Uzorci dizajna ontologija

Popović, Elvis

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Elvis Popović

UZORCI DIZAJNA ONTOLOGIJA

DIPLOMSKI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Elvis Popović

Matični broj: 46421/17–R

Studij: Baze podataka i baze znanja

UZORCI DIZAJNA ONTOLOGIJA

DIPLOMSKI RAD

Mentorica :

prof. dr. sc. Sandra Lovrenčić

Varaždin, rujan 2019.

Elvis Popović

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Smisao ontologija u računarstvu jest svrsishodni opis konceptualizacije određene domene iz realnog svijeta, s mogućnošću automatskog zaključivanja, što nalazi primjenu u bazama znanja i njihovoj tehnološkoj paradigmi - semantičkom webu. Semantički web se izražava pomoću standardnih jezika poput RDF, RDFS, OWL, pri čemu se osobita pažnja polaže na ravnotežu između ekspresivnosti i primjenjivosti automatskog zaključivanja koje svoje temelje ima u deskriptivnim logikama. U okviru semantičkog weba, postavljaju se novi zahtjevi na ontologije: prenosivost i ponovna iskoristivost što vodi ka uzorcima dizajna, kao malih provjerenih komponenti prilagođenih praktičnim zadacima. U radu se procjenjuje iskoristivost uzoraka dizajna, uspoređujući određene uzorke iz postojećih kataloga te ih primjenjujući na dvije male ontologije dizajnirane u tu svrhu.

Ključne riječi: automatsko zaključivanje; baza znanja; deskriptivne logike; OBO; ontologija; OWL; RDFS; semantički web; uzorci dizajna;

Sadržaj

1. Uvod

Pojam ontologije su računalne znanosti preuzele iz filozofije, gdje je ontologija "filozofijska disciplina koja proučava ono što jest i u čijem se središtu nalaze pitanja kao što su: Što uopće znači biti ili postojati? Što postoji? Koja su osnovna svojstva onoga što postoji? Koji su osnovni odnosi među stvarima koje postoje?" (Filozofski leksikon Leksikografskog zavoda "Miroslav Krleža", kao što citira Veljak, str. 1). Ontologija u tom kontekstu gotovo se i ne razlikuje od metafizike, te se ta dva pojma smatraju sinonimima, no informatičku ontologiju ipak treba razlikovati, jer ona predstavlja formalni sistem reprezentacije znanja u bazama znanja (Veljak, 2013, str. 2) s velikim utjecajem na semantički web. U kontekstu informacijskih sustava, računarske ontologije su artefakti koji opisuju određene aspekte svijeta (realnog, mogućeg, nestvarnog, nemogućeg, željenog), zbog neke svrhe (Gangemi i Presutti, 2009, str. 1). Nemogući svjetovi se mogu odnositi na kompleksne igre i simulacije, jer svrha informatičkih sustava već duže vrijeme nije isključivo vezana uz podršku poslovnim i realnim sustavima, kako se često smatra.

Kada govorimo o artefaktima, želimo reći da ontologije nisu same sebi svrha. One imaju zadatak definiranja znanja i strukture znanja, kako bi se tako definirano znanje moglo praktično iskoristiti za rješavanje određenih zadataka. Ontologije tako služe za dobivanje rješenja, ali same po sebi nisu rješenje. Ontologije imaju logičku strukturu, te moraju zadovoljiti domenu i zadatak, odnosno svrhu zbog koje su postavljene.
Pored prenosivosti (eng. *portability*), najvažniji zahtjev koji se polaže na ontologije jest ponovna iskoristivost (eng. *reusability*). Treba naglasiti da računalne ontologije nisu dosegle svoju punu zrelost, i još uvijek je potonja karakteristika teško ostvariva. Prema Gangemi i Presutti (2009), razlozi su slijedeći:

- veličina i kompleksnost postojećih ontologija
- nejasni razlozi odabira pojedinih načina oblikovanja postojećih ontologija
- nedostatak kriterija prilikom odabira pojedinih reinženjerskih postupaka
- krhkost alata koji bi dizajneru ontologija trebali biti od pomoći

