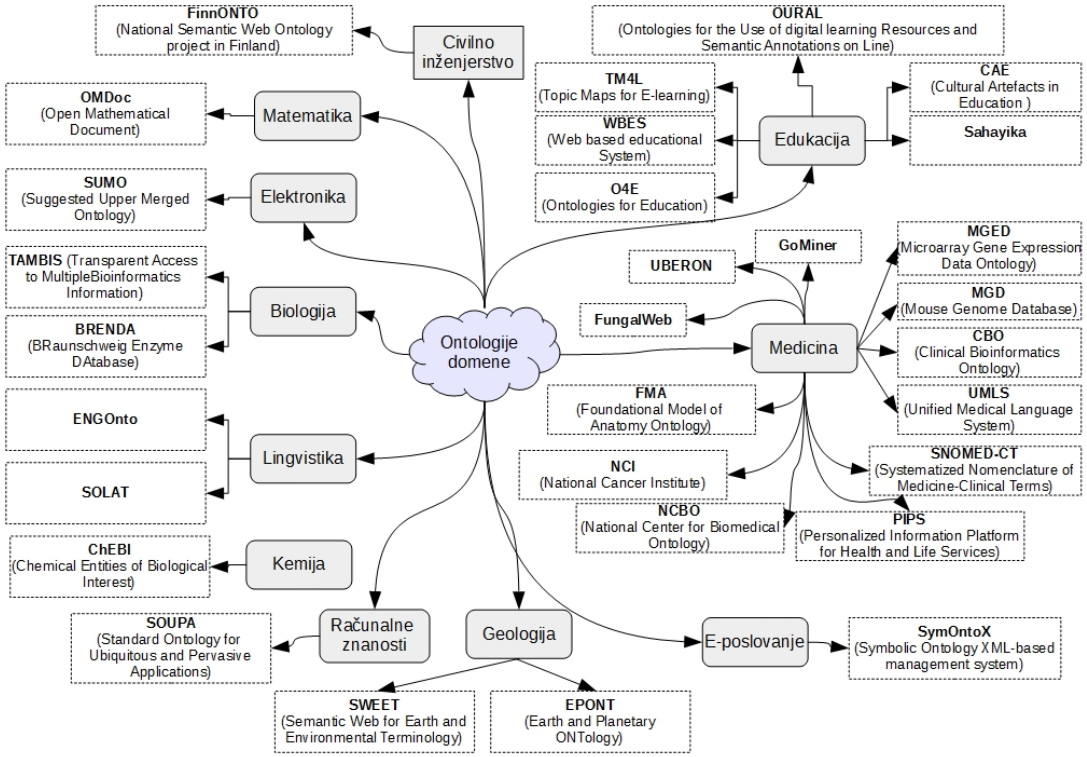
- Opća ontologija - predstavlja ontologiju koja za glavni cilj ima integraciju heterogenog znanja iz različitih izvora. Sastoji se od općih koncepata koji se ponavljaju u svim domenama, te je nezavisna o specifičnim domenama, a samim time i neupotrebljiva za specifičnost zadatka određene domene.

Tablica 2.3: Usporedba općih ontologija [45]

	Lokacija	Razvojni tim	Struktura	Jezik implementacije
BFO	http://www.ifomis.org/bfo	Smith, Grenon, Stenzhorn, Spear (IFOMIS)	36 klasa povezanih s "is_a" relacijama	OWL
Cyc	http://www.cyc.com/	Cycorp	Oko 300 000 koncepata, 3 000 000 pravila, 15 000 relacija	CycL, OWL
DOLCE	http://www.loa-cnr.it/DOLCE.html	Gruarino i ostali istraživači LOA-e	Oko 100 koncepata i 100 aksioma	Logika prvog reda, KIF, OWL
GFO	http://www.onto-med.de/ontologies/gfo.html	Onto-Med razvojna grupacija	79 klasa, 97 podklasa, 67 svojstava	Logika prvog reda, KIF, OWL
PROTON	http://proton.semanticweb.org/	Ontotext Lab, Sirma	300 koncepata i 100 svojstava	OWL Lite
Sowa	http://www.jfsowa.com/ontology/	Sowa	30 klasa, 30 aksioma	Modalni jezik prvog reda, KIF
SUMO	http://www.ontologyportal.org/	Niles, Pease i Menzel	20 000 termina i 60 000 aksioma	SUO-KIF, OWL

Postoji nekoliko razvijenih općih ontologija kao što je to prikazano u Tablici 2.3, a tu pripadaju BFO (Basic Formal Ontology), Cyc, DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering), GFO (General Formal Ontology), PROTON (PROTo ONtology), Sowa i SUMO (Sugested Upper Merged Ontology).

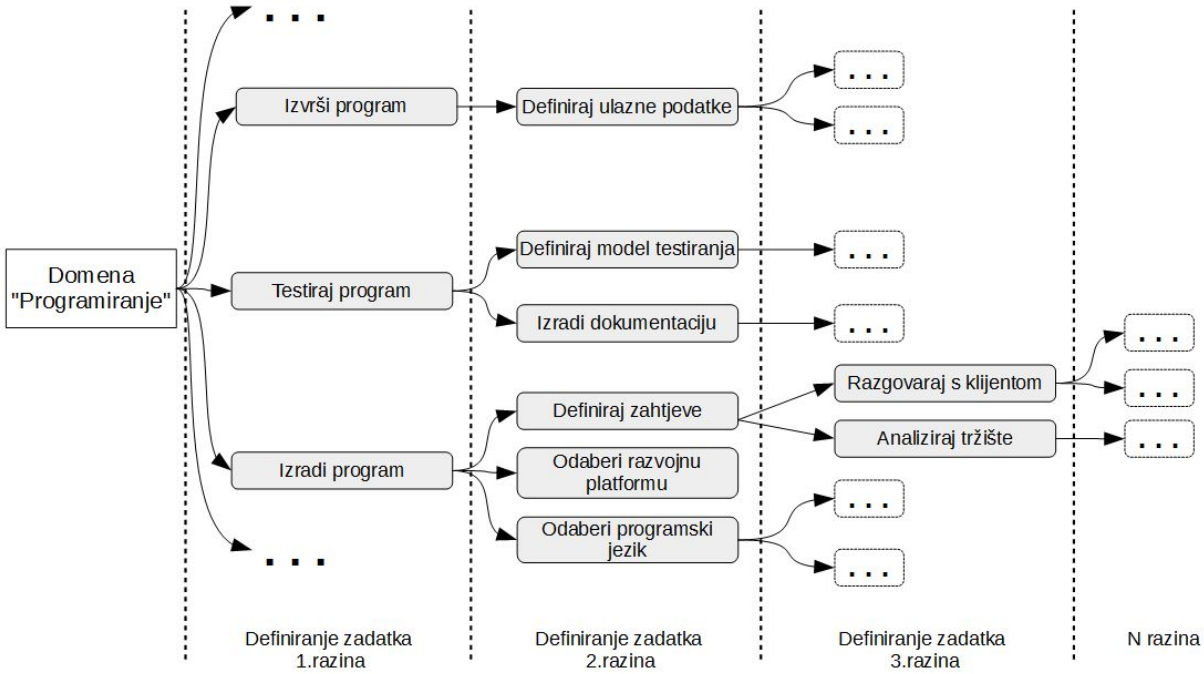
- Ontologija domene - predstavlja ontologiju koja prikazuje konceptualizaciju specifičnu za domenu koju obuhvaća. Drugim riječima, ontologije domene su formalni opis nekog modela podatka sastavljenog od klasa, instanci i njihovih međusobnih relacija koji opisuju određeno područje primjene. Razlikuju se od općih ontologija u tome što klase ontologije domene predstavljaju specijalizaciju klasa u općim ontologijama. Za primjer se može uzeti ontologija koja opisuje neko sveučilište gdje bi klase i svojstva studenata pripali općoj ontologiji, dok bi atributi koji opisuju pripadnost studenata određenoj ustanovi kao što je fakultet pripali ontologiji domene. Zbog svoje preciznosti, svaka promjena na ontologiji domene utjecat će na svaku komponentu povezanu s traženim podacima. Ontologije kao način reprezentacije znanja i sustavi s kojima su povezane koriste se u različitim domenama što je prikazano na Slici 2.12.



Slika 2.12: Područja primjene ontologija

- Ontologije zadatka - predstavljaju ontologije koje nastaju specijaliziranjem koncepata opisivanjem skupa definicija ili koncepata koji se tiču zadatka iz općih ontologija kako bi pružile predloške potrebne za stvaranje modela procesa, odnosno upotrebljivost zadatka određenog područja. Model procesa služi za rješavanje specifičnog problema neke domene primjene. Primjer hijerarhijske strukture modela zadatka prikazan je na Slici 2.13. Postoji nekoliko kriterija za izradu modela, a uključuju jednostavnost (potrebno je da svaki zadatak bude jasno definiran tako da ga bilo tko može razumjeti), koherentnost (svaki zadatak treba odgovarati svojoj definiciji) i proširivost (sve postojeće definicije zadataka trebale bi se moći proširiti novim terminima i izrazima na način da se ne uzrokuje kontradikcija unutar same definicije).

  Ontologije domene mogu biti povezane s ontogijom zadatka na način da ontologija zadatka sadrži znanje koje je potrebno kako bi se postigao određeni zadatak, dok ontologije domene opisuju znanje o tome gdje bi se određeni zadatak mogao primijeniti.



Slika 2.13: Hijerarhijska struktura modela zadatka

- Aplikacijske ontologije - predstavljaju ontologije koje su ovisne o aplikacijama koje ih upotrebljavaju, a sastoje se od specijaliziranih koncepata ontologija domene i ontologija zadatka. Takve ontologije nisu ponovno upotrebljive upravo zbog njihove specifičnosti (npr. ontologija za rješavanje kvara visokotlačne pumpe na automobilu).

Definiranje koncepata i svojstava može se izraziti u različitoj formi, odnosno, što je struktura same ontologije složenija raste i stupanj formalnosti koji se dijeli na:

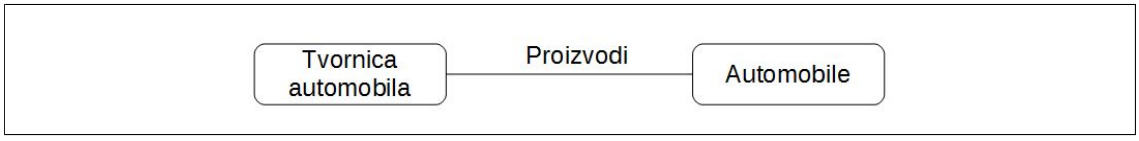
- Strogo neformalno - odnosi se na definicije izražene u prirodnom jeziku.

Automobil je proizvod tvornice automobila

- Polu neformalno - odnosi se na definicije izražene u prirodnom jeziku s određenim ograničenjima.

Tvornica automobila PROIZVODI automobile

- Polu fomalno - odnosi se na definicije izražene u okvirno temeljenom jeziku.



- Strogo formalno - ako su definicije izražene u logički temeljenom jeziku (npr. OWL).

```
<a:owl_objectproperty rdf:about="proizvodi" rdfs:label="proizvodi">
        <rdfs:range rdf:resource="Tvornica automobila"/>
        <rdfs:domain rdf:resource="Automobile"/>
</a:owl_objectproperty>
```

Dijelovi ontologija

U literaturi se mogu pronaći različite podjele komponenti ontologije premda sve vode istom cilju [15]. Stoga se može reći da su sljedeći elementi potrebni kako bi se ontologija opisala kroz strukturu i pravila, a da u isto vrijeme bude formalizirana u jeziku koji je strojno čitljiv [15], [43]:

- Na razini konceptualizacije modelira se znanje pomoću klasa, instanci, relacija, funkcija i aksioma.
- Na razini specifikacije formalno se specificira konceptualizacija korištenjem nekog formalnog jezika s kojim se ontologije konstruiraju, zapisuje znanje o specifičnosti domene te pravila o donošenju odluka za podršku i obradu zapisanog znanja.

Ontologija je skup $O := \{C, R, H^C, H^R, I_C, I_R, A^O\}$, gdje je

- C - skup koncepata.
- R - skup relacija između koncepata. $R_i \in R$, $R_i \rightarrow C$ x C.
- H^K - hijerarhija klasa u formi relacije $H^C \subseteq CxC$, gdje $H^C(C_1, C_2)$ predstavlja relaciju roditelj $\leftrightarrow$ dijete između koncepata C_1 i C_2.
- H^R - hijerarhiju relacija u formi relacije $H^R \subseteq R$ x R, gdje $H^R(R_1, R_2)$ znači da je R_1 podrelacija R_2.
- I_C i I_R - predstavlja dva različita skupa koncepata i relacija.
- A^O - skup aksioma.

Klase - glavni entiteti u ontologiji koji predstavljaju skupinu instanci koji su organizirani u taksonomiju. Prikaz taksonomije ovisi o domeni koja se opisuje, a najčešće je hijerarhijski strukturirana. Klase mogu imati i podklase kao specijalizaciju roditeljske klase, što je prikazano na Slici 2.10.
Instance - predstavljaju jednostavne objekte kao osnovnu jedinicu ontologije. Mogu modelirati konkretne ili više apstraktne modele neke stvari unutar domene (npr. računalo kao konkretni model ili članak o računalima kao apstraktniji model).
Aksiomi - tvrdnje koje uključuju pravila, a koje se koriste za provjeru konzistentnosti ontologije i prikaz znanja koje se ne može formalno opisati pomoću već navedenih komponenti ontologije. Aksiomi se dodaju kao prefiks u obliku implikacije, logičke jednakosti, konjukcije, disjunkcije,

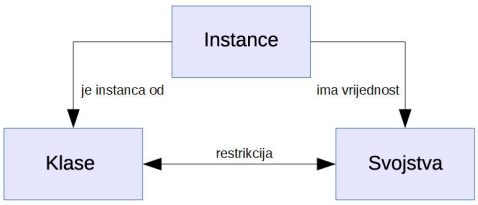
negacije, egzistencijalnog i univerzalnog kvantifikatora. Za primjer se može navesti definicija jednakosti jedne ili više klasa:

$$axiom ::=' EquivalentClasses('klasaID\{klasaID\}')'$$

Kao što je već i navedeno ranije, aksiomatizacija je jedan od tri bitna čimbenika prilikom razvoja ontologije koja se odnosi na stvaranje novog znanja.
Osim navedenih dijelova ontologije, navode se još i predlošci koji služe za prikaz, modeliranje i ugradnju znanja ontologije, a najčešće se koriste dvije osnovne skupine:

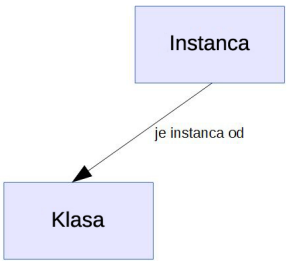
- Logika prvog reda.
- Deskriptivna logika.

Relacije - predstavljaju tip veze unutar hijerarhije klasa. Relacija R definirana je kao podskup kartezijevog produkta n skupova koncepata $C_1, C_2, ..., C_n$ i zapisuje se u obliku $R(C_1, C2, ..., C_n)$. Uz relacije, ontologija može sadržavati i informacije (uvjete) kao što su svojstva, ograničenja i isključive izraze koji trebaju biti zadovoljeni kako bi klase međusobno bile u relaciji [4], [11], [43], [18]. Gradivni elementi ontologije koji su navedeni ranije u disertaciji prikazani su na Slici 2.14.



Slika 2.14: Gradivni elementi ontologije

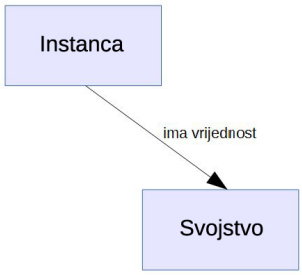
Relacija između instance i klase - S obzirom da skupina instanci definira klasu, relaciju pripadnosti *("je instanca od")* potrebno je eksplicitno navesti kako bi se pravilno obavila identifikacija članova pojedine klase [18]. Primjer relacije instanca - klasa prikazan je na Slici 2.15.



Slika 2.15: Relacija instanca - klasa

Relacija između instance i svojstva - svakoj instanci moguće je dodati neku vrijednost opisanu svojstvom. Relacija *("ima vrijednost")* omogućava specificiranja vrijednosti određenog

atributa instance. Za primjer se može uzeti da instanca ima vrijednost 4.500.000 kao broj stanovništva za svojstvo populacija Hrvatske. Također, tako dodijeljene relacije kasnije mogu omogućiti točnije definiranje anonimnih klasa koje će biti detaljnije opisane u nastavku ove disertacije [18]. Primjer relacije instanca - svojstvo prikazan je na Slici 2.16.



Slika 2.16: Relacija instanca - svojstvo

Relacije između klase i svojstva - Klase kao skupovi instanci mogu imati dodane restrikcije sukladno dodijeljenim svojstvima. Takav tip relacije omogućava promjenu sadržaja klase popunjavanjem dodatnim instancama definiranjem uvjeta pripadnosti ili čak promjenu hijerarhije definiranjem dodatne pripadnosti određene klase drugoj klasi [18]. Primjer relacije klasa - svojstvo prikazan je na Slici 2.17.



Slika 2.17: Relacija klasa - svojstvo

Logika

Logika [46], [47], [48], [49] je grčka riječ koja predstavlja znanost o zakonima saznavanja istine i metodama spoznaje (*logos* = riječ, smisao, govor). Pripada grani filozofije te je uz metodologiju jedna od osnovnih filozofskih disciplina. Sami počeci logike u Europi vežu se za Grčku i grčkog filozofa Aristotela kao njezinog začetnika, koji raspravlja o svim osnovnim pitanjima logike. Točnije govoreći, istraživanja Aristotela i ostalih filozofa najviše su se odnosila na deduktivnu logiku koja se, premda nije doživjela bitne promjene, dugo zadržala uz povremena dopunjavanja i sistematizaciju tijekom stoljeća. Početkom 19. stoljeća dolazi do razvoja induktivne logike, dok u drugoj polovici 19. stoljeća i početkom 20. stoljeća dolazi do razvoja simboličke logike (Matematička logika) koja predstavlja razvijeniji oblik deduktivne logike.
Logiku se može podijeliti na formalnu i neformalnu, gdje se u formalnoj razlikuju tradicionalna logika i moderniji oblici simboličke, matematičke logike. Matematička logika predstavlja granu matematike koja se bavi simboličkim prikazom tradicionalne logike na način da se izbjegne dvosmislenost točnim definiranjem pojmova što nije slučaj u tradicionalnoj logici. Za razvoj

matematičke logike najviše je zaslužan George Boole. On je u svojoj teoriji predstavio ideje da se prilikom rada s iskazima koriste oznake kao i da zakoni mišljenja imaju puno sličnosti sa zakonima aritmetike. Korištenjem unije, presjeka i komplementa zapisao je i dokazao osnovne zakone iskaznog računa, a oni su poznati pod nazivom aksiomi Booleove algebre. Danas je matematička logika osnova računarstva, te se na njoj temelje logički dijelovi procesora. Bilo koji formalni sustav smatra se logičkim ako u sebi sadrži dobro definiranu sintaksu (sintaksno prihvatljive definicije objekata jezika), dobro definiranu semantiku (dodjeljivanje značenja) i dobro definiranu teoriju dokaza (ovisnost o pravilima).

Propozicijska logika

Propozicijska logika [48], [50], [49] predstavlja jednu od najapstraktnijih i najjednostavnijih logika poznate još i kao izjavna logika. Može se reći da se propozicijska logika primarno odnosi na studiju o logičkim operatorima i veznicima kao načinu sastavljanja više manjih propozicija ili izmjeni cijelih propozicija u svrhu postizanja složenijih propozicija.
Propozicija predstavlja izjavu koja može biti istinita ili lažna, dok kod složenijih izjava, istinitost u potpunosti ovisi o istinitosti izjava za koje se također pretpostavlja da su istinite ili lažne. Za primjer se mogu uzeti sljedeće izjave:

Hrvatska je članica Europske unije.
Zagreb je glavni grad Hrvatske.
Godina ima 359 dana.

Također, kao što je i ranije navedeno, izjava može biti i složena što znači da sadržava jednu ili više izjava povezanih logičkim operatorima:

Zemlja se vrti oko Sunca ili se Sunce vrti oko Zemlje.

Iako se navedena rečenica odnosi na istinitu izjavu, može se primijetiti da se sastoji od dvije manje izjave ”Zemlja se vrti oko Sunca” i ”Sunce se vrti oko Zemlje” povezane ILI veznikom, gdje je prva izjava istinita, a druga lažna. Umjesto pisanja čitavih izjava, mogu se koristiti izjavne varijable koje predstavljaju izjavnu rečenicu. Npr. *p = ”Zemlja je okrugla”*.

Sintaksa propozicijske logike

Negacija ($\sim, \neg$)
Bilo koja izjava može kao prefiks imati oznaku negacije kako bi tvorila drugu izjavu koja predstavlja negaciju izvorne izjave.

$$I(\neg P) = \begin{cases} 1 & ako \quad I(P) = 0 \\ 0 & ako \quad I(P) = 1 \end{cases}$$

Izjava $\neg P$ je istinita ako i samo ako je P lažna.

Konjukcija $(\wedge, \&, \cdot)$

Mogu se kombinirati bilo koje dvije izjave kako bi tvorile treću koja se naziva konjukcijom izvornih izjava.

$$I(P \wedge Q) = \begin{cases} 1 & ako \quad I(P) = 1 \quad i \quad I(Q) = 1 \\ 0 & ako \quad I(P) = 0 \quad ili \quad I(Q) = 0 \end{cases}$$

Izjava $(P \wedge Q)$ je istinita ako su i P i Q izjave istinite.

Disjunkcija (uključiva) $(\vee, |, +)$

Mogu se kombinirati bilo koje dvije izjave kako bi tvorile treću koja se naziva disjunkcijom izvornih izjava. Moguće je definirati uključivu i isključivu disjunkciju

$$I(P \vee Q) = \begin{cases} 1 & ako \quad I(P) = 1 \quad ili \quad I(Q) = 1 \\ 0 & ako \quad I(P) = 0 \quad i \quad I(Q) = 0 \end{cases}$$

Izjava $(P \vee Q)$ je istinita ako je P izjava istinita ili je Q izjava istinita ili su obje istinite.

Disjunkcija (isključiva)

$$I(P \bigoplus Q) = \begin{cases} 1 & ako \quad (I(P) = 1 \quad i \quad I(Q) = 0) \quad ili \quad (I(P) = 0 \quad i \quad I(Q) = 1) \\ 0 & ako \quad (I(P) = 1 \quad i \quad I(Q) = 1) \quad ili \quad (I(P) = 0 \quad i \quad I(Q) = 0) \end{cases}$$

Izjava $(P \bigoplus Q)$ je istinita ako izjava P i izjava Q imaju različite vrijednosti.

Implikacija $(\Rightarrow, \supset, \rightarrow)$

Mogu se kombinirati bilo koje dvije izjave kako bi tvorile treću koja se naziva implikacijom izvornih izjava. Drugim riječima, implikacijom se govori da ako je izjava P istinita, tada je i izjava Q istinita.

$$I(P \Rightarrow Q) = \begin{cases} 0 & ako \quad I(P) = 1 \quad i \quad I(Q) = 0 \\ 1 & u \quad suprotnom \end{cases}$$

Izjava $(P \Rightarrow Q)$ je istinita u svim slučajevima osim ako je P izjava istinita, a Q izjava lažna.

Bikondicional $(\Leftrightarrow)$

Bikondicijskim uvjetom govori se da je izjava P istinita ako i samo ako je izjava Q istinita.

$$I(P \Leftrightarrow Q) = \begin{cases} 1 & ako \quad (I(P) = 1 \quad i \quad I(Q) = 1) \quad ili \quad (I(P) = 0 \quad i \quad I(Q) = 0) \\ 0 & u \quad suprotnom \end{cases}$$

Izjava $(P \Leftrightarrow Q)$ je istinita u svim slučajevima kada su izjave P i Q ili istinite ili lažne. U tom slučaju, za izjave P i Q se kaže da su logički jednake.

Tablica 2.4: Tablica istinitosti za veznike propozicijske logike

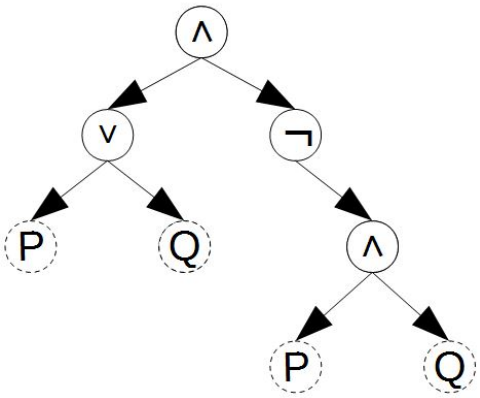
P	**Q**	$\neg$ **(samo P)**	$\wedge$	$\vee$	**+**	$\Rightarrow$	$\Leftrightarrow$
F	F	T	F	F	F	T	T
F	T	T	F	T	T	T	F
T	F	F	F	T	T	F	F
T	T	F	T	T	F	T	T

Koristeći tablicu istinitosti (Tablica 2.4) i izjavne varijable, može se navesti nekoliko primjera izjavnih rečenica:

P = Oblačno je.
Q = Pada kiša.
$(P \wedge Q)$ = Oblačno je i pada kiša.
$(P \vee Q) \wedge \neg(P \wedge Q)$ = Ili je oblačno ili pada kiša.
$(\neg P) \wedge (\neg Q)$ = Niti je oblačno niti pada kiša.

Svaka izjava ili kombinacija izjava može se evaluirati na način da se za izjavu kaže da li je istinita ili lažna. Neka funkcija $eval : I \rightarrow \{1,0\}$ određuje jednu moguću evaluaciju istinitosti cijele izjave ili elemenata izjave, što se još naziva i interpretacija izjave.
Za izjavu $(P \vee Q) \wedge \neg(P \wedge Q)$, postupak interpretacije je sljedeći:
$eval(P) = 0$;
$eval(Q) = 1$;
$= eval((eval(P) \vee eval(Q) \wedge \neg(eval(P) \wedge eval(Q))$
$= (0 \vee 1) \wedge \neg(0 \wedge 1)$
$= 1 \wedge \neg 0$
$= 1 \wedge 1$
$= 1$
Kako je izjava istinita za navedenu interpretaciju, može se zaključiti da interpretacija zadovoljava izjavu.

Također, bilo koju izjavnu rečenicu moguće je prikazati kao stablo gdje krajnji čvorovi predstavljaju varijable, odnosno izjave, dok ostali čvorovi predstavljaju logičke veznike. Stablo za izjavnu rečenicu $(P \vee Q) \wedge \neg(P \wedge Q)$ prikazano je na Slici 2.18.



Slika 2.18: Prikaz izjavne rečenice stablom

Međutim, veliki nedostatak propozicijske logike leži u vrlo ograničenoj ekspresivnosti. To se najviše odnosi na generalizaciju i ponavljajuće uzorke u izjavnim rečenicama. Za primjer se može uzeti rečenica *"Svi ljudi su sisavci"*. Navedena izjavna rečenica ne bi bila dovoljna u propozicijskoj logici, već bi se za svakog čovjeka morala napisati jedna izjavna rečenica. Kako bi se to izbjeglo, i kako bi to bilo moguće koristi se logika prvog reda.

Logika prvog reda

Logika prvog reda [51], [52], [53], [54], [49], [55] je predikatna logika izvorno razvijena kao teorija 1930-e godine isključivo za matematičke potrebe te predstavlja mehanizam zaključivanja opisan logičkim simbolima. Pojam predikatna logika odnosi se na logiku koja se prvenstveno bavi predikatima. Logika prvog reda samo dodatno određuje tip predikatne logike koji ovisi o skupu nelogičnih simbola. Drugim riječima, osim logike prvog reda postoji i logika drugog, trećeg, n-tog reda ali se mogu razmatrati kao cjelina zbog razlika koje su tehničke prirode.
Premda su neki od nedostataka propozicijske logike navedeni ranije u radu, na sljedećem primjeru je prikazan glavni razlog upotrebe logike prvog reda umjesto propozicijske logike.

"Ako su sve ceste dobre za voziti (P), i ako je makadam cesta (Q), tada su svi makadami dobri za voziti (R)."

Ako se navedenu izjavnu rečenicu rastavi na jednostavne izjave (propozicije) dobije se:

- P = (C $\rightarrow$ V) , gdje za C(x), x $\rightarrow$ cesta
- Q = (M $\rightarrow$ C) , gdje za M(x), x $\rightarrow$ makadam
- R = (M $\rightarrow$ V) , gdje za V(x), x $\rightarrow$ dobro za voziti

Prema tome, može se reći da je izraz $((P \wedge Q) \rightarrow R)$ ispravan.
Međutim, ako se želi reći da su samo neke makadamske ceste dobre za voziti dok ostale nisu, to bi bilo vrlo teško izraziti koristeći propozicijsku logiku. Koristeći logiku prvog reda, izjavna rečenica koja bi opisala takvu situaciju izgleda:

$$(\forall x)(C(x) \rightarrow V(x)), (\exists x)(M(x) \wedge (C(x)) \models (\exists x)M(x))$$

Detaljan opis korištenih simbola i strukture navedene rečenice slijedi u nastavku.
Logika prvog reda predstavlja proširenje propozicijske logike omogućavajući definiranje tvrdnji o objektima (ljudi, zemlja, računala, doktorat), svojstvima objekata (roditelj od, podskup od) i veza između objekata (je dio ustanove, je veći od, sadrži, dolazi poslije) što je čini znatno ekspresivnijim oblikom logike [51, 52, 53, 54]. Usporedba propozicijske logike i logike prvog reda prikazana je u Tablici 2.5. Bitno je napomenuti da razlike navedene u Tablici 2.5 ne predstavljaju sve razlike između te dvije logike već samo one bitne za opis u sklopu ove disertacije.

Tablica 2.5: Usporedba propozicijske logike i logike prvog reda

	Propozicijska logika	**Logika prvog reda**
Ontologija	činjenice (P, Q)	objekti, svojstva, relacije
Sintaksa	izjave, logički veznici	varijable i kvantifikacija (izjave imaju strukturu)
Semantika	tablica istinitosti	interpretacija

Sintaksa logike prvog reda dijeli se na skup logičkih i skup ne logičkih simbola.
Logički simboli

- Pomoćni simboli (interpunkcija) - ” (”, ”) ”, ” . ”, ” , ”. Primjer upotrebe ovih simbola bio bi promjena prioriteta veznika u izjavnoj rečenici.
- Veznici - koriste se za sastavljanje izjava, a sastoje se od istih znakova kao i u propozicijskoj logici. Konjukcija ($\wedge$), disjunkcija ($\vee$), negacija ($\neg$), implikacija ($\Rightarrow$), bikondicional ($\Leftrightarrow$). Tu još pripadaju ($\forall$) i ($\exists$), koji predstavljaju univerzalni, odnosno, egzistencijalni kvantifikator. Na primjeru:

  $\forall X$ p(X) čita se ”za sve X, p(X) je istina”,
  $\exists X$ p(X) čita se ”postoji X takav da je p(X) je istina”

  Također, egzistencijalni kvantifikator omogućava stvaranje izjave o nekom objektu bez eksplicitnog imenovanja istog.
- Varijable - proizvoljan konačan skup simbola $\{x_1, x_2, ..., x_n\}$

Nelogički simboli

- Predikatni simboli - koriste se za označavanje svojstava i relacija između objekata. Može primati i konačan broj argumenata. S obzirom na broj argumenata razlikuju se predikatni simboli bez argumenata koji se promatraju kao izjava u propozicijskoj logici, predikatni simboli s jednim argumentom koji se promatraju kao svojstvo objekta i predikatni simboli s više argumenata koji se promatraju kao označavanje relacija između objekata. Na primjeru:

Kvalitetan(Švicarski_nož)
Pjeva(Oliver,Cesarica)

- Funkcijski simboli - kao i kod predikatnih simbola, mogu imati ulazne argumente. Glavni zadatak je uparivanje ulaza u objekte. Za primjer se može uzeti izraz "MentorOd(Tomislav)", što bi za rezultat dalo objekt koji predstavlja mentora. Funkcijski simboli bez argumenata predstavljaju konstante koje se tumače kao objekti.

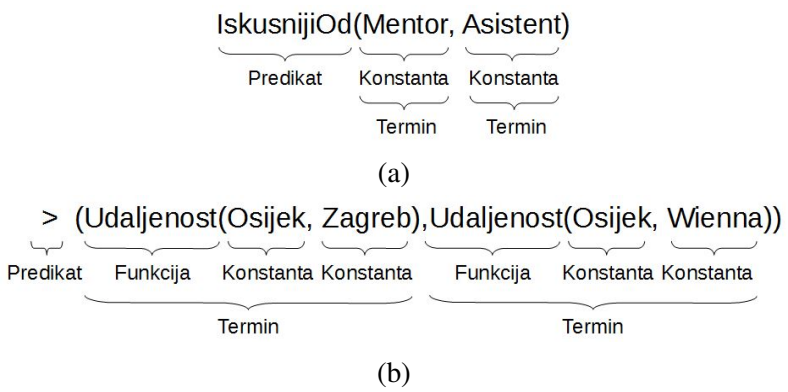
Kako bi se definirala izjavna rečenica u logici prvog reda, prvo je potrebno definirati osnovnu komponentu logike prvog reda, *termin*. Ako se kažemo da je C skup konstanti, F skup funkcijskih simbola i V skup varijabli, tada se skup termina T definira kao:

- Konstante ($C \subseteq T$) - predstavlja najjednostavniji tip termina i predstavljaju bilo koji tip objekta u skupu T.
- Varijable ($V \subseteq T$) - označavaju bilo koju vrijednost u skupu T.
- Funkcije (ako $f \in F$ i $t_1, t_2, ..., t_n \in T$, tada $f(t_1, t_2, ..., t_n) \in T$, gdje $n = ar(f)$). Ideja funkcije je ista kao i kod programiranja, a ta je da za određene ulazne argumente vrati neku vrijednost.

$funkcija(termin_1, termin_2, ..., termin_n)$

S obzirom na navedeno, definicija jednostavne rečenice prikazane na Slici 2.19 u logici prvog reda je kako slijedi:

- Ako su $\{t_1, t_2, ..., t_n\}$ termini i ako je P predikat s n ulaznih argumenata, tada je $P(t_1, t_2, ..., t_n)$ jednostavna rečenica.
- Ako su t_1 i t_2 termini, tada je $(t_1 = t_2)$ jednostavna rečenica.

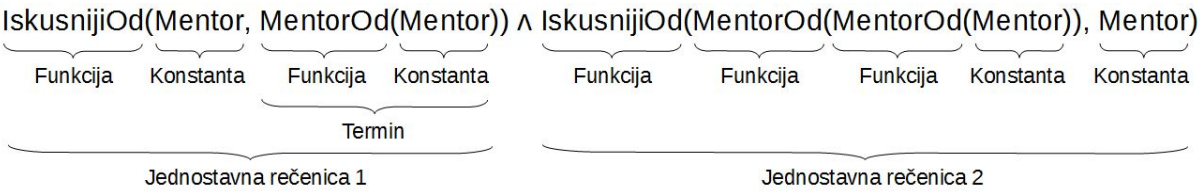


Slika 2.19: Primjer jednostavnih rečenica

Složenija rečenica u logici prvog reda prikazana na Slici 2.20 definira se koristeći veznike kao u propozicijskoj logici kako slijedi:

- Ako su I_1 i I_2 rečenice, tada su rečenice i sljedeći izrazi:

$$(\neg I_1),\ (I_1 \wedge I_2),\ (I_1 \vee I_2),\ (I_1 \Rightarrow I_2),\ (I_1 \Leftrightarrow I_2),\ (\forall x) I_1,\ (\exists x) I_1$$



Slika 2.20: Složena rečenica

Modeli logike prvog reda

Prvi korak u određivanju semantike neke logike je definiranje modela u kojima se interpretiraju definirane rečenice. Rečenice su istinite ili lažne uzimajući u obzir model koji je za logiku prvog reda uređena dvojka $\langle$ D, I $\rangle$ gdje:

- D predstavlja domenu, odnosno, konačan ili neodređeni skup objekata.
- I predstavlja funkciju interpretacije kojom se dodjeljuje značenje, odnosno, interpretacija svakoj konstanti (c), funkciji (f) ili predikatnom simbolu (p) kako slijedi:
  - Ako je c konstantni simbol, tada je $I(c)$ objekt u domeni D.
  - Ako je f funkcijski simbol, tada je $I(f) : D^n \rightarrow D$ interpretacija funkcije koja mapira n objekata u domenu D.
  - Ako je p predikatni simbol s određenim brojem argumenata, tada je $I(p) \subseteq D^n$. U slučaju predikatnog simbola bez argumenata, tada je $I(p) = \{true, false\}$.

Kao primjer se može uzeti rečenica ” IskusnijiOd(MENTOR, MentorOd(MENTOR))” kojom se želi reći da je mentor iskusniji od svog asistenta. Za navedenu rečenicu model M bi mogao izgledati:

$$D = \{MENTOR, ASISTENT, NULL\}$$
$$I(\text{Mentor}) = \text{MENTOR}$$
$$I(\text{IskusnijiOd}) = \{\langle MENTOR, ASISTENT\rangle\}$$

Kako bi se napravila interpretacija funkcije ”MentorOd”, dodjeljuju se vrijednosti za svaki ulaz:

$$I(\text{MentorOd})(\text{MENTOR}) = \text{ASISTENT}$$
$$I(\text{MentorOd})(\text{ASISTENT}) = \text{NULL}$$
$$I(\text{MentorOd})(\text{NULL}) = \text{NULL}$$

Deskriptivna logika

Opisna logika ili deskriptivna logika [51], [52], [56], [57], [58], [35], [59], [60], [61], [62] predstavlja formalni jezik za reprezentaciju znanja neke domene primjene na strukturiran i formalno razumljiv način utemeljen na matematičkoj logici koja je jedna od glavnih pristupa reprezentacije znanja uz neuronske mreže, semantičke mreže, okvire i pravila. Što se tiče ekspresivnosti, nalazi se između propozicijske logike i logike prvog reda koje su opisane ranije u disertaciji. Nastaje kao niz pokušaja i promašaja na području razvoja formalne ontologije u informacijskoj tehnologiji. Sve počinje 50-ih godina prošlog stoljeća rastom složenosti baza podataka, gdje dolazi do određenih problema ontološke naravi. Time sve više raste zanimanje u domeni umjetne inteligencije za razvojem ontologija.
Razvoj se 70-ih i 80-ih godina usmjerava na ekspertne sustave od kojih su najpoznatiji DENDRAL [16], [29] u području organske kemije i MYCIN [16] u području medicinske dijagnostike. Međutim, nastao je problem umjetnog razumijevanja prirodnih jezika kao što je engleski jezik. Kao rješenje tog problema nastaje Cyc projekt kao najambiciozniji projekt koji je temeljen na predstavljanju znanja jezicima temeljenim na formalnoj logici. Semantičke mreže i okvirni sustavi najbitniji su sustavi za predstavljanje znanja, no, glavni problem je nepostojanje formalne semantike što rezultira dvosmislenošću riječi. Prva formalna semantika za predstavljanje znanja nakon semantičkih mreža i okvirnih sustava je jezik KL-ONE. Njegovo glavno ograničenje je loša provodljivost zaključivanja u smislu neodlučivog zaključivanja. Upravo je KL-ONE [16], [28], [53], [63] jezik prethodnik deskriptivne logike kao grane formalne logike, a kao glavna prednost deskriptivne logike nameće se mogućnost zaključivanja, odnosno, korištenja eksplicitno definiranog znanja u svrhu donošenja novog znanja.
Deskriptivna logika služi kao teorijska osnova u implementaciji različitih sustava i predstavljanja znanja. Koristi se u različitim domenama kao što su medicinska informatika, programsko

inženjerstvo, procesiranje prirodnih jezika, web temeljeni informacijski sustavi i druge. Najraširenija takva implementacija uz KIF (Knowledge Interchange Format) je OWL.
Kao logički temeljeni formalizam, deskriptivna logika sadrži poveznice s drugim formalizmima kao što je predikatna logika prvog reda (FOL - First Order Logic) i modalna logika [64]. Većina jezika deskriptivne logike predstavljaju odlučive dijelove predikatne logike prvog reda (pojam odlučiv znači da se za bilo koji ulazni problem, na izlazu u konačnom vremenu dobije "da" ili "ne" odgovor). Izražavanje jezika deskriptivne logike u predikatnu logiku prvog reda, ukoliko je to moguće, provodi se na način da se klase prebacuju u izraze logike prvog reda s jednom slobodnom varijablom gdje naziv klase odgovara unarnom predikatu, naziv svojstva binarnom predikatu i nazivi instanci konstantama. S druge strane, $\mathscr{ALC}$ (Attributive Concept Language with Complements) koji predstavlja jednostavniji oblik jezika deskriptivne logike, može se gledati kao notacijska varijanta modalne logike K_m u kojoj nazivi klasa odgovaraju propozicijskim slovima, dok vrijednosti i egzistencijalni kvantifikatori odgovaraju modalnim operatorima. Osim logike prvog reda i modalne logike, postoji povezanost i s objektno orijentiranim jezicima za modeliranje kao što su ER (Entity Relationship) i UML (Unified Modeling Language) [64]. Relevantni pojmovi domene primjene definiraju se opisima klasa (eng. concept description), odnosno, izrazi su sastavljeni od atomičkih klasa/unarni predikati (eng. atomic concepts) i atomičkih svojstava/binarni predikati (eng. binary predicates) upotrebom konstrukata za svojstva i klase. Konstrukti za svojstva i klase sadržajno variraju ovisno o vrsti jezika deskriptivne logike, a u ovom podpoglavlju za primjere će se uzeti $\mathscr{ALC}$ kao najmanji propozicijski zatvoreni jezik deskriptivne logike [65].
Za početak se kao primjer može uzeti sljedeća definicija klase sastavljena korištenjem osnovnih konstrukata:

"Muškarac koji je oženjen sutkinjom, sva njegova djeca su ili suci ili odvjetnici"

$$Covjek \sqcap \neg Zena \sqcap (\exists ozenjen.Sudac) \sqcap (\forall imaDjecu.(Sudac \sqcup Odvjetnik)) \tag{2.1}$$

Navedena definicija u Izrazu 2.1 sadrži konstrukte disjunkcije ($\sqcup$) koja se interpretira kao unija, konjukcije ($\sqcap$) koja se interpretira kao presjek, negacije ($\neg$) koja se interpretira kao komplement te univerzalni ($\forall$) i egzistencijalni ($\exists$) kvantifikator. Kada bi se napravila kratka analiza navedenog opisa, primjerice instanca Marko pripadala bi $\exists$*oženjen.Sudac* ako postoji neka instanca koja je povezana s instancom Marko preko *udana* svojstva i da u isto vrijeme pripada klasi *Sudac*. Slično tome, Marko pripada $\forall imaDjecu.(Sudac \sqcup Odvjetnik)$ ako su sva njegova djeca (instance) povezane s instancom Marko preko *imaDjecu* svojstva i ako pripadaju klasama *Sudac* ili $Odvjetnik$.

Opisi klasa koriste se za stvaranje baze znanja u deskriptivnoj logici koja se dijeli na tri dijela kako slijedi [64], [59], [60], [61], [62]:

- TBox - terminološko znanje (eng. Terminological box).
- RBox - tvrdnje o svojstvima (eng. Role assertions), odnosi se na složenije jezike deskriptivne logike, dok se većina baza znanja sastoji samo od TBox i ABox komponente.
- ABox - tvrdnje o instancama (eng. Assertional box).

TBox

Sadrži terminologiju neke domene primjene opisivanjem relacija između klasa.