Izolirati dijelove velikih i zapetljanih ontologija, kako bi se iskoristio njihov dizajn za izgradnju novih ontologija, često je pretežak i vremenski zahtjevan zadatak za prosječnog korisnika. Testiranje može biti vrlo zahtjevno, što je u proturječju sa pojmom ponovne iskoristivosti. Često je jednostavnije sve dijelove ontologije izraditi iz nule, umjesto da se koristi postojeće rješenje iz neke kompleksne ontologije.
Postoje neke provjerene, relativno male ontologije namijenjene izgradnji drugih ontologija. Kao primjer ću spomenuti *FOAF*[*] deskriptivnu ontologiju za opis osoba, njihovog ponašanja i povezanosti, BibTeX ontologiju za opis bibliografskih elemenata, *DOLCE*[†] ontologiju orijentiranu na

[*] FOAF - ontologija osoba, "prijatelj prijatelja" (eng. *Friend of a Friend*)

[†] DOLCE - ontologija za lingvistički i kognitivni inženjering (eng. *Descriptive Ontology for Linguistic and Cognitive Engineering*)

ontološke kategorije prirodnih jezika i ljudskog zdravog razuma. Uz ontologije treba spomenuti i konvencije i prakse poput *SKOS*[‡] modela podataka za sistem organizacije znanja, kao što su tezaurusi, klasifikacijske sheme, sustavi naslova i taksonomije (Busch, 2015, str. 3). Ipak, takve jednostavne ontologije i konvencije često nisu dovoljno fleksibilne, pa primjerice SKOS uopće nije namijenjen razvoju složenijih ontologija (Busch, 2015, str. 5).
Stoga je daleko bolje osmisliti male, univerzalne, prilagodljive gradivne elemente koji će poslužiti za razvoj kompleksnih ontologija. Upravo iz tog razloga, pojavljuje se potreba za uzorcima dizajna ontologija, baš kao što su se zbog sličnih razloga uzorci dizajna pojavili i u softverskom inženjerstvu. Ovakvi uzorci imati će provjerenu logičku strukturu koja će olakšati potencijalnom korisniku izgradnju kvalitetnih ontologija. Sam pojam uzorka dizajna dolazi od britansko-američkog arhitekta, matematičara i teoretičara dizajna Christophera Wolfganga Alexandera (1936-), koji mu je pridijelio značenje zajedničke vodilje za rješavanje dizajnerskih problema.(Gangemi i Presutti, 2009, str. 3)

Uzorci dizajna su odgovor na zahtjeve prenosivosti i ponovne iskoristivosti. Povezani sa najboljom praksom oblikovanja ontologija, ti su uzorci istovremeno i mali i često primjenjivi. Pažljivo su testirani, njihova logička struktura je provjerena, a mogu biti i standardizirani i dobro dokumentirani. Tako postaju gradivni blokovi svake ontologije koju korisnik poželi oblikovati, uz minimalne zahtjeve za promjenama i prilagodbom. Ovi uzorke su oko 2005. godine, prema uzorima iz softverskog inženjerstva, uveli Aldo Gangemi, Eva Blomqvist i Kurt Sandkuhl koristeći najbolje prakse W3C konzorcija koji se tada bavio i semantičkim webom i ontologijama.(Hammar, 2017, str. 2)

U softverskom inženjerstvu je u jednom trenutku došlo do određene krize razvoja, a bila je uzrokovana sve kompleksnijim aplikacijama koje je trebalo razvijati u razumnom, obično sve kraćem vremenu.(Sehring i dr., 2006, str. 2)
Manifestirala se u probijanju rokova i budžeta, padu kvalitete finalnog proizvoda i sve skupljem testiranju i održavanju tako proizvedenog softvera. Uzorci dizajna proizašli su iz najbolje prakse najboljih razvojnih inženjera, sistematizirani su i imenovani, što je olakšalo komunikaciju među sudionicima koji su ih oblikovali, dopunjavali, profinjivali i koristili. Slično Ekstremnom programiranju u softverskom inženjerstvu, uzorci postaju osnova *Ekstremnog ontološkog dizajna* koji uključuje raščlanjivanje složenog problema po metodi "*podijeli pa ovladaj*", izbor prikladnog uzorka za konačne produkte raščlambe, i nakon toga, evaluaciju dobivenog rezultata. (Gangemi i Presutti, 2010, str. 24)

Kada govorimo o uzorcima dizajna, bez obzira radi li se o ontologijama ili softverskom inženjerstvu, najvažniji se pokazuje koncept kataloga. Uzorci uvijek pripadaju katalogu uzoraka. Svaki uzorak u katalogu ima svoje ime, namjenu, alternativno nazivlje, problem koji rješava, sam dizajn uzorka, veze među elementima, njihove odgovornosti i suradnje, te posljedice njegovog izbora odnosno primjene. Uzorci dizajna postaju nužnost za inženjersku primjenu, bez obzira radi li se o softverskom inženjerstvu ili inženjerstvu ontologija.(Gamma i dr., 1997, str. 10)
I dok je u softverskom inženjerstvu primjena uzoraka prepoznata u praksi, u ontologijama, koje su mlađe odnosno slabije razvijeno područje informacijskih znanosti, uzorci su još uvijek u pr-

[‡]SKOS - jednostavni sustav organizacije znanja (eng. *Simple Knowledge Organization System*)

vobitnoj fazi razvoja i usklađivanja.
Postoji nekoliko problema povezanih sa uzorcima dizajna, na koje nailaze istraživači ontologija prema Hammar (2017):

- Nedovoljno istražena kvaliteta samih predloženih i primjenjivanih uzoraka: pri tome se misli na nedostatak publikacija koje procjenjuju takve uzorke.
- Nedostatak metodologija i alata za zadatke fine granularnosti: obično se opisuju postupci za složene i veće zadatke. No kada se ti zadaci raščlane, potrebno je imati raspoložive metode za njihovo rješavanje.
- Nedostatak empirijske evaluacije metodologija koje se bave implementacijom ontologija: osobito je problematično praćenje ponašanja tih ontologija u različitim scenarijima i određivanje potrebnih adaptacija kako bi se prilagodile tim scenarijima.
- Nedostatak znanja primjene ontologija u scenarijima iz prakse: najčešće se ontologije odnose na sintetičke, akademske scenarije koji često ne pokrivaju praktične probleme.

Pored ovih problema, pojavljuje se i ključni problem nedovoljnog interesa prakse kako za same ontologije, tako i za njihovu sistematizaciju te uvođenje metodologija procjene najboljih praksi i posljedično uzoraka dizajna ontologija.
Moja motivacija da se posvetim ovom problemu proizašla je iz ovih izvora:

- želje da bolje istražim same uzorke dizajna ontologija, kako bih bolje upoznao i sam proces razvoja ontologija
- želje sa povežem postojeće znanje iz softverskog inženjerstva vezano uz uzorke dizajna, sa još uvijek mladom i nedovoljno razvijenom teorijom uzoraka dizajna ontologija zbog čega je moguće dati i svoj vlastiti doprinos na tom području
- potrebe da usavršim vještine izrade ontologija, jasno prepoznajući česte logičke pogreške te valjane smjernice za rješavanje čestih problema koji se u razvoju pojavljuju

Kako ćemo kasnije vidjeti, uzorci dizajna ontologija jesu podijeljeni u određene obitelji, ali nisu pročišćeni, niti je utvrđen njihov konačni broj. Svakim danom se pojavljuju novi uzorci sukladno određenim domenskim problemima. Često su uzorci ovisni o određenoj domeni (Falbo i dr., 2014, str. 4), kao što je slučaj sa uzorcima sadržaja *NeON kataloga*. Zbog sve te nedorečenosti i nezrelosti, ovo područje će svoj puni potencijal tek pokazati u budućnosti. U ovom trenutku to izgleda kao obećanje intenzivnog razvoja, tim više što za ontologijama i bazama znanja raste potreba u znanstvenoj zajednici. Stoga je potrebno pratiti sve promjene i razvoj na polju teorije i primjene uzoraka dizajna ontologija.