Definiranje podređenosti klasa (eng. concept inclusion)

$$K_1 \sqsubseteq K_2 \tag{2.2}$$

Izraz 2.2 označava podređenosti klase K_1 klasi K_2. Na primjer Slon $\sqsubseteq$ Životinja. Tako definirano znanje omogućava daljnje nasljeđivanje dodatnih činjenica o instancama ovisno o njihovim pripadnostima.

Jednakost klasa (eng. concept equivalence)

$$K_1 \equiv K_2 \tag{2.3}$$

Izraz 2.3 označava da klase K_1 i K_2 sadrže iste instance. Jednakost se također često upotrebljava i u situacijama kada se želi dodijeliti određeni skraćeni naziv nekoj kompleksnoj klasi. Za primjer se može uzeti kompleksna klasa iz izraza 2.1 i dodijeliti joj naziv "M1". Takva definicija prikazana je u Izrazu 2.4:

$$M1 \equiv Covjek \sqcap \neg Zena \sqcap (\exists ozenjen.Sudac) \sqcap (\forall imaDjecu.(Sudac \sqcup Odvjetnik)) \tag{2.4}$$

Kao primjer može se definirati situacija u kojoj je obalna država ona koja ima pristup moru, a neobalna država ona koja nema. Takav TBox prikazan je Izrazom 2.5:

$$\begin{aligned} \mathscr{T} \doteq \{&ObalnaZemlja \equiv Zemlja \sqcap \exists imaIzlazNa.More, \\ &NeObalnaZemlja \equiv Zemlja \sqcap \forall nemaIzlazNa.More\} \end{aligned} \tag{2.5}$$

ABox

Sadrži znanje o instancama u smislu pripadnosti određenoj klasi ili međusobnim relacijama između dvije ili više instanci. Neke od najčešće korištenih tvrdnji su:

Određivanje pripadnosti klasi (eng. concept assertions)

$$K(i) \tag{2.6}$$

Izraz 2.6 označava pripadnost instance i klasi K.
Postavljanje relacije između instanci (eng. role assertions)

$$R(i_1, i_2) \tag{2.7}$$

Izraz 2.7 označava da je instanca i_1 u relaciji R s instancom i_2.
Nejednakost instanci (eng. individual inequality)

$$i_1 \neq i_2 \tag{2.8}$$

Izraz 2.8 označava da su instance i_1 i i_2 različite. S obzirom da u deskriptivnoj logici različita imena instanci mogu predstavljati istu stvar, potrebno je eksplicitno navesti njihovu različitost.
Jednakost instanci (eng. individual equality)

$$i_1 = i_2 \tag{2.9}$$

Izraz 2.9 označava da su instance i_1 i i_2 jednake. Slično kao u prethodnom primjeru nejednakosti instanci, jednakost instanci eksplicitno se navodi ukoliko dvije instance različitog naziva predstavljaju istu stvar u bazi znanja.
Nastavno na primjer iz TBox-a, ABox definira činjenice o obalnim i ne obalnim zemljama Izrazom 2.10:

$$\begin{aligned}\mathscr{A} \doteq \{&NeObalnaZemlja(Srbija), Zemlja(Hrvatska), More(Jadransko_more)\\ &imaIzlazNa(Hrvatska, Jadransko_more)\}\end{aligned} \tag{2.10}$$

RBox
Sadrži znanje koje se odnosi na svojstva definiranih svojstava.
Podređenost svojstva (eng. role inclusion)

$$R_1 \sqsubseteq R_2 \tag{2.11}$$

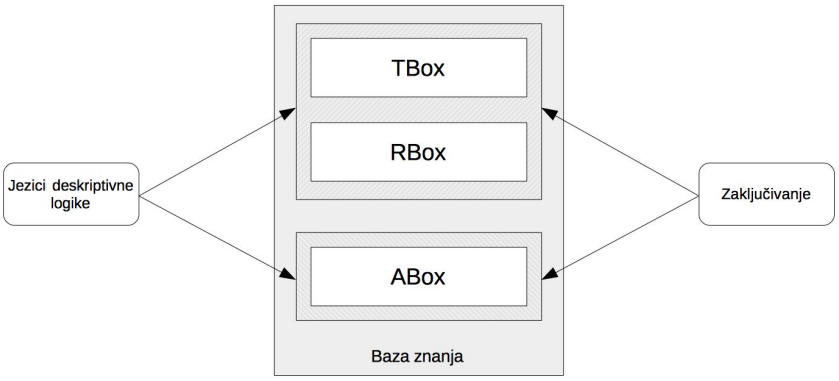
Izraz 2.11 označava da je svojstvo R_1 podsvojstvo svojstva R_2, odnosno, svaka instanca povezana preko svojstva R_1 također je povezana i preko svojstva R_2.
Nejednaka svojstva (eng. disjoint roles)

$$Disjoint(R_1, R_2) \tag{2.12}$$

Izraz 2.12 označava da svojstva R_1 i R_2 ne mogu biti jednaki, odnosno, instanca ne može biti

povezana preko oba svojstva u isto vrijeme.

$$\mathscr{R} \doteq \{Disjoint(imaIzlazna, nemaIzlazNa)\} \tag{2.13}$$



Slika 2.21: Dijelovi baze znanja deskriptivne logike

TBox, RBox i ABox kao dijelovi baze znanja deskriptivne logike prikazani su na Slici 2.21. Značenje klasa deskriptivne logike dano je pomoću interpretacije (eng. interpretation) koja je uređena dvojka $I = \langle \triangle^I, \cdot^I \rangle$ gdje je $\triangle^I$ domena, odnosno, neprazan po mogućnosti konačan skup instanci, te $\cdot^I$ interpretacijska funkcija. Interpretacijska funkcija uparuje svaku klasu koja se pojavljuje u TBox-u u podskup domene, svako svojstvo u binarnu relaciju domene i svaki naziv instance koja se pojavljuje u ABox-u u element domene. Značenje kompleksnih opisa klasa ovisi o konstruktima korištenim u stvaranju opisa [65]. Neki opći konstrukti koncepata i svojstava prikazani su u Tablici 2.6. Bitno je za napomenuti da ukoliko se iz Tablice 2.6 ukloni egzistencijalni kvantifikator, tada preostali konstrukti predstavljaju osnovni $\mathscr{AL}$ jezik deskriptivne logike, a što je vidljivo i u Tablici 2.8.

Tablica 2.6: Opći konstrukti koncepata i svojstava [64], [66]

Naziv	**Sintaksa**	**Semantika temeljena na interpretaciji** $I = \langle \triangle^I, \cdot^I \rangle$
Gornja klasa (eng. top concept)	$\top$	$\triangle^I$
Donja klasa (eng. bottom concept)	$\bot$	$\emptyset$
Enumeracija instanci (eng. nominal)	$\{a\}$	$\{a^I\}$
Presjek klasa (eng. intersection)	$K_1 \sqcap K_2$	$K_1^I \cap K_2^I$
Negacija (eng. concept complement)	$\neg K$	$\triangle^I \setminus \quad K^I$
Unija klasa (eng. union)	$K_1 \sqcup K_2$	$K_1^I \cup K_2^I$
Univerzalni kvantifikator (eng. value/universal restriction)	$\forall R.K$	$\{x \mid \forall y : \langle x,y \rangle \in R^I \quad implicira \quad y \in K^I\}$
Egzistencijalni kvantifikator (eng. existential restriction)	$\exists R.K$	$\{x \mid \exists y : \langle x,y \rangle \in R^I \quad implicira \quad y \in K^I\}$

Na primjeru opisa $Zemlja \sqcap \exists imaIzlazNa.More$ interpretira se kao presjek svih zemalja i skupa elemenata domene koja ima izlaz na more. Za interpretaciju I kaže se da je model od TBox ako zadovoljava sve definicije klasa u $\mathscr{T}$ (za svaku definiciju klase $A \equiv K$ u $\mathscr{T}$ uparuje A i K u isti podskup domene). Kod ABox-a na sličan način, za interpretaciju I kaže se da je model od ABox ako zadovoljava sve klase i svojstva u $\mathscr{A}$ (za svako svojstvo $r(a,b)$ interpretacija r sadrži par koji se sastoji od interpretacije a i b).

Tablica 2.7: Aksiomi deskriptivne logike [64], [66]

Naziv	Sintaksa	Kriterij zadovoljenja
Definicija klase (eng. concept definition)	$A \equiv K$	$A^{\mathscr{I}} = K^{\mathscr{I}}$
Podređenost klase (eng. concept inclusion)	$K_1 \sqsubseteq K_2$	$K_1^{\mathscr{I}} \sqsubseteq K_2^{\mathscr{I}}$
Dodjela instance (eng. concept assertion)	$K(i)$	$i^{\mathscr{I}} \in K^{\mathscr{I}}$
Dodjela svojstva (eng. role assertion)	$R(i_1, i_2)$	$\langle i_1^{\mathscr{I}}, i_2^{\mathscr{I}} \rangle \in R^{\mathscr{I}}$
Dodjela komplement svojstva (eng. negative role assertion)	$\neg R(i_1, i_2)$	$\langle i_1^{\mathscr{I}}, i_2^{\mathscr{I}} \rangle \notin R^{\mathscr{I}}$
Jednakost svojstva (eng. role equivalence)	$R_1 \equiv R_2$	$R_1^{\mathscr{I}} = R_2^{\mathscr{I}}$
Podređenost svojstva (eng. role hierarchy)	$R_1 \sqsubseteq R_2$	$R_1^{\mathscr{I}} \subseteq R_2^{\mathscr{I}}$
Općenita podređenost svojstva (eng. general role inclusion)	$R_1 \circ \cdots \circ R_k \sqsubseteq R_n$	$R_1^{\mathscr{I}} \circ \cdots \circ R_k^{\mathscr{I}} \subseteq R_n^{\mathscr{I}}$
Funkcijsko svojstvo, tranzirivno svojstvo, simetrično svojstvo, asimetrično svojstvo, refleksivno svojstvo, irefleksivno svojstvo	Fun(R), Tra(R), Sym(R), Asy(R), Ref(R), Irr(R)	$R^{\mathscr{I}}$ je funkcionalno, tranzitivno, simetrično, asimetrično, refleksivno i irefleksivno

Kao što je već navedeno ranije, ekspresivnost jezika za reprezentaciju znanja iz obitelji deskriptivnih jezika izražena je brojem konstrukata za reprezentaciju znanja. Jezik koji je spomenut i na temelju kojeg su rađeni primjeri je $\mathscr{ALC}$. $\mathscr{ALC}$ nastaje od osnovnog $\mathscr{AL}$ jezika deskriptivne logike. Dodavanjem dodatnih konstrukata u osnovni $\mathscr{AL}$ jezik dobiju se naprednije izvedenice kao što je to prikazano u Izrazu 2.14 i Tablici 2.8 [64], [66].

$$((\mathscr{ALC} \mid \mathscr{S})[\mathscr{H}] \mid \mathscr{SR})[\mathscr{O}][\varepsilon][\mathscr{I}][\mathscr{F} \mid \mathscr{N} \mid \mathscr{Q}] \tag{2.14}$$

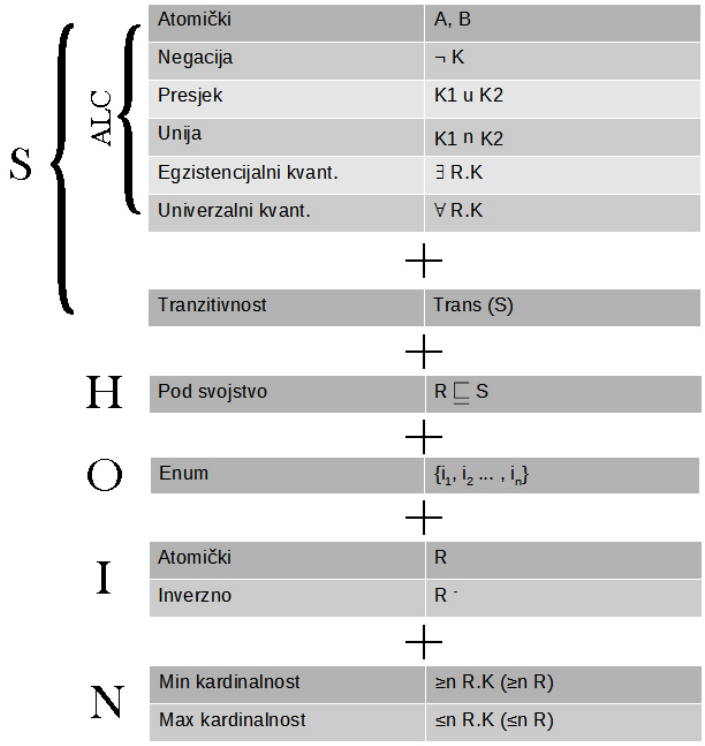
Tablica 2.8: Prikaz nekih konceptnih konstrukata deskriptivne logike [64], [66]

Naziv	Sintaksa	Semantika	Simbol
Gornja klasa (eng. top)	$\top$	$\triangle^{\mathscr{I}}$	$\mathscr{AL}$
Donja klasa (eng. bottom)	$\bot$	$\emptyset$	$\mathscr{AL}$
Atomička klasa (eng. atomic concept)	A	$A^{\mathscr{I}} \subseteq \triangle^{\mathscr{I}}$	$\mathscr{AL}$
Atomičko svojstvo (eng. atomic role)	R	$R^{\mathscr{I}} \subseteq \triangle^{\mathscr{I}} \; x \; \triangle^{\mathscr{I}}$	$\mathscr{AL}$
Atomička negacija (eng. atomic negation)	$\neg A$	$\triangle^{\mathscr{I}} \setminus \; A^{\mathscr{I}}$	$\mathscr{AL}$
Presjek (eng. intersection)	$K_1 \sqcap K_2$	$K_1^{\mathscr{I}} \cap K_2^{\mathscr{I}}$	$\mathscr{AL}$
Unverzalni kvantigikator (eng. value/universal restriction)	$\forall R.K$	$\{a \in \triangle^{\mathscr{I}} \mid \forall b.(a,b) \in R^I \quad \rightarrow, \quad y \in K^I\}$	$\mathscr{AL}$
Unija (eng. union)	$K_1 \sqcup K_2$	$K_1^I \cup K_2^I$	$\mathscr{U}$
Egzistencijalni kvantifikator (eng. existential restriction)	$\exists R.K$	$\{a \in \triangle^{\mathscr{I}} \mid \exists b.(a,b) \in R^I \quad \wedge \quad y \in K^I\}$	$\mathscr{E}$
Restrikcija brojeva bez ograničenja (eng. unqualified number restriction)	$\geqslant n\, R$ $\leqslant n\, R$ $= n\, R$	$\{a \in \triangle^{\mathscr{I}} \mid\mid \{b \in \triangle^{\mathscr{I}} \mid (a,b) \in R^{\mathscr{I}}\} \mid \geq n\}$ $\{a \in \triangle^{\mathscr{I}} \mid\mid \{b \in \triangle^{\mathscr{I}} \mid (a,b) \in R^{\mathscr{I}}\} \mid \leq n\}$ $\{a \in \triangle^{\mathscr{I}} \mid\mid \{b \in \triangle^{\mathscr{I}} \mid (a,b) \in R^{\mathscr{I}}\} \mid = n\}$	$\mathscr{N}$
Negacija (eng. negation)	$\neg K$	$\triangle^I \setminus \; K^I$	$\mathscr{C}$

Neka od dodatnih proširenja koja nisu navedena u Tablici 2.8, a nalaze se u Izrazu 2.14 su:

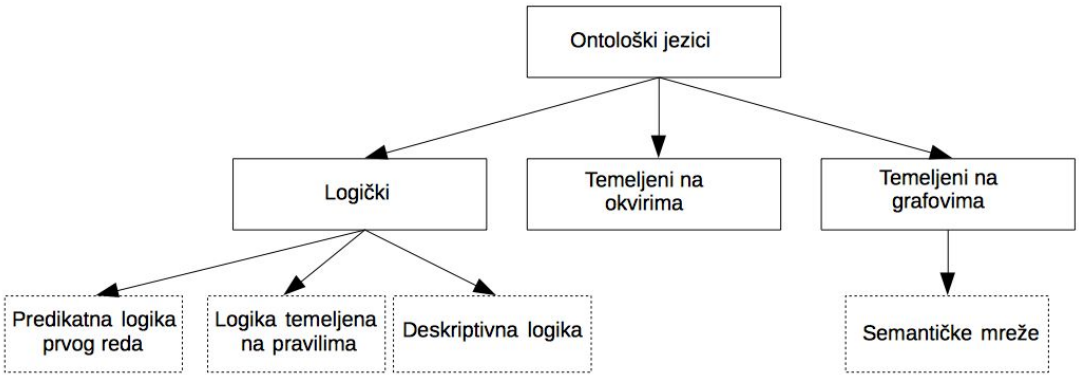
- $\mathscr{S}$ - proširenje dodavanjem tranzitivnih svojstava (ulančani aksiomi oblika $r \circ r \sqsubseteq r$ za $r \in R_N$).
- $\mathscr{SR}$ - proširenje dodavanjem RBox aksioma.
- $\mathscr{H}$ - proširenje dodavanjem hijerarhije svojstava ($R_1 \sqsubseteq R_2$).
- $\mathscr{I}$ - proširenje dodavanjem inverznog svojstva (R^-).
- $\mathscr{F}$ - proširenje dodavanjem podrške za definiranje funkcionalnosti svojstva ($T \sqsubseteq \leq 1.T$).
- $\mathscr{O}$ - proširenja dodavanjem podrške za definiranje klase enumeracijom ($\{i_1, ..., i_n\}$).

Na taj način definirani su jezici $\mathscr{SHIF}$, $\mathscr{SHOIN}$, $\mathscr{SROIQ}$ deskriptivne logike koji će se spominjati u sljedećem poglavlju u kojem će se raspravljati o OWL jeziku gdje su neke verzije OWL jezika upravo temeljene na nekima od njih. Primjer sastavljanja SHOIN podjezika deskriptivne logike prikazan je na Slici 2.22.



Slika 2.22: Primjer definiranja SHOIN podjezika deskriptivne logike

Klasifikacija ontoloških jezika



Slika 2.23: Podjela ontoloških jezika

Na Slici 2.23 prikazana je podjela ontoloških jezika u tri glavne skupine.

Logički

U ovu skupinu pripadaju ontološki jezici temeljeni na logici gdje je tako zadana ontologija model interpretacije koja daje značenje objektima ili kombinaciji objekata i relacija ontologije [20]. Jezici temeljeni na logici dodatno se dijele još na [20]:

Temeljeni na predikatnoj logici prvog reda

*K*IF - formalni jezik s punom logičkom izražajnošću prvog reda koji je svoju primjenu našao u području razvoja baza znanja i umjetne inteligencije. Nastao je kao rezultat "DARPA (Defense Advanced Research Projects Agency) Knowledge Sharing" projekta namijenjen razmjeni znanja između različitih računalnih programa (nezavisno o načinu implementacije i jeziku) tako što se prilikom komunikacije dva programa uparuje interna struktura podataka u KIF. Postoje dvije vrste KIF jezika, linearni i strukturni. Kod linearnog, svi izrazi predstavljeni su u obliku ASCII znakova, dok su kod strukturnog predstavljene kao strukturirani objekti. Neke od glavnih značajki su [67],[20],[35] [27], [28], [29],[37], [52], [68], [69], [70]:

- Deklarativna semantika - shvaćanje izraza definiranih u jeziku bez upotrebe interpretera.
- Logička opsežnost - definiranje rečenica u predikatnoj logici.
- Reprezentacija znanja o reprezentaciji znanja što omogućava implementaciju novih reprezentacijskih konstrukata bez promjene samog jezika.
- Prevodivost - mogućnost prebacivanja deklarativnog znanja u tipične jezike za reprezentaciju znanja i obrnuto.
- Čitljivost - opisivanje semantike reprezentacijskog jezika iako nije primarno namijenjen za tu svrhu.
- Mogućnost korištenja kao reprezentacijskog jezika.

*C*ycL - jezik Cyc projekta koji je s razvojem započeo 1984. godine, a kojem je glavna zadaća određivanje ontologije koja bi omogućila razvoj umjetne inteligencije na računalima. Predstavlja formalni jezik čija sintaksa proizlazi iz predikatne logike, a skup definiranih CycL rečenica koje se sastoje od termina tvore bazu znanja. Iako je još u procesu razvoja, trenutno je u upotrebi u različitim segmentima poslovne problematike s rezultatima koji nadmašuju sva slična programska rješenja. Ono što karakterizira ovaj jezik je visoka preciznost i dobra izražajnost. Trenutno baza znanja CycL jezika sadrži gotovo 200 000 termina [20], [35], [28], [29], [52], [68], [71], [72].

Temeljeni na deskriptivnoj logici

U ovu skupinu pripadaju jezici koji su još dodatno podijeljeni na jezike temeljene isključivo na deskriptivnoj logici, hibridne jezike i jezike semantičkog web-a.

Jezici temeljeni isključivo na deskriptivnoj logici

Ontologija se može izraziti u deskriptivnom jeziku uvođenjem dvije komponente, TBox i ABox gdje:

- TBox - navodi opća svojstva klasa i uloga koji se definiraju kako bi se opisala svojstva objekta koja trebaju biti zadovoljena kako bi mogao pripadati određenoj klasi [7], [20], [29], [57], [58], [72], [73], [74], [75], [76].
- ABox - sadrži tvrdnje o individualnim objektima ili parovima objekata [20], [29], [57], [58], [72], [74].

Detalji jezika temeljenih na deskriptivnoj logici navedeni su ranije u disertaciji.

Hibridni jezici

Hibridni sustavi predstavljaju specijalni tip sustava za reprezentaciju znanja koja se sastoji od dva ili više podsustava s ciljem kombiniranja znanja u svrhu poboljšanja reprezentativne podudarnosti i deduktivne moći. Karakterizira ih upravljanje samo jedne baze znanja, što znači da za određeni upit korisnika, hibridni sustav kombinira znanje i zaključke različitih podsustava kako bi se dobio rezultat. U ovoj skupini bit će opisana dva jezika, CARIN i AL-Log (Attributive Language and Datalog) kao najzanimljiviji zbog toga što oni popunjavaju prostor između deskriptivne logike i logike Hornovih klauzula [20], [77].

CARIN - Zbog svoje ekspresivne moći koja je u većini slučajeva dostatna, jezici temeljeni na Hornovim pravilima predstavljaju osnovu u različitim implementacijama umjetne inteligencije. Međutim, jedan od nedostataka je nedovoljna ekspresivnost potrebna za modeliranje kvalitetne domenske hijerarhijske strukture. Nasuprot tome, deskriptivna logika predstavlja skupinu jezika za reprezentaciju znanja koja je dizajnirana isključivo za modeliranje hijerarhijskih struktura. CARIN pripada skupini jezika koji kombinira deskriptivnu logiku (ALCNR kao proširenje ALC-a) i Hornova pravila. Upravo iz toga proizlazi nekoliko prednosti koje su bitne u samoj primjeni jezika, a to su dodana ekspresivna moć koja se odnosi na modeliranje hijerarhije te upitni jezik. Ono što karakterizira CARIN je tretiranje klasa i svojstava kao unarnih i binarnih predikata što omogućava postavljanje vezivnih upita izraženih preko DL (Description Logic) ABox-a [20], [78], [79].

AL-log - dvokomponentni sustav koji se sastoji od strukturalnog podsustava temeljenog na deskriptivnoj logici (ALC) i relacijskog podsustava temeljenog na Datalog jeziku. Između navedenih podsustava interakcija je ostvarena omogućavajući specifikaciju ograničenja Datalog rečenicama, dok se izražavanje ograničenja provodi upotrebom ALC deskriptivne logike. Strukturalni podsustav koji je podijeljen na dva dijela omogućava izražavanje znanja o klasama, svojstvima i instancama (ALC baza znanja). Prvi dio odnosi se na znanje o klasama od interesa i sadrži skup izjava koje opisuju bitna svojstva takvih klasa, dok se drugi dio odnosi na skup tvrdnji o pripadnosti instanci određenoj klasi i odnosi se na znanje o instancama klasa. Relacijski podsustav omogućava izražavanje relacijskog znanja u obliku ograničenih Datalog

izjava.
U usporedbi sa sličnim rješenjima AL-log pruža veću ekspresivnost u strukturalnoj komponenti i napredniji mehanizam zaključivanja. Kao jedna od glavnih značajki ističe se reprezentacija i zaključivanje nad nepotpunim znanjem koja je moguća zahvaljujući opširnom skupu jezičnih konstrukata u strukturalnoj komponenti [80], [77].

Jezici semantičkog web-a
Kako su ontologije glavni gradivni element semantičkog web-a razvijeno je nekoliko reprezentacijskih jezika. Neki od najpoznatijih su OIL (Ontology Inference Layer), DAML-OIL (DARPA Agent Markup Language - OIL), a u zadnje vrijeme najpopularniji jezik je OWL.

OIL - predstavlja jezik za opis ontologija na web-u čija je glavna svrha omogućavanje dijeljenja i razmjene ontologija. Na razvoj OIL jezika utjecale su različite struje iz područja umjetne inteligencije, pa prema tome, OIL jezik objedinjuje:

- Okvirno temeljena reprezentacija - definiranje klasa kao skupa nadklasa (eng. superclass) te skupova svojstava i ograničenja.
- Deskriptivna logika - skup operatora za formiranje klasa, formalna semantika i zaključivanje i proširenje okvirno temeljenih aksioma za modeliranje elemenata.
- Sintaksna razmjena (Web temeljeni jezici) - proširenje jezika OKBC/GFP (Open Knowledge Base Connectivity/Generic Frame Protocol), XOL (Ontology Exchange Language) i RDF Sheme u svrhu stvaranja osnovne klasne hijerarhije za aplikacije bez podrške za OIL jezik.

OIL ontologija predstavlja strukturu organiziranu u tri razine gdje se prva razina odnosi na instance, druga na definiranje ontologije i treća na informacije o svojstvima ontologije kao što su nazivi, autor, izdavač i drugi [81], [82], [83].

DAML+ OIL - predstavlja jezik koji je rezultat spajanja DAML jezika koji pruža veću ekspresivnost u definiranju ontoloških struktura u odnosu na RDF i RDF Shemu na kojima se temelji i OIL jezika sa sličnim ciljevima kao DAML uz nešto veću ekspresivnost.
Ontološki jezik kao takav, temelji se na objektno orijentiranom pristupu za opisivanje strukture neke domene u smislu klasa i svojstava. Formalno gledano, DAML+OIL može se promatrati kao ekvivalent deskriptivnoj logici gdje DAML+OIL ontologija odgovara DL terminologiji (TBox) [84], [85], [86], [82].

OWL - jezik za opisivanje ontologija na web-u razvijen pod okriljem W3C tako da ostane što je više kompatibilan s prethodnim jezicima, a predložen je kao standardni ontološki jezik za semantički web. Postoje tri različita OWL podjezika, a to su OWL Full, OWL DL i OWL Lite.

O OWL jeziku i njegovim verzijama detaljnije će se govoriti u nastavku disertacije [4], [20], [27], [52], [68], [76], [87], [88], [89], [90], [91].

Jezici temeljeni na okvirima

Jezici koji se temelje na pojmu okvira ili klasa gdje okviri predstavljaju strukturu podataka za reprezentaciju objekata ili klasa objekata. Svaki okvir u sebi sadrži i polja (eng. slots) koja opisuju atribute ili svojstva, a mogu opisivati i relacije između više okvira ako postoje. Jezici ovog tipa često se koriste u domeni obrade prirodnog jezika kao sustavi reprezentacije znanja [20]. Jezici koji pripadaju u ovu skupinu su Ontolingua, OCML (Operational Conceptual Modelling Language), OKBC/GFP, XOL i UML.

Ontolingua - jezik temeljen na kombinaciji KIF-a i okvirne ontologije, gdje okvirna ontologija predstavlja ontologiju za reprezentaciju znanja temeljenu na KIF-u koja omogućava specificiranje ontologije koristeći pojmove (klasa, instanca, klasa_od i druge) prateći paradigmu okvira. Ontolingua omogućava definiranje ontologija na bilo koji od sljedećih načina:

- Koristeći rječnik okvirne ontologije.
- Koristeći izraze KIF jezika.
- Koristeći kombinaciju oba jezika.

Ontologija opisana u Ontolingua jeziku kompatibilna je s različitim jezicima za reprezentaciju znanja, što znači da se definicije napisane u standardnom deklarativnom jeziku prevode u formu koju na ulazu zahtijevaju drugi sustavi za reprezentaciju znanja [20], [35], [37], [92], [93], [83], [37].

OCML - okvirno temeljeni reprezentacijski jezik usmjeren prema formalizaciji i operacionalizaciji modela znanja kao što su ontologije. Kombinira relacijski pristup s objektno orijentiranim pristupom uključujući osnovne proceduralne konstrukte. Tri su tipa konstrukata u OCML jeziku, funkcijski (konstante, varijable, riječi (eng. strings)), kontrolni (specificiraju radnje i tijek izvođenja kao što su petlje i uvjeti), logički (operatori (konjukcija, disjunkcija, negacija, implikacija) te egzistencijalni, odnosno, univezalni kvantifikatori).
Evaluaciju i izvršavanje modela u OCML jeziku obavlja zaključivač koji kombinira dokazivanje izraza (analogno Prolog-u, uključujući i povratak u slučaju pogreške), nasljeđivanje atributa obzirom na hijerarhijsku strukturu i proceduralne pozive prema CommonLisp funkcijskom jeziku [20], [29], [72], [94], [95], [83].

OKBC/GFP - predstavlja aplikacijsko programsko sučelje (eng. API - application programming interface) za pristup i interakciju s bazama znanja u različitim sustavima reprezentacije znanja na uniformni način. Nastaje kao nadogradnja na GFP te ga poboljšava u nekoliko točaka

kao što su mogućnost korištenja na različitim platformama, podrživost više različitih programskih jezika, eksplicitna obrada entiteta koji nisu okviri te bolja kontrola zaključaka kao i određivanja zadanih vrijednosti [20], [29], [96], [97], [98].

XOL - jezik posrednik koji dolazi kao nastavak Ontolingua jezika sa svrhom omogućavanja formata za razmjenu i prebacivanje ontologija između različitih sustava baza i razvojnih alata za ontologije. Sintaksa XOL jezika temelji se na XML jeziku dok se semantika definira kao podskup OKBC modela znanja (OKBC-Lite) [20], [21], [35], [28], [29], [56], [76].

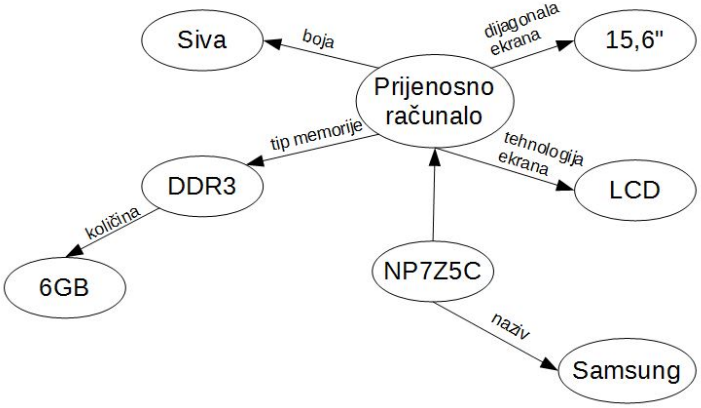
UML - jezik koji služi za modeliranje poslovnih sustava, informacijskih sustava, analizu i dizajniranje te druge slične procese. Gradivni elementi UML jezika su stvari koje predstavljaju apstrakciju u modelu, relacije koje povezuju stvari i dijagrami koji grupiraju skupine stvari koje međusobno koreliraju. Ono što se nameće kao nedostatak UML jezika je manjak formalno definirane semantike [20], [99], [100], [101], [102].

Jezici temeljeni na grafovima

U ovu skupinu pripadaju jezici koji omogućavaju prikaz problema na grafički i intuitivan način koji uvelike olakšava reprezentaciju znanja. Dijele se na semantičke mreže, konceptualne grafove, OML/CKML (Ontology Markup Language/Conceptual Knowledge Markup Language) i predmetne mape [20].

Semantičke mreže - predstavljaju dvodimenzionalnu reprezentaciju znanja koja se sastoji od skupa čvorova (eng. nodes) koji predstavljaju objekte, poveznice (eng. edges) koje izražavaju relacije/odnose između objekata (Na primjer ”is-a” ili ”has-a”) te predstavljaju osnovnu strukturu za reprezentaciju znanja. Također, svakoj poveznici moguće je dodijeliti naziv (eng. link label).

Deklarativna grafička reprezentacija zajednička je svim semantičkim mrežama, a može se koristiti za reprezentaciju znanja ili kao podrška automatiziranim sustavima za zaključivanje. S obzirom da semantičke mreže na neki način oponašaju način na koji se strukturiraju informacije u ljudskom mozgu, vrlo su popularan način reprezentacije znanja u području umjetne inteligencije [20], [103].



Slika 2.24: Jednostavan primjer semantičke mreže

Konceptualni grafovi (eng. conceptual graph) - formalizam temeljen na semantičkim mrežama za grafičku reprezentaciju logičkih opisa. Za razliku od semantičkih mreža, semantika se temelji na predikatnoj logici. Glavna svrha je opis koncepata i međusobnih relacija tako da bude jasno čitljiv, a u isto vrijeme formalan. Predstavlja se bipartitnim grafom gdje se koncepti prikazuju pravokutnicima, a relacije elipsama. Poveznice određuju postojanost i orijentaciju relacije, a također i smjer čitanja [20], [29], [76], [104], [105].



Slika 2.25: Jednostavan primjer konceptualnog grafa

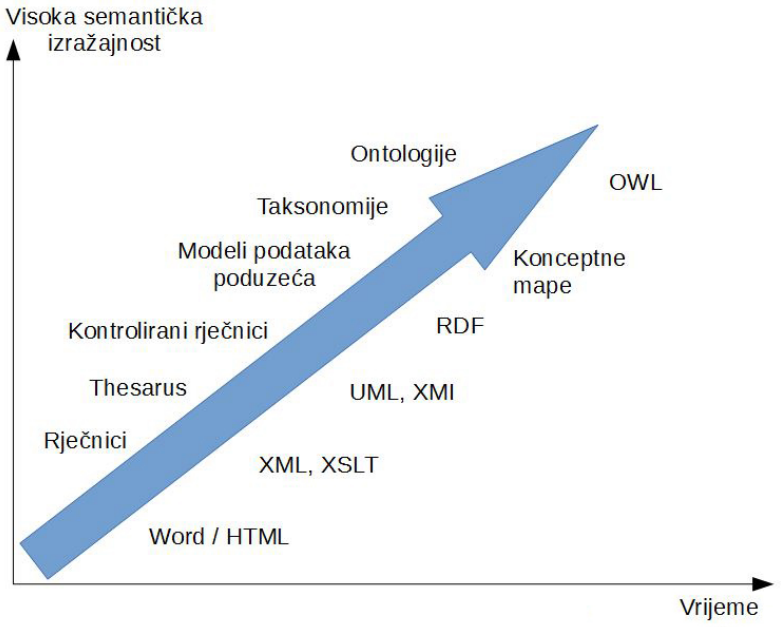
OML/CKML - predstavlja konceptualni jezik za označavanje čija je svrha omogućavanje web agentima lakše i efikasnije prikupljanje znanja. Dijeli se na dva dijela. CKML omogućava konceptualni okvir znanja za reprezentaciju distribuiranih informacija. Temelji se na konceptualnim grafovima, formalnoj analizi koncepata i toku informacija te je usko povezan s deskriptivnom logikom kao načinom modeliranja ontologija. Drugi dio, OML predstavlja ontološku strukturu koja uključuje klase, relacije, instance i ograničenja. Sadrži tri razine izražavanja ograničenja, hijerarhijska, relacijska (račun binarnih relacija) i logičke izraze [15], [20], [76], [106], [107].

Predmetne mape - predstavljaju standard za integraciju znanja, a sastoje se od predmeta i asocijacija kojim se opisuju stvari od interesa te odnosi između predmeta i pojava. Predmeti mogu sadržavati više od jednog naziva, a mogu se kategorizirati po tipu ili podtipu. Također, predmeti mogu biti povezani s internim sadržajem kao što su opisi i vrijednosti podataka te eksternim sadržajem kao što su neke datoteke ili mrežne stranice. Mogu se koristiti za realizaciju različitih organizacijskih shema od jednostavnih taksonomija do semantički naprednih ontologija [20], [30].

Poglavlje 3

Jezik za predstavljanje znanja OWL

Gledano kroz povijest, ontologije se prvo spominju u domeni filozofije. Pojavom Interneta devedesetih godina prošlog stoljeća došlo je do pojave SHOE (Simple HTML Ontology Extension) jezika koji omogućava uključivanje značajki ontologije u HTML dokumente. Nakon razvoja tog prvog web temeljenog jezika, dolazi do razvoja i ostalih jezika koji se spominju ranije u radu, a njihova podjela prikazana je na Slici 2.23.
Razvoj većeg broja jezika, zajedno s nedostatkom standardizacije došlo je do razvoja ontologija u sličnim domenama koje se razlikuju sintaktički i semantički. To uvelike otežava njihovo korištenje pa često da dolazi do potrebe prilagođavanja određenoj primjeni što, naravno, oduzima relativno puno vremena. DAML+OIL predstavlja prvi jezik inspiriran deskriptivnom logikom uz dodatak RDF / RDF Sheme, a koji je trebao biti u potpunosti integriran u izgradnju semantičkog web-a. Međutim, kako u vrijeme kada se razvijao DAML+OIL jezik, RDF i RDFS nisu imali formalu semantiku, nikako se nije mogla napraviti takva veza između dva jezika. Kako bi se taj problem riješio, 2001. godine dva odvojena tima *RDF Core Working Group* i *Web Ontology Working Group* počinju raditi na ažuriranju RDF preporuka i osiguravanju formalne semantike za RDF, odnosno, razvoj ontološkog jezika za web kompatibilnog s novom verzijom RDF-a. Kao rezultat dva istraživanja, 2004. godine nastaje RDF s implementiranom formalnom semantikom i OWL(Web Ontology Language) jezik. Iste te godine, W3C standardizira korištenje OWL jezika koji postaje bitan čimbenik u budućem razvoju semantičkog web-a, a u isto vrijeme rješava problem interoperabilnosti nastalim nagomilavanjem jezika [108].



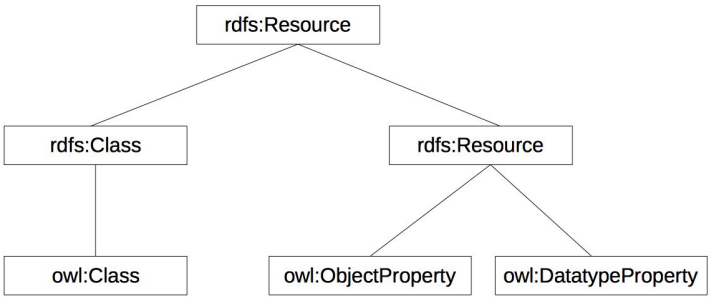
Slika 3.1: Grafički prikaz jezika za označavanje i ontoloških jezika [94]

Prilikom stvaranja OWL jezika postojale su određene struje koje su utjecale na njegov razvoj. Te struje proizlaze iz već utvrđenih formalizama i paradigmi reprezentacije znanja kao i iz već postojećih ontoloških jezika i jezika semantičkog web-a. Jedan od značajnijih utjecaja imao je jezik prethodnik OWL-u, DAML+OIL. Ostali značajniji utjecaji odnose se na deskriptivnu logiku, okvirno temeljene jezike i RDF. Utjecaj deskriptivne logike najviše se odnosio na formalnu semantiku, izbor jezičnih konstrukata i integraciju tipova podataka. OWL Lite i OWL DL kao podskupina OWL jezika koji će biti opisani u nastavku mogu se sagledati kao ekspresivna deskriptivna logika, gdje je OWL Lite ekvivalent SHIF(D), a OWL DL ekvivalent SHOIN(D) koji zapravo pripadaju jezicima iz obitelji jezika deskriptivne logike. Utjecaj okvirnih jezika odnosi se na jednostavnost čitanja i shvaćanja ontologija korisnika koji nemaju iskustva s deskriptivnom logikom. Apstraktna sintaksa kojom je dana formalna specifikacija i semantika OWL jezika slična je onoj OIL jezika, što je čini puno lakšom za čitanje u usporedbi s RDF/XML sintaksom. Zadnji veći utjecaj na razvoj OWL jezika odnosi se na potrebu za održavanjem maksimalne kompatibilnosti s postojećim web jezicima, a pri tome se najviše misli na RDF. To ima smisla iz razloga što RDF/RDF Shema već sadržava osnovna svojstva klasa i svojstva temeljnih ontoloških jezika. Isto tako omogućava definiranje relacija podklase i podsvojstva. Osim toga, RDF/RDFS jezik prethodi razvoju OWL jezika, te postoje zajednice koje rade na razvoju opisa izraženih u tom jeziku.

Međutim, koliko god bila idealna ideja da OWL bude ekstenzija RDF/RDF Sheme, to nije moguće jer dolazi do problema između ekspresivne moći RDF/RDFS i efikasnog zaključivanja. Razlog tome je velika ekspresivnost elemenata RDF/RDFS jezika (npr. rdfs:Class i rdfs:Property) koji bi doveli do nekontroliranog ponašanja prilikom izvođenja zaključaka.

Unatoč tome, u OWL jeziku se koristi jedan dio RDF/RDFS elemenata [108], [56], [32] što je prikazano na Slici 3.2:

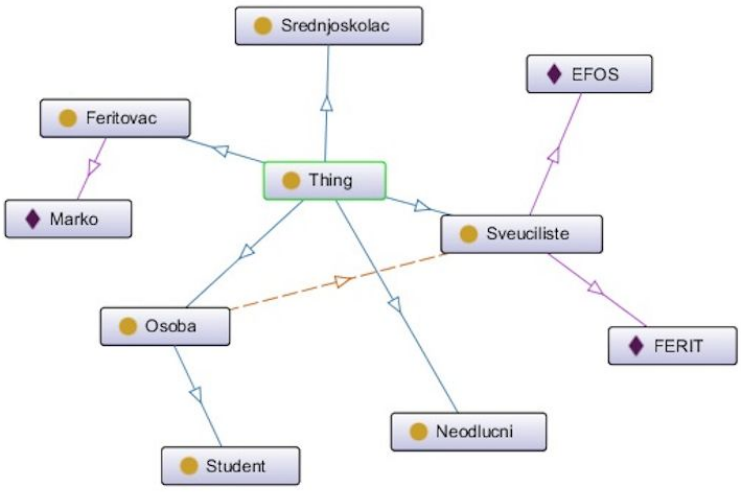
- Svi podjezici OWL-a koriste RDF sintaksu.
- Instance se definiraju koristeći RDF opise.
- Konstrukti kao što su owl:Class, owl:DatatypeProperty i owl:ObjectProperty predstavljaju specijalizaciju istih u RDF jeziku.



Slika 3.2: Podklasni odnos između OWL jezika i RDF/RDFS jezika [32]

Na primjeru jednostavne ontologije koja je prikazana na Slici 3.3 koristeći RDF/RDFS moguće je:

- Definirati klase *Sveučilište*, *Osoba*, *Student* i *Feritovac*.
- Definirati da je klasa *Student* podklasa klase *Osoba*.
- Definirati instance *FERIT* i *EFOS* kao dio klase *Sveučilište*.
- Definirati svojstvo *pripadnost* s domenom (eng. domain) *Osoba* i dosegom (eng. range) *Sveučilište*.
- Definirati svojstvo *godina* s domenom *Osoba* i dosegom *integer*.
- Definirati instancu *Marko* koja pripada klasi *Feritovac* s vrijednosti svojstva *godina* 25.