U ovom radu detaljno ću iznijeti teorijsku pozadinu koja je stajala iza razvoja semantičkog weba, ali i zaključivača (zaključivači, eng. *reasoners*) koji se koriste u ontologijama, uključujući i logiku okvira (F-logiku), proširenje Hornove klauzule na generalni logički program, te jezike poput Ruleloga, ErgoAI platforme i open source verzije pod imenom Ergo Lite odnosno FLORA-2. Zatim ću razraditi pozadinu semantičkog weba, deskriptivne logike i njihove izražajnosti, te jezike za prikaz i dijeljenje znanja poput RDFS i ontoloških OWL jezika.

Opisat ću GOF i POSA kataloge uzoraka dizajna u softverskom inženjerstvu, a nakon toga i obitelji uzoraka dizajna ontologija. Uz upotrebu dijagramskih formalizama i formalizma deskriptivnih logika opisati najzanimljivije uzorke za svaku obitelj u okviru kataloga NeOn projekta, sve uzorke iz Sourceforge kataloga otvorenih biomedicinskih ontologija kao i neke uzorke primjenjive na višeagnente sustave. Nastojat ću povezati uzorke iz različitih kataloga, pri čemu će stožerni katalog biti katalog biomedicinskih ontologija budući da je jezgrovit, lako shvatljiv i široko primjenjiv. Navest ću najznačajnije dostupne alate za razvoj ontologija i uz upotrebu alata Protégé opisati ontologiju o pizzama i DOLCE Ultralite ontologiju navodeći uzorke koji se u njima mogu uočiti.

U konačnici ću prikazati dvije male ontologije nastojeći koristiti metode primjene uzoraka dizajna. Ontologije koje ću razraditi su ontologija uzoraka dizajna četvorice autora (GOF) u softverskom inženjerstvu, uz prijedlog poveznica sa težinskim faktorima, te ontologija ekološkog zbrinjavanja otpada (EZO) bazirana na projektnom zadatku u okviru predmeta "Uzorci dizajna" za akademsku godinu 2018/2019. Opisat ću glavne dijelove dizajna, uz osvrt na uzorke koji su upotrijebljeni u njegovoj razradi. Pri tome ću koristiti Protégé i FaCT++ zaključivač kao osnovne alate, Visual Paradigm za izradu dijagrama te LaTeX MikTeX i Overleaf LaTeX editore.

Svi dijagrami u radu, ako nisu preuzeti iz vanjskih izvora, nastali su upotrebom Tikz paketa unutar samog LaTeX-a te Visual Paradigm-a.

2. Ontologije i semantički web

Nemoguće je govoriti o uzorcima dizajna ontologija, ukoliko se ne pojasni pojam ontologije te uloga ontologija, odnosno vokabulara u semantičkom webu. Potrebno je objasniti i semantički web kao tehnologiju za prijenos znanja, a koja se razvijala paralelno sa klasičnim webom od samih njegovih početaka. Semantički web je vrsta baze znanja sa svrhom dijeljenja znanja na webu. Za razliku od klasičnog weba koji ima istog tvorca, Tima Bernersa Leea, semantički web je namijenjen strojnom razumijevanju značenja sadržaja i automatskom zaključivanju. Znanje, odnosno baze znanja kritični su elementi za upravo aktualnu tehnologiju umjetne inteligencije ali i tehnologije ekspertnih sustava odnosno sustava za potporu odlučivanju. Jedna od njihovih reprezentacija su ontologije, koje su definicije strukture znanja sadržanog u bazama znanja a time i semantičkom webu.

Pojam ontologije dolazi iz filozofije gdje označava granu filozofije koja opisuje sve ono što postoji i po čemu postoji (Veljak, 2013, str. 1), odnosno prirodu i strukturu stvarnosti (Guarino, Oberle i Staab, 2009, str. 1), te ih treba razlikovati od pojma *epistemologije*, teorije spoznaje koja govori o samom znanju. Osim toga, ontologija se razlikuje od epistemologije i po tome što je ontološka opstojnost predmeta spoznaje nužno povezana sa znanstveno utemeljenom spoznajom.(Veljak, 2013, str. 5)
Jednako tako, ontologija se poistovjećuje sa metafizikom, ili smatra njezinim dijelom, no postoje suptilne jezične razlike, budući da je predmet ontologije uži od predmeta metafizike. Ontologija je tako dio metafizike koji se bavi bićem općenito i njegovim transcendentalnim svojstvima, odnosno ontologija je specijalna metafizika.(Veljak, 2013, str. 1)