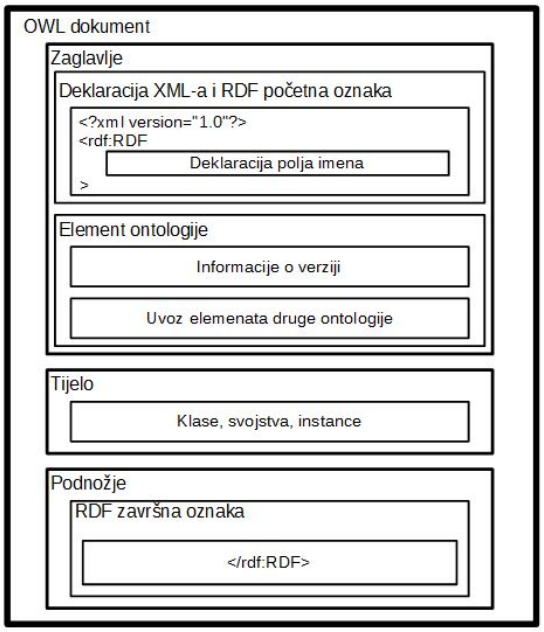


Slika 3.3: Primjer jednostavne ontologije

Za razliku od RDF/RDFS, za istu ontologiju na Slici 3.3 s OWL jezikom dodatno je moguće:

- Definirati klasu *Sveučilište* kao različitu (eng. disjoint) od klase *Osoba*.
- Definirati instancu *FERIT* kao različitu (eng. distinct individual) od instance *EFOS*.
- Definirati svojstvo *imaStudenta* kao inverzno svojstvo *pripadnost*.
- Definirati klasu *Srednjoškolac* koja će sadržavati sve instance klase *Osoba* koje nisu povezane preko svojstva *pripadnost*. Odnosno, sve one osobe koje još nisu upisale fakultet (pod pretpostavkom da će svaka osoba nakon završetka srednje škole upisati fakultet).
- Definirati klasu *Neodlučni* koja će sadržavati sve instance klase *Osoba* koji su povezani preko svojstva *pripadnost* više od jednom. Odnosno, sve one osobe koje pohađaju više od jednog fakulteta.
- Definirati svojstvo *godina* kao funkcionalno.

OWL [27], [52], [68], [87], [88], [89], [90], [91], [109] predstavlja deskriptivni jezik za opisivanje ontologija na web-u, odnosno, za opisivanje klasa, njihovih međusobnih relacija, svojstava i karakteristika svojstava. Osnovna namjena mu je prikazivanje značenja riječi i njihovih odnosa u rečenicama što znači da su sve informacije unutar dokumenata u isto vrijeme strojno obradive i razumljive ljudima. Temelji se na DAML+OIL jeziku te ima više mogućnosti za iskazivanje značenja i semantike u usporedbi s XML/RDF/RDFS. Kao najnovije razvijeni standard postaje jedan od najpopularnijih ontoloških jezika [13], [68], [90] zbog svoje izrazite ekspresivnosti i formalne semantike što uvelike poboljšava razumijevanje sadržaja. S obzirom da je OWL temeljen na RDF-u, OWL ontologija može se predstaviti kao graf, a taj graf se onda može predstaviti u obliku RDF trojki.



Slika 3.4: Struktura OWL dokumenta [18]

Na Slici 3.4 prikazana je struktura OWL dokumenta koja se dijeli na tri glavna dijela [18], [32]:

- Zaglavlje.
- Tijelo.
- Podnožje.

Zaglavlje

Svaki OWL dokument započinje zaglavljem čija je glavna svrha definiranje RDF početne oznake, informacije verzije kao i informacije o uvozima drugih ontologija.
S obzirom da su OWL ontologije predstavljene u RDF/XML formatu, preporučeno je svaki OWL dokument započeti s XML deklaracijom u kojoj je navedena verzija XML-a kao i kodiranje (eng. encoding).
RDF početna oznaka kao jedina početna oznaka u OWL dokumentu sadrži deklaraciju imenskog prostora. Imenski prostor ontologije navodi skraćene reference koje se nalaze unutar OWL dokumenta, a u svrhu onemogućavanja dvosmislenosti oznaka kao i lakše čitljivosti dokumenta. Postoje tri tipa imenskih prostora unutar zaglavlja OWL dokumenta:

- Standardne reference imenskog prostora.
- Imenski prostori povezani s uvoznim izjavama.
- Imenski prostori za identifikaciju ontologije koja se trenutno opisuje.

Prema tome, na primjeru se mogu prikazati definicije imenskih prostora RDF specifikacije, XML sheme, RDFS specifikacije te OWL specifikacije:

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns="http://www.w3.org/2002/07/owl#"
```

Također, ukoliko ontologija sadrži dijelove druge ontologije, moguće je definirati takav imenski prostor:

```
xmlns:food="http://www.food.org/pizzaSlavonska#"
```

Drugi dio zaglavlja odnosi se na informacije o verziji i elementima uvoza. Kako se ontologije s vremenom razvijaju, dolazi do promjena u strukturi ili u semantičkom smislu što dovodi do potrebe o evidentiranju takvih promjena. U tu svrhu, OWL omogućava unos informacija o verzijama OWL dokumenata koje uključuju tekst verzije, indikacije o prethodnim verzijama, izjave kompatibilnosti s prethodnim verzijama te deklaracije o zastarjelim klasama ili svojstvima.

Na primjer, ukoliko postoji sustav koji se oslanja na jednu verziju ontologije, a u međuvremenu dođe do razvoja novije verzije, to može uzrokovati određene probleme uslijed problema kompatibilnosti. Kako bi se provjerilo da li je uzrok problema uistinu OWL dokument, potrebno je provjeriti prethodnu verziju tog dokumenta koja se upisuje pod oznaku *"owl:priorVersion"*.

```
<owl:Ontology rdf:about="">
<owl:priorVersion rdf:resource="http://www.link.org/naziv_ontologije-ont"/>
</owl:Ontology>
```

Također, ukoliko se pouzdano zna da svi konstrukti nove ontologije imaju isto značenje kao u prethodnoj verziji, tada je to potrebno eksplicitno navesti unutar *"backwardCompatibileWith"* oznake.

```
<owl:backwardCompatibileWith rdf:resource="staraVerzijaOntURI"/>
```

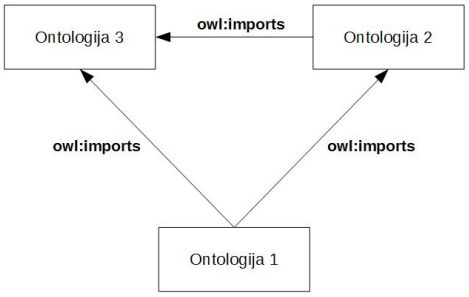
Tablica 3.1: Konstrukti verzioniranja [18]

Konstrukt	**Svrha**
owl:versionInfo	Informacije o verziji.
owl:priorVersion	Informacije o prethodnoj verziji.
owl:backwardCompatibileWith	Informacije o kompatibilnosti trenutne verzije s prethodnom verzijom.
owl:incompatibileWith	Informacije o ne kompatibilnosti trenutne verzije s prethodnom verzijom.
owl:DeprecatedClass	Informacije o zastarjelim klasama koje se ne bi trebale koristiti u narednim verzijama.
owl:DeprecatedProperty	Informacije o zastarjelim svojstvima koja se ne bi trebala koristiti u narednim verzijama.

U Tablici 3.1 navedeni su svi dostupni konstrukti za verzioniranje OWL dokumenata.
Kao jedno od glavnih prednosti OWL-a je mogućnost proširivanja postojeće ontologije uvozom elemenata druge ontologije. U tu svrhu koristi se oznaka *"owl:imports"* koja definira URI vanjske ontologije koja se koristi kao uvoz u trenutnu verziju ontologije.

```
<owl:imports rdf:resource="URI_Ontologije_koja_se_uvozi"/>
```

Osim uvoza jedne ontologije, moguć je uvoz dodatnih ontologija prilikom čega treba obratiti pozornost na tranzitivno svojstvo *"owl:imports"* oznake. Što znači da ako ontologija_1 uvozi ontologiju_2, a ontologija_2 uvozi ontologiju_3, tada ontologija_1 uvozi ontologije 2 i 3 kao što je prikazano na Slici 3.5 [18]. Ako dvije ontologije uvoze jedna drugu, to znači da su one jednake.



Slika 3.5: Tranzitivnost uvoza

Tijelo

OWL omogućava definiranje sljedećih elemenata:

- Klase - glavni gradivni blokovi OWL ontologije, a predstavljaju resurse, odnosno, instancu ili skupinu instanci. Dodatni opis klase definira se korištenjem rječnika iz RDF/RDFS jezika ili OWL-a. Postoje dvije vrste specijalnih klasa, *owl:Thing* i *owl:Nothing*, gdje *owl:Thing* predstavlja korijensku klasu koja je nadklasa svim ostalim klasama i *owl:Nothing* koja predstavlja praznu klasu i podklasu svih ostalih klasa.

```
<owl: Class rdf: ID="Class1" />
<owl: Class rdf: ID="Class2" />
<rdfs: subClassOf rdf: resource="#Class1" />
</owl: Class>
```

- Svojstva - koriste se za definiranje relacija između instanci. Postoje dva tipa svojstva u OWL-u, svojstvo objekata i svojstvo tipa podatka. Svojstvo objekata kao instance predefinirane OWL klase definira vezu između instanci dvije različite klase, dok svojstvo tipa podatka definira odnos između instanci i tipa podatka. Kao što je to slučaj kod klasa, i svojstva se mogu opisivati dodavanjem pod svojstava koristeći element *subPropertyOf*. Osim toga, moguće je definirati još stvari kao što je definiranje taksonomije, domene (eng. domain) i dosega (eng. range) itd.

```
<owl: ObjectProperty rdf: ID="hasPDescriptor" />
<rdfs: domain rdf: resource="#P" />
<rdfs: range rdf: resource="#PDescriptor" />
</owl: ObjectProperty>
```

- Instance - u literaturi poznate još kao i individue predstavljaju specifične objekte koji pripadaju klasama i koriste se za izražavanje semantike klasa i svojstava. Također, povezani su i s drugim instancama definiranjem svojstava. Za definiranje instance koriste se dvije činjenice/aksioma, a to su aksiom o svojstvu instance i pripadnosti klase te aksiom o identitetu instance.

Podnožje

Na kraju svega, svaki OWL dokument završava sa zatvarajućom RDF oznakom *"</rdf:RDF>"*.

3.1 Podjela OWL jezika

OWL jezik dijeli se na tri podjezika [7], [38], [87], [110], [111], [32], [108], [56] što je prikazano na Slici 3.6:

Jezik OWL Lite

OWL Lite predstavlja podskup OWL DL jezika znatno niže složenostii koji omogućava samo osnove za konstrukciju podklasne hijerarhije što znači da isključuje nabrajajuće klase, disjunktne izjave i kardinalnost (ograničena na 0 ili 1). Takva postava rezultira boljim svojstvima zaključivača. S obzirom da je OWL Lite podskup OWL DL jezika, to znači da je svaka OWL Lite ontologija valjana OWL DL ontologija što isto vrijedi i za zaključke. U Tablici 3.2 navedena je apstraktna sintaksa OWL Lite podjezika.

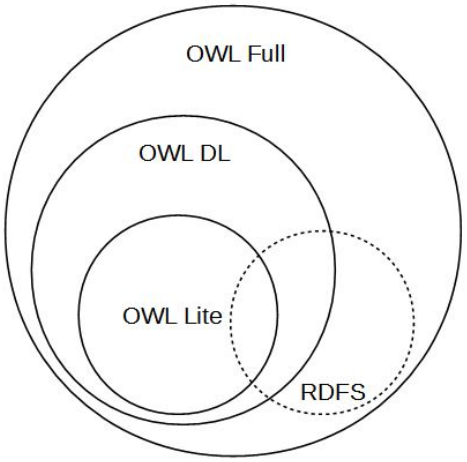
Jezik OWL DL

OWL DL s podrškom deskriptivne logike predstavlja podskup OWL Full podjezika s manje ograničenja u odnosu na OWL Lite. Namijenjen je korisnicima koji zahtijevaju veliku ekspresivnost uz zadržavanje računalne potpunosti (jamstvo da će se svi zaključci izračunati) te odlučivosti (garancija da će svi izračuni biti izvršeni u konačnom vremenu). Sadrži sve OWL jezične konstrukte uz određena ograničenja (npr. klasa se može definirati kao podklasa jedne ili više drugih klasa, ali klasa ne može biti instanca druge klase). Svaka OWL DL ontologija valjana je OWL Full ontologija, a isto vrijedi i za zaključke.

U Tablicama 3.3 i 3.4 navedena je apstraktna sintaksa OWL DL podjezika.

Jezik OWL Full

OWL Full predstavlja vrlo izražajni jezik bez ograničenja razvijen kao proširenje RDF-a uz potpunu kompatibilnost s RDF-om (sintaktička i semantička). U OWL Full podjeziku, klasa se u isto vrijeme može gledati kao skup instanci i kao instanca. Kod OWL Full podjezika ne postoji jamstvo računalne potpunosti i odlučivosti. Svaki OWL Full dokument valjani je RDF dokument.



Slika 3.6: Podjela OWL jezika

Tablica 3.2: Svojstva jezika OWL Lite [112]

OWL apstraktna sintaksa	**DL sintaksa**
Klasa (K djelomično K_1 ... K_n)	$K \sqsubseteq K_i$
Klasa (K potpuna K_1 ... K_n)	$K \equiv K_1 \sqcap ... \ K_n$
Jednake klase (K_1 ... K_n)	$K_1 \equiv ... \equiv K_n$
Svojstvo objekta (R super(R_1) ... super (R_n)	$R \sqsubseteq R_i$
domena (K_1) ... domena (K_n)	$\top \sqsubseteq \forall R^{-}.K_i$
doseg (K_1) ... doseg (K_n)	$\top \sqsubseteq \forall R.K_i$
[inverz od (R_0)]	$R \equiv ({}^{-}R_0)$
[simetrično (R_0)]	$R \equiv ({}^{-}R)$
[funkcijsko]	$\top \sqsubseteq \leqslant 1R$
[inverz funkcijsko]	$\top \sqsubseteq \leqslant 1R^{-}$
[tranzitivno])	$Trans\ (R)$
Podsvojstvo od (R_1 R_2)	$R_1 \sqsubseteq R_2$
Jednako svojstvo (R_1 ... R_n)	$R_1 \equiv ... \equiv R_n$
Instanca (o pripada (K_1) ... pripada (K_n)	$o \in K_i$
vrijednost (R_1 o_1) ... vrijednost (R_n o_n))	$\langle o, o_i \rangle \in R_i$
Jednake instance (o_1 ... o_n)	$o_1 = ... = i_n$
Različite instance (o_1 ... o_n)	$o_i \neq o_j, i \neq j$
Opisi (K)	
A (URI referenca), owl:Thing, owl:Nothing	A ,$\top$,$\bot$
restrikcija (R neke vrijednosti iz (K))	$\exists R.K$
restrikcija (R sve vrijednosti iz (K))	$\forall R.K$
restrikcija (R min kardinalnost (0,1))	$\geqslant 0R$,$\geqslant 1R$
restrikcija (R max kardinalnost (0,1))	$\leqslant 0R$,$\leqslant 1R$

Tablica 3.3: Svojstva jezika OWL DL prvi dio [108], [56]

OWL apstraktna sintaksa	**DL sintaksa**
Opisi (K)	
A (URI referenca)	A
owl:Thing	$\top$
owl:Nothing	$\bot$
presjek od (K_1 K_2...)	$K_1 \sqcap K_2$
unija od (K_1 K_2...)	$K_1 \sqcup K_2$
komplement od (K)	$\neg K$
jedan od (o_1 ...)	$\{o_1, ... \}$
restrikcija (R neke vrijednosti iz (K))	$\exists R.K$
restrikcija (R sve vrijednosti iz (K))	$\forall R.K$
restrikcija (R ima vrijednost (o))	$R : o$
restrikcija (R min kardinalnost (n))	$\geqslant nR$
restrikcija (R max kardinalnost (n))	$\leqslant nR$
restrikcija (U neke vrijednosti iz (D))	$\exists U.D$
restrikcija (U sve vrijednosti iz (D))	$\forall U.D$
restrikcija (U ima vrijednost (v))	$U : v$
restrikcija (U min kardinalnost (n))	$\geqslant nU$
restrikcija (U max kardinalnost (n))	$\leqslant nU$
Raspon podataka (D)	
D (URI referenca)	D
jedan od (v_1 ...,)	$\{v_1 ... \}$
Svojstvo objekta (R)	
R (URI referenca)	R
	R^-
Svojstvo tipa podatka (U)	
U (URI referenca)	U
Instance (o)	
o (URI referenca)	o
Vrijednost podatka (v)	
v (RDF literal)	v

Tablica 3.4: Svojstva jezika OWL DL drugi dio [108], [56]

OWL apstraktna sintaksa	**DL sintaksa**
Klasa (K djelomično K_1 ... K_n)	$K \sqsubseteq K_i$
Klasa (K potpuna K_1 ... K_n)	$K \equiv K_1 \sqcap ... \ K_n$
Navedena klasa (K o_1 ... o_n)	$K \equiv \{o_1, ..., o_n\}$
Podklasa od (K_1 ... K_2)	$K_1 \sqsubseteq K_2$
Jednake klase (K_1 ... K_n)	$K_1 \equiv ... \equiv K_n$
Različite klase (K_1 ... K_n)	$K_i \sqcap K_j = \bot, i \neq j$
Tip podatka (D)	
Svojstvo objekta	
Svojstvo objekta (R super(R_1) ... super (R_n)	$R \sqsubseteq R_i$
domena (K_1) ... domena (K_n)	$\geqslant 1 \ R \sqsubseteq K_i$
doseg (K_1) ... doseg (K_n)	$\top \sqsubseteq \forall R.K_i$
[inverz od (R_0)]	$R \equiv (^{-}R_0)$
[simetrično]	$R \equiv (^{-}R)$
[funkcijsko]	$\top \sqsubseteq \leqslant 1R$
[inverz funkcijsko]	$\top \sqsubseteq \leqslant 1R^{-}$
[tranzitivno])	$Trans\ (R)$
Podsvojstvo od (R_1 R_2)	$R_1 \sqsubseteq R_2$
Jednako svojstvo (R_1 ... R_n)	$R_1 \equiv ... \equiv R_n$
Svojstvo tipa podatka	
U super (U_1) ... super (U_n)	$U \sqsubseteq U_i$
domena (K_1) ... domena (K_m)	$\geqslant 1 \ U \sqsubseteq K_i$
doseg (D_1) ... doseg (D_i)	$\top \sqsubseteq \forall U.D_i$
[funkcijsko])	$\top \sqsubseteq \leqslant 1U$
Podsvojstvo od ($U_1 U_2$)	$U_1 \sqsubseteq U_2$
Jednako svojstvo (U_1 ... U_n)	$U_1 \equiv ... \equiv U_n$
Anotacije	
Svojstvo anotacije (S)	
Instance	
Instanca (o pripada (K_1) ... pripada (K_n)	$o \in K_i$
vrijednost (R_1 o_1) ... vrijednost (R_n o_n)	$\langle o, o_i \rangle \in R_i$
vrijednost (U_1 v_1) ... (U_n v_n))	$\langle o, v_i \rangle \in U_i$
Jednake instance (o_1 ... o_n)	$o_1 = ... = o_n$
Različite instance (o_1 ... o_n)	$o_i \neq o_j, i \neq j$

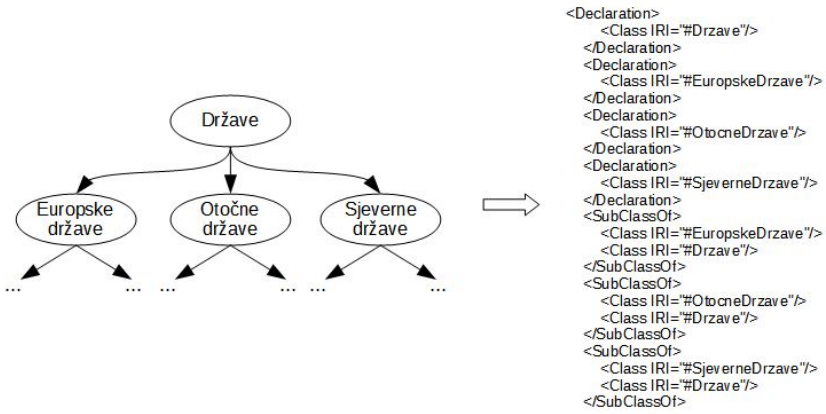
3.2 Definiranje klasa

S obzirom na postavljene ciljeve ove disertacije, fokus se stavlja na hijerarhijsku strukturu, točnije, na tipove i način definiranja klasa kao gradivnih elemenata OWL ontologije [18], [113]. Ostale mogućnosti OWL jezika koje se mogu vidjeti u Tablicama 3.2, 3.3 i 3.4 kao i opisi istih, mogu se pronaći u [18], [113].
Može se reći da postoje dva tipa klasa:

- Primitivna klasa
- Kompleksna klasa

Primitivna klasa

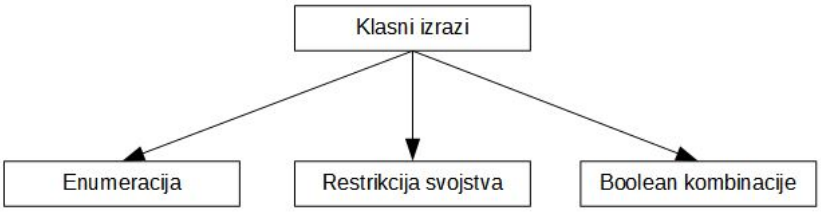
Prvi korak u stvaranju bilo koje ontologije je definiranje primitivnih klasa. Primitivna klasa predstavlja jednostavnu imenovanu klasu (eng. named class) koja u svojoj definiciji ne sadrži ništa. Ona može biti postavljena kao nadklasa drugih klasa ili kao podklasa drugih klasa kao što je to prikazano na Slici 3.7.



Slika 3.7: Primjer primitivnih klasa

Kompleksna klasa

Kompleksna klasa predstavlja nastavak na primitivnu klasu, a razlika je u tome što može sadržavati neke dodatne klasne izraze (eng. class expressions) prikazane na Slici 3.8 [18].



Slika 3.8: Komponente klasnih izraza

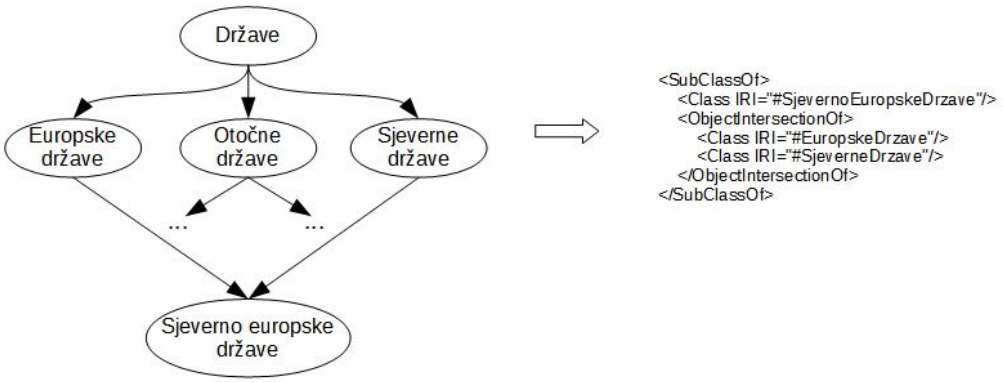
Enumeracija (eng. enumerated Classes) - predstavlja klasu s predefiniranim skupom članova (instanci). To znači da osim navedenih članova niti jedan drugi član ne može biti dio te klase. Kako bi se instance mogle dodati, one već trebaju biti stvorene unutar ontologije, a takav način dodjeljivanja preporučuje se za situacije kada je već unaprijed poznat izgled klase. Dobar primjer za to je definiranje klase koja opisuje dane u tjednu kao što je prikazano na Slici 3.9.

```
<EquivalentClasses>
    <Class IRI="#DaniUTjednu"/>
    <ObjectOneOf>
      <NamedIndividual IRI="#Ponedjeljak"/>
      <NamedIndividual IRI="#Srijeda"/>
      <NamedIndividual IRI="#Nedjelja"/>
      <NamedIndividual IRI="#Cetvrtak"/>
      <NamedIndividual IRI="#Petak"/>
      <NamedIndividual IRI="#Utorak"/>
      <NamedIndividual IRI="#Subota"/>
    </ObjectOneOf>
  </EquivalentClasses>
```

Slika 3.9: Primjer enumeracije klase

Bitno je napomenuti da tako definirana klasa pripada skupini anonimnih klasa o kojima će se još dodatno pisati u nastavku disertacije.
Booleove kombinacije - s obzirom da su klase zapravo skupovi, moguće je njima manipulirati booleovim kombinacijama koristeći:

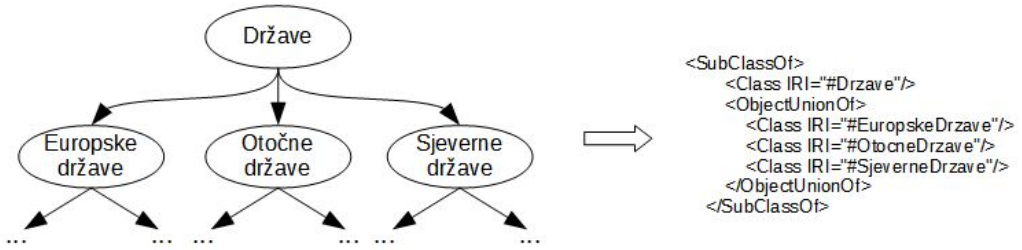
- Presjek (eng. intersection) - presjek klase definira se kombiniranjem dvije ili više drugih klasa koristeći *I* (AND, ⊓) operator. Primjer definicije presjeka prikazan je na Slici 3.10. Tako definirana klasa u sebi sadržava sve zajedničke instance korištenih klasa u presjeku. Za primjer, ako postoje definirane klase *EuropskeDržave* i *SjeverneDržave*, tada bi klasa *SjevernoEuropskeZemlje* bila presjek klasa *EuropskeDržave* i *SjeverneDržave* kao što je prikazano na Slici 5.9.



Slika 3.10: Primjer presjeka

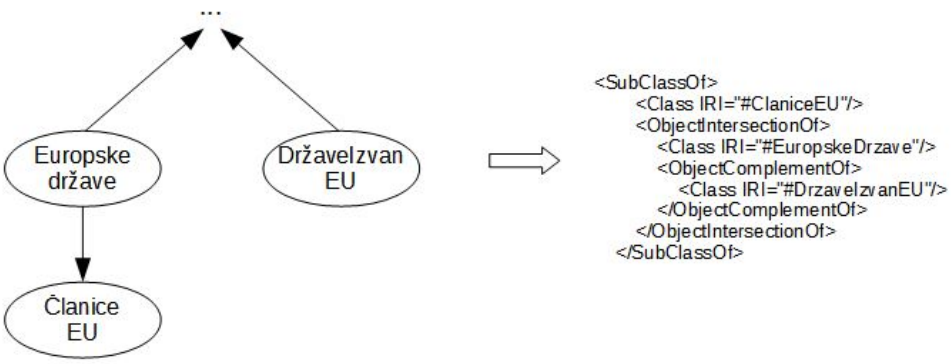
- Uniju (eng. union) - unija klasa definira se kombiniranjem dvije ili više drugih klasa koristeći *ILI* (OR, ⊔) operator. Tako definirana klasa u sebi sadržava sve instance svih korištenih klasa u uniji.

Za primjer, ako postoji klasa *Države* i ako treba sadržavati sve države, tada se može definirati kao unija ostalih država kao što je to prikazano na Slici 3.11.



Slika 3.11: Primjer unije

- Komplement (eng. complement) - koristi se kako bi se definirala klasa definiranjem svih instanci koje ne pripadaju toj klasi što je ekvivalent logičkoj negaciji *NE* (NOT, ¬). Za primjer, ako postoji klasa *EuropskeDržave* i klasa *DržaveIzvanEU* koja predstavlja sve države koje nisu članice Europske unije tada se klasa *ČlaniceEU* koja sadrži sve instance (države) koje ne pripadaju klasi *DržaveIzvanEU* definira kao što je prikazano na Slici 3.12.



Slika 3.12: Primjer komplementa

Slično tome, postoji mogućnost definiranja različitih klasa (eng. disjoint classes). Ukoliko se želi definirati da instanca ne može biti član više od jedne klase u određenoj skupini klasa tada se koristi navedeno svojstvo. S obzirom da se za klase u OWL jeziku pretpostavlja da se preklapaju, nije moguće biti siguran kako instanca nije član neke klase samo zbog toga što nije eksplicitno navedena kao član. Kako bi se osiguralo da su dvije ili više klasa u potpunosti različite potrebno je to eksplicitno navesti koristeći svojstvo različitosti. Za primjer, ako postoje klase *ČlaniceEU* i *DržaveIzvanEU* tada je jasno da se u te dvije klase ne mogu nalaziti iste instance. Primjer definicije različitosti klasa prikazano je na Slici 3.13.

```
<DisjointClasses>
    <Class IRI="#ClaniceEU"/>
    <Class IRI="#DrzaveIzvanEU"/>
</DisjointClasses>
```

Slika 3.13: Primjer definiranja različitosti klasa

Restrikcija svojstva - predstavlja definiranje klase na način da se u obzir uzme svojstvo po kojem instance sudjeluju u nekoj klasi. Klasa koja je definirana restrikcijom naziva se još i anonimnom klasom. Restrikcije se mogu podijeliti u tri glavne skupine:

- Kvantifikatorske restrikcije (eng. quantifier restrictions)
- Restrikcije kardinalnosti (eng. cardinality restrictions)
- *imaVrijednost* restrikcija (eng. hasValue restrictions)

U kvantifikatorske restrikcije mogu se smjestiti egzistencijalna restrikcija (eng. existential restriction) i univerzalna restrikcija (eng. universal restriction).
Egzistencijalna restrikcija ($\exists$) prikazana na Slici 3.14 koristi se za definiranje klase instanci koje imaju barem jednu vezu preko zadanog svojstva s instancama koje su članovi zadane klase.

```
<ObjectSomeValuesFrom>
    <ObjectProperty IRI="#nazivSvojstva"/>
    <Class IRI="#ciljanaKlasa"/>
</ObjectSomeValuesFrom>
```

Slika 3.14: Primjer definiranja egzistencijalne restrikcije

Univerzalna restrikcija ($\forall$) prikazana na Slici 3.15 koristi se za definiranje klasa instanci koje jedino za zadano svojstvo imaju vezu preko tog svojstva s instancama zadane klase.

```
<ObjectAllValuesFrom>
    <ObjectProperty IRI="#nazivSvojstva"/>
    <Class IRI="#ciljanaKlasa"/>
</ObjectAllValuesFrom>
```

Slika 3.15: Primjer definiranja univerzalne restrikcije

Restrikcije kardinalnosti mogu se podijeliti na minimalnu (eng. minimum cardinality) i maksimalnu (eng. maximum cardinality).
Minimalna kardinalnost prikazana na Slici 3.16 koristi se prilikom definiranja klase instanci koje imaju barem *n* (zadani broj) semantički različitih instanci ili vrijednosti podataka za zadano svojstvo.

```
<ObjectMinCardinality cardinality="3">
        <ObjectProperty IRI="#nazivSvojstva"/>
</ObjectMinCardinality>
```

Slika 3.16: Primjer definiranja minimalne kardinalnosti

Maksimalna kardinalnost prikazana na Slici 3.17 koristi se prilikom definiranja klase instanci koje imaju najviše *n* (zadani broj) semantički različitih instanci ili vrijednosti podataka za zadano svojstvo.

```
<ObjectMaxCardinality cardinality="1">
        <ObjectProperty IRI="#nazivSvojstva"/>
</ObjectMaxCardinality>
```

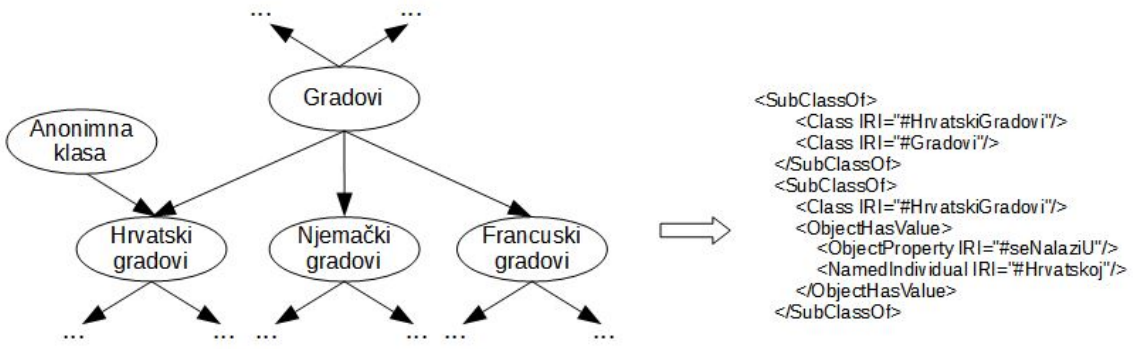
Slika 3.17: Primjer definiranja maksimalne kardinalnosti

Restrikcija *imaVrijednost* prikazana na Slici 3.18 koristi se prilikom definiranja klasa temeljem vrijednosti svojstava instanci.

```
<ObjectHasValue>
        <ObjectProperty IRI="#nazivSvojstva"/>
        <NamedIndividual IRI="#vrijednost"/>
</ObjectHasValue>
```

Slika 3.18: Primjer definiranja *imaVrijednost* restrikcije

Obzirom na navedene restrikcije, anonimna klasa bi bila bilo koja klasa definirana upravo korištenjem nekih od restrikcija. Anonimna klasa predstavlja specijalni oblik klase i uglavnom se i ne prikazuje u samoj strukturi ali ima jednaki utjecaj kao i primjerice primitivna klasa. Za primjer, ako postoji klasa *HrvatskiGradovi* i ako je ona definirana kao podklasa klase *Gradovi*. Tada se još dodatno može reći da u klasu *HrvatskiGradovi* pripadaju i sve instance koje su preko svojstva *seNalaziU* povezane s vrijednosti *Hrvatskoj*. To se postiže definiranjem dodatne anonimne klase koja u sebi sadrži sve takve instance. Drugim riječima, klasa *HrvatskiGradovi* definirana je kao podklasa klase Gradovi i anonimne klase koja u sebi sadrži sve gradove koji se nalaze u Hrvatskoj. Slika 3.19 prikazuje takav slučaj.

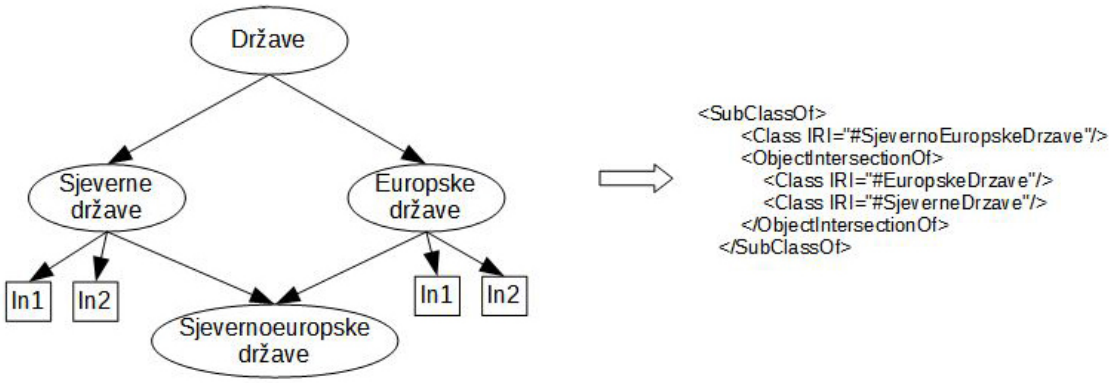


Slika 3.19: Primjer definiranja anonimne klase

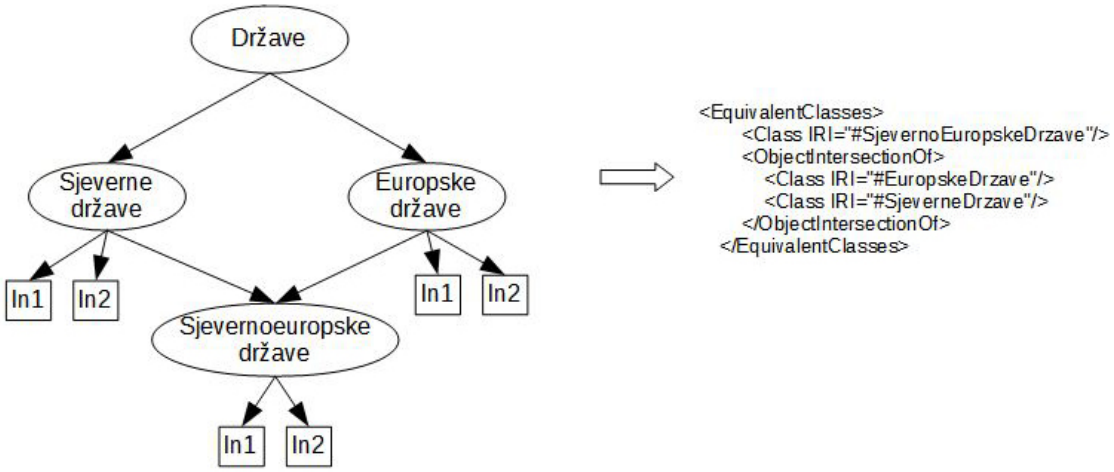
Uz sve navedeno, a koje se odnosi na tipove klasa (primitivna i kompleksna), bitno je napomenuti da se klase mogu definirati na dva načina ovisno o pripadnosti instanci:

- Korištenjem samo nužnih uvjeta.
- Korištenjem nužnih i dovoljnih uvjeta.

Ako postoji neka klasa opisana nužnim uvjetima, za potrebe primjera može se reći da je to *Klasa1*, tada se može reći da instance članovi te klase trebaju zadovoljavati uvjete. Međutim, ne može se reći da bilo koja instanca koja se nalazi negdje dalje u ontologiji, a koja zadovoljava uvjete klase *Klasa1* treba biti član *Klasa1*. U slučaju da se *Klasa1* definira koristeći nužne i dovoljne uvjete, tada se može reći da instance članovi te klase trebaju zadovoljavati uvjete i da bilo koja instanca koja se nalazi dalje u ontologiji, a koja zadovoljava uvjete *Klasa1* treba biti član *Klasa1*. Klasa definirana nužnim i dovoljnim uvjetima označava se *Equivalent class* oznakom, a naziva se definiranom klasom. U slučaju samo nužnih uvjeta, klasa se naziva primitivnom s tim da je bitno napomenuti da se naziv primitivna ne odnosi na tip klase primitivna koji je naveden malo ranije u tekstu. Drugim riječima, i primitivna i kompleksna klasa mogu biti definirane klase koristeći samo nužne ili nužne i dovoljne uvjete.



Slika 3.20: Primjer definiranja klase nužnim uvjetima



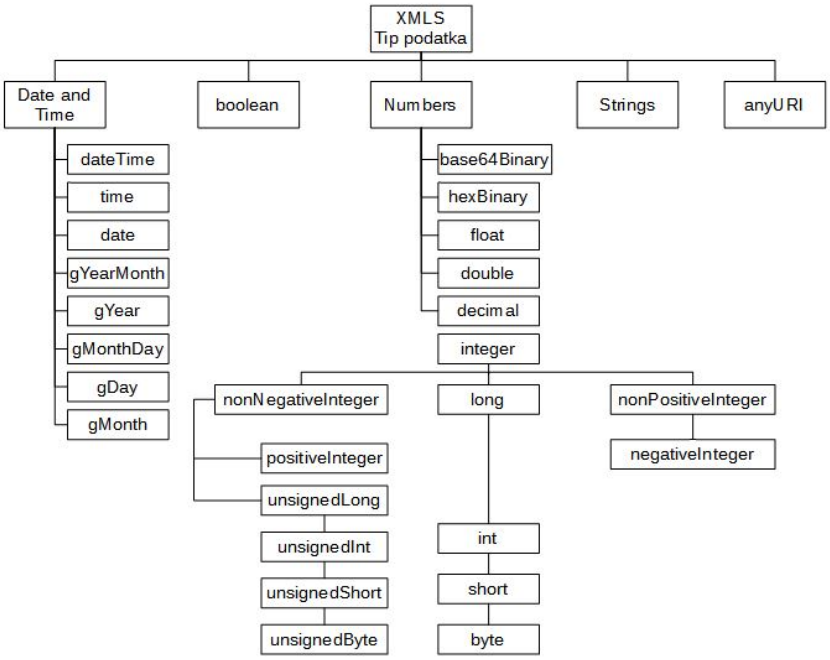
Slika 3.21: Primjer definiranja klase nužnim i dovoljnim uvjetima

Na Slikama 3.20 i 3.21 prikazan je primjer definiranja klase presjekom dviju drugih klasa gdje je na Slici 3.20 klasa *SjevernoEuropskeDrzave* definirana koristeći nužne uvjete, dok je na Slici 3.21 ista klasa definirana nužnim i dovoljnim uvjetima. Osim toga, klase od kojih se sastoji presjek *SjeverneDrzave* i *EuropskeDrzave* sadrže instance *In1* i *In2*. Vidljivo je da klasa definirana nužnim uvjetima ne sadrži niti jednu od navedenih instanci, dok su instance *In1* i *In2* kod klase definirane nužnim i dovoljnim uvjetima dodijeljene toj klasi.

3.3 OWL 2

Iako je OWL sam po sebi dovoljan te je doživio uspjeh među korisnicima, pronađeni su određeni problemi u dizajnu jezika. Ti problemi ne predstavljaju zapreku korištenju, već samo potrebu za dodatnom revizijom jezika što je dovelo do razvoja OWL 2 jezika koji je standardiziran 2009. godine. OWL 2 predstavlja proširenje OWL jezika, a odnosi se na mali skup dodataka koje su zatražili korisnici. Neke od tih promjena odnose se i na razlike u već navedenim profilima (podjezicima) OWL jezika. Sve promjene uvedene u OWL2 jezik već podržavaju različiti zaključivači kao što su HermiT, Pellet te FaCT++.
Neki razlozi razvoja OWL 2 jezika, odnosno problema prepoznatih od strane korisnika i dizajnera OWL jezika su [114]:

- Ograničenje ekspresivnosti
  - Uvjetovana restrikcija kardinalnosti (eng. qualified cardinality restriction) - manjak mogućnosti ograničavanja nekog broja vrijednosti svojstva prema zadanom tipu. Na primjer, izjavu *"ima barem troje djece koja su muškarci"* nije moguće definirati.
  - Ekspresivnost relacija (eng. relational expressivity) - manjak konstrukata za definiranje kompleksnih svojstava. Ti problemi odnose se na tranzitivnost svojstva (ako *a* sadrži *b* i *b* je dio *c*, tada *a* također sadrži *c*), kao i na refleksivnost, irefleksivnost, simetričnost, asimetričnost te različitost (eng. disjoint) svojstva.
  - Tipovi podataka - manjak ekspresivne moći prilikom opisivanja klasa čije su instance povezane s konkretnim vrijednostima kao što su cjelobrojne vrijednosti (eng. integer). OWL se oslanja na ugrađene tipove podataka po uzoru na XML Shemu koji su prikazani na Slici 3.22 [18].
  - Ključevi (eng. keys) - manjak definiranja po ključu kakav postoji primjerice u sustavima baza podataka. Prema tome, u OWL-u nije moguće definirati da je svaki građanin Republike Hrvatske unikatno identificiran preko OIB broja.



Slika 3.22: Tipovi podataka XML Sheme

- Sintaksni problemi - odnose se na dizajn sintakse OWL jezika, a u svrhu poboljšanja jasnoće čitanja i izražavanja. S obzirom da OWL dolazi s dvije normativne sintakse (Apstraktna sintaksa i OWL RDF), odnos između te dvije sintakse stvara probleme prilikom transformacije iz jedne u drugu.
  - Okvirno temeljena paradigma - odnosi se na probleme u odnosima između okvira i aksioma apstraktne sintakse, s obzirom da su na razvoj OWL apstraktne sintakse značajno utjecali okvirno temeljeni ontološki jezici.
  - Poravnanje s konstruktima DL-a - konstrukti OWL apstraktne sintakse ne odgovaraju u potpunosti konstruktima DL-a što dovodi do konfuzije kod korisnika.
  - Određivanje tipa entiteta ontologije - manjak dostatnih informacija koje bi riješile problem neodređenosti tipa entiteta ontologije (klasa, svojstvo, instanca). Na primjeru:

    ```
    ...
    restriction (imaBrata
    someValuesFrom(Muškarac))
    ...
    ```

    nije jasno predstavlja li *"imaBrata"* svojstvo podatka, a *"Muškarac"* tip podatka ili *"imaBrata"* predstavlja svojstvo objekta, a *"Muškarac"* klasu.
  - Problemi s OWL RDF - bez obzira što je veliki broj ontologija pisan u OWL RDF sintaksi, pokazala se kao prilično teška za korištenje u samoj praksi. Jedan od tih problema je konstrukcija RDF izjavne rečenice, tzv. triplet kojim se opisuju semantičke veze između resursa, dok se većina OWL konstrukata ne mogu predsta-

viti u obliku tripleta bez uvođenja novih objekata. Nadalje, tripleti koji odgovaraju jednom OWL konstruktu, trebaju se grupirati premda postoji mogućnost da su razbacani kroz nekoliko dokumenata, što narušava efikasnost i skalabilnost parsera (potrebno je spremanje u radnu memoriju).

– Meta-modeliranje - odnosi se na probleme u razlikovanju između klase i instance što dovodi do potrebe razvoja bolje semantike meta-modeliranja. Za primjer se mogu uzeti dvije izjave:

```
ClassAssertion(Vojnik, Pero)
ClassAssertion(Umirovljeni, Vojnik)
```

gdje se u prvoj izjavi *"Vojnik"* uzima kao klasa, dok se u drugoj izjavi *"Vojnik"* uzima kao instanca.

- OWL Full - predstavlja najekspresivniji podjezik OWL jezika, gdje je svaki RDF graf sintaktički važeći OWL Full dokument. Nepostojanje sintaktičkih restrikcija u korištenju ugrađenog OWL rječnika dovodi do problema računalne neodlučnosti. S druge strane, zbog složenosti ne postoji niti jedna kompletna implementacija OWL Full ontologije [114].
- OWL Lite - iako je OWL Lite definiran kao podjezik OWL DL jezika, te izgleda kao puno jednostavnija inačica i dalje stvara problem računalna složenost zaključivanja (implicitne negacije u aksiomima).

3.4 Podjela OWL 2 jezika

Kao kod OWL jezika, i OWL 2 jezik sastoji se od nekoliko verzija podjezika [114], [115] kao što je prikazano na Slici 3.23:

OWL 2 Full

OWL 2 Full predstavlja najekspresivniji podjezik OWL 2 jezika uključuje sve OWL 2 DL podjezike zajedno s RDFS konstruktima bez ograničenja. Međutim, kao što je to slučaj kod OWL jezika, OWL 2 Full ne jamči da će se proces zaključivanja završiti.

OWL 2 EL

OWL 2 EL predstavlja podjezik temeljen na $\mathcal{EL}$ opisnoj logici, točnije njezinom nasljedniku $\mathcal{EL}$++. Dizajniran je u svrhu omogućavanja efikasnog zaključivanja (u polinomnom vremenu

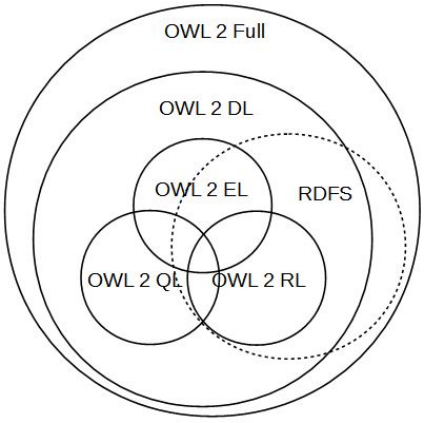
ovisno o veličini ontologije) nad ontologijama koje sadrže izuzetno veliki broj klasa i svojstava s naglaskom na spajanje klasa (podklasno svojstvo) i *SomeValuesFrom* restrikcije kao glavnim značajkama modeliranja. To se najviše odnosi na ontologije u medicini, pa se kao primjer mogu navesti ontologije gena (Gene Ontology) s više od 25 000 klasa i SNOMED-CT ontologija medicinskih izraza s više od 500 000 klasa.

OWL 2 QL

OWL 2 QL predstavlja podjezik temeljen na OWL DL jeziku, dizajniran za primjenu zaključivanja nad velikim količinama podataka. Za razliku od OWL 2 EL podjezika koji je namijenjen za slučaj kada je broj klasa i svojstava velik, OWL QL namijenjen je za rad s ontologijama koje sadrže izuzetno veliki broj instanci. Glavna motivacija za razvoj ovog podjezika je zadržavanje podataka unutar neke od baza podataka (npr. SQL), te omogućiti translaciju zaključivanja u upite na bazu podataka. Kako bi se to postiglo, ekspresivnost samog jezika je smanjena (npr. zabranjena je upotreba disjunkcije i *AllValuesFrom* restrikcija).

OWL 2 RL

OWL 2 RL predstavlja podjezik koji treba u isto vrijeme omogućiti prilagodljivi odnos između procesa zaključivanja temeljenog na pravilima i ekspresivnosti ontologije. Dizajniran je na taj način da se nekoliko zadataka zaključivanja može implementirati kao skup pravila u sustavu ulančavanja prema naprijed. Idealan je za implementaciju poslovne logike u pravila (ako/onda) nad već postojećim RDF podacima.



Slika 3.23: Podjela OWL 2 jezika

Poglavlje 4

Algoritam evidencijskog zaključivanja

Teorija dokaza prvi put se počinje istraživati 60-ih godina prošlog stoljeća, a do sada je našla vrlo široku primjenu u različitim područjima kao što su umjetna inteligencija, ekspertni sustavi, prepoznavanje uzoraka, baze podataka, procjena rizika i između ostalog, u metodama višekriterijskog odlučivanja. Metode višekriterijskog odlučivanja odnose se na donošenje određenih odluka s prisutnim, a često konfliktnim kriterijima. Generalno govoreći, postoji dva glavna tipa MCDM (Multiple-Criteria Decision Making) problema, a odnose se na konačni broj alternativnih rješenja i na beskonačan broj rješenja ovisno o definiranom problemu. Također, metode višekriterijskog odlučivanja dijele se na dva dijela [2]:

- Nekompenzacijske - nije dozvoljeno balansiranje vrijednosti između atributa, odnosno, nepovoljna vrijednost u jednom atributu ne može biti protuvrijednost povoljnoj vrijednosti u drugom atributu. Svaki atribut stoji sam za sebe, stoga se usporedba pravi na bazi atribut-atribut. Metode u ovoj kategoriji karakterizira jednostavnost. Neke od metoda koje pripadaju ovoj skupini su Maxmin, Maxmax i disjunktivna metoda ograničenja.
- Kompenzacijske - dozvoljavaju balansiranje između atributa. Lagani pad u jednom atributu prihvatljiv je ako je kompenziran poboljšanjem u jednom ili više atributa. Neke od metoda koje pripadaju ovoj skupini su metoda bodovanja, metoda podudarnosti i metoda evidencijskog zaključivanja.

Evidencijsko zaključivanje najnovija je metoda višekriterijskog odlučivanja temeljena na De mpster-Shafer teoriji koja umjesto inače korištene matrice odluke koristi proširenu matricu odluke u kojoj se zadani atributi opisuju distribuiranim ocjenama uključujući stupanj uvjerenja. S obzirom na to, kao rezultat se od subjektivne procjene dobije objektivna ocjena [2], [3]. Nastavno na to, može se pretpostaviti da procjena nekog sustava može biti loša, dovoljno dobra, dobra, vrlo dobra i odlična što odgovara ocjenama od 1 do 5. Prilikom procjene nekog atributa, za primjer se može reći da se radi o prijenosnom računalu, ocjena se može zadati na sljedeći način:

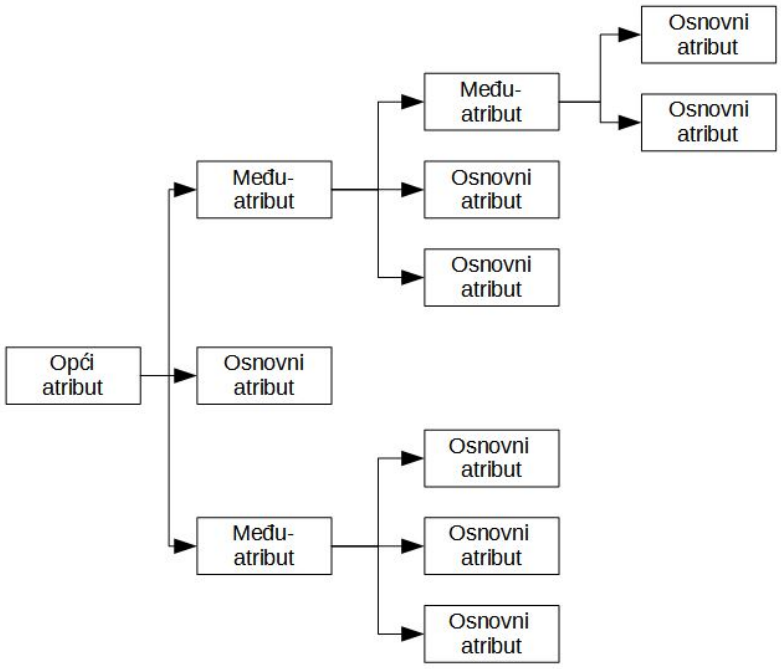
- 30% siguran da je računalo u dobrom stanju, 40% siguran da je računalo u vrlo dobrom

stanju i 10% siguran da je računalo u odličnom stanju.

Iz navedene procjene može se vidjeti da je ukupan stupanj uvjerenja 0.8, a razlika od 0.2 koja nedostaje predstavlja stupanj nesigurnosti (eng. uncertainty). Da je primjerice ocjena bila 100% siguran da je računalo u dobrom stanju ili 50% siguran da je računalo u dobrom stanju i 50% siguran da je računalo u odličnom stanju, tada bi procjena atributa bila potpuna te bi stupanj nesigurnosti bio 0.

4.1 Postupak procjene stanja

Kako bi se napravila procjena nekog sustava, potrebno je načiniti specifikaciju elemenata tog sustava i prikazati u obliku hijerarhijske strukture (taksonomije). Hijerarhijska struktura se može načiniti ručno ili izvući iz neke druge već gotove strukture kao što je ontologija. Prednost ontologije u ovom slučaju bi bila njezina velika ekspresivnost što jamči detaljno opisani sustav. Tako stvorena struktura u sebi sadrži tri tipa čvorova, opći atribut, međuatribut i osnovni atribut. Opći atribut odnosi se na skup svih atributa i predstavlja sustav koji se ocjenjuje. Međuatribut se može opisati kao opći atribut niže razine, on predstavlja skupinu (specijalizacija strukture) osnovnih ili međuatributa. Osnovni atribut predstavlja najnižu razinu, odnosno, predstavlja predmet ocjenjivanja. Primjer takve strukture prikazana je na Slici 4.1.



Slika 4.1: Hijerarhijska organizacija atributa

Matematički gledano, pristup evidencijskog zaključivanja može se predstaviti na sljedeći način [116], [2], [117], [118]:

Skup osnovnih atributa predstavljenih općim atributom y ili međuatributom definira se kao:

$$E = e_1, ...e_i, ...e_L \ , \quad e_i(i = 1, ..., L), \tag{4.1}$$

gdje L predstavlja broj osnovnih atributa. Također, težine atributa koje su bitne prilikom procjene stanja predstavljene su s:

$$\omega = \{\omega_1, ..., \omega_i, ..., \omega_L\}, \tag{4.2}$$

gdje ω_i predstavlja relativnu težinu i-tog osnovnog atributa e_i s vrijednošću između 0 i 1.

$$1 \geq \omega_i \geq 0, \sum_{i=1}^{L} \omega_i = 1. \tag{4.3}$$

Za procjenu stanja atributa skup mogućih ocjena stanja definira se kao uređeni skup elemenata H:

$$H_n(n = 1, ..., N), \tag{4.4}$$

gdje ocjena H_{n+1} predstavlja veću ocjenu od ocjene H_n.
Nakon definicije skupa osnovnih atributa, težina i mogućih ocjena može se definirati procjena i-tog elementa skupa osnovnih atributa koja je predstavljena izrazom:

$$S(e_i) = (H_n, \beta_{n,i}) \ , \quad \sum_{n=1}^{N} \beta_{n,i} \leq 1, \tag{4.5}$$
$$(n = 1, ..., N), \quad (i = 1, ..., L),$$

gdje $\beta_{n,i}$ predstavlja stupanj uvjerenja za kojeg vrijedi:

$$1 \geq \beta_{n,i} \geq 0. \tag{4.6}$$

Ako je $\beta = 1$, to znači da je procjena i-tog elementa $S(e_i)$ potpuna. Ukoliko $\beta = 0$, radi se o specijalnom slučaju koji opisuje potpuni nedostatak ocjene za dani atribut. Potpuni nedostatak ocjene, ili djelomični nedostatak ocjene vrlo je česta pojava u procjeni nekog sustava i omogućavanje takvog načina ocjenjivanja i jest jedna od glavnih prednosti algoritma za evidencijsko zaključivanje.
Ako je β_n stupanj uvjerenja na koju je procijenjen opći atribut i ako je H_n ocjena, potrebno je izračunati stupanj uvjerenja β_n uzimajući u obzir procjene svih osnovnih atributa e_i. Taj postupak naziva se agregacijski proces.

U tu svrhu, računa se matrica osnovnih težina:

$$m_{n,i} = \omega_i \beta_{n,i}\ , \quad 1 \geq \omega_i \geq 0, \sum_{i=1}^{L} \omega_i = 1, \tag{4.7}$$

$$m_{H,i} = 1 - \sum_{n=1}^{N} m_{n,i} = 1 - \omega_i \sum_{n=1}^{N} \beta_{n,i}, \tag{4.8}$$

gdje $m_{n,i}$ predstavlja stupanj kojom i-ti atribut e_i podupire sud da se osnovni atribut može procijeniti na vrijednost unaprijed definirane ocjene H_n. $m_{H,i}$ predstavlja nedodijeljenu vrijednost vjerojatnosti za sve dodijeljene ocjene promatranog atributa e_i.
Uz pretpostavku da su $E_{I,(i)}$ podskup prvih i atributa $E_{I,(i)} = \{e_1, e_2, ..., e_i\}$, $m_{n,I(i)}$ težinska vjerojatnost definirana kao stupanj kojim svi i atributi podupiru sud kojim je opći atribut y procijenjen na ocjenu H_n i $m_{H,I(i)}$ nedodijeljene vrijednosti pojedinim ocjenama nakon procjene svih osnovnih atributa $E_I(i)$, težinske vjerojatnosti $m_n, I(i)$, $m_H, I(i)$ za $E_I(i)$ mogu se dobiti iz osnovne težinske vjerojatnosti $m_{n,j}$ i $m_{H,j}$. Na osnovu navedenog, algoritam evidencijskog zaključivanja može se prikazati izrazima:

$$m_{n,I(i+1)} = K_{I,(i+1)}(m_{n,I(i)} m_{n,i+1} + m_{n,I(i)} m_{H,i+1} + m_{H,I(i)} m_{n,i+1}), \tag{4.9}$$

$$n = 1, ..., N,$$

$$m_{H,I(i+1)} = K_{I,(i+1)} m_{H,I(i)} m_{H,i+1}, \tag{4.10}$$

$$K_{I(i+1)} = \left[1 - \sum_{t=1}^{N} \sum_{j=1 j \neq t}^{N} m_{t,I(i)} m_{j,i+1}\right]^{-1}, \tag{4.11}$$

$$i = 1, ..., L-1,$$

gdje $K_{I(i+1)}$ predstavlja normirajući koeficijent tako da uvjet $\sum_{n=1}^{N} m_{n,I(i+1)} + m_{H,I(i+1)} = 1$ bude zadovoljen. Na kraju algoritma, izračunava se kombinirani stupanj uvjerenja β_n i stupanj nepotpunosti procjene β_H za opći atribut y izrazima:

$$\beta_n = m_{n,I(L)}, \tag{4.12}$$

$$n = 1, ..., N,$$

$$\beta_H = m_{H,I(L)} = 1 - \sum_{n=1}^{N} \beta_n. \tag{4.13}$$

Navedeni agregacijski proces predstavlja osnovni algoritam za evidencijsko zaključivanje. Kako bi rezultat bio objektivniji definirana su četiri dodatna aksioma kao proširenje osnovnog algoritma za evidencijsko zaključivanje [117], [118]:

1. Aksiom nezavisnostii – opći atribut y, se ne može procijeniti ocjenom H_n ukoliko niti jedan od osnovnih atributa skupa E nije procijenjen ocjenom H_n. Ako je $\beta_{n,i} = 0$ za sve $i = 1,...,L$ tada je $\beta_n = 0$.
2. Aksiom koncenzusa – opći atribut y trebao bi biti precizno ocijenjen ocjenom H_n ako su svi osnovni atributi skupa E precizno ocijenjeni ocjenom H_n. Ako je $\beta_{k,i} = 1$ i $\beta_{n,i} = 0$ $(n \neq k)$ za sve $i = 1,...,L$ i $n = 1,...,N$, tada je $\beta_k = 1$ i $\beta_n = 0$ $(n = 1,...,N, n \neq k)$.
3. Aksiom potpunosti – ako su svi osnovni atributi skupa E procijenjeni u potpunosti na određeni skup ocjena, opći atribut y bi također trebao biti procijenjen na isti podskup ocjena.
4. Aksiom nepotpunosti – ako je procjena nekog od osnovnih atributa skupa E nepotpuna do određenog stupnja, opći atribut y će biti procijenjen nepotpunom ocjenom.

Kako bi se ispunili navedeni aksiomi, razvijen je novi pristup koji bi trebao pružiti pouzdanu agregaciju potpunih i nepotpunih informacija koristeći novu težinsku normizaciju:

$$\sum_{i=1}^{L} \omega_i = 1. \tag{4.14}$$

Nova težinska normizacija zadovoljava aksiom koncenzusa. U proširenom algoritmu, nedodijeljene vrijednosti vjerojatnosti $m_{H,i}$ dijele se na dva dijela s obzirom na relativne težine atributa i nepotpunost procjene.

$$\overline{m}_{H,i} = 1 - \omega_i. \tag{4.15}$$

$$\widetilde{m}_{h,i} = \omega_i (1 - \sum_{i=1}^{N} \beta_{n,i}). \tag{4.16}$$

$$\widetilde{m}_{H,i} + \overline{m}_{H,i} = m_{H,i}. \tag{4.17}$$

Vrijednost $\overline{m}_{H,i}$ predstavlja stupanj kojim ostali atributi sudjeluju u procjeni te ovisi o težini i-tog atributa. $\overline{m}_{H,i}$ će imati vrijednost 1 ukoliko težina osnovnog atributa $e_i = 0$ ili $\omega_i = 0$. Ako osnovni atribut e_i dominira procjenom ili je $\omega_i = 1$, $\overline{m}_{H,i}$ će imati vrijednost 0.
Posljedica nepotpunosti procjene osnovnih atributa $\widetilde{m}_{h,i}$ predstavlja drugi dio nedodijeljene vrijednosti vjerojatnosti $m_{H,i}$ koji nije dodijeljen niti jednoj ocjeni. $\widetilde{m}_{h,i} = 0$ ako je procjena osnovnog atributa $S(e_i)$ potpuna, dok će u suprotnom $\widetilde{m}_{h,i}$ imati vrijednost proporcionalnu ω_i između 0 i 1.

Ako se pretpostavi da su težinske vjerojatnosti nastale agregacijom prvih i procjena $m_{n,I(i)}(n = 1,...,N)$, $\widetilde{m}_{H,I(i)}$ i $\overline{m}_{H,I(i)}$, tada se prošireni algoritam evidencijskog zaključivanja može prikazati kao rekurzija koja za $(i+1)$ procjenu uzima u obzir prvih i procjena izrazom:

$$m_{n,I(i+1)} = K_{I(i+1)} \left[m_{n,I(i)} m_{n,i+1} + m_{H,I(i)} m_{n,i+1} + m_{n,I(i)} m_{H,i+1} \right], \tag{4.18}$$

$$m_{H,I(i)} = \widetilde{m}_{H,I(i)} + \overline{m}_{H,I(i)}, \tag{4.19}$$

$$\widetilde{m}_{H,I(i+1)} = K_{I(i+1)} \left[\widetilde{m}_{H,I(i)} \widetilde{m}_{H,i+1} + \overline{m}_{H,I(i)} \widetilde{m}_{H,i+1} + \widetilde{m}_{H,I(i)} \overline{m}_{H,i+1} \right], \tag{4.20}$$

$$\overline{m}_{H,I(i+1)} = K_{I(i+1)} \left[\overline{m}_{H,I(i)} \overline{m}_{H,i+1} \right], \tag{4.21}$$

$$K_{I(i+1)} = \left[1 - \sum_{t=1}^{N} \sum_{j=1 j \neq t}^{N} m_{t,I(i)} m_{j,i+1} \right]^{-1}, \tag{4.22}$$

$$i = \{1,...,L-1\}.$$

Nakon obavljenih svih L procjena, uz upotrebu normizacijskog procesa računa se kombinirani stupanj uvjerenja izrazima:

$$\beta_n = \frac{m_{n,I(L)}}{1 - \overline{m}_{H,I(L)}}, \tag{4.23}$$

$$n = 1,...,N,$$

$$\beta_H = \frac{\widetilde{m}_{H,I(L)}}{1 - \overline{m}_{H,I(L)}}, \tag{4.24}$$

gdje β_n predstavlja stupanj uvjerenja za ocjenu H_n koja je dodijeljena procjenom i β_H koji predstavlja nedodijeljeni stupanj uvjerenja, odnosno, nepotpunost u ukupnom procesu procjene. Kako bi se istaknula razlika u procjenama, odnosno, kako bi se ukupna procjena mogla usporediti s određenim referentnim vrijednostima računa se ukupna ocjena (eng. utility number) kako bi se prikazala ekvivalentna numerička vrijednost pojedinih ocjena dobivenih procesom agregacije. Ako $U(H_n)$ predstavlja ukupnu ocjenu procjene H_n, s tim da vrijedi $U(H_{n+1}) > U(H_n)$, ukupna ocjena procjene $U(H_n)$ može se izračunati metodom dodjeljivanja vjerojatnosti ili korištenjem regresijskog modela s parcijalnim ocjenama ili usporedbama.

$$u(y) = \sum_{n=1}^{N} \beta_n u(H_n). \tag{4.25}$$

Za izračun ukupne ocjene prikazan izrazom 4.25 treba vrijediti uvjet da su procjene potpune, odnosno β_H treba biti jednaka 0. Opći atribut y može biti procijenjen na raspon ocjena koje su u intervalu $[\beta_n, (\beta_n + \beta_H]$, gdje stupanj uvjerenja β_n predstavlja donju granicu procjene na koju se može procijeniti opći atribut y, dok je gornja granica definirana mjerom plauzibilnosti za H_n, odnosno $(\beta_n + \beta_H)$. Ako je procjena potpuna, interval se reducira samo na vrijednost β_n što znači da je interval stupnjeva uvjerenja direktno ovisan o nedodijeljenom stupnju uvjerenja β_H. Za sve ostale slučajeve, vrijednost procjene na koju se opći atribut y može procijeniti nalazi se u intervalu β_n do $(\beta_n + \beta_H)$.
Prema navedenom, definiraju se najmanja, srednja i najveća vrijednost koje jednoznačno karakteriziraju procjenu općeg atributa y izrazima:

$$u_{min}(y) = (\beta_1 + \beta_H)u(H_1) + \sum_{n=2}^{N} \beta_n u(H_n), \tag{4.26}$$

$$u_{avg}(y) = \frac{u_{max}(y) - u_{min}(y)}{2}, \tag{4.27}$$

$$u_{max}(y) = \sum_{n=1}^{N-1} \beta_n u(H_n) + (\beta_N + \beta_H)u(H_N). \tag{4.28}$$

Za slučaj kada nedodijeljeni stupanj uvjerenja $\beta_H = 0$, vrijedi $u(y) = u_{max}(y) = u_{min}(y) = u_{avg}(y)$. U ostalim slučajevima, za ocjenu atributa a_1 kaže se da je bolja od ocjene atributa a_2 ako vrijedi $u_{min}(y(a_1)) > u_{max}(y(a_2))$. Također, ako vrijedi $u_{min}(y(a_1)) = u_{min}(y(a_2))$ i $u_{max}(y(a_1)) = u_{max}(y(a_2))$, tada se može reći da su atributi jednaki.

Poglavlje 5

Model prilagodbe ontološke strukture u taksonomsku strukturu temeljen na HermiT zaključivaču

Potreba za distribuiranim i heterogenim sustavima zahtjeva stvaranje više od jedne ontologije kojima će se pristupati iz različitih sustava i okruženja. Ontologije kao ključni dio semantičkog web-a koriste se za modeliranje različitih reprezentacija znanja, a neke od primjena navedene su ranije u poglavlju koje opisuje ontologije. Upravo ta popularnost ontologija i stvaranje više različitih ontologija u istim domenama dovodi do heterogenosti ontologija i proturječnosti. Kao riješenje tog problema navode se različite tehnike interoperabilnosti koje će se navesti i kratko opisati. Prema tome, tehnike interoperabilnosti su [31], [73], [40], [119], [120], [121], [41], [122]:

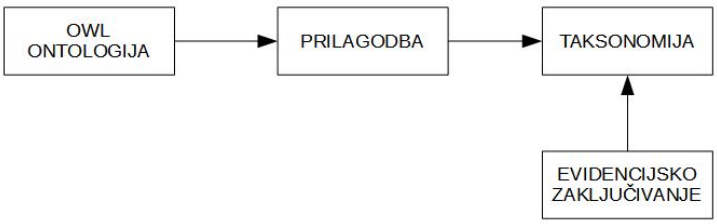
- Transformacija – odnosi se na proces stvaranja nove ontologije prema zahtjevima zadanim prethodnom ontologijom. Prilikom izvođenja transformacije moguće je postići promjene kako u reprezentacijskom formalizmu tako i u samoj semantici.
- Translacija – odnosi se na proces koji se odvija nad dvije ontologije gdje se translatiranjem jedne ontologije dobiva nova ontologija koja koristi reprezentaciju i semantiku druge ontologije. Dobivena ontologija koristi se za drugačije svrhe od prvotno namijenjene. Postoji dvije vrste translacije gdje se prva odnosi na sintaktičku translaciju, a druga na semantičku translaciju.
- Stapanje – odnosi se na proces stvaranja jedne ontologije iz jednog ili više postojećih izvora ontologija iste domene. Tako stvorena ontologija je unikatna bez obzira na preklapajuće domene te sadrži sve informacije svih izvora.
- Uparivanje – odnosi se na alternativni način stapanja ontologija tako da se interpretira skupina sličnosti između dvije ili više ontologija sličnih ili istih domena. Označavanje sličnosti može se obaviti ručno, a postoji i nekoliko polu-automatiziranih metoda mapira-

nja. Postoji tri vrste uparivanja ontologija, a to su uparivanje između integrirane globalne ontologije i lokalne ontologije, uparivanje između lokalnih ontologija i uparivanje ontologije prilikom stapanja i poravnanja.

- Integracija i poravnanje – odnosi se na proces stvaranja ontologije iz jednog ili više izvora različitih domena. Postoje tri slučaja integracije nad ontologijama, a to su integracija prilikom kreiranja nove ontologije, integracija ontologija stapanjem različitih ontologija u jednu koja obuhvaća sve i integracija ontologija u aplikacije primjene.

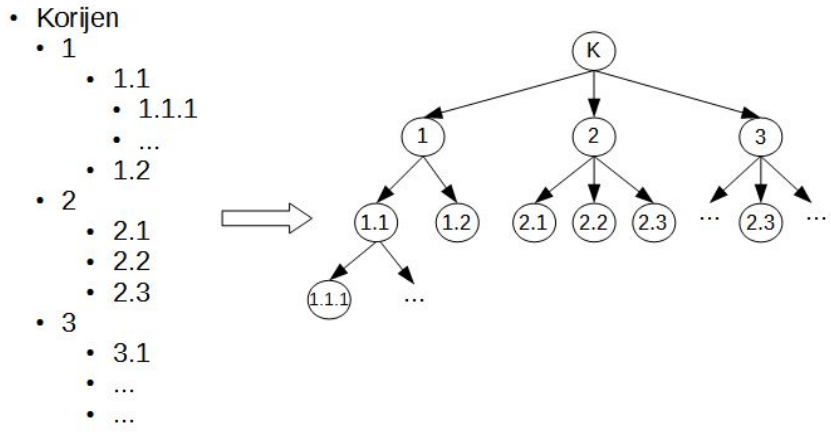
  Poravnanje ontologija predstavlja proces stvaranja poveznica između dvije ontologije tako da se stvaraju dodatni aksiomi koji opisuju veze između koncepata u različitim ontologijama bez mijenjanja samog značenja izvorne ontologije.

Glavni cilj ove disertacije je prilagodba OWL ontologije u taksonomiju koja ispunjava sve uvjete koji su potrebni za primjenu evidencijskog zaključivanja na tako prilagođenoj strukturi kao što je prikazano na Slici 5.1. Kako bi taksonomija ispunjavala uvjete potrebne za primjenu evidencijskog zaključivanja, treba zadovoljavati pravila općeg stabla gdje postoji jedan korijenski čvor dok ostali čvorovi mogu imati maksimalno jednog roditelja i veći broj nasljednika.



Slika 5.1: Proces prilagodbe

Taksonomije predstavljaju jednostavnu strukturu podataka koja se često poistovjećuje sa stablima. Sastoji se od proizvoljnog broja čvorova gdje se prvi čvor naziva korijenski čvor. Svaki čvor predstavlja neki resurs, odnosno, skupinu ili podskupinu resursa koji su povezani relacijama s drugim čvorovima ili skupinama čvorova. Razlika između taksonomije i ontologije je u samoj kompleksnosti strukture, gdje ontologija koristi znatno kompleksnije konstrukte za definiranje strukture te je prikazana grafom, dok je taksonomija prikazana jednostavnim roditelj-dijete (eng. parent-child) vezama.



Slika 5.2: Jednostavna taksonomija

Kako je već i spomenuto ranije, taksonomija se često uspoređuje sa stablima, ali to ne znači da zadovoljava pravila općeg stabla što je potrebno postići. Drugim riječima, struktura izvorne ontologije s klasama koje imaju višestruke roditelje, povratne veze i ostalo, može biti prikazana taksonomijom. To znači da je cilj dobiti taksonomiju koja ne sadržava takve slučajeve, odnosno, da zadovoljava pravila općeg stabla. Na taj način, stovrila bi se veza između OWL ontologije i primjene evidencijskog zaključivanja.
Kako bi se takva veza ostvarila, potrebno je riješiti određene probleme koji se nameću:

- Problem velike ekspresivnosti ontologije.
- Problem strukturalne prirode.

Iako su oba problema usko vezana, potrebno je odrediti ispravan pristup svakom od njih. Kada bi se kao rješenje problema razmatrala velika ekspresivnost ontologije, to bi značilo da je potrebno mijenjati način na koji algoritam za evidencijsko zaključivanje funkcionira, s obzirom da nisu podržani različiti konstrukti koji se koriste u razvoju ontologije. Kao primjer takvog problema može se navesti definicija anonimne klase (eng. complex class) unutar ontologije. Problem strukturalne prirode odnosi se na strukturu podataka koja opisuje neku domenu primjene. Ontologije se predstavljaju grafom kao strukturom podataka, dok je za primjenu algoritma za evidencijsko zaključivanje na ulaz potrebno dovesti taksonomsku strukturu koja opisuje tu istu domenu. Samim time, dolazi se do zaključka kako bi idealan pristup problemu bio rješavanje problema strukturalne prirode na način da se maksimalno zadrži ekspresivnost zadane ontologije.
Kada se priča o promjeni strukture ontologije najčešće se misli na tehnike interoperabilnosti koji se spominju ranije u disertaciji. S obzirom da su sve tehnike interoperabilnosti ranije već navedene i opisane, u Tablici 5.1 prikazani su rezultati koji se dobiju primjenom određene tehnike interoperabilnosti.

Tablica 5.1: Usporedba tehnika interoperabilnosti

Način	**Ulaz**	**Izlaz**
Translacija	Dvije ontologije	Ontologija
Transformacija	Jedna ontologija	Druga struktura
Uparivanje	Dvije ontologije	Ontologija
Stapanje	1+ Ontologija	Ontologija
Integracija	1+ Ontologija	Ontologija
Poravnanje	Dvije ontologije	Ontologija

Iz Tablice 5.1 moguće je vidjeti da metoda transformacije najviše odgovara kao pristup rješavanju navedenog problema s obzirom da rezultat transformacije može biti bilo koja struktura podataka pa tako i taksonomija koja je potrebna za primjenu evidencijskog zaključivanja.

5.1 Pregled postojećeg područja istraživanja

Prije početka rješavanja problema transformacije OWL ontologije u taksonomiju, potrebno je načiniti kompletan pregled trenutnog stanja istraživanja u području transformacije OWL ontologija. Postoji nekoliko tipova transformacija s različitim izlaznim strukturama kako slijedi:

- OWL transformacija u relacijske baze podataka.
- OWL transformacija u OBO (Open Biomedical Ontologies).
- OWL transformacija u konceptualne mape.
- OWL transformacija u HLA objekte (Higl Level Architecture Objects).
- OWL transformacija u skladište podataka (eng. DataWarehouse).
- OWL transformacija u Bayesove mreže (eng. Bayesian networks).
- OWL transformacija u RSM (Resource Space Model).

Transformacija ontologije zapisane u jeziku OWL u relacijske baze podataka

A.Gali et al. predlažu pretvaranje OWL ontologija u relacijske baze zbog ubrzanja performansi gdje se prilikom pretvaranja smanjuje broj spajanja (eng. join). S obzirom da su operacije spajanja vremenski vrlo skupe, prilikom korisnikovog upita značajno se povećavaju performanse [97].

S.Auer et al. razvijaju metodologiju za transformaciju SHOIN pravila za zaključivanje u upite relacijske baze podataka. Također, razvijaju se sheme baze podataka i sheme kodiranja (eng. encoding) za zaključivanje visokih performansi [58].

I.Astrova et al. predlažu pristup za automatsku transformaciju ontologija u objektno orijentiranu bazu podataka. Može se primijeniti na bilo koji objektno orijentirani DBMS (Database Management System) koji podržava sql3 (Structured Query Language) standard. Može uparivati sve konstrukte ontologije, klase, naslijeđene klase, kardinalnost i svojstva objekata [123].

U drugom radu istog autora, razvijena je metoda koja prati 12 pravila, a rješava postojeće probleme u transformaciji ontologija temeljenih na OWL-u u relacijske baze podataka. Navedeni problemi su ignoriranje ograničenja, neimplementiranost rješenja, zahtjevi za interakcijom korisnika te nemogućnost analiziranja gubitaka pretvorene strukture u odnosu na izvornu [124].

E.Vysniauskas et al. razvijaju algoritam za transformaciju OWL ontologije u RDB (Relational Database) shemu. U algoritmu se klase ontologije mapiraju u relacijske tablice, svojstva u relacije, a atributi i ograničenja u meta-podatke. Algoritam je moguće primijeniti na OWL Lite i djelomično na OWL DL sintaksu [125].

Transformacija ontologije zapisane u jeziku OWL u OBO

C.Golbreich et al. predlažu omogućavanje interoperabilnosti između OBO i alata semantičkog web-a. Implementirano je proširenje OWL API-a s OBO parserom. U algoritmu, OBO izrazi mapiraju se u OWL klase, OBO tipovi relacija u OWL svojstva i OBO instance u OWL instance [126].

S.H.Tirmizi et al. razvili su alat koji koristi implementirana pravila za transformaciju OBO ontologije u OWL ontologiju i obrnuto bez gubitaka znanja [127].

V.Dritsou et al. razvili su algoritam s mogućnošću transformacije OWL ontologije u OBO bez gubitka bitnih informacija koje se mogu prikazati u OBO. Algoritam se temelji na prebacivanju OWL ontologija u RDF/XML sintaksu [88].

Transformacija ontologije zapisane u jeziku OWL u konceptualne mape

Rad O.Verhodubs et al. temelji se na razvoju i opisu algoritma za transformaciju OWL ontologije u pravila. Otkriveno je sedam slučajeva gdje se dijelovi OWL koda mogu prebaciti u pravila. Kao drugi dio rada navodi se doprinos u evaluaciji efikasnosti algoritma [128].

O.Vasilecas et al. predlažu automatsku transformaciju OWL ontologija u konceptualni mo-

del koji se zatim može transformirati u kod izvediv u MS SQL (Microsoft SQL), Oracle, DB2 ili drugim modernim DBMS-ovima [129].

V.Graudina et al. opisuju početnu etapu u istraživanju identifikacije odgovarajućih elemenata OWL ontologije i konceptne mape. OWL elementi koji nisu u relaciji s konceptnom mapom ili nisu jednoznačno razumljivi bez primjera odbacuju se. Postoji još slučajeva odbacivanja koji se mogu pronaći u tablici navedenoj u samom radu [130].

U drugom radu istog autora, razvijen je algoritam za generiranje konceptne mape iz OWL ontologije temeljen na definiranju sličnosti između elemenata OWL ontologija i odgovarajućih elemenata konceptne mape [131].

Transformacija ontologije zapisane u jeziku OWL u HLA objekte

Rad Ö.ÖzdİkİŞ et al. temelji se na razvoju alata s korisničkim sučeljem koji omogućava konfiguriranje uparivanja OWL konstrukata u HLA OMT (Object Model Template) konstrukte. Zatim se definicije uparivanja primjenjuju na OWL ontologije, a kao rezultat algoritma dobije se HLA model prikazan u formi XML dokumenta [132].

Transformacija ontologije zapisane u jeziku OWL u skladište podataka

Rad S.Talebzadeh et al. temelji se na prijedlogu algoritma koji se sastoji od pet koraka za potpuno automatsku transformaciju OWL ontologije u skladište podataka. Predloženi algoritam u ovisnosti o klasama, dostupnim relacijama i podacima spremljenim u OWL ontologiju stvara tablice skladišta podataka te ubacuje podatke u njih [133].

V.Nebot et al. predlažu poluautomatsko ekstraktiranje i kombinaciju podataka izraženih u OWL ontologiji (RDF/XML format) u činjeničnu tablicu skladišta podataka gdje je svaka činjenica semantički validna kombinacija dimenzijskih vrijednosti s odgovarajućim prikupljenim mjerama [134].

Y.Laadidi et al. predlažu pristup definiranja činjenične tablice dimenzionalnog modela gdje predložena metoda dijeli strukturu OWL ontologije u dva dijela. Prvo dio odnosi se na pojednostavljenje procesa fokusiranja na bitne koncepte i podatke, dok se drugi dio odnosi na izradu dimenzionalno činjeničnog modela prema rezultatima OWL strukture iz prvog koraka. Navedeni pristup omogućava administratoru skladišta podataka više fleksibilnosti koristeći složene izvore OWL ontologije [135].

M.Gulić predlaže poluautomatsku metodu za pretvaranje OWL ontologije u zvijezda shemu. Metoda transformira elemente ontologije nakon što korisnik odabere koje će klase ontologije predstavljati činjenice kao i koji će podaci biti uključeni u skladište podataka kako bi se izbjeglo popunjavanje nepotrebnim podacima [136].

U znanstvenoj literaturi postoji još nekoliko radova koji su direktno ili indirektno vezani za pretvaranje OWL ontologija u skladište podataka ali nisu u potpunosti orijentirani prema tome. To su sljedeći radovi:

J.Pardillo et al. opisuje načine na koje se skladište podataka može poboljšati korištenjem OWL ontologija. Neki od razloga koji pogoduju upravljanju ontološkog znanja i alatima za poboljšanje izvođenja poslova u skladištenju podataka su validiranje podatkovnih instanci i modela, razumijevanje značenja notacija, ponovna upotreba znanja različitih domena, bolja integracija podataka, analiza i drugi koji se mogu pronaći u radu [137].

Rad L.E.Sarraj et al. temelji se na stvaranju OWL ontologije koja se odnosi na strukturu skladišta podataka, resursa i koncepata domene. Obrađuju dva ključna pitanja, a to su pitanja nadležnosti koje ontologija uzima u obzir te koje klase koje tvore ontologiju doprinose zaključivačima u analizi i razumijevanju indikatora skladišta podataka [74].

V.Vrbanić predlaže stvaranje modela ontološkog pristupa poboljšanja skladišta podatka korištenjem same semantike, odnosno, poboljšanja sustava baze podataka procesiranjem upita na sintaktičkoj i semantičkoj razini [138].

V.Nebot et al. definiraju osnovu za multidimenzionalnu analizu podataka semantičkog weba u skladište podataka. Definirano je nekoliko glavnih doprinosa rada, razvojni okvir za dizajn i izgradnju semantičkog skladišta podataka, definiranje zahtjeva i primjera iz kojih se vidi korist korištenja takvog sustava, metodologija za dizajn i automatsko generiranje i validaciju multidimenzionalne integrirane ontologije, automatska konstrukcija multidimenzionalne kocke te proučavanje nekoliko alternativa za implementaciju predloženog sustava [139].

M.Thenmozhi et al. razvijaju pristup razvoju hibridnog razvojnog okvira koji koriste OWL ontologije za stvaranje multidimenzionalnih shema [140].

V.Nicolicin et al. predlažu poboljšanje performansi odziva upita u skladištu podataka izmjenama alokacije cache memorije korištenjem OWL ontologije [141].

Transformacija ontologije zapisane u jeziku OWL u Bayesove mreže

Rad Z.Ding et al.[142] temelji se na modeliranju nesigurnosti u OWL ontologiji temeljenoj na bayesovim mrežama što uključuje proširenje OWL-a, transformaciju OWL ontologije u strukturu bayesove mreže korištenjem transformacijskih pravila i izrada uvjetnih vjerojatnosnih tablica bayesove mreže korištenjem IPFP (Iterative Proportional Fitting Procedure) metode.

Transformacija ontologije zapisane u jeziku OWL u RSM

H.Zhuge et al.[143] predlažu automatsku transformaciju između OWL ontološkog opisa u RS (Resource Space) kako bi se povećala sveukupna efikasnost RSM-a. U algoritmu, instance OWL ontologije pretvaraju se u resurse RSM-a, a odnosi nasljeđivanja i svojstva u osi RSM-a.

5.2 Izrada modela

Detaljnim pregledom radova navedenih u pregledu postojećeg područja istraživanja, napravljen je popis pokrivenosti definicija i aksioma ontologije za pojedine metode. Ti podaci prikazani su u Tablici 5.2, a odnose se na sve metode za koje su bile dostupne informacije.

Tablica 5.2: Usporedba metoda transformacije OWL ontologija

			DB				**CM**			**DW**		**HLA**	**RSM**
			[[97]]	[[123],[124]]	[[144]]	[[125]]	[[128]]	[[130]]	[[131]]	[[136]]	[[133]]	[[132]]	[[143]]
Klasa	**definicija**	Klasa	X	X	X	X	X	X	X	X	X	X	X
		Enumeracija		X								X	
		Restrikcija	X	X	X	X	X	X				X	X
		Presjek						X	X			X	
		Unija	X		X				X			X	X
		Komplement	X				X		X			X	
	aksiomi	podKlasa	X	X	X	X	X	X	X	X	X	X	X
		Jednakost					X	X	X				X
		Nejednakost	X		X	X							
Svojstvo	**definicija**	Svojstvo	X	X	X	X	X	X	X	X	X	X	X
		Domena, Range				X		X	X		X		X
		podSvojstvo									X		
	aksiomi	Funkcionalno		X	X			X	X				
		Jednakost, Inverz		X	X			X	X				
		Tranzitivno, Simetrično						X	X				
Instanca	**definicija**	Instanca	X	X	X		X	X	X	X	X		X
	aksiomi	Jednakost		X									

Iz Tablice 5.2 vidljivo je da sve metode imaju neke zajedničke funkcionalnosti, a odnose se na detekciju klasa, podklasa, svojstava i instanci. Međutim, ostali slučajevi nisu jednako

pokriveni ili nisu navedeni u radovima u kojima su opisane metode. Na osnovu toga, mogu se navesti sljedeća ograničenja navedenih metoda u odnosu na ciljeve koji se žele postići u disertaciji:

- Selektivan odabir aksioma.
- Pokrivenost razine ekspresivnosti OWL jezika.
- Detekcija anonimnih klasa.
- Rješavanje specijalnih slučajeva kao što je presjek i jednakost klasa.

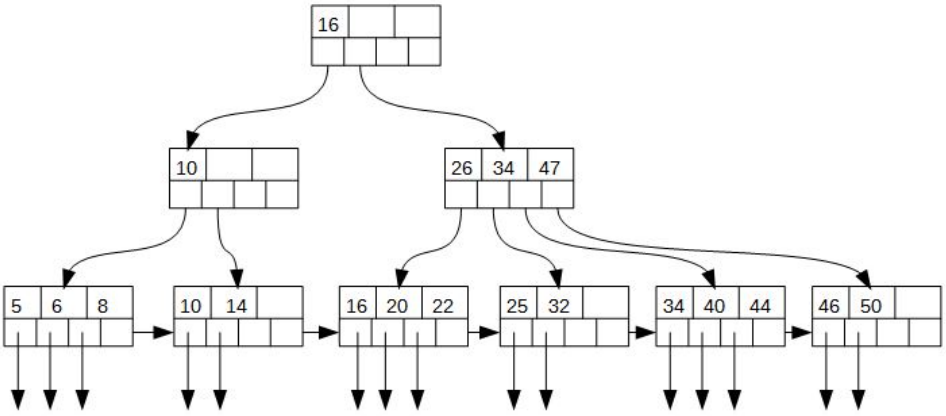
Kako je poznat tip strukture potreban za primjenu algoritma evidencijskog zaključivanja, osim analize pokrivenosti elemenata ontologije prilikom transformacije pojedinih metoda, potrebno je napraviti i analizu izlaznih struktura svake od metoda.

Tablica 5.3: Izlazne strukture metoda transformacija ontologija

Izvor	**Rezultat transformacije**	**Struktura podataka**
Ontologija	DB	B+ stablo
	CM	Stablo, zvijezda, graf
	DW	UB stablo
	Bayes	Graf
	OBO	Graf
	HLA	Graf
	RSM	Resursni model

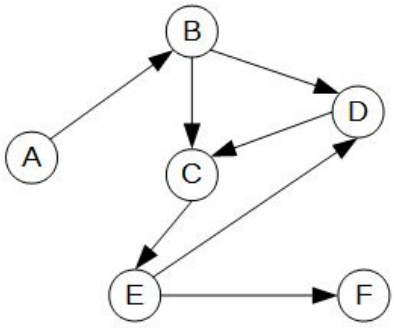
U Tablici 5.3 prikazani su rezultati analize izlaznih struktura pojedinih metoda, gdje su metode grupirane ovisno o cilju (baza podataka, predmetne mape i drugi).

B+ stablo [145], [146] – predstavlja strukturu podataka, odnosno, samobalansirajuće stablo traženja. Definirano je kao nastavak B-stabla (generalizacija poredanog binarnog stabla), gdje svaki od čvorova može imati više od dva nasljednika. Često se koristi u implementacijama baza podataka. Dijeli se na unutarnje čvorove (eng. non-leaf nodes) koji sadrže poredanu listu ključeva i pokazivača na čvorove niže razine i vanjske čvorove (eng. leaf nodes) u koje se spremaju podaci. Primjer jednog B+ stabla prikazan je na Slici 5.3.



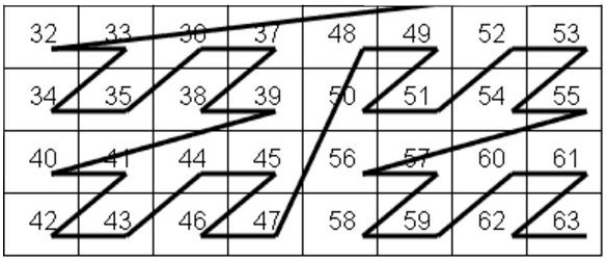
Slika 5.3: B+ stablo

Graf [146] – predstavlja složenu strukturu podataka, odnosno, uređeni skup konačnog broja čvorova i međusobnih relacija. Postoje dvije vrste grafova, usmjereni (eng. directed graph) i neusmjereni (eng. undirected graph). Čvorovi unutar grafa mogu biti povezani na različite načine, pa tako svaki čvor može imati više ulaznih i izlaznih relacija, a ono što može dodatno otežati rad s grafom je postojanje ciklusa. O ciklusima u ontologijama će se govoriti u nastavku disertacije. S obzirom da su ontologije predstavljene usmjerenim grafom, na Slici 5.4 prikazan je primjer usmjerenog grafa.



Slika 5.4: Primjer usmjerenog grafa

UB-stablo (eng. Universal B-tree) [147] – predstavlja balansirano stablo za efikasno pretraživanje podataka. Zapravo, radi se o B+ stablu s razlikom što UB-stabla služe za spremanje višedimenzijskih podataka u Z-smjeru (Slika 5.5, dok B+ stabla služe za spremanje jednodimenzijskih podataka (linearno poredani ključevi). Kao primjer jednog od najčešće korištenog višedimenzijskog podatka je svakako GPS (Global Positioning System) podatak (tri dimenzije, visina + geografska širina + geografska dužina).

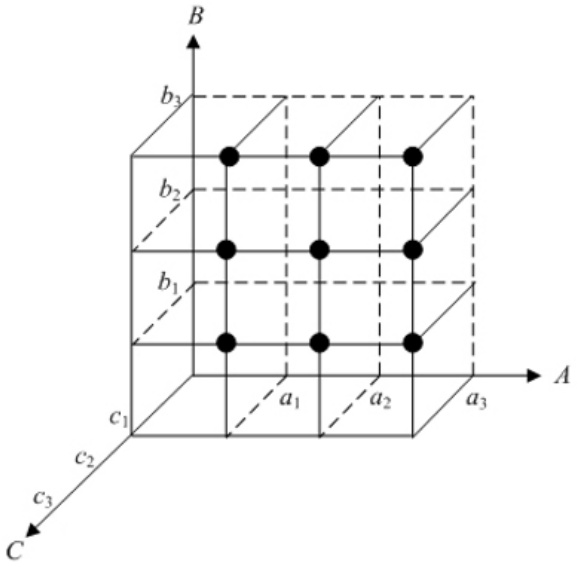


Slika 5.5: Primjer Z-krivulje[147]

Resursni model

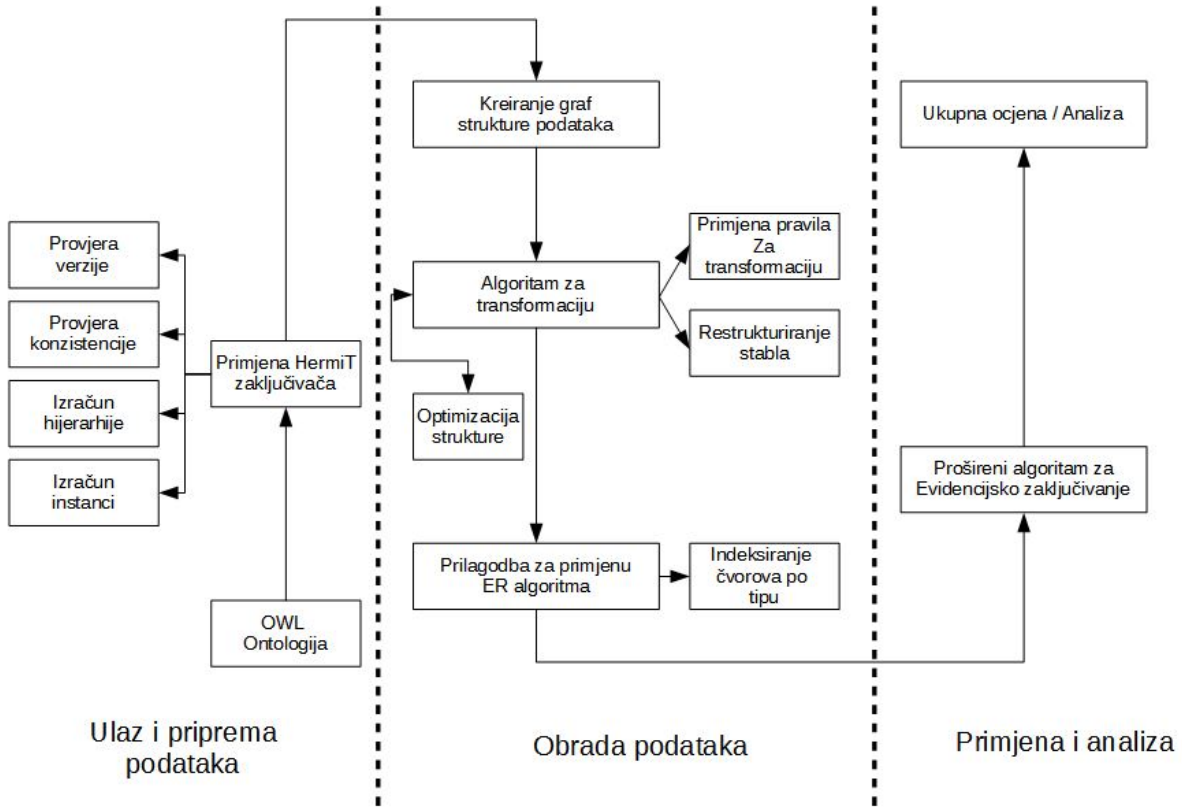
Osnovni koncepti i notacije vezani za resursni model definirane su kao [148]:

- Resursni prostor predstavlja n-dimenzionalni prostor prikazan na Slici 5.6 gdje se svaka točka odnosi na jedan unikatno određeni resurs *RS* ili na skup međusobno povezanih resursa *RS* $(X_1, X_2, ..., X_n)$. X_i predstavlja naziv osi, dok $X_i = \langle C_{i1}, C_{i2}, ..., C_{in} \rangle$ predstavlja os s poredanim koordinatama gdje C označava naziv kordinate.
- Koordinata C predstavlja klasu resursa *R(C)*. Za kordinatu C se kaže da je neovisna ako naziv ne predstavlja sinonim kordinate C'.
- Dvije osi se mogu nazvati istim ako su im nazivi isti i ako su nazivi svih pripadajućih koordinata isti.
- Ako dvije osi $X_1 = \langle C_{11}, C_{12}, ..., C_{1n} \rangle$ i $X_2 = \langle C_{21}, C_{22}, ..., C_{2n} \rangle$ imaju iste nazive ali različite kordinate tada se mogu spojiti u jednu na način $X = X_1 \cup X_2 = \langle C_{11}, C_{12}, ..., C_{1n}, C_{21}, C_{22}, ..., C2n \rangle$.
- Os X može se razdvojiti na dvije osi X' i X'' na način da se razdvoji kordinatni skup X tako da je $X = X' \cup X''$.



Slika 5.6: Primjer višedimenzijskog prostora

Analizom dobivenih izlaznih struktura dostupnih transformacija OWL ontologija može se doći do zaključka da niti jedna nije podobna za primjenu algoritma za evidencijsko zaključivanje. Razloga za to je više, a neki od njih su ograničenja minimalnog broja nasljednika, te raspored podataka kod B+ i UB stabla, povratna veza i višestruki broj roditelja kod grafova i n-dimenzionalnog prostora resursnog modela. Jedina izlazna struktura koja bi se mogla eventualno iskoristiti je ona koja nastaje prilikom transformacije OWL ontologije u konceptne mape, s obzirom da postoji mogućnost dobivanja općeg stabla. Međutim, opće stablo se dobije samo u slučaju kada je i ontologija, odnosno, graf kojim je predstavljena postavljen tako da zadovoljava pravila općeg stabla. Situacije kada je graf tako postavljen događaju se rijetko pa se samim time nije dobro na to osloniti.
Kako bi se izbjegli nedostaci navedenih metoda izrađen je model transformacije ontološke strukture u taksonomsku strukturu koji je prikazan na Slici 5.7.



Slika 5.7: Model transformacije ontološke strukture u taksonomsku strukturu

Izrađeni model prikazan na Slici 5.7 podijeljen je na tri glavna dijela:

- Ulaz i priprema podataka.
- Obrada podataka.
- Primjena i analiza.

Ulaz i priprema podataka

Prvi dio modela transformacije, ulaz i priprema podataka, dodatno se dijeli na dva dijela, a to su učitavanje ontologije i primjena zaključivača. Kako algoritam za evidencijsko zaključivanje služi za ocjenjivanje nekog sustava, postrojenja, modela i slično, potrebno je što vjernije prikazati reprezentaciju ocjenjivanog sustava. To znači da je, na neki način, potrebno OWL ontologiju koja u sebi sadrži sve prednosti semantičke ekspresivnosti transformirati u jednostavniju strukturu (taksonomiju) bez bitnih gubitaka. Zbog toga, zaključivač predstavlja jedan od najbitnijih dijelova modela i u ovom slučaju primarno služi kako bi se maksimalno sačuvala ekspresivnost ontologije nad kojom se obavlja transformacija u taksonomiju. Osim već navedenog primarnog razloga upotrebe zaključivača, postoji i nekoliko drugih razloga. Provjerava se verzija učitane ontologije kao i konzistencija. Provjera konzistencije bitan je korak jer ona osigurava točnu reprezentaciju bez konfliktnih i duplih tvrdnji što uvelike umanjuje mogućnost krive interpretacije. Na primjer, ako postoji definicija klase koja u sebi sadrži instance koje predstavljaju države koje nemaju izlaz na more te ako se u nju doda instanca koja predstavlja državu s izlazom na more (npr. Hrvatska), tada takva ontologija neće proći provjeru konzistencije.

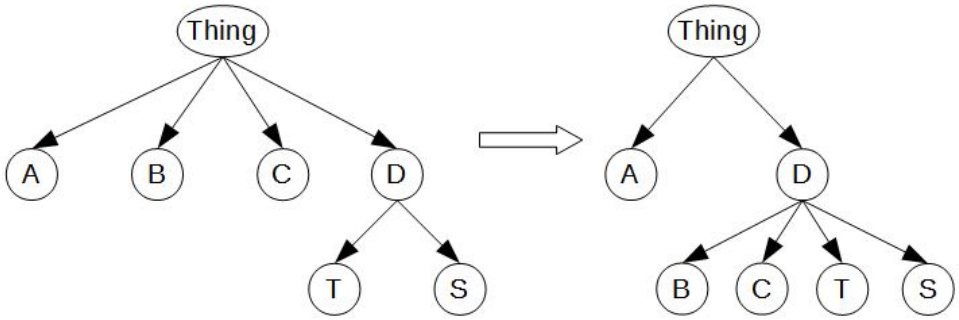
Danas su dostupni razni zaključivači kao što su Pellet, FaCT, FaCT++, Snorocket, RACER, SWRL-IQ, ELK, HermiT, TROWL i CEL [149]. U svrhu ulaza i pripreme podataka u modelu za transformaciju ontologija koristi se HermiT zaključivač iz nekoliko bitnih razloga.

HermiT zaključivač [114], [150], [151] predstavlja javno dostupan zaključivač, te je uključen u standardne biblioteke Protégé alata za razvoj ontologija. Potpuno je razvijen u skladu s OWL 2 semantikom koja predstavlja standard od 2009. godine (W3C). Temelji se na hypertableau izračunu koji omogućava izbjegavanje nedeterminističkog ponašanja što se znalo događati kod drugih vrlo popularnih zaključivača kao što su Pellet i FaCT++, a koji su temeljeni na tableau izračunu. Podržava široki skup standardnih i novih optimizacijskih metoda kako bi se poboljšale performanse zaključivanja nad stvarnim ontologijama. Također, podržava sve značajke OWL 2 jezika, uključujući i novo dodane tipove podataka kao i klasifikaciju objekata i svojstava. Testovi performansi pokazali su da HermiT zaključivač ne daje najbolje rezultate što se tiče brzine zaključivanja, ali se pokazao kao robustniji u usporedbi s ostalim zaključivačima koji su imali poteškoća sa složenijim ontologijama. Što se tiče komunikacije i upravljanja sa zaključivačem, ona se može provesti na tri načina, a to je preko ugrađenog Java sučelja, OWL API-a i komandne linije. U slučaju ove disertacije, sva komunikacija s zaključivačem obavlja se preko OWL API-a.

S obzirom da je u OWL jeziku na više načina moguće definirati klase, ovisno o načinu definicije dolazi do promjene u samom izgledu strukture prije i poslije primjene zaključivača. Prema tome, razlikuju se osnovni i izvedeni model strukture koji se također koriste i u alatima za razvoj

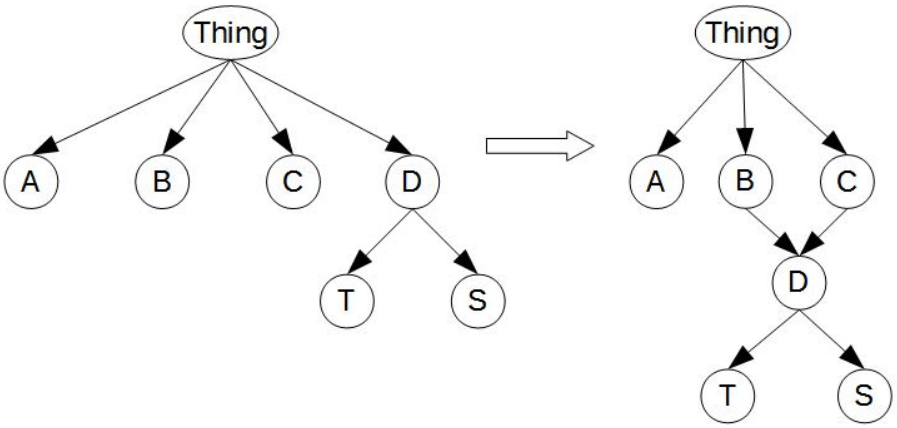
ontologija kao što je Protégé:

- Osnovni model - odnosi se na izvornu strukturu koju postavlja korisnik.
- Izvedeni model (nasljeđeni) - odnosi se na strukturu temeljenu na primjeni nekog od dostupnih zaključivača (u ovom slučaju HermiT zaključivača). Takav model vjerno prikazuje strukturu ontologije s pripadajućim instancama onako kako je zamišljena (zaključivač uzima u obzir aksiome).



Slika 5.8: Primjer osnovnog modela s definiranom unijom i izvedenog modela

Na Slici 5.8 prikazan je jednostavan primjer na kojem se vidi razlika između osnovnog modela i izvedenog modela. Klase A, B, C, T i S predstavljaju primitivne klase, dok klasa D predstavlja kompleksnu klasu, definiranu kao uniju klasa B i C ($D \equiv B \sqcup C$). S obzirom da je klasa D definirana kao skup svih instanci koje su ili članovi klase B ili članovi klase C, u izvedenom modelu zaključivač je postavio klasu D kao nadklasu klasama B i C.



Slika 5.9: Primjer osnovnog modela s definiranim presjekom i izvedenog modela

Slično kao na prethodnoj slici, Slika 5.9 prikazuje jednostavan primjer definiranja presjeka. Klase A, B, C, T i S predstavljaju primitivne klase, dok klasa D predstavlja kompleksnu klasu, definiranu kao presjek klasa B i C ($D \equiv B \sqcap C$). S obzirom da je klasa D definirana kao skup svih instanci koje su članovi i klase B i klase C, u izvedenom modelu zaključivač je postavio klasu D kao podklasu klasa B i C.

Struktura prikazana u izvedenom modelu je struktura koja će se dalje obrađivati u dijelu obrade podataka modela za transformaciju ontologije.

Obrada podataka

Nakon pripreme podataka, izvedenu strukturu iz zaključivača potrebno je transformirati tako da zadovoljava sve zahtjeve za primjenu algoritma evidencijskog zaključivanja.
Prema tome, obrada podataka dijeli se na tri ključna dijela:

- Stvaranje graf strukture podataka iz izvedene strukture.
- Primjena algoritma za transformaciju.
- Prilagodba za primjenu algoritma evidencijskog zaključivanja.

Stvaranje graf strukture

Stvaranje graf strukture podataka sastoji se od postupka parsiranja XML dokumenta dobivenog kao rezultat zaključivača. U tom dokumentu nalaze se sve klase ontologije zajedno s pripadajućim instancama.