Ontologiju u filozofiji treba razlikovati od ontologije u informacijskim znanostima jer potonja istražuje formalne sisteme reprezentacije.(Veljak, 2013, str. 2). U daljnjem tekstu pod pojmom ontologije ću podrazumijevati informatičku ontologiju. U informacijskim znanostima ali i lingvistici, ontologije su modeli realnog svijeta, odnosno određene problemske domene zajedno sa ograničenjima koje su prisutne u toj domeni. Vrlo često su bliskije modelima podataka nego konkretnim implementacijama. Ontologije opisuju strukturalna svojstva objekata, a ne zanima ju njihovo postojanje.(Guarino, 1995, str. 2)
Treba ih razlikovati od baza znanja kojima su temelj. Istovremeno, ontologije su posebne vrste baza znanja, pri čemu se više različitih baza znanja mogu odnositi na istu ontologiju.(Guarino, 1995, str. 5)

Prema najkraćoj mogućoj definiciji, ontologije su formalne eksplicitne specifikacije dijeljenih konceptualizacija (Marchetti i dr., 2009, str. 1). Tom Gruber (kao što citiraju Guarino, Oberle i Staab (2009), str. 3) konceptualizaciju vidi kao "apstraktan, pojednostavljen pogled na svijet koji želimo reprezentirati zbog nekog razloga".

Prema Gruberu, "ontologije su primitivi pomoću kojih se modelira domena određenog znanja ili predmeta razgovora".(Gruber, 2009) Ti primitivi su najčešće klase odnosno skupovi, atributi odnosno svojstva, i relacije među objektima odnosno predstavnicima pojedinih klasa (Gruber, 2009). Ontologiju čine formalni, eksplicitni opisi koncepata u domeni, odnosno klasa, svojstava svakog koncepta tj. slotova, te ograničenja nad slotovima koja se nazivaju i facetama.

Svojstva se nazivaju i ulogama (eng. *roles*), koje u deskriptivnoj logici označavaju binarne relacije.(Noy i McGuinness, 2014, str. 3)

Ontologije nisu nužno vezane uz semantički web. Njihova opća svrha je opis, odnosno specifikacija koncepata i relacija u okviru određene domene, što može biti vezano za lingvistiku, medicinu i druge primjene. Ipak, u semantičkom webu su postale ključni element, budući da bez njih, RDF trojke čine samo još jedan oblik baze podataka bez mogućnosti specificiranja koncepata, ograničenja i aksioma, odnosno formalnih specifikacija koncepata domene. S ontologijama se povezuju sustavi za automatsko zaključivanje. Upravo zaključivanje i jest razlog zašto specifikacije moraju biti formalne.(Obitko, 2007)

Baze znanja su kovanica koja označava vrstu baza podataka koja se razlikuje od danas uobičajenih relacijskih baza koje podatke čuvaju u tablicama. Tablice su reprezentanti relacija, one se mogu povezivati, i nad takvim povezanim strukturama se mogu postavljati jednostavni ali i složeni upiti. Baza znanja je posebna vrsta baza podataka koja se sastoji od 3 dijela:(L. Naykhanova i I. Naykhanova, 2018, str. 3)

- Baze činjenica - skup parametara određenog zadatka
- Produkcijskih pravila - uvjete i akcije koje se poduzimaju ukoliko je ispunjen uvjet
- Ontologije - povezanost pravila i situacija, te znanje o poslovnim pravilima i zakonima

"Baze znanja se mogu smatrati proširenjem ontologija specifičnim znanjem vezanim uz konkretnu primjenu."
(Martínez, Taboada i Mira, 2003, str. 1)

Baza znanja u općenitijem smislu je dinamična i opisuje pored strukture i stanja u toj domeni. To ne znači da i same ontologije nisu dinamične. One svoju strukturu mijenjaju daleko brže nego što to čini struktura objektnog programa u softverskom inženjerstvu koja se može mijenjati tek pri revizijama softvera.

2.1. Logika prvog reda

Ontološki jezici poput jezika OWL utemeljeni su na deskriptivnim logikama. Njihov je temelj logika 1. reda odnosno račun predikata. "Izvorno je razvijena kao teorija1930-e godine isključivo za matematičke potrebe te predstavlja mehanizam zaključivanja opisan logičkim simbolima.(Galba, 2016, str. 30)

U logici 1. reda postoje i objekti i relacije među njima (predikati), pri čemu objekti mogu imati vrlo složenu unutrašnju strukturu. Logika prvog reda je neodlučiva, tj. *ne postoji test* kojim bi se za svaku formulu u konačno mnogo koraka moglo ispitati je li formula valjana. Upravo je neodlučivost i složenost pri zaključivanju jedna od najvećih prepreka kod primjene u semantičkom webu. U računu sudova i logici 1. reda važan je pojam rečenice odnosno formule. Dobro oblikovana formula se u logici 1. reda definira induktivno na slijedeći način:

Definicija 2.1.1 *Definicija formule za račun predikata(Čubrilo, 1985)*

1) Svaka atomarna formula je formula.
2) Ako su F i G formule, tada su $(\neg F)$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$ $(F \leftrightarrow G)$ *također formule.*
3) Ako je F formula, a x varijabla, tada su i riječi $(\forall x F)$ *i* $(\exists x F)$ *također formule.*
4) riječ je σ*-formula (ili samo formula) ako i samo ako je nastala primjenom konačno mnogo puta pravila od 1) do 3).*

Formula se gradi na osnovu abecede koja se u logici 1. reda definira na slijedeći način:

Definicija 2.1.2 *Definicija abecede za račun predikata Abeceda* $\mathcal{A}$ *jezika računa predikata* $\mathcal{L}(RP)$ *je unija slijedećih skupova (Vuković, 2007, str. 124):*

$$A_1 = \{c_i : i \in I \subseteq \mathbb{N}\} - najviše\ prebrojiv\ skup\ konstantnih\ simbola$$
$$A_2 = \{x_j : j \in J \subseteq \mathbb{N}\} - najviše\ prebrojiv\ skup\ individualnih\ varijabli$$
$$A_3 = \{f_k^{m_k} : k \in K \subseteq \mathbb{N}\} - najviše\ prebrojiv\ skup\ fun.\ simbola\ konačnih\ kratnosti\ m_k.$$
$$A_4 = \{P_l^{n_l} : l \in L \subseteq \mathbb{N}\} - najviše\ prebrojiv\ skup\ relacijskih\ simbola\ konačne\ kratnosti\ n_l.$$
$$A_5 = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\} - skup\ logičkih\ veznika$$
$$A_6 = \{\forall, \exists\} - skup\ kvantifikatora\ (univerzal\ i\ egzistencijal)$$
$$A_7 = \{(,)\} - skup\ pomoćnih\ simbola\ (lijeva\ i\ desna\ zagrada)$$

U logici 1. reda atomarna formula je zamjena za literal u logici sudova i definira se na slijedeći način:

Definicija 2.1.3 *Atomarna formula - atom (Vuković, 2007, str. 126)*
Neka je P neki n-arni relacijski simbol (predikat), a $t_1, \ldots, t_n$ *su termi. Tada je riječ* $P(t_1, \ldots, t_n)$ *atomarna formula.*

Logika 1. reda je definirana jezikom, skupom aksioma i pravilima izvoda. I dok se abeceda odnosi na sintaksu logike, interpretacije se odnose na semantiku koja svakoj formuli pridružuje vrijednost iz domene interpretacije $\mathcal{D}$. Glavni zadatak je omogućiti automatsko izvođenje logičkih posljedica, kroz postupke normalizacije i unifikacije te postupke dobivanja rezolventi. Kako bi se pojednostavilo automatsko zaključivanje i logičko programiranje uvode se logičke klauzule i linearna rezolucija definitnih klauzula (SLD rezolucija, eng. *Selective Linear Definite*). SLD rezolucija, kao restrikcija općeg pravila zaključivanja i osnova automatskog zaključivanja u logičkom programiranju, nije potpuna na skupu svih formula logike 1. reda, ona je potpuna na skupu Hornovih klauzula. Općenito su klauzule u logici 1. reda konačni disjunkti atomarnih formula, a ako sadrže najviše jedan pozitivni literal odnosno atomarnu formulu, takva se klauzula naziva Hornovom klauzulom. Primjer takve klauzule je $\neg F_1 \vee \neg F_2 \vee \neg F_3 \vee G$ gdje je G pozitivni literal u računu sudova, a u logici 1. reda analogno atomarna formula. Ukoliko Hornova klauzula sadrži točno jedan pozitivni literal, takva se klauzula naziva definitna klauzula, pri čemu se pozitivni literal piše na početku, i naziva glavom klauzule a ostali literali čine tijelo klauzule. Formalno se zapisuje $G \leftarrow F_1, F_2, F_3$ što se jednostavno može pokazati da slijedi iz