```
...
<SubClassOf>
      <Class abbreviatedIRI=":A"/>
      <Class IRI="Thing"/>
</SubClassOf>
...
<SubClassOf>
      <Class abbreviatedIRI=":B"/>
      <Class IRI="A"/>
</SubClassOf>
...
<ClassAssertion>
   <Class abbreviatedIRI=":A"/>
   <NamedIndividual abbreviatedIRI=":Ind1"/>
</ClassAssertion>
...
```

Slika 5.10: Primjer XML dokumenta kao rezultat pokretanja zaključivača

Na Slici 5.10 nalazi se skraćeni primjer XML dokumenta koji se dobije nakon primjene zaključivača. Jasno se može vidjeti način zapisivanja izračunate hijerarhije i pripadajućih instanci. Prema tome, iz primjera se može vidjeti da je klasa *A* podklasa klase *Thing*, klasa *B* je podklasa klase *A* i da instanca *Ind1* pripada klasi *A*. Kada se pogleda sadržaj rezultirajućeg XML dokumenta, moguće je doći do zaključka da je puno toga izgubljeno s obzirom da dokument sadrži samo hijerarhiju i pripadajuće instance bez aksioma i ostalih definicija koje čine ontologiju vrlo ekspresivnom. Međutim, prilikom izračuna hijerarhije zaključivač uzima u obzir sve aksiome i definicije izvorne ontologije i na osnovu toga izračunava točan položaj pojedine klase i pri-

padajuće instance. Kako je od složenije strukture kao što je ontologija cilj dobiti jednostavnu taksonomsku strukturu, upravo to i je razlog primjene zaključivača. Prema tome, gubitak dijela ontologije u izlaznom XML dokumentu (definicije i aksiomi) ne predstavlja nikakav problem.

Na osnovu izračunate hijerarhije, dobiveni skup čvorova može se prikazati kao usmjereni graf G:

$$G = (V, E), \tag{5.1}$$

$$E \subseteq VxV, (u, v) \in E, \tag{5.2}$$

$$a_{i,j} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{inače} \end{cases}, \tag{5.3}$$

gdje G predstavlja neprazan skup čvorova V i skup usmjerenih veza (eng. edges) $E \subseteq VxV$. Svaka usmjerena veza $(u, v) \in E$ ima svoj početak (čvor u) i kraj (čvor v). S obzirom na nekoliko mogućnosti prikaza usmjerenog grafa, odabran je prikaz pomoću matrice susjedstva $a_{i,j}$. Pseudo kod za stvaranje hijerarhijske strukture iz rezultata zaključivača dan je u Algoritmu 1.

Algoritam 1: Stvaranje hijerarhije

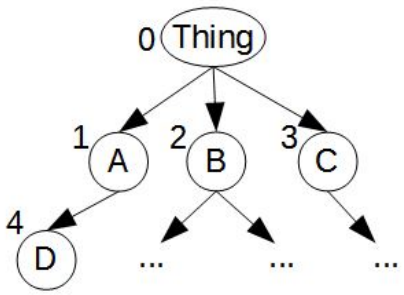
```
Učitavanje XML datoteke;
LE - lista elemenata;
H - lista čvorova hijerarhije;
nl - svi SubClassOf elementi XML dokumenta;
nl_pod - svi SubClassOf podcovori;
for i = 0 to nl.length do
    for j = 0 to nl_pod.length do
        if tip čvora == ELEMENT_NODE then
            dohvati naziv čvora;
            dodaj u LE;
        end
    end
    if H.length == 0 then
        dodaj LE[1];
        listi nasljednika dodaj LE[0];
    end
    if H.length != 0 then
        if H ne sadrži LE[1] then
            dodaj LE[1];
            listi nasljednika dodaj LE[0];
        end
        if H sadrži LE[1] then
            dohvati listu nasljednika za čvor LE[1];
            listi nasljednika dodaj LE[0];
        end
    end
end
```

Međutim, nakon učinjenog prvog koraka postupak stvaranje točne hijerarhije nije gotov. S obzirom da su čvorovi ontologije, uključujući i njihove nasljednike, razbacani bez nekog reda, potrebno je točno indeksirati sve čvorove kako bi se mogla uspješno napraviti reprezentacija ontologije u graf strukturi podataka. Postupak indeksiranja prikazan je u pseudo kodu Algoritam 2.

Algoritam 2: Indeksiranje čvorova

```
I_N - lista naziva cvorova;
I_B - lista indeksa za cvorove;
L_N - lista nasljednika za cvor;
indeks = 0;
sp = 0;
L_N = dohvati listu svih nasljednika za Thing;
Dodaj Thing u I_N;
indeks++;
for i = 0 to L_N.length do
    dodaj naziv u I_N;
    dodaj indeks u I_B;
    indeks++;
end
sp=indeks; for i = 1 to sp.length do
    L_N = dohvati nasljednike za I_N[i];
    if L_N.length != 0 then
        dodaj naziv u I_N;
        dodaj indeks u I_B;
        indeks++;
    end
    sp = indeks;
end
```

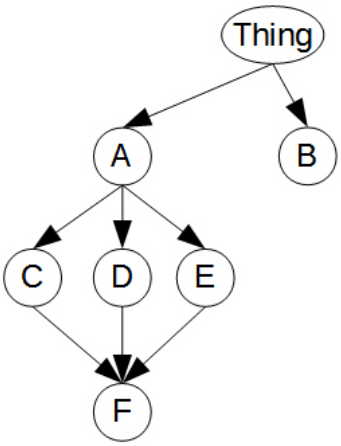
Na način kako je to prikazano u Algoritamu 2 sastavljanje hijerarhijske strukture kreće od korijenskoga čvora Thing i nastavlja se preko njegovih nasljednika do kraja strukture što rezultira točno poredanom graf strukturom kao što je prikazano na Slici 5.11.



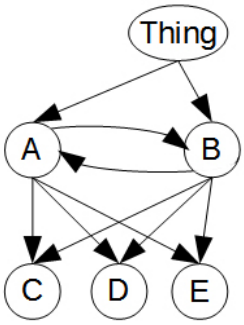
Slika 5.11: Primjer indeksiranje graf strukture

Primjena algoritma za transformaciju

Stvorena struktura iz prethodnog koraka predstavlja točnu reprezentaciju ontologije, međutim, u određenim situacijama pojavljuju se problemi zbog kojih nije moguće primijeniti algoritam evidencijskog zaključivanja jer nisu zadovoljna svojstva taksonomije. To se odnosi na glavne dvije skupine problema, problem višestrukih roditelja i problem povratne veze (jednakost klasa).



Slika 5.12: Problem višestrukih roditelja



Slika 5.13: Problem povrtne veze

Višestruki roditelji nastaju ukoliko korisnik definira višestruke roditelje određene klase ili ukoliko se radi o kompleksnoj klasi koja je definirana kao presjek dvije ili više klasa. Povratna veza, iako se u pravilu izbjegava, nastaje ukoliko korisnik definira da su dvije klase jednake ili ukoliko zaključivač izračuna da su dvije klase jednake. Osim veze između takve dvije klase, nastaje i problem križanja gdje je svaka od jednakih klasa roditelj istoj ili skupini klasa. Bitno je napomenuti da se ciklusi u ontologijama mogu dogoditi jedino između dvije klase koje su definirane kao jednake uglavnom na istoj razini. Drugim riječima, ciklus u ontologiji razlikuje se od ciklusa koji se inače može dogoditi u grafu i koji može dovesti do nemogućnosti stabilizacije strukture.

Logički gledano to i ima smisla, jer klase s pripadajućim podklasama definiraju se tako da podklase specijaliziraju svoju nadklasu. Ako se primjerice opisuje proizvodni proces, opisuje se tako da se krene od konačnog rezultata prema manjim dijelovima koji ga čine. Stoga nema smisla da jedan mali dio u proizvodnom procesu ima povratnu vezu na klasu koju čini. U konačnici, može se zaključiti kako bi zajednički problem svih navedenih situacija zapravo bio problem višestrukih roditelja. Glavni zadatak algoritma za transformaciju je pojednostavljenje dijelova hijerarhije prikazanih na Slikama 5.12 i 5.13, a koji nisu dozvoljeni u taksonomiji. Kako bi zahtjevi taksonomije bili zadovoljeni, mogu se definirati dva osnovna pravila koja je potrebno pratiti:

- Pravilo acikličke strukture - iako ontološka struktura uglavnom teži acikličkoj strukturi, u određenim situacijama mogu se pojaviti ciklusi koje je potrebno ukloniti.
- Pravilo jednog roditelja - kod ontologija (i ostalih sličnih struktura kao primjerice predmetne mape) vrlo često se pojavljuje nekoliko roditelja za jedan čvor. Takve slučajeve potrebno je izolirati i na odgovarajući način riješiti.

S obzirom da je moguće definirati dva tipa klasa, primitivnu i kompleksnu, i da su iz XML dokumenta koji se dobije iz zaključivača dostupne samo informacije o strukturi i pripadajućim instancama, potrebno je na neki način dohvatiti informacije o izvornoj strukturi. U tu svrhu, u algoritam je potrebno proslijediti i XML dokument u koji je spremljena izvorna struktura, a primjer kako taj dokument izgleda prikazan je na Slici 5.14.

```
...
<Declaration>
    <ObjectProperty IRI="#hasNesto2"/>
</Declaration>
<EquivalentClasses>
    <Class IRI="#H"/>
    <ObjectIntersectionOf>
      <Class IRI="#B"/>
      <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasNesto"/>
        <Class IRI="#B"/>
      </ObjectSomeValuesFrom>
    </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#A"/>
    <ObjectIntersectionOf>
      <Class IRI="#B"/>
      <Class IRI="#C"/>
    </ObjectIntersectionOf>
</EquivalentClasses>
...
```

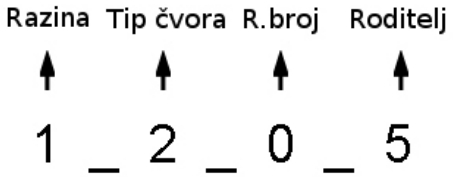
Slika 5.14: Primjer XML dokumenta izvorne strukture

Na Slici 5.14 prikazan je dio XML dokumenta izvorne strukture iz kojeg se mogu vidjeti definicije dvije kompleksne klase H i A gdje je klasa H definirana kao presjek klase B i ano-

nimne klase (restrikcija na klasu B) i klasa A koja je definirana kao presjek klasa B i C. Ako se za primjer uzme klasa A i ako se na to još doda da je klasa A podklasa klase G, ukupan broj roditelja bit će tri (B, C i G). Kada bi se gledala samo hijerarhija koju je izračunao zaključivač vidjelo bi se da su B, C i G klase zaista roditelji klase A, međutim, razlog zbog čega su baš one roditelji ostao bi nepoznanica. To se ne smije dogoditi s obzirom da se čvorovi s višestrukim roditeljima ne rješavaju jednako u svim slučajevima. Iz tog razloga, kombinacijom XML dokumenata koji predstavljaju izvornu strukturu koja sadrži sve definicije i aksiome i XML dokumenta iz zaključivača moguće je prepoznati radi li se o primitivnim klasama ili kompleksnim kao u primjeru.
Prilikom restrukturiranja grafa kako bi zadovoljio svojstva taksonomije ostaje prostora i za dodatnu optimizaciju strukture. Optimizacija strukture treba se provesti ponajviše zbog primjene evidencijskog zaključivanja. Kako je ukupna ocjena evidencijskog zaključivanja vrlo bitna jer može utjecati na određene promjene u sustavu koji se ocjenjuje treba pripaziti da svaka klasa predstavlja zasebni dio sustava. Stoga se optimizacija odnosi na brisanje nepotrebnih čvorova ili na redukciju veza s ciljem postizanja što točnije ocjene na kraju evidencijskog zaključivanja. Rezultat optimizacije trebao bi rezultirati apsolutno pojednostavljenom i čistom strukturom.
O optimizaciji strukture i algoritmu transformacije detaljno će se govoriti u nastavku disertacije.

Prilagodba za primjenu algoritma evidencijskog zaključivanja

Zadnji korak u dijelu obrade podataka je prilagodba za primjenu algoritma za evidencijsko zaključivanje. Kako bi se evidencijsko zaključivanje moglo pravilno primijeniti, nad taksonomijom je potrebno napraviti indeksiranje čvorova kako bi se moglo točno znati o kojem se tipu čvora radi. U poglavlju 4 navedeno je da unutar strukture nad kojom se primjenjuje algoritam evidencijskog zaključivanja mogu postojati tri tipa čvorova, opći atribut, međuatribut i osnovni atribut. Na opći atribut i međuatribut se može gledati kao na isti tip čvora, gdje opći atribut predstavlja ukupnu ocjenu nekog ocjenjivanog sustava, dok međuatributi predstavljaju skupove osnovnih atributa koji se ocjenjuju, odnosno, međuatribut se može shvatiti kao opći atribut niže razine. Osnovni atributi predstavljaju objekte ocjenjivanja, čiji skup ocjena direktno utječe prvo na međuatribut kojem pripadaju i u konačnici na ukupnu ocjenu općeg atributa.
Ako se pogleda taksonomija dobivena transformacijom ontologije, tada se može vidjeti da strukturu čine klase s jednom korijenskom klasom *Thing* i instance. S obzirom da instance predstavljaju objekte klasa, one se mogu izjednačiti s osnovnim atributima, dok se klase mogu izjednačiti s međuatributim, odnosno općim atributom za klasu *Thing*.



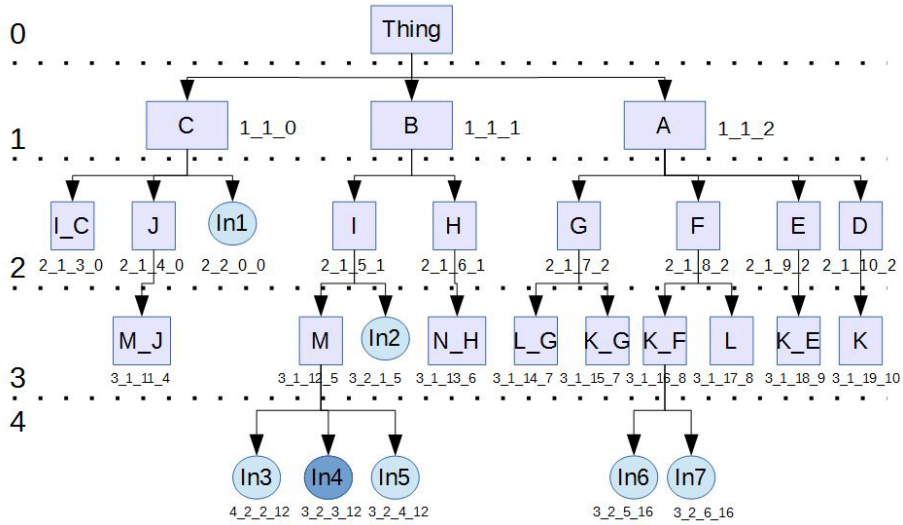
Slika 5.15: Primjer indeksiranja čvorova

Slika 5.15 prikazuje način indeksiranja čvorova taksonomije kao dio prilagodbe za primjenu evidencijskog zaključivanja. Prvi broj predstavlja razinu u taksonomiji na kojoj se nalazi promatrani čvor. Drugi broj predstavlja tip čvora, gdje su opći atributi zajedno sa međuatributima označeni brojem 1, dok se osnovni atributi označavaju brojem 2. Treći broj predstavlja redni broj pojavljivanja čvora određenog tipa. I na kraju, Zadnji broj predstavlja oznaku roditelja promatranog čvora. Ovakav način indeksiranja nije jedini, te se ista stvar može učiniti i na druge načine. Međutim, na ovakav način, otvara se mogućnost primjene različitih upita po nekim od navedenih parametara ukoliko se struktura spremi u neku od baza podataka.

Algoritam 3: Indeksiranje čvorova

```
red - lista posjećenih čvorova;
non - ukupni broj čvorova;
posjeceni - polje za označavanje posjećenih čvorova;
i - indeks promatrani čvor;
dodaj u posjeceni pocetni cvor;
while red != prazan do
    if i == Thing then
        dodaj čvor;
        dodaj redni_broj;
        redni_broj++;
        dodaj razinu;
        razina++;
    end
    if i != Thing then
        razina = dohvati razinu čvora i;
        razina+=1;
        dodaj redni_broj;
        redni_broj++;
    end
    if i == Thing then
        djeca = dohvati sve nasljednike čvora i;
        for k = 0 to djeca.length do
            dodaj čvor;
            dodaj razinu;
            dodaj roditelja;
        end
    else
        djeca = dohvati sve nasljednike čvora i;
        for k = 0 to djeca.length do
            dodaj čvor;
            dodaj razinu;
            dodaj roditelja;
        end
    end
    while i <= non do
        if posjeceni[i] == 0 then
            dodaj u red i;
            posjeceni[i]=1;
        end
        i++;
    end
end
```

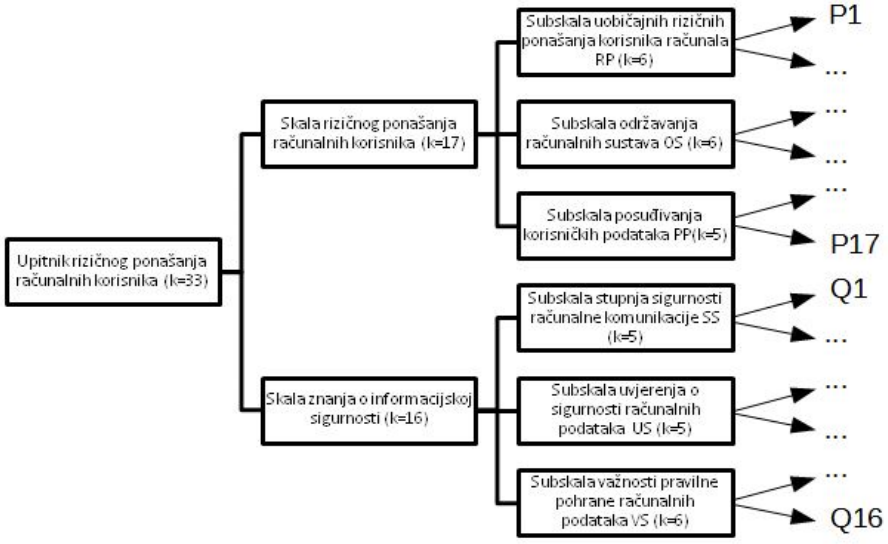
Algoritam 3 prikazuje pseudokod za indeksiranje taksonomske strukture kojim se dobije struktura prikazana na Slici 5.16.



Slika 5.16: Primjer strukture za evidencijsko zaključivanje

Primjena i analiza

S obzirom na postupak primjene evidencijskog zaključivanja navedenog u poglavlju 4, primjer primjene može se prikazati na taksonomiji o informacijskoj sigurnosti korisnika na Internetu.



Slika 5.17: Taksonomija o informacijskoj sigurnosti korisnika na Internetu

Slika 5.17 prikazuje taksonomiju koja opisuje informacijsku sigurnost korisnika na Internetu [152]. Opisana je sa sveukupno devet međuatributa i 33 osnovna atributa koji su zapravo objekti ocjenjivanja prilikom primjene evidencijskog zaključivanja. Osnovni atributi P1 - P17 i Q1 - Q16 predstavljaju različita pitanja o sigurnosti za koje se dodjeljuju ocjene na temelju kojih evidencijsko zaključivanje izračunava ukupnu ocjenu za opći atribut *"Upitnik rizičnog ponašanja računalnih korisnika"*. Za svaki osnovni atribut dodjeljuju se ocjene od 1 - 5 kako je

i opisano u poglavlju 4 koje su predstavljene kao loša, dovoljna, prosječna, dobra i odlična znakovima (P, I, A, G, E). Nesigurnost, odnosno, neodređenost se računa ovisno o danim ocjenama za pojedini osnovni atribut.

Tablica 5.4: Izračun evidencijskog zaključivanja

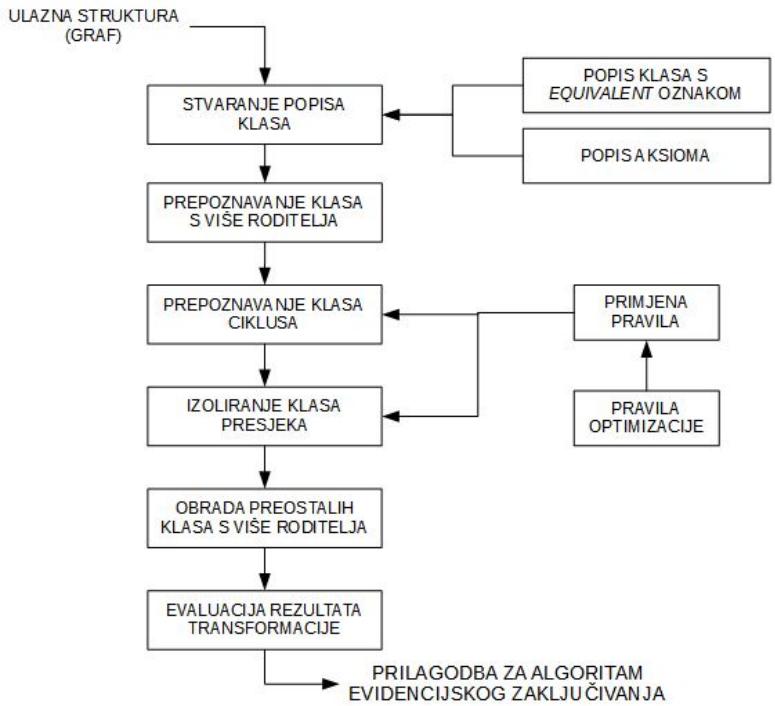
Ontološki nazivi	**Instanca**	**Podklasa**	**Podklasa**	**Thing**	
ER nazivi	**Osnovni atribut**	**Međuatribut**	**Međuatribut**	**Opći atribut**	**Konačna ocjena**
P1, P4	G	G (0.371)			
P2, P3, P5	E	E (0.629)			
P6	I	I (0.156)	I (0.085)		
P7, P8	G	A (0.156)	A (0.045)		
P14	E	G (0.344)	G (0.222)		
P16	E	E (0.344)	E (0.618)		
P17	A		H (0.031)	P (0.090)	
P9	-	I (0.138)		I (0.038)	
P10, P12 - P15	E	E (0.747)		A (0.114)	U = 0.772
P11	I	H (0.115)		G (0.370)	(0.765 - 0.779)
Q1, Q3	E	P (0.191)		E (0.375)	
Q2	G	A (0.191)		H (0.013)	
Q4	P	G (0.191)	P (0.194)		
Q5	A	E (0.429)	A (0.194)		
Q6, Q8	P	P (0.409)	G (0.481)		
Q7, Q10	A	A (0.409)	E (0.131)		
Q9	G	G (0.182)			
Q11 - Q16	G	G (1.00)			

U tablici 5.4 prikazan je postupak izračuna ukupne ocjene evidencijskog zaključivanja gdje se prvo ocjenjuju osnovni atributi, a zatim se agregacijskim postupkom prikazanim u poglavlju 4 računaju vrijednosti podskala predstavljenih međuatributima. U tablici se može primijetiti i ocjena H koja predstavlja nesigurnost za dani opći atribut P9. Nakon izračuna ocjena svih međuatributa, računa se konačna ocjena (eng. utility number) s vrijednostima u_{min}(y) = 0.765, u_{max}(y) = 0.779 i u_{avg}(y) = 0.772 prema izrazima 4.26, 4.27 i 4.28. S obzirom da je u navedenom slučaju stupanj uvjerenja β_H != 0 zbog unesene nesigurnosti za opći atribut P9, uzima se srednja vrijednost u_{avg}(y) = 0.772 kao konačna ocjena. Dobivena konačna ocjena dalje se uspoređuje s nekim referentnim vrijednostima određenim za ocjenjivani sustav. To može ručno napraviti ekspert koji ocjenjuje sustav ili automatski primjenom inteligentnih agenata [153].

Poglavlje 6

Algoritam transformacije ontologije i optimizacija dobivene taksonomije u skladu sa zahtjevima evidencijskog zaključivanja

Najbitniji dio modela za transformaciju upravo je algoritam za transformaciju koji omogućava prilagodbu ontološke strukture zapisanu u OWL jeziku u taksonomsku strukturu koja zadovoljava uvjete za primjenu evidencijskog zaključivanja.



Slika 6.1: Koraci algoritma za transformaciju OWL ontologije

Na Slici 6.1 prikazani su koraci algoritma za transformaciju koji se mogu podijeliti u tri glavna dijela:

- Stvaranje popisa klasa ulazne strukture.
- Prilagodbe graf strukture podataka.
- Evaluacija rezultata transformacije.

Stvaranje popisa klasa ulazne strukture

Nakon što se dobiju izračunati podaci iz HermiT zaključivača stvara se hijerarhijska struktura što je prikazano Algoritmima 1 i 2. S obzirom da takva struktura može sadržavati problematične situacije prikazane na Slikama 5.12 i 5.13, prvi korak u rješavanju tih problema je stvaranje popisa svih klasa koje u tome sudjeluju. To se općenito odnosi na klase s više od jednog roditelja, a tu pripadaju i klase presjeka kao i klase koje su jednake.
Popis svih definiranih klasa dobije se iz izvornog dokumenta u kojem je definirana ontologija, a primjer takvog dokumenta prikazan je na Slici 5.14. Prilikom stvaranja popisa definiranih klasa potrebno je obratiti pozornost na definicije i aksiome, pa se stoga provjeravaju sljedeće oznake:

- ObjectIntersectionOf (presjek).
- ObjectSomeValuesFrom (egzistencijalna restrikcija).
- ObjectComplementOf (komplement).
- ObjectOneOf (enumeracija).
- ObjectUnionOf (unija).
- ObjectAllValuesFrom (univerzalna restrikcija).
- ObjectHasValue (*imaVrijednost* restrikcija).
- ObjectMinCardinality (minimalna kardinalnost).
- ObjectMaxCardinality (maksimalna kardinalnost).

Nakon prepoznavanja definiranih čvorova, potrebno je napraviti popis svih klasa koji imaju više od jednog roditelja. S obzirom da je graf prikazan matricom susjedstva, potrebno je provjeriti koji čvorovi imaju više od jedne *"true"* vrijednosti u svojim poljima. To je prikazano u Algoritmu 4.

Algoritam 4: Prepoznavanje čvorova s više roditelja

```
brojac=0;
for i = 0 to cvorovi.length do
    for j = 0 to cvorovi.length do
        if matrica[i][j] then
            brojac++;
        end
    end
    if brojac>1 then
        dodaj čvor;
    end
    brojac=0;
end
```

Nakon ovog koraka poznate su sve definirane klase, te su također poznate i sve klase koje imaju više od jednog roditelja. Na taj način, točno se može odrediti koja od klasa s više roditelja pripada definiranim klasama, a koja ne.

Prilagodba graf strukture podataka

Kako bi se transformacija uspješno provela, potrebno je definirati određena pravila. Za početak, potrebno je definirati tipove klasa koje se mogu pojaviti u OWL ontologiji:

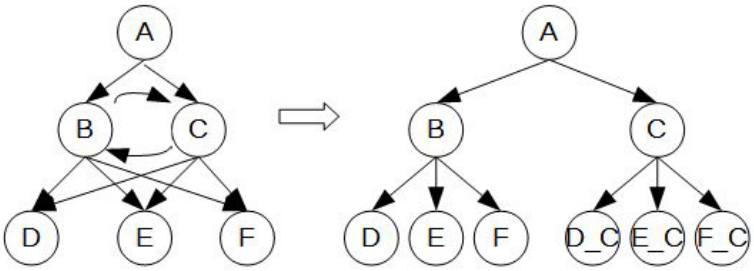
- Definicija 1. Klasa se može nazvati definiranom ako u sebi sadrži dovoljan i nužan uvjet.
- Definicija 2. Klase se može nazvati primitivnom ako u sebi sadrži samo nužan uvjet.
- Definicija 3. Klasa se može nazvati anonimnom ako je definirana korištenjem nekih od mogućih ograničenja.

Ranije u disertaciji opisana je razlika između definirane i primitivne klase, a ona se odnosi na način dodjele instanci. S obzirom da su instance objekti ocjenjivanja u ovom slučaju, predstavljaju bitan element koji se ne smije izgubiti u transformaciji. Također, anonimna klasa sadrži instance prema definiranoj restrikciji, pa je u procesu transformacije bitno znati razlučiti radi li se o anonimnoj klasi ili ne.
Kako bi se primjenom evidencijskog zaključivanja dobila što točnija ocjena, potrebno je prilikom transformacije što je moguće više umanjiti stvaranje nepotrebnih klasa kopija. Prema tome, definirana su dodatna dva pravila optimizacije:

1. Ako su dvije ili više klasa definirane kao jednake, za *n-1* klasa, gdje *n* predstavlja ukupni broj jednakih klasa, izlazne relacije se brišu.
2. Klasa koja predstavlja rezultat presjeka dvije ili više klasa, ovisno o tipu klase prema definicijama 1-3 može se kompletno ukloniti iz grafa.

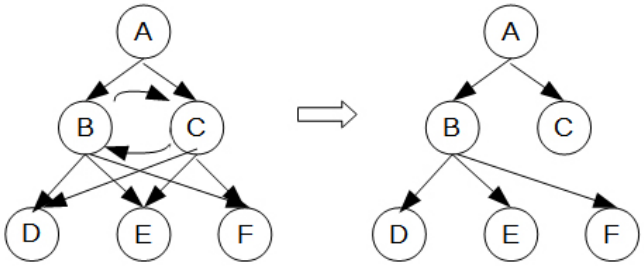
Pravilo jednakih klasa

Ako su dvije ili više klasa za koje vrijedi definicija 1 definirane kao jednake (ciklička veza), klase nasljednici se kopiraju.

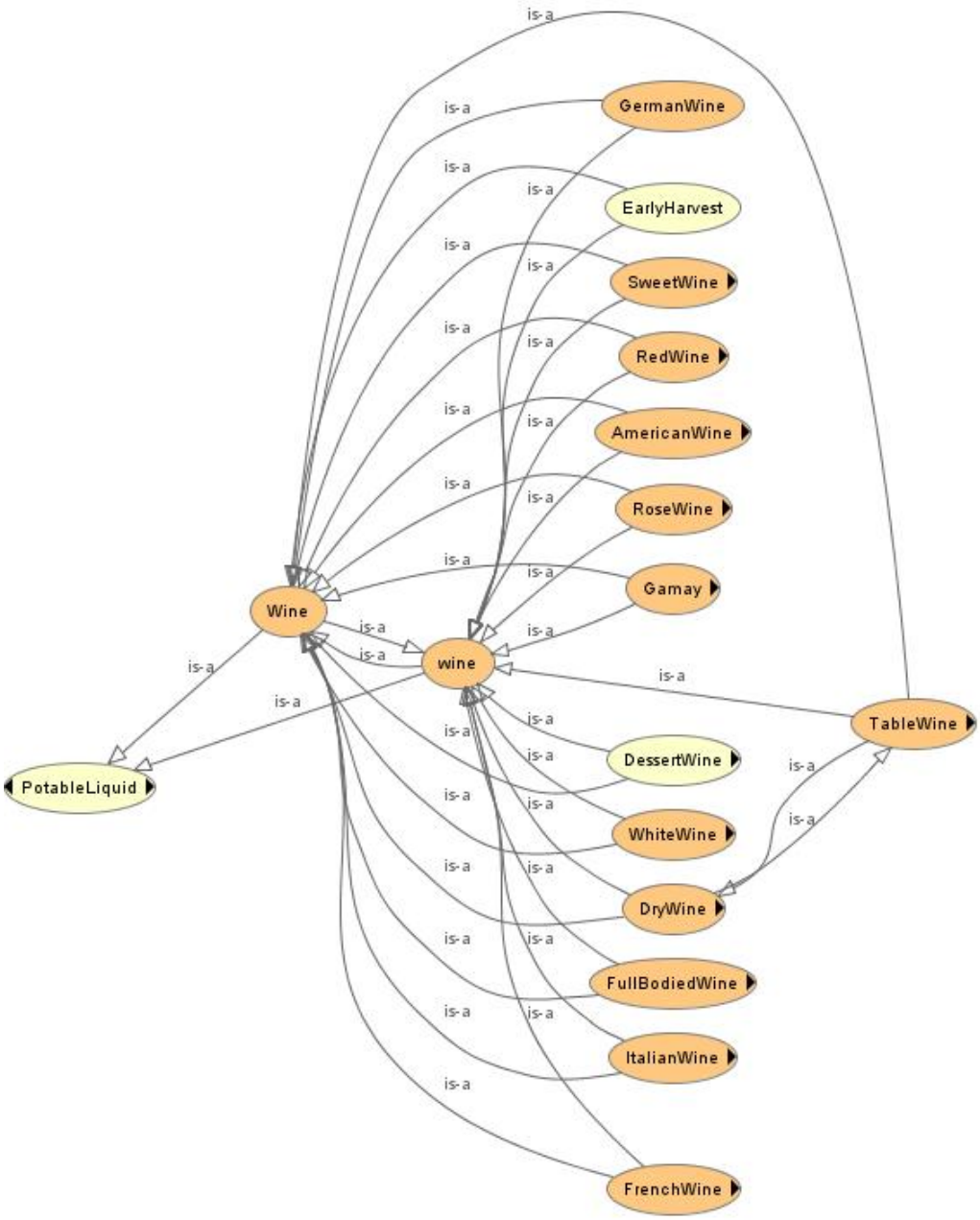


Slika 6.2: Prvo pravilo transformacije

Međutim, ako su klase definirane kao jednake kao što je to prikazano na Slici 6.2, to znači da klase *B* i *C* predstavljaju istu stvar. Prema tome, može se primijeniti prvo pravilo optimizacije gdje se relacije jedne od klasa jednakosti brišu što je prikazano na Slici 6.3. Time neće doći do gubitka, a prilikom ocjenjivanja u algoritmu evidencijskog zaključivanja, ocjena klase kojoj nisu obrisane relacije može se dodijeliti klasi kojoj jesu. O tome hoće li će se ocjena dodati drugoj klasi jednakosti može odlučiti sam ekspert koji će koristiti evidencijsko zaključivanje, a odluka može ovisiti o tome imaju li jednake klase zajedničkog ili različitog roditelja.



Slika 6.3: Prvo pravilo transformacije nakon primjene prvog pravila optimizacije



Slika 6.4: Primjer definicije jednakih klasa

Na Slici 6.4 prikazana je definicija dvije jednake klase *wine* i *Wine* u OWL ontologiji koja opisuje vrste vina. Jednake klase nalaze se u XML dokumentu koji se dobije nakon primjene HermiT zaključivača, a izgled dokumenta prikazan je na Slici 6.5. Također, Algoritam 5 prikazuje pseudokod za prepoznavanje jednakih klasa iz dokumenta prikazanog na Slici 6.5.

```
...
EquivalentClasses( <http://www.co-ode.org/ontologies/wine/wine.owl#Wine>
        <http://www.co-ode.org/ontologies/wine/wine.owl#wine> )
...
```

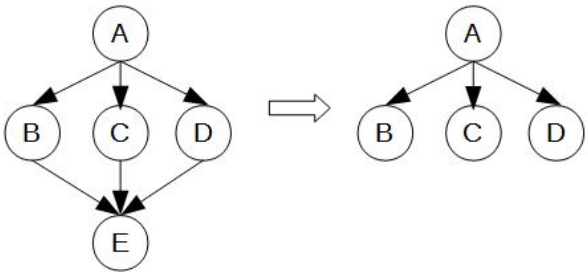
Slika 6.5: Primjer XML dokumenta iz zaključivača s popisom jednakih klasa

Algoritam 5: Prepoznavanje jednakih klasa

```
br = učitaj XML dokument;
while linija = br.readLine() != null) do
    if oznaka == EquivalentClasses then
        dodaj čvor;
    end
end
```

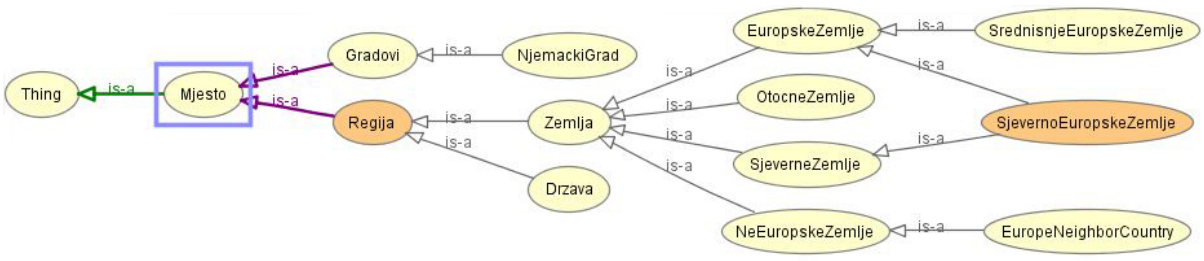
Pravilo presjeka klasa

Ako za klasu vrijedi definicija 1 i ako je rezultat presjeka dvije ili više klasa, klasa se briše kao i sve ulazne relacije.



Slika 6.6: Primjer definicije presjeka klasa

Na Slici 6.6 prikazan je klasični primjer presjeka više klasa, gdje je klasa E definirana kao presjek klasa B, C i D, odnosno, $E = B \sqcap C \sqcap D$. Uz pretpostavku da je raspored instanci po klasama $B = (I1, I2, I3)$, $C = (I1, I2)$ i $D = (I1, I2, I4)$, klasa E će s obzirom da je definirana koristeći nužan i dovoljan uvjet, osim inicijalno dodijeljenih instanci, sadržavati i instance $I1$ i $I2$. Ako bi se čvor kopirao čime bi se riješio problem višestrukih roditelja, to bi značilo kako bi svaka od klasa B, C i D za podklasu imale klasu E. S gledišta same strukture sve bi trebalo biti u redu, međutim, kako se na toj strukturi treba primijeniti algoritam evidencijskog zaključivanja koji na temelju instanci klasa računa ukupnu ocjenu nekog sustava, jasno je da će kopije istih instanci višestruko utjecati na veličinu ukupne ocjene. Jedna od takvih situacija prikazana je na Slici 6.7.

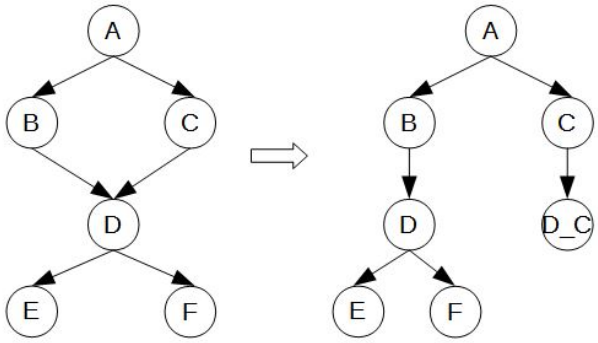


Slika 6.7: Primjer definicije presjeka klasa

Ako se pretpostavi da klase *EuropskeZemlje* i *SjeverneZemlje* sadržavaju instancu *Svedska*, tada će i klasa *SjevernoEuropskeZemlje* sadržavati instancu *Svedska*. Isto tako, ako pretpostavimo da se ocjenjuje standard zemalja, ukoliko se napravi kopija klase *SjevernoEuropskeZemlje* ocjena za *Svedska* će višestruko utjecati na ukupnu ocjenu. S obzirom da klasa *SjevernoEuropskeZemlje* sadrži sve instance koje se već nalaze u klasama roditeljima, ona se može ukloniti prema drugom pravilu optimizacije.
Međutim, brisanje bilo koje klase prilikom transformacije dovodi do određenog gubitka u rezultirajućoj strukturi. Stoga je potrebno je napraviti dodatne provjere kako bi brisanje određenog čvora bilo u potpunosti sigurno s minimalnim gubicima.

Podpravilo 1

Ako klasa presjeka ima definirane ili naslijeđene (na temelju zaključivanja) nasljednike, tada neće doći do brisanja. Takva situacija prikazana je na Slici 6.8.

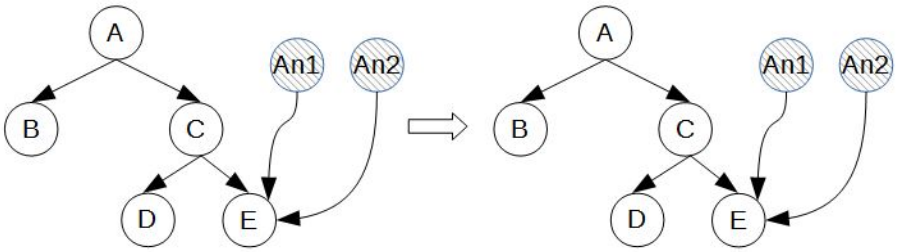


Slika 6.8: Podpravilo 1

Na Slici 6.8 klasa D definirana je kao presjek klasa B i C ($D \equiv B \sqcap C$), te ima dodijeljena dva nasljednika E i F ($E \sqsubseteq D$, $F \sqsubseteq D$). S obzirom da klase E i F mogu imati dodatna značenja i sadržavati bitne instance, brisanje klase D dovelo bi do prevelikog gubitka u rezultirajućoj strukturi. Zbog toga se klasa presjeka D kopira ($(B \sqcap C) \sqsubseteq B$, $(B \sqcap C) \sqsubseteq C$).

Podpravilo 2

Ako se klasa presjeka sastoji samo od klasa za koje vrijedi definicija 3, tada neće doći do brisanja čvora.

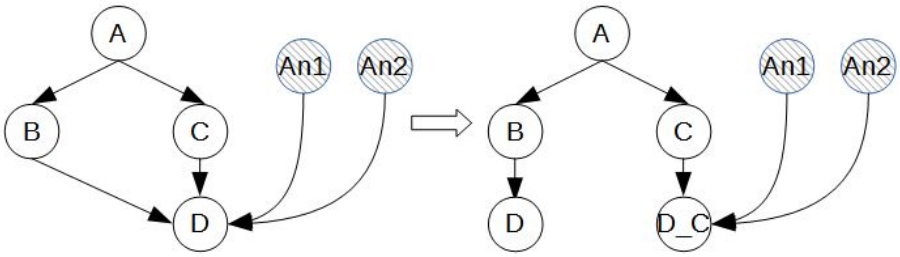


Slika 6.9: Podpravilo 2

Na Slici 6.9 klasa E definirana je kao presjek dvije anonimne klase *An1* i *An2* ($E \sqsubseteq (\forall svojstvo.S1 \sqcap \forall svojstvo.S2)$ i kao podklasa klase C ($E \sqsubseteq C$). S obzirom da se anonimne klase ne ucrtavaju u samu strukturu, već samo predstavljaju skup instanci koje zadovoljavaju zadanu restrikciju, brisanjem klase E koja u sebi sadrži instance dobivene presjekom anonimnih klasa, došlo bi do prevelikog gubitka u rezultirajućoj strukturi. Prema tome, u ovom slučaju ne dolazi do bilo kakve promjene.

Podpravilo 3

Ako se klasa presjeka sastoji samo od klasa za koje vrijedi definicija 3 i ako ima više roditelja, tada neće doći do brisanja.

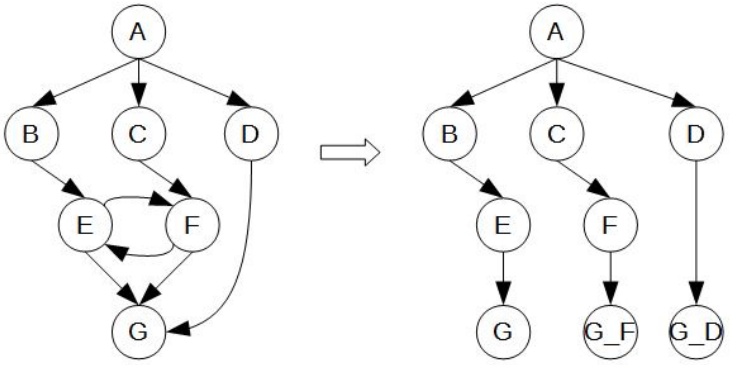


Slika 6.10: Podpravilo 3

Na Slici 6.10 klasa E definirana je kao presjek dvije anonimne klase *An1* i *An2* ($E \sqsubseteq (\forall svojstvo.S1 \sqcap \forall svojstvo.S2)$ i kao podklasa klase C i B ($E \sqsubseteq C$, $E \sqsubseteq B$). U ovom slučaju, s obzirom na višestruke roditelje doći će do kopiranja klase D kako ne bi došlo do trajnog gubitka instanci dobivenih presjekom anonimnih klasa.

Podpravilo 4

Ako klasa presjeka kao jednog od roditelja ima definiranu klasu koja ima definiranu jednakost s drugom klasom, tada neće doći do brisanja.



Slika 6.11: Podpravilo 4

Na Slici 6.11 klase E i F definirane su kao jednake ($E \equiv F$, $E \sqsubseteq B$, $F \sqsubseteq C$) dok je klasa G definirana kao presjek klasa E i D ($G \equiv E \sqcap D$) s tim da je veza klase G i F ($G \sqsubseteq F$) nastala kao rezultat jednakosti klasa E i F. U takvoj situaciji, kada bi došlo do brisanja klase E nastao bi preveliki gubitak s obzirom da jednake klase mogu imati različite roditelje kao što je prikazano u ovom slučaju, pa se klasa G kopira.

Podpravilo 5

Ovo pravilo odnosi se na raspodjelu instanci klase presjeka i njezinih roditelja. Ukoliko klasa presjeka sadrži neku instancu koja nije dio njezinih roditelja, tada se taj čvor ne smije obrisati zbog očitih gubitaka. Ukoliko je sadržaj instanci isti, i ako su provjere podpravila 1-4 prošle, tada se klasa presjeka može obrisati.

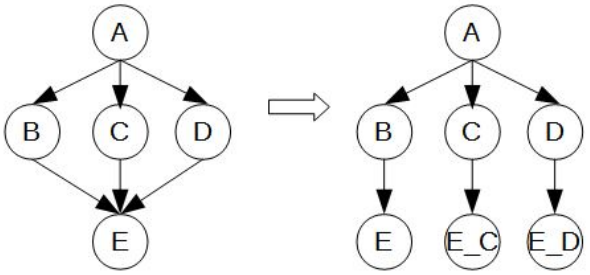
Algoritam 6: Prepoznavanje klase presjeka

```
VR - lista čvorova s više roditelja;
EK - lista definiranih klasa (dovoljni i nužni uvjeti);
AK - lista aksioma definiranih klasa;
for i = 0 to VR.length do
    if EK sadrži VR[i] then
        if AK sadrži oznaku ObjectIntersectionOf za VR[i] then
            dodaj čvor presjeka;
        end
    end
end
```

Algoritam 6 prikazuje postupak prepoznavanja klasa presjeka.

Pravilo višestrukih roditelja

Ako za klasu vrijedi definicija 2, i ako ima dvije ili više klasa roditelja, napraviti kopiju klase za *n-1* relacija, gdje *n* predstavlja ukupni broj ulaznih relacija.

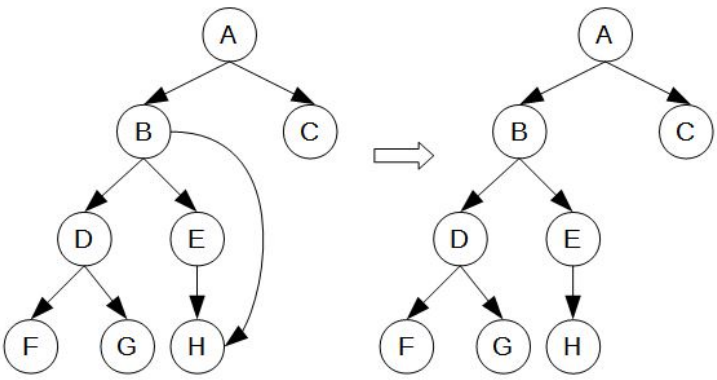


Slika 6.12: Primjer višestrukih roditelja

Na Slici 6.12 prikazana je situacija kada klasa ima višestruke roditelje. U ovom slučaju klasa E definirana je kao podklasa klasa B, C i D ($E \sqsubseteq B, E \sqsubseteq C, E \sqsubseteq D$). Ovo pravilo odnosi se na sam kraj algoritma za transformaciju. Drugim riječima, kada sve ostale provjere klasa prođu, ostatak se obrađuje na način da se klase koje imaju više roditelja kopiraju. Tu pripadaju sve klase koje su na taj način inicijalno definirane, klase koje imaju naslijeđene roditelje te sve kompleksne klase koje su definirane samo nužnim uvjetom (prema definiciji 2). Ovakav slučaj se najčešće i pojavljuje u ontologijama.

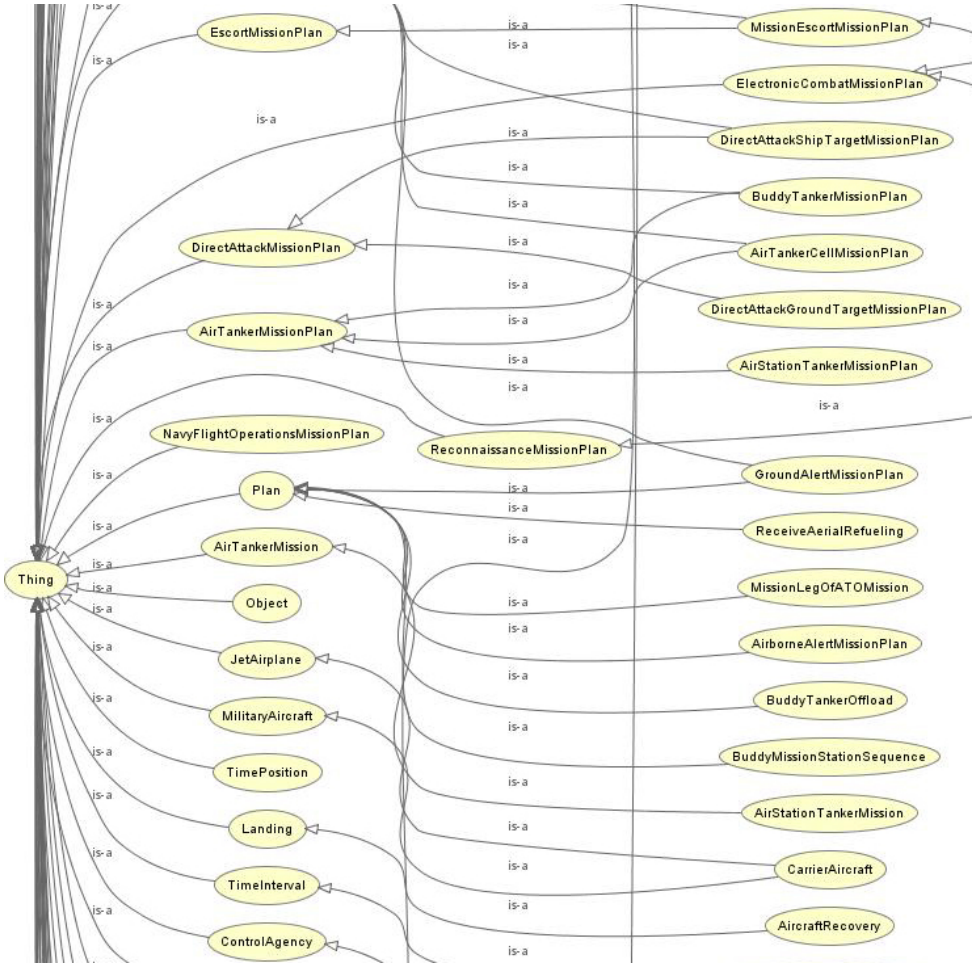
Podpravilo 1

Ako se pretpostavi da *r* predstavlja razinu stabla na kojoj se nalazi promatrana klasa, te ukoliko klasa ima višestruke roditelje gdje je jedan ili više roditelja na *r- 2* (ili više) razini, potrebno je provjeriti pripada li takva klasa istom podstablu. Ukoliko pripada istom podstablu, relacija s tom klasom se briše dok se u suprotnom kopira. Ukoliko sve klase roditelji pripadaju *r-1* razini, tada će doći do kopiranja klase. Jedna od takvih mogućih situacija prikazana je na Slici 6.13.



Slika 6.13: Podpravilo 1

Potrebno je napomenuti da se ovakve situacije gotovo nikada neće dogoditi iz razloga što su u ontologiji sve klase vezane *is-a* relacijom pa takve situacije zaključivač riješi na način koji je opisan u podpravilu 1. Također, u svim testiranjima koja su se provela u okviru ove disertacije nije se niti jednom pojavila ovakva situacija. S obzirom da nisu testirane sve moguće situacije, postoji mogućnost da se pod određenim uvjetima takva situacija i pojavi. S obzirom na to, ovo pravilo je ugrađeno u algoritam za transformaciju.



Slika 6.14: Primjer višestrukih roditelja

Na Slici 6.14 prikazan je isječak jedne ontologije na kojoj se može vidjeti primjer višestrukih roditelja klasa.

Evaluacija rezultata transformacije

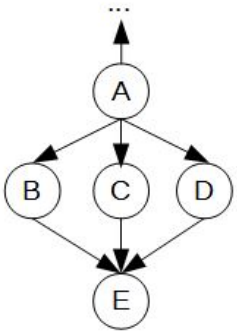
Evaluacija rezultata transformacije predstavlja zadnji potvrdni dio algoritma za transformaciju prije nego je taksonomska struktura spremna za primjenu evidencijskog zaključivanja. Detalji o evaluaciji rezultata nalaze se u sljedećem poglavlju.

6.1 Ovisnost pravila transformacije

Prema onom što je navedeno, pravila su podijeljena kako slijedi:

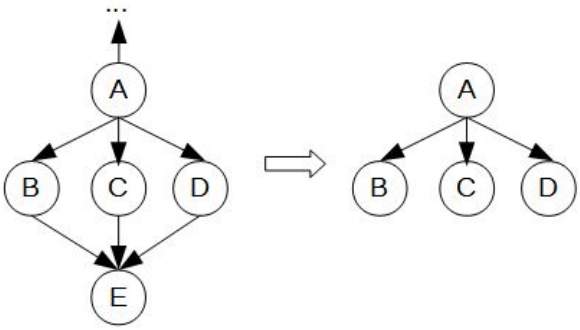
1. Pravilo jednakih klasa - Ako su dvije ili više klasa definirane kao jednake (ciklička veza), klase nasljednici se kopiraju. Na ovom pravilu moguće je primijeniti prvo pravilo optimizacije.
2. Pravilo presjeka klasa - Ako za klasu vrijedi definicija 1 i ako je rezultat presjeka dvije ili više klasa, klasa se prema drugom pravilu optimizacije briše ako niti jedno od sljedećih podpravila nije zadovoljeno:
   (a) Ako klasa presjeka ima definirane ili naslijeđene (na temelju zaključivanja) nasljednike, tada dolazi do kopiranja klase.
   (b) Ako se klasa presjeka sastoji samo od klasa za koje vrijedi definicija 3 (anonimna klasa), tada neće doći do promjene uz uvjet samo jednog roditelja.
   (c) Ako se klasa presjeka sastoji samo od klasa za koje vrijedi definicija 3 (anonimna klasa) i ako ima dva ili više roditelja, tada dolazi do kopiranja klase.
   (d) Ako klasa presjeka kao jednog od roditelja ima definiranu klasu koja ima definiranu jednakost s drugom klasom, tada će doći do kopiranja klase.
   (e) Ako sadržaj instanci klase presjeka nije jednak sa sadržajem klasa roditeljima s kojima treba biti obrisana relacija, tada će doći do kopiranja uz uvjet dva ili više roditelja.
3. Pravilo višestrukih roditelja - Ako za klasu vrijedi definicija 2 (primitivna, nužni uvjeti), i ako ima dvije ili više klasa roditelja, tada će doći do kopiranja za *n-1* relacija, gdje *n* predstavlja ukupni broj ulaznih relacija.
   (a) Ako se klasa roditelj nalazi na dvije ili više razina iznad promatrane klase i ako se nalazi u istom podstablu, relacija se briše.

Ovisnost pravila može se promatrati na dva načina. Jedan način je mogućnost izvođenja jednog pravila u ovisnosti da li je izvedeno jedno ili više drugih prije njega, dok je drugi način ovisnost izgleda rezultirajuće strukture o redoslijedu izvođenja pravila.
Vezano za prvi način, izvođenje bilo kojeg od navedenih pravila nije uvjet za izvođenje drugog. Drugim riječima, ukoliko se ontologija sastoji samo od klasa s višestrukim roditeljima, tada će se upotrijebiti samo treće pravilo. To isto vrijedi i za OWL ontologiju koja je inicijalno sastavljena tako da zadovoljava sve uvjete za primjenu algoritma za evidencijsko zaključivanje, odnosno, neće se upotrijebiti niti jedno pravilo.
Vezano za drugi način, način na koji trenutno radi algoritam za transformaciju dovest će do drugačije rezultirajuće strukture ako se promijeni redoslijed izvođenja pravila. Prema tome, za primjer se može uzeti sljedeća struktura prikazana na Slici 6.15.



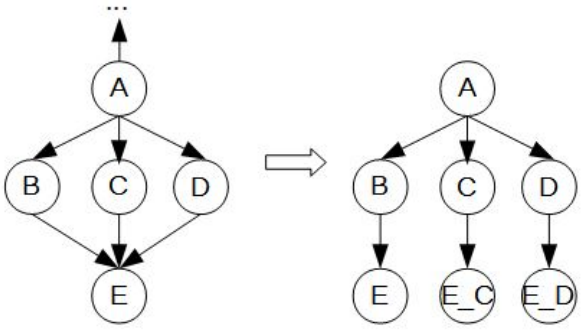
Slika 6.15: Primjer presjeka klasa

Na Slici 6.15 prikazana je definicija presjeka klasa, gdje je klasa E definirana kao presjek klasa B, C i D ($E \equiv B \sqcap C \sqcap D$). Prema trenutnom načinu na koji algoritam za transformaciju funkcionira, za navedeni slučaj bi se primjenilo drugo pravilo koje regulira slučajeve presjeka, a koje bi za rezultat dalo strukturu prikazanu na Slici 6.16.



Slika 6.16: Primjer rješavanja klase presjeka

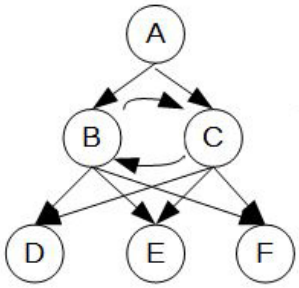
Ukoliko se u ovom slučaju primjeni treće pravilo prije drugog dobije se struktura prikazana na Slici 6.17.



Slika 6.17: Primjer rješavanja klase presjeka

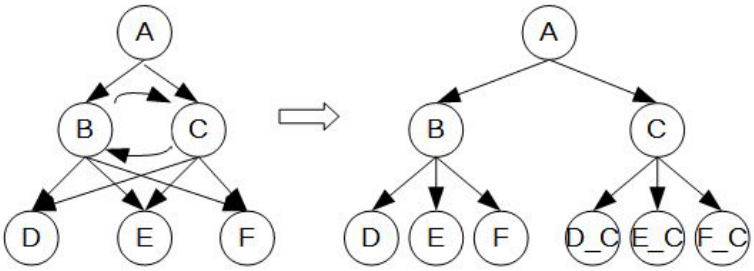
Iz rezultata je jasno vidljivo kako bi umjesto brisanja klase E došlo do kopiranja. Što se tiče konačne strukture, ona i dalje zadovoljava pravila za primjenu algoritma za evidencijsko zaklju-

čivanje, jedino što dolazi do nepotrebnog kopiranja čvorova što će u konačnici dovesti do lošije kvalitete ukupne ocjene kao rezultata evidencijskog zaključivanja.
Drugi primjer prikazan je na Slici 6.18.



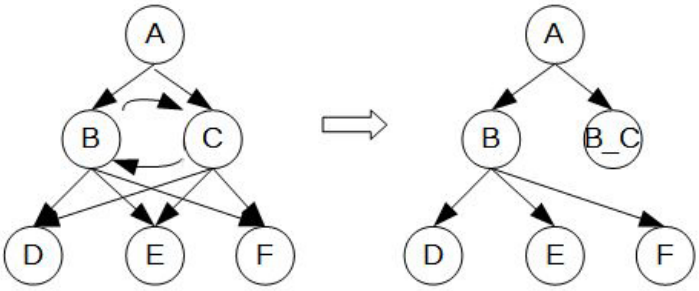
Slika 6.18: Primjer jednakih klasa

Prema trenutnom načinu na koji algoritam funkcionira, za navedeni slučaj bi se primjenilo prvo pravilo koje regulira slučajeve jednakosti klasa, a koje bi za rezultat dalo strukturu prikazanu na Slici 6.19.



Slika 6.19: Primjer rješavanja jednakosti klase

Ukoliko se u ovom slučaju primjeni treće pravilo prije prvog dobije se struktura prikazana na Slici 6.20.



Slika 6.20: Primjer rješavanja jednakosti klase

Iz rezultata je jasno vidljivo kako bi umjesto smanjivanja broja veza kod jednakih klasa došlo do nepotrebnog kopiranja klasa. Isto kao i u prošlom primjeru, konačna struktura i dalje zado-

voljava pravila za primjenu algoritma za evidencijsko zaključivanje.
Iz prikazanog, vidljivo je da se problem ovisnosti pravila odnosi na primjenu trećeg pravila, odnosno, pravila višestrukih roditelja. S obzirom na trenutno stanje, redoslijed izvođenja algoritma koji je prikazan na Slici 6.1 nužno se treba poštivati. U suprotnom će se dobiti pogrešan rezultat.

6.2 Primjena algoritma za transformaciju

Kako bi se utvrdilo radi li algoritam za transformaciju zajedno s pravilima ispravno potrebno je napraviti niz testova na skupu ulaznih ontologija. Većina ontologija korištenih u ovom testiranju preuzeta je s online repozitorija sveučilišta Oxford. Postoje i drugi repozitoriji, međutim, nisu svi dostupni za slobodno korištenje. U konačnici, skup ontologija koji je predviđen za testiranje sasvim je dovoljan za provjeru ispravnosti transformacija. Prema tome, u Tablici 6.1 navedene su testne ontologije zajedno s informacijama kao što je IRI (Internationalized Resource Identifier) ontologije, te metrikama koje opisuju definicije i aksiome korištene u sastavljanju ontologije. Uz podatke kao što su ukupan broj aksioma, logičkih aksioma, klasa, svojstva objekata i svojstva tipa podatka nalaze se još i podaci o ukupnom broju instanci koji će kasnije služiti kao provjera je li došlo do gubitka u pretvorbi te podatak o DL ekspresivnosti koji zapravo ovisi o tipu korištenih aksioma u samoj definiciji ontologije.

Tablica 6.1: Testne ontologije - metrike

			Metrike (ukupan broj)						
R.br	**Naziv**	**IRI**	**Aksiomi**	**Logički aksiomi**	**Klase**	**Svojstva objekta**	**Svojstva tipa podatka**	**Instance**	**DL ekspresivnost**
1.	BIOPAX	http://www.cs.ox.ac.uk/isg/ontologies/UID/00007.owl	2644	2095	41	33	37	323	ALCHN(D)
2.	BIOPAX Reactome	http://www.cs.ox.ac.uk/isg/ontologies/UID/00009.owl	746	496	41	33	37	30	ALCHN(D)
3.	ABAv1.0	http://www.cs.ox.ac.uk/isg/ontologies/lib/BioPortal/ ABA_v1.0/v1.0/ABA_v1.0.owl	6178	3441	913	2	0	0	ALCI
4.	Functional Partitipation	http://www.cs.ox.ac.uk/isg/ontologies/UID/00017.owl	1757	1162	96	259	0	0	SHIN
5.	CommonSense Mapping	http://www.cs.ox.ac.uk/isg/ontologies/UID/00013.owl	1937	1259	126	273	2	0	SHIN(D)
6.	IMGT	http://www.imgt.org/download/ IMGT-ONTOLOGY/IMGT-ONTOLOGY-v1-0-1.owl	3841	2114	292	23	5	0	SHIN(D)
7.	LUBM	http://www.cs.ox.ac.uk/isg/ontologies/ lib/LUBM/lubm/2009-05-29/00347.owl	117960	100636	43	25	7	17174	ALEHI+(D)
8.	SEMINTE	http://www.cs.ox.ac.uk/isg/ontologies/UID/00776.owl	83476	65459	60	16	0	17941	ALCOIF
9.	AEO	http://www.cs.ox.ac.uk/isg/ontologies/UID/00002.owl	5089	3472	760	47	16	16	SHIN(D)
10.	EMELD	http://www.cs.ox.ac.uk/isg/ontologies/UID/00075.owl	1302	319	156	7	1	190	ALHN(D)
11.	GARDINER	http://www.cs.ox.ac.uk/isg/ontologies/UID/00055.owl	602	244	153	22	0	0	SHIN
12.	MovieDB	http://www.cs.ox.ac.uk/isg/ontologies/UID/00053.owl	2027	1129	96	332	46	0	ALH(D)
13.	MGEDO	http://www.cs.ox.ac.uk/isg/ontologies/UID/00082.owl	3391	1402	229	104	6	658	ALCOF(D)
14.	Mission Model	http://www.cs.ox.ac.uk/isg/ontologies/UID/00137.owl	516	200	146	18	1	41	ALH
15.	Transportation	http://www.cs.ox.ac.uk/isg/ontologies/UID/00148.owl	2016	798	445	92	4	155	ALH
16.	VICODI	http://www.cs.ox.ac.uk/isg/ontologies/UID/00779.owl	149848	116404	194	10	2	33238	ALHI(D)
17.	DolceLite	http://www.cs.ox.ac.uk/isg/ontologies/UID/00015.owl	534	349	37	70	0	0	SHI
18.	Extended DnS	http://www.cs.ox.ac.uk/isg/ontologies/UID/00016.owl	1606	1055	96	229	0	0	SHIN
19.	Erlangen	http://www.cs.ox.ac.uk/isg/ontologies/UID/00025.owl	1546	970	87	260	4	0	SHIN(D)
20.	Food	http://www.cs.ox.ac.uk/isg/ontologies/UID/00781.owl	1252	889	138	16	1	206	SHOIN(D)

U Tablici 6.2 prikazane su dodatne informacije o testnim ontologijama. Te informacije sastoje se od ukupnog broja korištenih aksioma, a koji direktno utječu najviše na samu strukturu ontologije. Prema tome, navedeni su ukupni broj aksioma za podklase kojim se definiraju roditelji klase, aksiom za jednakost klasa koji se ne odnosi isključivo na jednake klase (ciklička veza) već i na ostale tipove kao što su presjeci, unije i restrikcije. Drugim riječima, klase opisane tim aksiomom sadrže nužne i dovoljne uvjete. I na kraju, navodi se ukupan broj aksioma za različite klase.

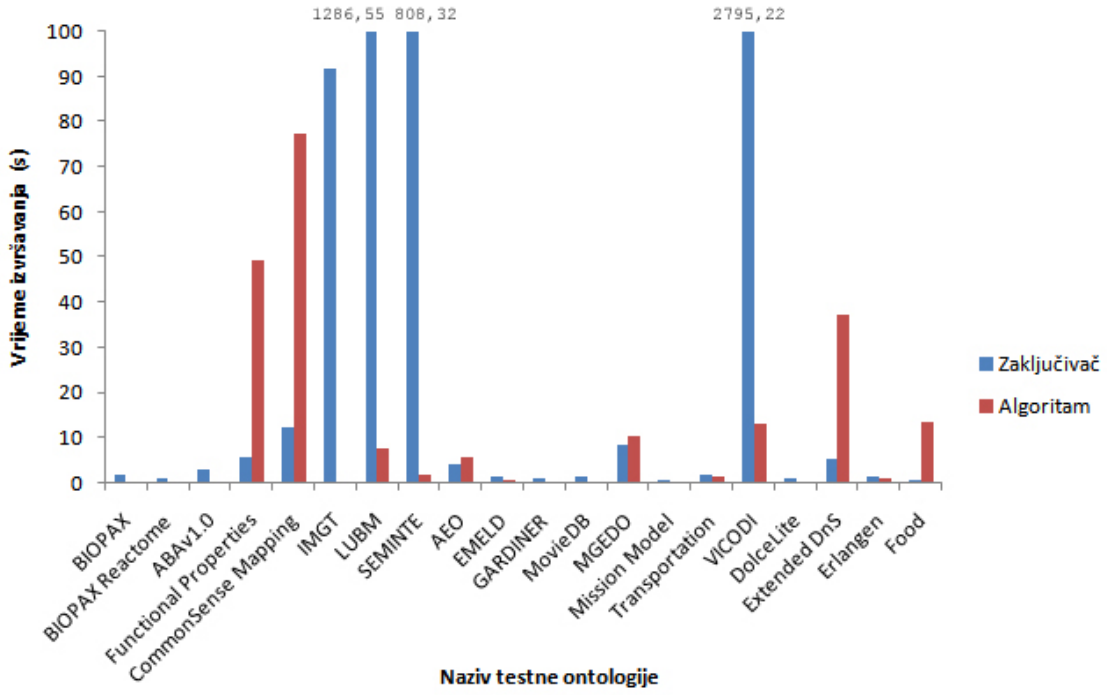
Tablica 6.2: Testne ontologije - Aksiomi klasa

		Aksiomi klasa		
R.br	**Naziv**	**Pod klase**	**Jednake klase**	**Različite klase**
1.	BIOPAX	108	0	85
2.	BIOPAX Reactome	108	0	85
3.	ABAv1.0	911	0	2525
4.	Functional Partitipation	166	24	41
5.	CommonSense Mapping	204	27	42
6.	IMGT	550	38	1477
7.	LUBM	36	6	0
8.	SEMINTE	55	0	113
9.	AEO	1357	3	1957
10.	EMELD	119	0	0
11.	GARDINER	197	1	10
12.	MovieDB	95	0	0
13.	MGEDO	452	0	0
14.	Mission Model	121	0	0
15.	Transportation	452	0	0
16.	VICODI	193	0	0
17.	DolceLite	73	3	18
18.	Extended DnS	166	24	41
19.	Erlangen	186	0	6
20.	Food	228	88	39

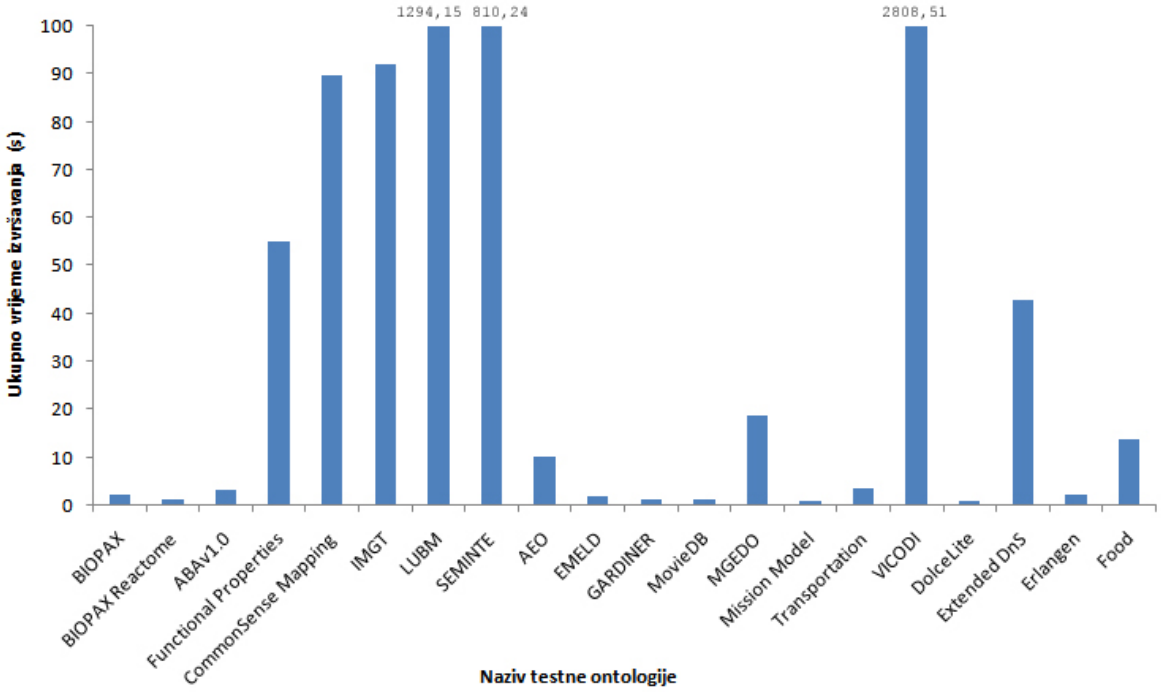
Tablica 6.3 prikazuje vremena izvršavanja za algoritam transformacije na skupu od 20 različitih ontologija. Navedena su pojedinačna vremena za zaključivač i algoritam te ukupno vrijeme.

Tablica 6.3: Vremena izvršavanja

R.br	Naziv	Vrijeme trajanja zaključivača (s)	Vrijeme trajanja algoritma (s)	Ukupno
1.	BIOPAX	1,73	0,34	2,07
2.	BIOPAX Reactome	1,05	0,22	1,27
3.	ABAv1.0	3,11	0,23	3,34
4.	Functional Partitipation	5,76	49,33	55,09
5.	CommonSense Mapping	12,42	77,32	89,74
6.	IMGT	91,85	0,17	92,02
7.	LUBM	1286,55	7,60	1294,15
8.	SEMINTE	808,32	1,92	810,24
9.	AEO	4,31	5,76	10,07
10.	EMELD	1,24	0,71	1,95
11.	GARDINER	0,86	0,24	1,10
12.	MovieDB	1,27	0,05	1,32
13.	MGEDO	8,43	10,36	18,79
14.	Mission Model	0,74	0,25	0,99
15.	Transportation	1,87	1,52	3,39
16.	VICODI	2795,22	13,29	2808,51
17.	DolceLite	0,88	0,02	0,90
18.	Extended DnS	5,46	37,43	42,89
19.	Erlangen	1,35	0,90	2,25
20.	Food	0,46	13,33	13,79



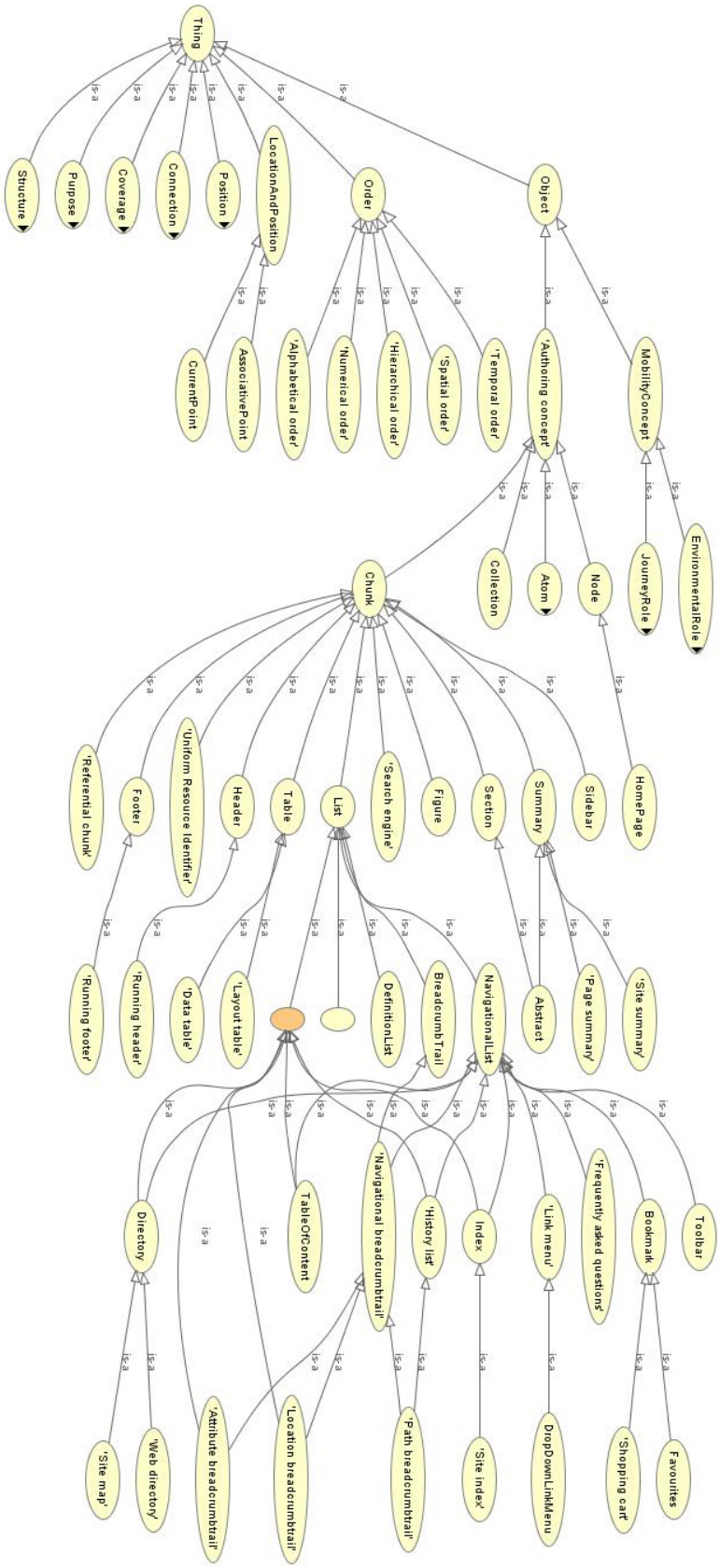
Slika 6.21: Pojedinačna vremena izvršavanja



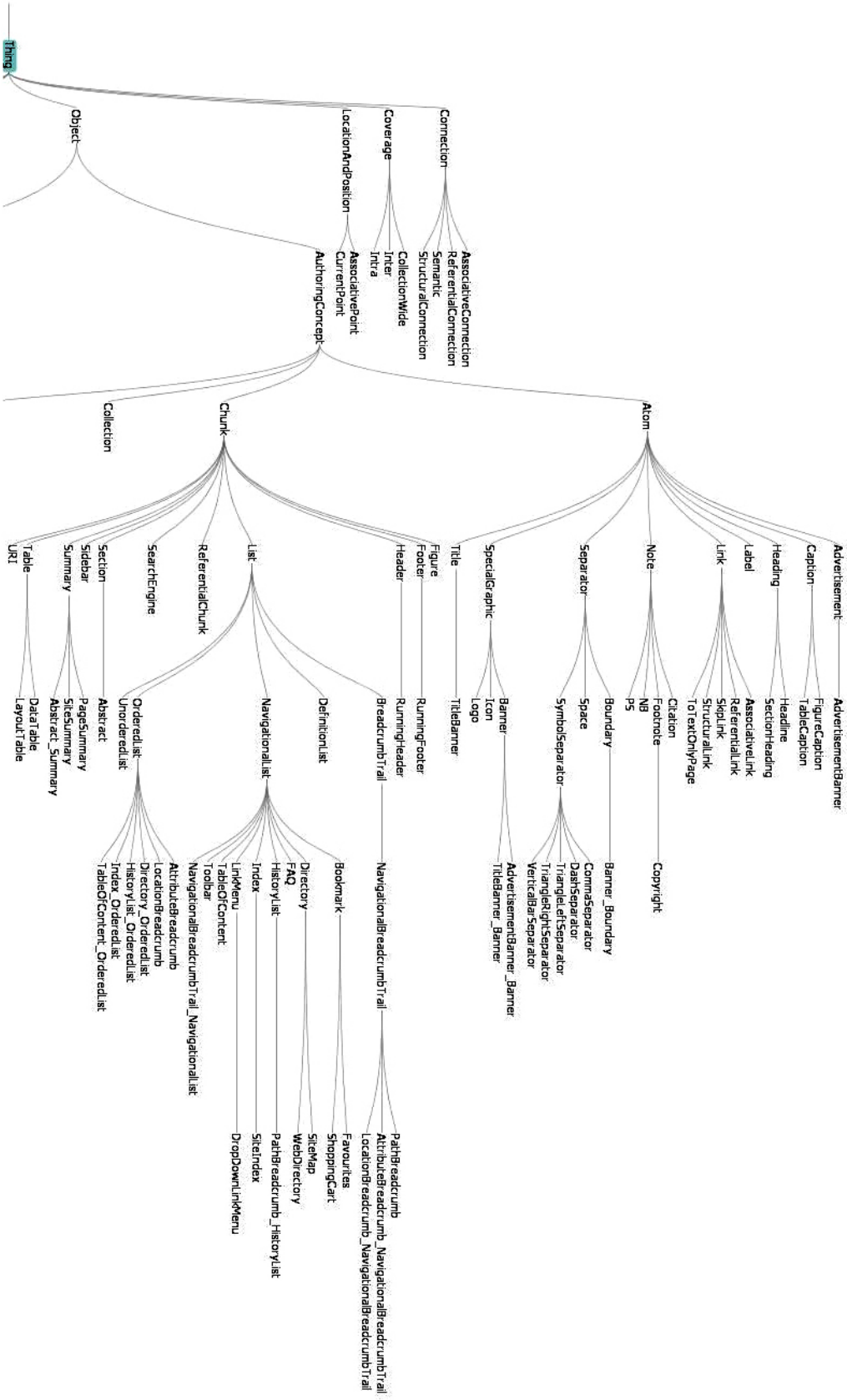
Slika 6.22: Ukupna vremena izvršavanja

Iz grafa na Slici 6.21 vidljivo je da vrijeme izvršavanja zaključivača može trajati znatno duže od vremena izvršavanja algoritma za transformaciju što je i logično s obzirom na znatno veću kompleksnost zaključivača. Vrijeme trajanja zaključivača ovisno je o broju aksioma korištenih u ontologiji što se najviše očituje nad ontologijama LUBM , SEMINTE i VICODI. Vrijeme trajanja izvođenja algoritma za transformaciju najviše ovisi o broju elemenata koji čine

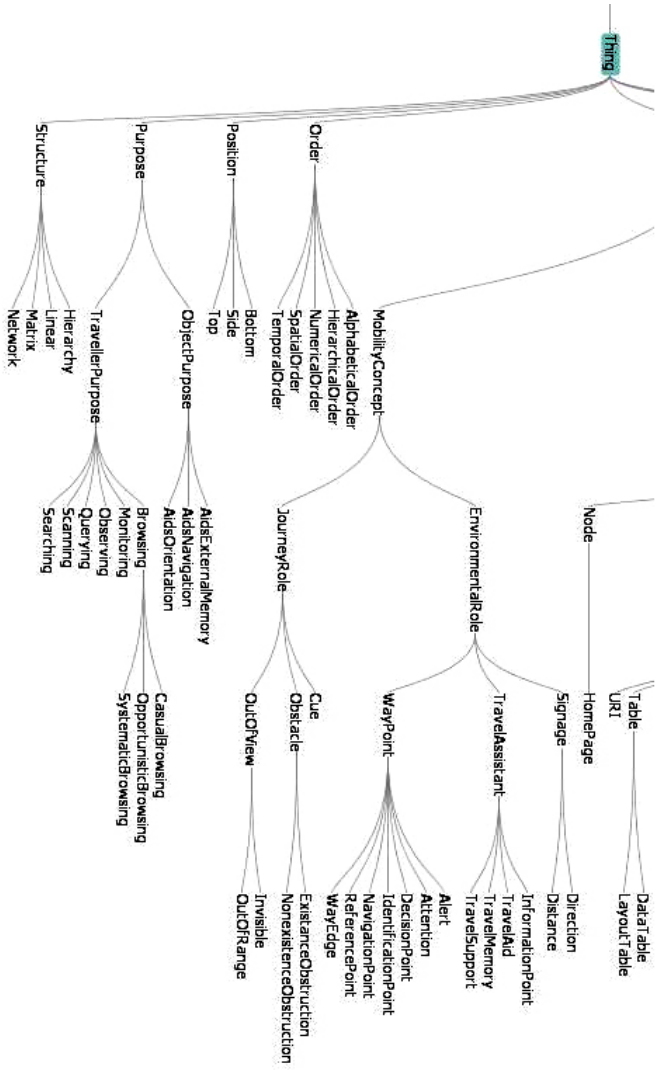
strukturu, odnosno, ukupnom broju klasa i pripadajućim instancama. Bitno je napomenuti da samo vrijeme izvođenja nije toliko bitno s obzirom da algoritam transformacije nije predviđen za korištenje u sustavima stvarnog vremena, već je dovoljno samo jednom napraviti transformaciju kako bi se mogla raditi daljna obrada dobivenih podataka. Na Slici 6.22 prikazana su zbrojena vremena izvođenja algoritma za transformaciju i HermiT zaključivača. Algoritam transformacije razvijen je u programskom jeziku Java dok za svrhe zaključivanja koristi javno dostupnu biblioteku HermiT zaključivača koja se može pronaći na Internetu [154]
U nastavku ovog poglavlja na Slikama 6.23, 6.24 i 6.25 prikazan je jednostavniji primjer ulazne ontologije (http://www.cs.ox.ac.uk/isg/ontologies/UID/00055.owl) i pripadajući rezultat algoritma za transformaciju.



Slika 6.23: Primjer ulazne ontologije



Slika 6.24: Rezultat transformacije prvi dio



Slika 6.25: Rezultat transformacije drugi dio

Na Slici 6.24 i 6.25 prikzan je rezultat transformacije ulazne ontologije prikazan na Slici 6.23. Ulazna ontologija predstavlja jednostavniji primjer ontologije koja se sastoji od sveukupno 153 klase (nisu sve vidljive zbog lakšeg pregleda). Može se primjetiti da ontologija ne zadovoljava pravila općeg stabla s obzirom na klase s višestrukim roditeljima. Neke od takvih klasa su "Abstract", "Index", "History List", "Directory" i slične. U izlaznoj taksonomiji jasno se može vidjet da su navedeni problemi riješeni te da su napravljene kopije tih klasa.

Poglavlje 7

Kriterij vrednovanja rezultata transformacije ontologije u taksonomsku strukturu

Evaluacija ontologije vrlo je bitan postupak u raznim koracima razvoja ontologije. Kao primjer kada bi se trebala raditi evaluacija mogu se navesti sljedeće situacije:

- Postupak stvaranja ontologije od početka.
- Postupak dopune već postojeće ontologije.
- Prilikom primjene nekih od tehnika interoperabilnosti.

Razlozi za evaluaciju ontologije mogu biti različiti, a tiču se određivanja životnog ciklusa ontologije, dupliciranja instanci, gubitka relacija, razlike u reprezentaciji i slično. Prema [155], [156] postoje definirane sljedeće metrike za evaluaciju ontologije:

- Preciznost – kriterij za određivanje podudara li se novo znanje s izvornim znanjem o određenoj domeni.
- Prilagodljivost - mjeri mogućnost korištenja ontologije u nekom drugom kontekstu s mogućnošću proširenja.
- Jasnoća – mjeri efikasnost predstavljanja namijenjenoga značenja i definiranih termina.
- Kohezija – mjeri modularnost ontologije, odnosno, stupanj povezanosti koncepata.
- Potpunost – mjeri uspješnost pokrivenosti neke domene.
- Računalna djelotvornost – mjeri brzinu kojom određeni alati mogu raditi s ontologijom (odnosi se na zaključivače).
- Konciznost – mjeri koliko nebitnih elemenata u odnosu na domenu sadrži ontologija.
- Konzistencija – mjeri koliko konfliktnih elemenata ontologija sadrži.
- Kombiniranost – mjeri broj uvezenih klasa koje se slažu s klasama u ontologiji.
- Pokrivenost – mjeri koliko dobro ontologija predstavlja domenu.

Metode temeljene na metrikama su:

- *OntoMetric* – okvir temeljen na hijerarhiji. Sastoji se od 160 karakteristika kroz 5 područja za evaluaciju kvalitete koja se odnose na sadržaj ontologije, jezik, metodologiju razvoja, alata za izradu i troška korištenja.
- *AKTiveRank* – temeljen na pronalasku skupa srodnih ontologija na temelju četiri mjere (podudarnost klasa, gustoći, semantičkoj sličnosti i različitosti).
- *ODEval* – detektira sintaksne probleme, postojanje ciklusa, nedovršivost i redundanciju klasa.
- *Oqual* – način ocjenjivanja ontologije temeljen na 3 dimenzije. Strukturalna dimenzija koja se odnosi na sintaksnu i formalnu semantiku. Funkcionalna dimenzija koja se odnosi na relacije između ontologija i predviđene primjene. I na kraju, dimenzija iskoristivosti koja se odnosi na efikasnost predstavljanja znanja.
- *OntoClean* – temelji se na definiranju skupine značajki (robusnost, identitet, cjelovitost i ovisnost) kojima se identificiraju dijelovi koji se trebaju ponovno preispitati.
- *OntoQA* – metoda temeljena na značajkama. Podijeljena je u dvije skupine, metrike sheme i metrike instanci. Metrika sheme mjeri dizajn i mogućnost reprezentacije znanja dok metrika instanci mjeri smještaj i popunjenost instanci.

Osim metoda temeljenih na metrikama, postoje još metode temeljene na evoluciji i metode temeljene na pravilima. Metode temeljene na evoluciji bave se verzioniranjem ontologija kroz vrijeme i na taj način se mjeri njihova kvaliteta. Metode temeljene na pravilima evaluiraju kvalitetu ontologije koristeći pravila koja su ugrađena u ontologiju i pravila koja definira korisnik kako bi se pronašle konfliktnosti ukoliko postoje u ontologiji.
Osim navedenih metoda za evaluaciju kvalitete ontologije, pregledom literature i radova iz područja rješavanja interoperabilnosti navodi se još jedan način mjerenja kvalitete, a to se odnosi na mjerenje temeljeno na rekonstrukciji. Mjerenje temeljeno na rekonstrukciji odnosi se na pokušaje rekonstruiranja rezultirajuće ontologije na temelju dostupnih pravila i hijerarhije. Ovisno o sličnosti s izvornom ontologijom definira se uspješnost, odnosno, dobiva se ocjena kvalitete.
S obzirom na dobivenu taksonomiju kao rezultat transformacije, može se zaključiti da niti jedna od navedenih metoda evaluacije ontolgija nije primjenjiva. Razlog za to nalazi se u činjenici da dobivena taksonomija, iako se strukturalno ne razlikuje s izvornom ontologijom, ne sadrži niti jedan aksiom / definiciju izvorne ontologije. Ono što preostaje je promatranje razlike u strukturama izvorne ontologije i rezultirajuće taksonomije kako bi se utvrdilo je li došlo do određenog gubitka prilikom transformacije. S obzirom da se radi o usporedbi grafa kao strukture podataka i taksonomije koja zadovoljava pravila općeg stabla, uz još dodatne zahtjeve za primjenu algoritma evidencijskog zaključivanja, u literaturi nije pronađena metoda koja bi zadovoljila takvu provjeru. Sukladno tome, definirana su tri svojstva koja rezultirajuća taksonomija treba zadovoljiti kako bi se ustvrdilo da je transformacija bila uspješna.

Nazivi definirana tri svojstva su kako slijedi:

- Svojstvo klasa.
- Svojstvo veza.
- Svojstvo povezanosti.

Svojstvo klasa

S obzirom na navedena pravila transformacije, jasno je vidljivo da algoritam za transformaciju ima tendenciju dodavanja novih ili brisanja postojećih klasa. Međutim, ne smije se dogoditi dodavanje novih klasa ili brisanja klasa ukoliko to nije u okviru definiranih pravila. Kako bi se utvrdilo da do toga nije došlo, definirano je pravilo svojstva klasa S_K izrazom:

$$S_K = \frac{BK_G - BKP}{BK_T - BKK}, \tag{7.1}$$

$$S_K \leq 1,$$

gdje je:

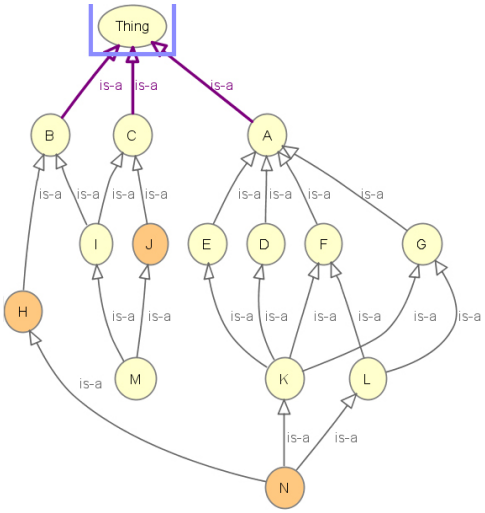
- BK_G - broj klasa izvorne ontologije.
- BKP - broj klasa presjeka.
- BK_T - broj klasa rezultirajuće taksonomije.
- BKK - broj klasa kopija.

Klase koje su definirane kao presjek dvije ili više drugih klasa bitne su zbog njihovog utjecaja na konačnu strukturu, odnosno, mogu uzrokovati brisanje klase ili kopiranje klase ovisno o zadovoljenim pravilima. Definicije klasa presjeka nalaze se u XML dokumentu prikazanom na Slici 5.14 i one su poznate odmah na početku. S druge strane, sve ostale klase koje imaju više od dva roditelja kopirat će se kako bi bilo zadovoljeno pravilo jednog roditelja. To rezultira stvaranjem viška klasa u taksonomiji. Kako bi se poništili utjecaji ova dva tipa klasa, od ukupnog broja klasa izvorne ontologije oduzima se ukupan broj definiranih klasa presjeka bez obzira je li došlo do njihovog brisanja ili kopiranja u rezultirajućoj taksonomiji, dok se s druge strane od ukupnog broja klasa taksonomije oduzima ukupan broj kopiranih klasa. Kao rezultat, svojstvo klasa treba biti manje ili jednako 1, u suprotnom, svojstvo nije zadovoljeno. Uz pretpostavku da je transformacija bila uspješna, S_K će imati vrijednost 1:

- Ukoliko je izvorna ontologija već zadana tako da zadovoljava sva pravila. Na taj način neće doći do promjene prilikom transformacije pa će BKP i BKK imati vrijednost 0, odnosno, vrijedit će $BK_G = BK_T$.
- Ukoliko sve klase presjeka zadovoljavaju pravilo za uklanjanje iz strukture. Na taj način vrijedit će $BK_G = BK_T$ neovisno o BKK koji se oduzima od BK_T.

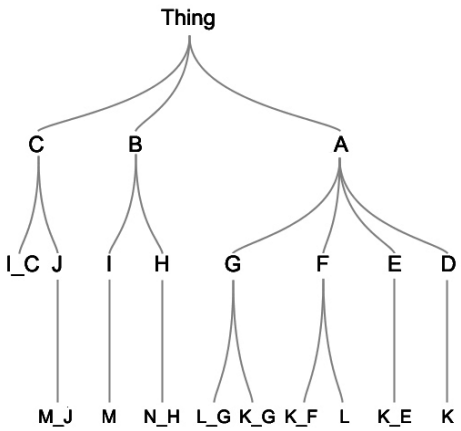
- Ukoliko dođe samo do kopiranja čvorova (bez postojanja klasa presjeka). Na taj način BKP će imati vrijednost 0, odnosno, vrijedit će $BK_G = BK_T$.

U svim ostalim slučajevima uz pretpostavku da je transformacija bila uspješna vrijednost S_K će biti manja od 1. Ukoliko je vrijednost S_K na kraju transformacije veća od 1 to znači da je u postupku transformacije došlo do nepotrebnog brisanja klase. Drugim riječima, dolazi do gubitka u rezultirajućoj taksonomiji u odnosu na izvorni graf pa samim time svojstvo klasa neće biti zadovoljeno. Za primjer se može uzeti jednostavna ontologija prikazana na Slici 7.1.



Slika 7.1: Jednostavna ontologija

Na Slici 7.1 prikazana je jednostavna ontologija gdje su klase N ($N = K \sqcap L$, $N \sqsubseteq H$), J ($J = J \sqcap Anonimna$, $J \sqsubseteq C$) i H ($H = B \sqcap Anonimna$) definirane kao presjeci. Isto tako, može se primijetiti kako klase I, M, K i L imaju više od jednog roditelja, odnosno, definirane su kao podklase više različitih klasa. Primjenom pravila prilikom transformacije dobije se struktura prikazana na Slici 7.2.



Slika 7.2: Rezultirajuća taksonomija

Prema onome što je prikazano, ulazna ontologija sastoji se od ukupno 15 klasa, od čega su tri klase definirane kao presjeci. Kod rezultirajuće taksonomije može se vidjeti da je došlo do kopiranja klasa *I*, *M*, *N*, *L* i *K*.

Stoga bi varijable za izračun svojstva klasa bile zadane kao $BK_G = 15$, $BKP = 3$, $BK_T = 21$ i $BKK = 7$. Primjenom formule za izračun svojstva klasa dobila bi se vrijednost:

$$S_K = \frac{15-3}{21-7} = 0,85.$$

Vrijednost S_K je manja ili jednaka od 1 što znači da rezultirajuća taksonomija zadovoljava svojstvo klasa.

Svojstvo veza

Sljedeće svojstvo za provjeru uspješnosti transformacije odnosi se na svojstvo veza između klasa. Svojstvo veza nadovezuje se na svojstvo klasa te zajedno tvore potporu za svojstvo povezanosti s obzirom da su informacije za izračun ta dva svojstva dostupna odmah nakon transformacije i znatno su manje računalno zahtjevni od izračuna svojstva povezanosti. Cilj svojstva veza je određivanje je li došlo do stvaranja dodatnih veza koje nisu obuhvaćene postavljenim pravilima. S obzirom da su poznati ukupni brojevi veza izvorne ontologije i rezultirajuće taksonomije, definirana je formula za izračun svojstva veza S_V:

$$S_V = \frac{BV_G}{BV_T}, \tag{7.2}$$

$$S_V \leq 1,$$

gdje je:

- BV_G - broj veza izvorne ontologije.
- BV_T - broj veza rezultirajuće taksonomije.

Kako bi svojstvo veza bilo zadovoljeno, vrijednost S_V treba biti manja ili jednaka 1. Uz pretpostavku da je transformacija bila uspješna S_V imat će vrijednost 1:

- Ukoliko je izvorna ontologija već zadana tako da zadovoljava sva pravila.
- Ukoliko prilikom transformacije dođe samo do kopiranja klasa.

Za oba navedena slučaja vrijedit će $BV_G = BV_T$. U svim ostalim slučajevima uz pretpostavku da je transformacija bila uspješna vrijednost S_V bit će manja od 1. Kao i kod svojstva klasa, ukoliko je vrijednost S_V na kraju transfromacije veća od 1 to znači da je u postupku transformacije došlo do nepotrebnog brisanja klase, a samim time došlo je i do gubitka u rezultirajućoj taksonomiji. U tom slučaju, svojstvo veza neće biti zadovoljeno. Ako se za primjer uzme ulazna ontologija prikazana na Slici 7.1 i rezultirajuća taksonomija prikazana na Slici 7.2, varijable za

izračun svojstva veza bile bi zadane kao $BV_G = 22$ i $BV_T = 20$. Može se primjetiti da je broj veza taksonomije BV_T manji od broja veza izvornog grafa BV_G . Razlog tome je klasa presjeka N koja je uklonjena iz strukture taksonomije. Primjenom formule za izračun svojstva veza dobila bi se vrijednost:

$$S_V = \frac{22}{20} = 0,91.$$

Vrijednost S_V je manja ili jednaka od 1 što znači da rezultirajuća taksonomija zadovoljava svojstvo veza.

Svojstvo povezanosti

Ukoliko vrijednosti izračunatih svojstava klasa i veza zadovoljavaju prije spomenute uvjete, potrebno je još izračunati svojstvo povezanosti. Ono se odnosi na provjeru relacija između klasa između izvorne ontologije i klasa rezultirajuće taksonomije. Drugim riječima, za svaku relaciju između bilo koje klase i korijenske klase izvorne ontologije, treba postojati ista takva relacija u rezultirajućoj taksonomiji. Na taj način provjerava se konzistentnost i kvaliteta transformacije. Postupak provjere za svojstvo povezanosti može se pokazati na primjeru ulazne ontologije prikazane na Slici 7.1 i rezultirajućoj taksonomiji na Slici 7.2 za klasu *M*. Sve putanje klase *M* prema korijenskoj klasi *Thing* za izvornu ontologiju su kako slijedi:

M - I - C - Thing,
M - J - C - Thing,
M - I - B - Thing.

S druge strane, sve putanje klase *M* prema korijenskoj klasi *Thing* za rezultirajuću taksonomiju su kako slijedi:

M - I - B - Thing.

Iz navedenog se jasno može vidjeti da putanje ne odgovaraju. Međutim, s obzirom na pravila transformacije i utjecaj klasa presjeka i kopija klasa potrebno je proći kroz još nekoliko provjera kako bi se konačno moglo utvrditi prolazi li rezultirajuća taksonomija provjeru. Dodatne provjere u putanjama uključuju:

- Je li klasa definirana kao presjek?
- Je li klasa definira kao jednaka s jednom ili više klasa (ciklička veza)?
- Je li klasa s višestrukim roditeljskim klasama?

Ako prilikom transformacije postoji bilo koji od navedenih slučajeva doći će do promjene rezultirajuće taksonomije što ne znači nužno da će doći i do promjene u reprezentaciji. Ukoliko dođe do promjene u reprezentaciji, ova provjera bi to trebala biti u mogućnosti otkriti. Prema tome, putanje koje je potrebno dodatno provjeriti su:

M - I - C - Thing,
M - J - C - Thing.

Prva na redu je putanja *M - I - C - Thing*. S obzirom da je definicija klase $I = I \sqsubseteq B$, $I \sqsubseteq C$, to znači da se radi o klasi s višestrukim roditeljima.

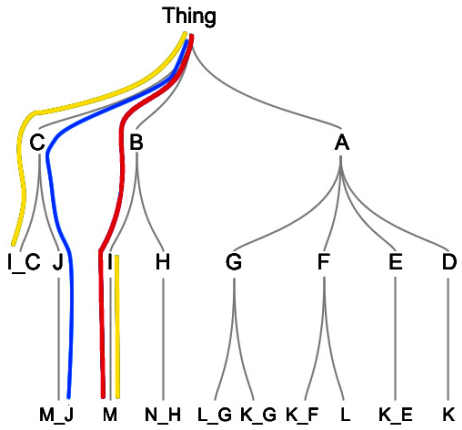
Prema tome, treba provjeriti postoji li kopija klase *C* i koji je njezin naziv, te na osnovu toga provjeriti podudara li se putanja od te klase do korijenske klase s ostatkom putanje koja se provjerava. Ukoliko putanja odgovara, provjeravana putanja zadovoljava svojstvo povezanosti te se prelazi na sljedeću. Rezultat toga prikazan je kako slijedi:

M - I - C - Thing - izvorna ontologija
I_C - C - Thing - rezultirajuća taksonomija

Kako bi provjera bila uspješna, klasa *M* ne treba izravno biti povezana s klasom *I_C* jer bi tako došlo do nepotrebnog kopiranja. S obzirom da klasa *I* i klasa *I_C* predstavljaju istu stvar, nije došlo do gubitka u reprezentaciji.
Na isti način, provjerava se preostala putanja *M - J - C - Thing*. Klasa *M* definirana je kao nasljednik klasa *J* i *I*, odnosno, kao klasa s višestrukim roditeljima. Prema pravilima, trebalo bi doći do kopiranja klase *M*, pa se prema tome provjerava postoji li kopija te klase i koji je njezin naziv. Ukoliko postoji kopija, potrebno je provjeriti da li putanja od te klase do korijenskog čvora odgovara ostatku putanje provjeravane putanje.

M - J - C - Thing,
M_J - J - C - Thing.

Prema prikazanom, može se vidjeti da putanja zadovoljava svojstvo povezanosti. Ukoliko sve putanje zadovoljavaju, može se reći da je rezultirajuća taksonomija prošla svojstvo povezanosti. Slika 7.3 prikazuje objašnjeni postupak gdje crvena, plava i žuta boja predstavljaju putanje za klasu M prema korijenskom čvoru u rezultirajućoj taksonomiji (M-I-B-Thing, M-J-C-Thing i M-I-C-Thing).



Slika 7.3: Provjera svojstva povezanosti

Tablica 7.1: Rezultati provjere svojstava za skup ulaznih ontologija

R.br	Naziv	BIG	BIT	BVG	BVT	SV	BKG	BKT	BKP	BKK	BOK	SC	SP	BVR
1.	BIOPAX	323	323	41	41	1	41	42	0	1	0	1	1	0
2.	BIOPAX Reactome	30	30	41	41	1	41	42	0	1	0	1	1	0
3.	ABAv1.0	0	0	913	913	1	914	914	0	0	0	1	1	0
4.	Functional Partitipation	0	0	101	101	1	96	102	5	6	0	0,94	1	0
5.	CommonSense Mapping	0	0	131	131	1	126	132	5	6	0	0,96	1	0
6.	IMGT	0	0	291	291	1	292	292	0	0	0	1	1	0
7.	LUBM	17174	17174	44	44	1	44	45	0	1	0	1	1	0
8.	SEMINTE	17941	17941	59	59	1	60	60	0	0	0	1	1	0
9.	AEO	16	16	836	836	1	759	837	0	78	0	1	1	0
10.	EMELD	190	190	155	155	1	156	156	0	0	0	1	1	0
11.	GARDINER	0	0	164	164	1	153	165	0	12	0	1	1	0
12.	MovieDB	0	0	96	96	1	97	97	0	0	0	1	1	0
13.	MGEDO	658	658	254	254	1	229	255	0	26	0	1	1	0
14.	Mission Model	41	41	178	178	1	147	179	0	32	0	1	1	0
15.	Transportation	155	155	525	525	1	446	526	0	80	0	1	1	0
16.	VICODI	33238	33238	194	194	1	195	195	0	0	0	1	1	0
17.	DolceLite	0	0	37	37	1	38	38	0	0	0	1	1	0
18.	Extended DnS	0	0	101	101	1	96	102	5	6	0	0,95	1	0
19.	Erlangen	0	0	100	100	1	87	101	0	14	0	1	1	0
20.	Food	206	206	204	173	0,85	138	174	40	36	0	0,71	1	0

Prema navedenim svojstvima, Tablica 7.1 prikazuje rezultate testiranja algoritma za transformaciju nad skupom ulaznih ontologija gdje je:

- BIG - Broj instanci izvorne ontologije.
- BIT - Broj instanci rezultirajuće taksonomije.
- BVG - Broj veza izvorne ontologije.
- BVT - Broj veza rezultirajuće taksonomije.
- SV - Svojstvo veza.
- BKG - Broj klasa izvorne ontologije.
- BKT - Broj klasa rezultirajuće taksonomije.
- BKP - Broj klasa presjeka.
- BKK - Broj klasa kopija.
- BOK - Broj obrisanih klasa.
- SK - Svojstvo klasa.
- SP - Svojstvo povezanosti.
- BVR - Preostali broj čvorova s više roditelja.

Iz Tablice 7.1 može se vidjeti da su na skupu ulaznih ontologija sve transformacije uspješno obavljene. To se može zaključiti iz vrijednosti svojstava *SV*, *SK* i *SP* koje su u dozvoljenim granicama. Osim vrijednosti svojstava, u tablici se nalaze i vrijednosti *BIT* i *BIG* koje ukazuju na to da nije došlo do gubitka instanci klasa koje su vrlo bitne u daljnjoj obradi evidencijskim zaključivanjem s obzirom da one predstavljaju objekte ocjenjivanja. Na kraju, kao još jedan parametar po kojem se može utvrditi uspješnost transformacije i vrijednost *BVR* koja ukazuje na to da u rezultirajućoj taksonomiji ne postoji više niti jedna klasa koja ima više od jednog roditelja. Ostale vrijednosti u tablici osim vrijednosti *BOK* predstavljaju elemente koji se uvrštavaju u formule za izračuna svojstva klasa i svojstva veza. Vrijednost *BOK* ukazuje na to da na skupu ulaznih ontologija prilikom transformacije nije došlo do brisanja niti jedne klase. Takav rezultat ne iznenađuje s obzirom da se klase brišu samo u specifičnim slučajevima koji se nisu našli u testnim ontologijama.

Poglavlje 8

Zaključak

Modeliranje podataka predstavlja vrlo značajan korak u razvoju nekog sustava, bilo da se radi o nekom informacijskom sustavu, bazi podataka ili nećem trećem. Uz način modeliranja vrlo je bitno odabrati i odgovarajuću metodu i jezik modeliranja podataka koji je u mogućnosti ispuniti sve zahtjeve. U današnje vrijeme, kada su informacije dostupne na svakom koraku, postoji mogućnost da model kojeg se želi napraviti već u nekom obliku i postoji. Ponovno iskorištavanje odlika je svih novijih metoda modeliranja podataka pa tako i ontologija. Upravo se na to i odnosi tema ove disertacije. S obzirom da ontologije predstavljaju vrlo napredan način modeliranja znanja, na način da se svakom elementu dodaje značenje i da se na osnovu zaključivača mogu dobiti kvalitetniji rezultati upita, bilo bi loše tako modelirane podatke ne iskoristiti u druge svrhe ako je to moguće. Kao jedno od takvih primjena, u ovoj doktorskoj disertaciji razmatra se evidencijsko zaključivanje. Evidencijsko zaključivanje predstavlja metodu za višekriterijsko odlučivanje koja na temelju distribuiranih ocjena, uključujući i stupanj uvjerenja, daje ukupnu ocjenu (procjenu) nekog sustava. Ako, primjerice, postoji ontologija koja opisuje sigurnost računalnog sustava, tada se određenim postupcima prilagodbe može, na toj već postojećoj ontologiji, primijeniti evidencijsko zaključivanje kako bi se dobio stupanj procjene računalnog sustava.

8.1 Metodologija istraživanja

Na temelju zahtjeva, u ovoj disertaciji upotrijebljene su sljedeće metodologije istraživanja:

- Pregled razvoja i upotrebe ontologija kao metode modeliranja podataka te usporedba s uobičajenim metodama modeliranja.
- Pregled metoda ocjenjivanja temeljenih na upotrebi algoritma za evidencijsko zaključivanje.
- Pregled postojećih metoda transformacije ontologija u drugu strukturu kao i njihova ograničenja u odnosu na predviđeni cilj.

- Razvoj novog modela transformacije ontologije.
- Razvoj algoritma transformacije ontološke strukture u taksonomsku strukturu temeljenog na pravilima uz mogućnost optimizacije i restrukturiranja taksonomske strukture prema zahtjevima za primjenu evidencijskog zaključivanja.
- Definiranje kriterija i razvoj algoritma za evaluaciju novog modela transformacije.
- Ispitivanje novorazvijenog modela.

8.2 Model prilagodbe ontološke strukture u taksonomsku strukturu

Iz strukture doktorske disertacije može se vidjeti da se provedeno istraživanje dijeli na dva bitna dijela. Prvi dio odnosi se na proučavanje ontologija kao naprednog načina modeliranja podataka, odnosno, baza znanja. Drugi dio odnosi na proučavanje postojećih metoda višekriterijskog odlučivanja, odnosno, evidencijskog zaključivanja i primjene istog. O prednostima oba dijela detaljno je raspravljano u radu. Kako bi se evidencijsko zaključivanje moglo primijeniti nad ontologijama potrebno je napraviti određene prilagodbe. U tu svrhu predložen je model prilagodbe OWL ontološke strukture u taksonomsku strukturu. Model je podijeljen na tri glavna dijela:

- Ulaz i priprema podataka.
- Obrada podataka.
- Primjena i analiza.

Ulaz i priprema podataka

Glavni dio ulaza i pripreme podataka je primjena HermiT zaključivača. U podpoglavlju 5.1 kratko su opisani radovi koji se bave transformacijom OWL ontologije u neku drugu strukturu podataka. Na osnovu tog pregleda sastavljena je Tablica 5.2 koja prikazuje funkcionalnosti pojedinih metoda. S obzirom da na strukturu ontologije, osim same raspodjele klasa i podklasa utječu i dodijeljene definicije i aksiomi bilo je potrebno proučiti na koji način navedene metode gledaju na takve slučajeve, a nedostaci su prikazani u Tablici 5.2. Isto tako, bitan je i tip izlazne strukture dobiven nekim od metoda koje se spominju u pregledu. Tablica 5.3 prikazuje tipove izlaznih struktura po metodama. Vidljivo je da strukture ne zadovoljavaju zahtjeve taksonomije koja je potrebna za primjenu evidencijskog zaključivanja, pri čemu se misli na taksonomiju koja zadovoljava pravila općeg stabla. Utvrđeno je da zaključivač, u ovom slučaju HermiT daje točne izračune strukture s obzirom na sve definicije i aksiome dodijeljene OWL ontologiji. Bitno je napomenuti da je izlaz iz zaključivača i dalje ontologija samo s raspodijeljenim klasama i instancama prema pravilima zadanih definicijama i aksiomima ontologije.

Prema tome, predstavlja značajan korak prije primjene algoritma za transformaciju.

Obrada podataka

Glavni dio obrade podataka čini algoritam za transformaciju ontološke strukture u taksonomsku strukturu. Kako bi algoritam uspješno prilagodio OWL ontologiju prikazanu grafom u taksonomsku strukturu koja zadovoljava pravila općeg stabla potrebno je definirati određena pravila. Evidencijsko zaključivanje koristi se za procjenu stanja nekog procesa ili sustava, stoga je za konačnu ocjenu dobivenu evidencijskim zaključivanjem vrlo bitno da bude što točnija. Prema tome, prilikom definiranja pravila potrebno je bilo uzeti u obzir i kako će se to odraziti na ocjenjivanje unutar evidencijskog zaključivanja. Definirana su sveukupno tri glavna pravila s odgovarajućim podpravilima. U radu su navedena, objašnjena i testirana sva pravila. Utvrđeno je da pravila zadovoljavaju sve potrebe vezane za prilagodbu ontološke strukture, a rezultati testnih transformacija prikazani su u Tablici 6.3. Kako je i navedeno u disertaciji, izvođenje jednog pravila nije uvjet za izvođenje drugog. Međutim, drugačiji redoslijed izvođenja pravila može rezultirati drugačijom izlaznom taksonomijom. Trenutno, spomenuto i ne predstavlja neki problem, premda je moguće definirati pravila tako da redoslijed izvođenja bude nezavisan. To bi se moglo izvesti tako da se prije izvršavanja radnje nad klasama predviđene pravilom provjeri o kakvoj se klasi radi. Primjerice, ako se radi o klasi presjeka, tada bi bilo potrebno dodati dodatno pravilo koje će onemogućiti primjenu trećeg pravila koje se odnosi na klase višestrukih roditelja prije provjere za drugo pravilo koje se odnosi na klase presjeka. Dodatna pravila i provjere tog tipa samo bi dodatno povećalo kompleksnost algoritma te povećalo vrijeme izvođenja transformacije, a s obzirom da je i trenutni raspored izvođenja sasvim dovoljan za uspješnu transformaciju, bilo kakve promjene takvog tipa otvorene su za buduće istraživanje.
Nakon provedene transformacije potrebno je provjeriti ispravnost rezultirajuće taksonomske strukture. U okviru ove disertacije definirana su tri svojstva koja rezultirajuća taksonomija treba zadovoljiti kako bi se mogla utvrditi uspješnost transformacije. Svojstvo klasa i svojstvo veza služe kao potpora svojstvu povezanosti zbog različite kompleksnosti izvršavanja. Drugim riječima, ako rezultirajuća taksonomija ne prođe ta dva svojstva, svojstvo povezanosti neće se niti provjeravati već će se odmah evidentirati neispravnost transformacije. Ispitivanjem je utvrđeno da su sve testne OWL ontologije prikazane u Tablici 6.1 uspješno prošle transformaciju. Rezultati transformacije prikazani u Tablici 7.1 sadrže vrijednosti sva tri svojstva uz informacije o broju ulaznih i izlaznih klasa, broju ulaznih i izlaznih instanci prema kojima se može zaključiti da nije došlo do gubitaka bitnih elemenata u transformaciji.

Primjena i analiza

Primjena i analiza odnosi se na daljnju obradu dobivene rezultirajuće taksonomije primjenom evidencijskog zaključivanja. U okviru ove disertacije, za potrebe ispitivanja implementirana je metoda evidencijskog zaključivanja kako bi se mogao provjeriti način izračuna. Takav primjer izračuna prikazan je u potpoglavlju 5.2. S obzirom da evidencijsko zaključivanje nije automatizirani proces već proces u kojem ekspert ocjenjuje neki sustav, rezultati uspješnosti izvođenja evidencijskog zaključivanja nisu nigdje prikazani. Međutim, sve strukture dobivene transformacijom zadovoljavaju sve zahtjeve i spremne su za primjenu evidencijskog zaključivanja. Sustav koji će zaokružiti cijeli proces planira se izraditi u budućem istraživanju.

8.3 Ograničenja algoritma za transformaciju

Testiranjem predloženog algoritma za transformaciju uočena su određena ograničenja. Ta ograničenja mogu se podijeliti u dvije skupine:

- Ograničenja vezana za OWL2 jezik.
- Ograničenja vezana za uvežene ontologije.

Trenutni algoritam za transformaciju podržava aksiome i definicije vezane za OWL jezik, odnosno, OWL1 jezik. Novouvedeni aksiomi unutar OWL2 jezika neće biti prepoznati što može rezultirati neuspjelom transformacijom ili uspješnom transformacijom s pogreškom u interpretaciji klasa. Prema tome, korištenje ontologija koje u sebi sadržavaju aksiome OWL2 jezika, a koji se odnose na hijerarhiju nije preporučeno.
Ograničenja vezana za uvežene ontologije odnose se na ontologije koje sadrže klase i aksiome neke druge ontologije. Na primjer, može se definirati vršna ontologija *food* i ontologija *pizza* koja implementira već definirane elemente *food* ontologije. S obzirom da se u ontologiji *pizza* ne nalaze definicije klasa *food* ontologije algoritam nije u mogućnosti prepoznati o kakvoj se klasi radi pa će samim time transformacija biti uspješna ali s pogreškom u interpretaciji klasa, a samim time i netočna. Drugim riječima, algoritam trenutno funkcionira tako da se sve definicije klasa trebaju nalaziti u ontologiji koja se obrađuje kako bi transformacija bila u potpunosti uspješna.
Iako izravno ne predstavlja ograničenje, bitno je napomenuti da trenutni algoritam transformacije zanemaruje *Nothing* klasu s obzirom da podklase *Nothing* klase predstavljaju one klase koje nisu prošle test zadovoljivosti prilikom obrade zaključivačem. Tijekom daljnjeg istraživanja svakako se planira proučavanje mogućnosti daljnje obrade *Nothing* klase.

8.4 Znanstveni doprinosi istraživanja i budući rad

Tri su znanstvena doprinosa ovog istraživanja:

- Model prilagodbe ontološke strukture u taksonomsku strukturu temeljen na HermiT zaključivaču.
- Kriterij vrednovanja rezultata transformacije ontologije u taksonomsku strukturu.
- Algoritam transformacije ontologije i optimizacija dobivene taksonomije prema zahtjevima evidencijskog zaključivanja.

Daljnji razvoj predloženog modela i algoritma za transformaciju ići će u smjeru:

- Poboljšanje u smislu podržavanja aksioma i definicija OWL2 jezika.
- Dohvaćanja svih definicija uveženih ontologija.
- Implementaciji cjelokupnog rješenja koja omogućuje zaokružen proces primjene algoritma transformacije, primjene evidencijskog zaključivanja i spremanja rezultata i ocjena.
- Proučavanja utjecaja *Nothing* klase.

Literatura

[1] T.Dillon, E.Chang, M.Hadzic, P.Wongthongtham, "Differentiating conceptual modelling from data modelling, knowledge modelling and ontology modelling and a notation for ontology modelling", APCCM '08: proceedings of the fifth on Asia-Pacific conference on conceptual modelling, 2008, str. 7-17.

[2] L.Xu, J.Yang, "Introduction to multi-criteria decision making and the evidential reasoning approach", Manchester School of Management, 2001.

[3] J.Yang, D.Xu, X.Xie, A.K.Maddulapalli, "Evidence theory and multiple criteria decision analysis the evidential reasoning approach", Proceedings of The 2010 Workshop on the Theory of Belief Functions, 2010.

[4] Paunović, L., Stokić, A., "Uticaj ontologija na funkcionalnost web-a", INFOTEH-JAHORINA, Vol. 11, 2012, str. 920-924.

[5] Sudeepthi, G., Anuradha, G., Babu, M., "A survey on semantic web search engine", IJCSI International Journal of Computer Science, Vol. 9, No. 1, 3 2012, str. 241-245.

[6] Saleem, A., "Semantic web vision: survey of ontology mapping systems and evaluation of progress", Doktorski rad, 2006.

[7] Prcela, M., "Predstavljanje ontologija na web-u", Institut Ruđer Bošković, 2008.

[8] Nayak, A., Agarwal, J., Yadav, V. K., Pasha, S., "Enterprise architecture for semantic web mining in education", Second International Conference on Computer and Electrical Engineering, 2009, str. 23-26.

[9] Štritof, T., "Vizualizacija znanja na webu", 2009.

[10] Rezk, E., Foufou, S., "A survey of semantic web concepts applied in web services and big data", IEEE, 2014, str. 767-772.

[11] Stanko, S., "Protégé".

[12] Kalyanpur, A., Golbeck, J., Banerjee, J., Hendler, J., "Owl: Capturing semantic information using a standardized web ontology language", Multilingual Computing and Technology Magazine, Vol. 15, No. 7, 2004.

[13] Hepp, M., "Semantic web and semantic web services: Father and son or indivisible twins?", IEEE Internet Comput, Vol. 10, No. 2, 2006, str. 85-88.

[14] Luka, B., "Approaches for generating owl ontologies from relational databases.", INFOTEH-JAHORINA, Vol. 10, 3 2011, str. 552-556.

[15] Obitko, M., Marik, V., "Ontologies for multi-agent systems in manufacturing domain", Proceedings. 13th Int. Work. Database Expert Syst. Appl., 2002.

[16] C.Legg, "Ontologies on the semantic web", Annual Review of Information Science and Technology, 2007, str. 407-452.

[17] Quboa, Q. K., Saraee, M., "A state-of-the-art survey on semantic web mining", Intelligent Information Management, Vol. 5, 2013, str. 10-17.

[18] L.W.Lacy, OWL: Representing Information Using the Web Ontology Language. Trafford, 2004.

[19] T.Berners-Lee, "Uniform resource identifier (uri): Generic syntax", dostupno na: https://tools.ietf.org/html/rfc3986

[20] Lenzerini, M., Milano, D., Poggi, A., "State of the art report 1 . ontology representation & reasoning".

[21] Karp, P. D., Chaudhri, V. K., Thomere, J., "Xol: An xml-based ontology exchange language", 2000.

[22] Yang, K., Steele, R., Lo, A., "An ontology for xml schema to ontology mapping representation.".

[23] W.Fan, "Xml constraints: Specification, analysis, and applications", DEXA Workshops, 2005, str. 805-809.

[24] L.Kim, "Xml integrated development environments", Altova, Inc. & Altova GmbH, 2002.

[25] D.Tidwell, "Introduction to xml", dostupno na: http://www.ibm.com/developerworks/xml/tutorials/xmlintro/xmlintro.html 2002.

[26] Programming XML Toolkit for iSeries, IBM, 2005, version 5 Release 5.

[27] Kalibatiene, D., Vasilecas, O., “Survey on ontology languages”, Artificial Intelligent Systems and Machine Learning, 2011, str. 124-141.

[28] Fensel, D., “Ontologies: A silver bullet for knowledge management and electronic commerce”, 2000.

[29] Aubrecht, P., “Ontology transformations between formalisms”, 2005.

[30] Presutti, V., Garshol, L. M., Vitali, F., Pepper, S., Gessa, N., “Towards the definition of guidelines for rdf and topic maps interoperability”, CEUR Workshop Proc., Vol. 185, 2005, str. 83-88.

[31] Ding, L., Kolari, P., Ding, Z., Avancha, S., “Using ontologies in the semantic web: A survey”, Springer, 2007, str. 79-113.

[32] G.Antoniou, Harmelen, F., “Web ontology language: Owl”, Handbook on Ontologies in Information Systems, 2003, str. 67-92.

[33] Chandrasekaran, B., Josephson, J. R., Benjamins, V. R., “What are ontologies, and why do we need them?”, IEEE Intelligent Systems and Their Applications, Vol. 14, 1999, str. 20-26.

[34] T.Lawson, “A conception of ontology”, 2004.

[35] Sireteanu, A. N., “A survey of web ontology languages and semantic web services”, Scientific Annals of the „Alexandru Ioan Cuza” University of Iaşi, Vol. 60, No. 1, 2013, str. 187-198.

[36] Spyns, P., Meersman, R., Jarrar, M., “Data modelling versus ontology engineering”, ACM SIGMOD Record, Vol. 31, No. 4, 2002.

[37] Gruber, T. R., “A translation approach to portable ontology specifications”, Knowledge Creation Diffusion Utilization, 1993.

[38] de Vergara, J. E. L., Villagrá, V. A., Berrocal, J., “Applying the web ontology language to management information definitions”, COMMAG, Vol. 68, 2004, str. 68-74.

[39] K.K.Breitman, do Prado Leite, J., “Ontology as a requirements engineering product”, Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International, Vol. 3, 2003, str. 309-319.

[40] G.Flouris, D.Manakanatas, H.Kondylakis, D.Plexousakis, G.Antoniou, “Ontology change: classification and survey”, The Knowledge Engineering Review, Vol. 00:0, 2007, str. 1-29.

[41] Noy, N., McGuiness, D., “Ontology development 101: A guide to creating your first ontology”, 2001.

[42] Ovchinnikova, E., Kuehnberger, K., “Aspects of automatic ontology extension: Adapting and regeneralizing dynamic updates”, AOW ’06 Proceedings of the second Australasian workshop on Advances in ontologies, Vol. 72, 2006, str. 51-60.

[43] K.Klarin, “Korištenje ontologija pri razvoju informacijskog sustava”.

[44] D.Jurić, “Metode učenja ontologija - trenutno stanje i problemi”.

[45] V.Mascardi, V.Cordi, P.Rosso, “A comparison of upper ontologies”, Tech. Rep., 2007.

[46] Vuković, M., Matematička logika. Element, 2009.

[47] Petrović, G., Logika. Element, 2008.

[48] Benthem, J., Ditmarsch, H., Eijck, J., J.Jaspars, Logic in Action, 2014.

[49] M.Ben-Ari, Mathematical Logic for Computer Science. Springer, 2012.

[50] S.Baronett, Logic. Oxford University Press, 2012.

[51] S.Staab, R.Studer, Handbook on Ontologies, 2007.

[52] Marchetti, A., Ronzano, F., Tesconi, M., Minutoli, S., “Formalizing knowledge by ontologies: Owl and kif”, 2008.

[53] B.Glimm, “A query language for web ontologies”.

[54] A.Fern, “Syntax and semantics syntax of fo logic”, 2010.

[55] M.S.Rozalia, Matematička logika. Univerzitet u Novom Sadu, 2012.

[56] Horrocks, I., Patel-Schneider, P., van Harmelen, F., “From shiq and rdf to owl:the making of a web ontology language - google search”, Journal of Web Semantics, Vol. 1, No. 1, 2003, str. 7-26.

[57] Taye, M. M., “Web-based ontology languages and its based description logics”, The Research Bulleting of Jordan ACM, Vol. 2, No. 2, 2012, str. 1-9.

[58] Auer, S., Ives, Z., “Integrating ontologies and relational data”, Technical Reports (CIS), 2007.

[59] F.Baader, I.Horrocks, U.Sattler, Description Logics. Elsevier, 2007, ch. Chapter 3, str. 1-45.

[60] F.Baader, D.Calvanese, D.McGuiness, D.Nardi, P.F.Patel-Schneider, The Description Logic Handbook: Theory Implementation and Applications. Cambridge University Press, 2003, ch. Basic Description Logics, str. 43-95.

[61] S.Rudolph, "Foundations of description logics", Reasoning Web 2011. LNCS, Vol. 6848, 2011, str. 384-417.

[62] M.Krötzsch, F.Simančík, I.Horrocks, "A description logic primer", Computing Research Repository (CoRR), 2012.

[63] Törmä, S., Villstedt, J., Lehtinen, V., Oliver, I., Luukkala, V., Semantic Web Services - A Survey. Helsinki University of Technology, Laboratory of Software Technology, 2008.

[64] A.Krisnadhi, P.Hitzler, "Description logics".

[65] B.Sertkaya, "A survey on how description logic ontologies benefit from formal concept analysis", Kryszkiewicz and Obiedkov, 2010, str. 2-21.

[66] F.Baader, D.Calvanese, D.McGuiness, D.Nardi, P.F.Patel-Schneider, The Description Logic Handbook: Theory Implementation and Applications. Cambridge University Press, 2003, ch. Description Logic Terminology, str. 485-505.

[67] M.R.Genesereth, R.E.Fikes, Knowledge Interchange Format, 1992.

[68] Maniraj, V., Sivakumar, D., "Ontology languages - a review", International Journal of Computer Theory and Engineering, 2010, Vol. 2, No. 6, 2010, str. 887-891.

[69] Cranefield, S., Purvis, M., "Uml as an ontology modelling language", IJCAI, 1999.

[70] P.Hayes, C.Menzel, "A semantics for the knowledge interchange format", IJCAI 2001, 2001.

[71] Uschold, M., Gruninger, M., "Ontologies: principles, methods and applications", The Knowledge Engineering Review, Vol. 11, No. 2, 1996.

[72] Nguyen, V., "Ontologies and information systems: A literature survey", Defence Science and Technology Organisation, 2011.

[73] Konstantinou, N., D.E.Spanos, Mitrou, N., "Ontology and database mapping: a survey of current implementations and future directions", Journal of Web Engineering, Vol. 7, No. 1, 2008.

[74] Sarraj, L. E., Espinasse, B., Libourel, T., Rodier, S., "Towards ontology-driven approach for data warehouse analysis", The Eight International Conference on Software Engineering Advances, 2013, str. 426-431.

[75] M.Fang, “Maintaining integrity constraints in semantic web”, 2013.

[76] Antoniou, G., Franconi, E., Harmelen, F. V., “Introduction to semantic web ontology languages”, 2005.

[77] Lisi, F. A., Malerba, D., “Ideal refinement of descriptions in al-log”, Proceedings of the 19th International Conference on Inductive Logic Programming, 2003, str. 215-232.

[78] Levy, A. Y., Rousset, M., “Carin: A representation language combining horn rules and description logics marie-christine rousset first introduction”, ECAI 96, 12th European Conference on Artificial Intelligence, 1996.

[79] T.Mailis, G.Stoilos, G.Stamou, “Expressive reasoning with horn rules and fuzzy description logics”, Proceedings of the 1st international conference on Web reasoning and rule systems, 2007.

[80] F.Donini, M.Lenzerini, D.Nardi, A.Schaerf, “Al-log: Integrating datalog and description logics”, Journal of Intelligent Information Systems (JIIS), 1998.

[81] I.Horrocks, D.Fensel, J.Broekstra, S.Decker, M.Erdmann, C.Goble, Harmelen, F., M.Klein, S.Staab, R.Studer, E.Motta, “The ontology inference layer oil”, technical report, Vrije Universiteit Amsterdam, 2000.

[82] S.Bechhofer, C.Goble, I.Horrocks, “Daml+oil is not enough”, Proceedings of the International Semantic Web Working Symposium (SWWS), 2001.

[83] O.Corcho, A.Gómez-Pérez, “A roadmap to ontology specification languages”, 12th international conference on knowledge engineering and knowledge management (EKAW-2000), 2000.

[84] I.Horrocks, “Daml+oil: A reason-able web ontology language”, International Conference on Extending Database Technology (EDBT 2002), Lecture Notes in Computer Science (LNCS), Vol. 1091, 2002, str. 11-28.

[85] D.L.McGuinness, R.Fikes, J.Hendler, L.A.Stein, “Daml+oil: An ontology language for the semantic web”, IEEE Intelligent Systems, 2002, str. 72-80.

[86] H.Nottelmann, N.Fuhr, “pdaml+oil: A probabilistic extension to daml+oil based on probabilistic datalog”, Proceedings Information Processing and Management of Uncertainty in Knowledge-Based Systems, 2004.

[87] Alalwan, N., Zedan, H., Siewe, F., “Generating owl ontology for database integration”, 2009 Third International Conference on Advances in Semantic Processing, 2009, str. 22-31.

[88] Dritsou, V., Mitraka, E., Topalis, P., Louis, C., “Getting the best from two worlds: Converting between obo and owl formats”.

[89] Chujai, P., Kerdprasop, N., Kerdprasop, K., “On transforming the er model to ontology using protégé owl tool,”, International Journal of Computer Theory and Engineering, Vol. 6, No. 6, 2014, str. 484-489.

[90] Humaira, A., Tabbasum, N., Ayesha, S., “A survey on automatic mapping of ontology to relational database schema”, Research Journal of Recent Sciences, Vol. 4, No. 4, 2015, str. 66-70.

[91] Haase, P., Stojanovic, L., “Consistent evolution of owl ontologies”, The Semantic Web: Research and Applications, 2005.

[92] Youn, S., Mcleod, D., “Ontology development tools for ontology- based knowledge management”, 2006.

[93] R.Gruber, T., “Ontolingua: A mechanism to support portable ontologies”, 1990.

[94] A.Obiniyi, O.Oyelade, G.Auta, “Ontology languages, development tools and repositories : Towards a unifying platform”, International Journal of Computer Science and Artificial Intelligence, Vol. 4, No. 3, 2014, str. 68-74.

[95] E.Motta, “An overview of the ocml modelling language”, Proceedings of the 8th Workshop on Knowledge Engineering Methods and Languages, KEML 1998, 1998.

[96] Serrano, J. M., Serrat, J., Strassner, J., “Ontology-based reasoning for supporting context-aware services on autonomic networks”, IEEE International Conference on Communications, 2007., 2007, str. 2097-2102.

[97] Gali, A., C.Chen, K.Claypool, Uceda-Sosa, R., “From ontology to relational databases”, 2004.

[98] Chaudhri, V. K., Fikes, R., Farquhar, A., Rice, J. P., Karp, P. D., Rice, J. P., “Okbc: A programmatic foundation for knowledge base interoperability”, Proceedings of the National Conference on Artificial Intelligence, 1998, str. 600-607.

[99] Zhou, H., Yang, H., Hugill, A., “An ontology-based approach to reengineering enterprise software for cloud computing”, Computer Software and Applications Conference, 2010, str. 383-388.

[100] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J., “Uml for ontology development why use uml to develop ontologies?”, 2001.

[101] Arnarsdóttir, K., Berre, A., Hahn, A., Missikoff, M., Taglino, F., "Semantic mapping: ontology-based vs. model-based approach alternative or complementary approaches?", 2006.

[102] Zedlitz, J., Luttenberger, N., "Transforming between uml conceptual models and owl2 ontologies", Workshop on Foundations, Technologies and Applications of the Geospatial Web, Terra Cognita 2012 - In Conjunction with the 11th International Semantic Web Conference, ISWC 2012, 2012, str. 15-26.

[103] The Cognitive Brain. MIT Press, 1991, ch. Building a Semantic Network, str. 99-115.

[104] J.F.Sowa, Conceptual Graphs, 2008.

[105] S.Polovina, "An introduction to conceptual graphs", ICCS 2007. LNCS (LNAI), Vol. 4604, 2007, str. 1-14.

[106] R.E.Kent, "Conceptual knowledge markup language: An introduction", Netnomics, Vol. 2, 3 2000, str. 139-169.

[107] R.E.Kent, "Conceptual knowledge markup language: The central core", Electronic Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), 1999.

[108] I.Horrocks, P.F.Patel-Schneider, D.L.McGuinness, C.A.Welty, The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition). Cambridge University Press, 2007, ch. OWL: a Description Logic Based Ontology Language for the Semantic Web, str. 1-32.

[109] J.Hebeler, M.Fisher, R.Blace, A.Perez-Lopez, Semantic Web Programming. John Wiley & Sons, 2011.

[110] "Owl web ontology language overview", dostupno na: http://www.w3.org/TR/owl-features/ Online, zadnji pristup 11.07.2016.

[111] Solic, K., Jovic, F., Blazevic, D., "An approach to the assessment of potentially risky behavior of ict systems", Tehnički vjesnik - Technical Gazette, 2013, str. 335-342.

[112] de Bruijn, J., A.Polleres, D.Fensel, "Wsml deliverable, d20 v0.1, owl lite-", WSML Working Draft, 2004.

[113] J.Heflin, "An introduction to the owl web ontology language", Lecture Notes, Lehigh University.

[114] Cuenca-Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U., "Owl2: the next step for owl", Journal of Web Semantics, 2008, str. 309-322.

[115] K.Sengupta, P.Hitzler, "Web ontology language (owl)", Encyclopedia of Social Network Analysis and Mining, 2014, dostupno na: http://corescholar.libraries.wright.edu/cse/184

[116] R.P.Srivastava, "An introduction to evidential reasoning for decision making under uncertainty: Bayesian and belief functions perspectives", International Journal of ccounting Information Systems, Vol. 12, 2010, str. 126-135.

[117] J.B.Yang, D.L.Xu, "On the evidential reasoning algorithm for multiple attribute decision analysis under uncertainty", IEEE Transactions on systems, Man and Cybernetics, Vol. 32, No. 3, 2002, str. 289-304.

[118] Blažević, D., "Predviđanje održavanja tehničkog sustava procjenom stanja", Doktorski rad, Sveučilište J.J.Strossmayera u Osijeku, Elektrotehnički fakultet Osijek, 2012.

[119] Tulasi, R. L., Rao, M. S., "Survey on techniques for ontology interoperability in semantic", 2014.

[120] R.L.Tulasi, M.S.Rao, "Tools and techniques for ontology interoperability: A survey", International Journal of Computer Science and Information Security (IJCSIS), Vol. 12, No. 8, 2014, str. 93-98.

[121] N.Choi, I.Y.Song, H.Han, "A survey on ontology mapping", SIGMOD Record, Vol. 35, No. 3, 2006, str. 34-41.

[122] Kalfoglou, Y., Schorlemmer, M., "Ontology mapping: the state of the art", The Knowledge Engineering Review, Vol. 18, No. 1, 2003, str. 1-31.

[123] Astrova, I., Kalja, A., "Storing owl ontologies in sql3 object-relational databases", WSEAS International Conference on APPLIED INFORMATICS AND COMMUNICATIONS, 2008, str. 99-103.

[124] Astrova, I., Korda, N., Kalja, A., "Storing owl ontologies in sql relational databases", Engineering and Technology, Vol. 1, No. 4, 2007, str. 1261-1266.

[125] Vysniauskas, E., Nemuraite, L., "Transforming ontology representation from owl to relational database", Information technology and control, Vol. 35, No. 3, 2006, str. 333-343.

[126] Golbreich, C., Horridge, M., Horrocks, I., Motik, B., Shearer, R., "Obo and owl: Leveraging semantic web technologies for the life sciences", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2007.

[127] Tirmizi, S. H., Miranker, D., "Obo2owl: Roundtrip between obo and owl", 2006.

[128] Verhodubs, O., Grundspenkis, J., "Algorithm of ontology transformation to concept map for usage in semantic web expert system", Applied Computer Systems, 2013, str. 80-87.

[129] Vasilecas, O., Bugaite, D., Trinkunas, J., "On approach for enterprise ontology transformation into conceptual model", International Conference on Computer Systems and Technologies - CompSysTech'06, 2006.

[130] Graudina, V., "Owl ontology transformation into concept map,", Computer Science, Vol. 34, 2008, str. 80-92.

[131] Graudina, V., Grundspenkis, J., "Concept map generation from owl ontologies", Concept Mapping: Connecting Educators. Proceedings of the 3rd International Conference on Concept Mapping, 2008, str. 173-180.

[132] Ozdikis, O., Durak, U., Oguztuzun, H., "Tool support for transformation from an owl ontology to an hla object model", Proceedings of the 3rd International Conference on Simulation Tools and Techniques, 2010.

[133] Talebzadeh, S., Seyyedi, M. A., Salajegheh, A., "Automated creating a data warehouse from unstructured semantic data", International Journal of Computer Applications, Vol. 88, No. 10, 2014, str. 19-25.

[134] Nebot, V., Berlanga, R., "Building data warehouses with semantic data", Proceedings of the 2010 EDBT/ICDT Workshops, Vol. 8, No. 2, 2010, str. 150-157.

[135] Laadidi, Y., Bahaj, M., "Designing a data warehouse from owl sources", International journal of Soft Computing and Software Engineering, Vol. 5, No. 3, 2015, str. 55-63.

[136] Gulić, M., "Transformation of owl ontology sources into data warehouse", 2013.

[137] Pardillo, J., Mazon, J. N., "Using ontologies for the design of data warehouses", International Journal of Database Management Systems, Vol. 3, No. 2, 2011, str. 73-87.

[138] V.Vrbanić, "Ontology assisted data warehouse system".

[139] Nebot, V., Berlanga, R., Perez, J. M., Aramburu, M. J., "Multidimensional integrated ontologies: A framework for designing semantic data warehouses", Journal on Data Semantics XIII, 2009, str. 1-36.

[140] Thenmozhi, M., Vivekanandan, K., "An ontology based hybrid approach to derive multidimensional schema for data warehouse", International Journal of Computer Applications, Vol. 54, No. 8, 2012, str. 36-42.

[141] Nicolicin-Georgescu, V., Benatier, V., Lehn, R., Briand, H., "An ontology-based autonomic system for improving data warehouses by cache allocation management", 2009.

[142] Ding, Z., Peng, Y., Pan, R., "A bayesian approach to uncertainty modelling in owl ontology", 2007.

[143] Zhuge, H., Shi, P., Xing, Y., He, C., "Transformation from owl description to resource space model", 1st Asian Semantic Web Conference (ASWC 2006), 2006.

[144] M.R.C.Louhdi, H.Behja, Alaoui, S., "Transformation rules for building owl ontologies from relational databases", Computer Science & Information Technology, 2013, str. 271-283.

[145] D.Zhang, K.P.Baclawski, V.J.Tsotras, "B+-tree", Encyclopedia of Database Systems, 2009, str. 197-200.

[146] T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein, Introduction to Algorithms. MIT Press, 2009.

[147] R.Fenk, V.Markl, R.Bayer, "Interval processing with the ub-tree", Proc. IDEAS, 2002, str. 12-22.

[148] H.Zhuge, "Resource space model, its design method and applications", The Journal of Systems and Software, 2004, str. 71-81.

[149] S.Abburu, "A survey on ontology reasoners and comparison", Int.J.Comput.Appl, Vol. 57, No. 17, 2012, str. 33-39.

[150] I.Horrocks, B.Motik, Z.Wang, "The hermit owl reasoner", CEUR Workshop Proc, 2012.

[151] B.Glimm, I.Horrocks, B.Motik, G.Stoilos, Z.Wang, "Hermit:an owl 2 reasoner", J.Autom.Reason, Vol. 53, No. 3, 2014, str. 245-269.

[152] T.Galba, K.Solic, I.Lukic, "An information security and privacy self-assessment (ispsa) tool for internet users", Acta Polytechnica Hungarica, Vol. 12, No. 7, 2015, str. 149-162.

[153] K.Solic, "Model za procjenu razine sigurnosti računalnog sustava zasnovan na ontologiji i algoritmu za evidencijsko zaključivanje", Doktorski rad, Sveučilište J.J.Strossmayera u Osijeku, Elektrotehnički fakultet Osijek, 2013.

[154] "Hermit zaključivač", dostupno na: http://www.hermit-reasoner.com/ Online, zadnji pristup 11.07.2016.

[155] H.Hlomani, D.Stacey, “Approaches, methods, metrics, measures and subjectivity in ontology evaluation: A survey”, Semantic Web Journal (that), 2014, str. 1-5.

[156] S.Tartir, B.Arpinar, A.P.Sheth, “Ontological evaluation and validation”, Theory and Applications of Ontology: Computer Applications, 2010, str. 115-130.

Popis kratica

ABox - Assertional box
ALC - Attributive Concept Language with Complements
AL-log - Attributive Language and Datalog
API - Application Programming Interface
BFO - Basic Formal Ontology
DAML - DARPA Agent Markup Language
DARPA - Defense Advanced Research Projects Agency
DBMS - Database Management System
DL - Description Logic
DOLCE - Descriptive Ontology for Linguistic and Cognitive Engineering
DTD - Document Type Definition
ER - Entity Relationship
FOL - First Order Logic
GFO - General Formal Ontology
GPS - Global Positioning System
HLA - High Level Architecture Objects
HTML - HyperText Markup Language
IRI - Internationalised Resource Identifier
KIF - Knowledge Interchange Format
MCDM - Multiple-Criteria Decision Making
MS SQL - Microsoft SQL
OBO - Open Biomedical Ontologies
OIL - Ontology Inference Layer
OKBC/GFP - Open Knowledge Base Connectivity/Generic Frame Protocol
OML/CKML - Ontology Markup Language/Conceptual Knowledge Markup Language
OMT - Object Model Template
OWL - Ontology Web Language
PROTON - PROToONtology
RBox - Role assertion

RDB - Relational Database
RDF - Resource Description Framework
RDFS - Resource Description Framework Schema
RS - Resource Space
RSM - Resource Space Model
SGML - Standardized Generalised Markup Language
SHOE - Simple HTML Ontology Extension
SQL - Structured Query Language
SUMO - Sugested Upper Merged Ontology
TBox - Terminological box
TURTLE - Terse RDF Triple Language
UB - Universal B-tree
UML - Unified Modeling Language
URI - Uniform Resource Identifier
W3C - World Wide Web Consortium
WWW - World Wide Web
XML - eXtensible Markup Language
XOL - Ontology Exchange Language

Popis slika

1.1. Prilagodba ontološke strukture 3

2.1. Podjela WWW po verzijama 7
2.2. Usporedba trenutnog web-a i ideje semantičkog web-a. 9
2.3. Informacijska interoperabilnost [7],[16] 10
2.4. Primjer jedinstvenog identifikatora resursa 12
2.5. Identifikacija dijela resursa fragmentom 13
2.6. Evolucija jezika za označavanje 14
2.7. Definiranje oznaka 14
2.8. Jednostavna RDF shema 15
2.9. RDF izjavna rečenica [18] 16
2.10. Specijalizacija/Generalizacija hijerarhijske strukture 18
2.11. Definicija ontologije 20
2.12. Područja primjene ontologija 22
2.13. Hijerarhijska struktura modela zadatka 23
2.14. Gradivni elementi ontologije 25
2.15. Relacija instanca - klasa 25
2.16. Relacija instanca - svojstvo 26
2.17. Relacija klasa - svojstvo 26
2.18. Prikaz izjavne rečenice stablom 30
2.20. Složena rečenica 33
2.21. Dijelovi baze znanja deskriptivne logike 38
2.22. Primjer definiranja SHOIN podjezika deskriptivne logike 41
2.23. Podjela ontoloških jezika 41
2.24. Jednostavan primjer semantičke mreže 47
2.25. Jednostavan primjer konceptualnog grafa 47

3.1. Grafički prikaz jezika za označavanje i ontoloških jezika [94] 49
3.2. Podklasni odnos između OWL jezika i RDF/RDFS jezika [32] 50
3.3. Primjer jednostavne ontologije 50

3.4. Struktura OWL dokumenta [18] 51
3.5. Tranzitivnost uvoza 54
3.6. Podjela OWL jezika 56
3.7. Primjer primitivnih klasa 60
3.8. Komponente klasnih izraza 60
3.9. Primjer enumeracije klase 61
3.10. Primjer presjeka 61
3.11. Primjer unije 62
3.12. Primjer komplementa 62
3.13. Primjer definiranja različitosti klasa 63
3.14. Primjer definiranja egzistencijalne restrikcije 63
3.15. Primjer definiranja univerzalne restrikcije 63
3.16. Primjer definiranja minimalne kardinalnosti 64
3.17. Primjer definiranja maksimalne kardinalnosti 64
3.18. Primjer definiranja *imaVrijednost* restrikcije 64
3.19. Primjer definiranja anonimne klase 64
3.20. Primjer definiranja klase nužnim uvjetima 65
3.21. Primjer definiranja klase nužnim i dovoljnim uvjetima 65
3.22. Tipovi podataka XML Sheme 67
3.23. Podjela OWL 2 jezika 69

4.1. Hijerarhijska organizacija atributa 71

5.1. Proces prilagodbe 78
5.2. Jednostavna taksonomija 79
5.3. B+ stablo 86
5.4. Primjer usmjerenog grafa 86
5.5. Primjer Z-krivulje[147] 87
5.6. Primjer višedimenzijskog prostora 87
5.7. Model transformacije ontološke strukture u taksonomsku strukturu 88
5.8. Primjer osnovnog modela s definiranom unijom i izvedenog modela 90
5.9. Primjer osnovnog modela s definiranim presjekom i izvedenog modela 90
5.10. Primjer XML dokumenta kao rezultat pokretanja zaključivača 91
5.11. Primjer indeksiranje graf strukture 93
5.12. Problem višestrukih roditelja 94
5.13. Problem povrtne veze 94
5.14. Primjer XML dokumenta izvorne strukture 95
5.15. Primjer indeksiranja čvorova 97

5.16. Primjer strukture za evidencijsko zaključivanje 99
5.17. Taksonomija o informacijskoj sigurnosti korisnika na Internetu 99

6.1. Koraci algoritma za transformaciju OWL ontologije 101
6.2. Prvo pravilo transformacije 104
6.3. Prvo pravilo transformacije nakon primjene prvog pravila optimizacije 104
6.4. Primjer definicije jednakih klasa 105
6.5. Primjer XML dokumenta iz zaključivača s popisom jednakih klasa 105
6.6. Primjer definicije presjeka klasa 106
6.7. Primjer definicije presjeka klasa 106
6.8. Podpravilo 1 107
6.9. Podpravilo 2 108
6.10. Podpravilo 3 108
6.11. Podpravilo 4 109
6.12. Primjer višestrukih roditelja 110
6.13. Podpravilo 1 110
6.14. Primjer višestrukih roditelja 111
6.15. Primjer presjeka klasa 113
6.16. Primjer rješavanja klase presjeka 113
6.17. Primjer rješavanja klase presjeka 113
6.18. Primjer jednakih klasa 114
6.19. Primjer rješavanja jednakosti klase 114
6.20. Primjer rješavanja jednakosti klase 114
6.21. Pojedinačna vremena izvršavanja 119
6.22. Ukupna vremena izvršavanja 119
6.23. Primjer ulazne ontologije 121
6.24. Rezultat transformacije prvi dio 122
6.25. Rezultat transformacije drugi dio 123

7.1. Jednostavna ontologija 127
7.2. Rezultirajuća taksonomija 127
7.3. Provjera svojstva povezanosti 131

Popis tablica

2.1. Predefinirane klase [18] 17
2.2. Temeljna svojstva RDFS-a [18] 18
2.3. Usporedba općih ontologija [45] 21
2.4. Tablica istinitosti za veznike propozicijske logike 29
2.5. Usporedba propozicijske logike i logike prvog reda 31
2.6. Opći konstrukti koncepata i svojstava [64], [66] 38
2.7. Aksiomi deskriptivne logike [64], [66] 39
2.8. Prikaz nekih konceptnih konstrukata deskriptivne logike [64], [66] 40

3.1. Konstrukti verzioniranja [18] 53
3.2. Svojstva jezika OWL Lite [112] 57
3.3. Svojstva jezika OWL DL prvi dio [108], [56] 58
3.4. Svojstva jezika OWL DL drugi dio [108], [56] 59

5.1. Usporedba tehnika interoperabilnosti 80
5.2. Usporedba metoda transformacije OWL ontologija 84
5.3. Izlazne strukture metoda transformacija ontologija 85
5.4. Izračun evidencijskog zaključivanja 100

6.1. Testne ontologije - metrike 116
6.2. Testne ontologije - Aksiomi klasa 117
6.3. Vremena izvršavanja 118

7.1. Rezultati provjere svojstava za skup ulaznih ontologija 131

Životopis

Tomislav Galba, rođen je 27.09.1987. godine u Osijeku. 2005. godine upisuje Elektrotehnički fakultet Osijek, Sveučilišta J.J.Strossmayera u Osijeku. Fakultet završava 2010. godine i stječe titulu magistar inženjer računarstva. 2011. godine počinje raditi na Elektrotehničkom fakultetu u Osijeku kao asistent, a iste godine upisuje poslijediplomski studij, smjer Komunikacije i informatika. Primarno se bavi istraživanjem ontologija i ostalim načinima modeliranja podataka, dok se još bavi efikasnim algoritmima i strukturama podataka.