početne klauzule. Ukoliko definitna klauzula nema tijelo, već samo jedan pozitivni literal odnosno glavu, takva se Hornova klauzula naziva jedinična klauzula ili činjenica. Označavamo ju $F \leftarrow$. Sa druge strane, Hornova klauzula koja ima samo tijelo, odnosno samo negativne literale, naziva se negativna klauzula, ciljna klauzula ili definitni cilj. Ona se označava formalno na slijedeće načine:

$$\begin{aligned}
&\leftarrow F_1, F_2, F_3, \ldots, F_n \equiv \\
&\leftarrow F_1 \wedge F_2 \wedge F_3 \wedge \ldots \wedge F_n \equiv \\
&\vee \neg(F_1 \wedge F_2 \wedge F_3 \wedge \ldots F_n) \equiv \\
&\vee \neg F_1 \vee \neg F_2 \vee \neg F_3 \vee \ldots \vee \neg F_n
\end{aligned}$$

Ako ciljna klauzula nema niti jedan literal, onda se označava kao prazna klauzula posebnom oznakom $\square$. Činjenice možemo smatrati posebnom vrstom programskih klauzula bez tijela (imaju samo glavu $F_0 \leftarrow$).

Klauzule na taj način mogu tvoriti programe koji imaju izražajnu snagu Turingovog stroja, što znači da je njima moguće ostvariti sve parcijalno rekurzivne funkcije pa tako i bilo koja procedura programa sa slobodnim pristupom memoriji (Von Neumanov stroj) koji je uobičajeno napisan kao niz asemblerskih programskih instrukcija. Ova je činjenica osnova logičkog programiranja zasnovanog na Hornovim klauzulama i SLD rezoluciji*.

Klauzule u SLD rezoluciji zapravo mogu biti ili definitna programska klauzula ili definitna ciljna klauzula. Samo za napomenu, *procedura* je u logici 1. reda skup programskih klauzula kojima su glave označene istim relacijskim simbolom, dok se skup takvih procedura naziva *logičkim programom*. Procedura koja se sastoji samo od činjenica, a koje se sastoje samo od funkcijskih i konstantnih simbola, dakle, koje ne sadrže varijable, naziva se *bazom znanja*.(Čačić, Paradžik i Vuković, 2014, str. 9)

2.2. Logike okvira

Logike okvira ili F-Logike imaju veću izražajnost od logike 1. reda, pa uz snažne F-jezike također mogu biti upotrijebljene za prikaz znanja i razvoj ontologija, no u ovom trenutku su manje popularne od raširenih jezika zasnovanih na manje izražajnim ali i jednostavnijim za automatsko zaključivanje - deskriptivnim logikama i jezicima poput OWL-a. Na slici 1 (str. 11) prikazani su konstrukti F-logike iz kojih se jasno vidi orijentiranost na objektno programiranje. Uočavamo klase, objekte, svojstva odnosno atribute, metode kao atribute sa parametrima, mehanizam nasljeđivanja i sve drugo što se veže uz objektno orijentirani pristup popularan u softverskom inženjerstvu, a što nije svojstveno logičkom programiranju i jezicima poput PROLOG-a, bar ne u njihovom osnovnom obliku. Povećanje izražajnosti deskriptivnih logika prema Balaban (1995) ima za svrhu postići slijedeće ciljeve:

- proširenje mehanizma za prikaz i zaključivanje: relacije koje ne ovise o taksonomijama

*SLD rezolucija (eng. *Selective Linear Definite*) je linearna rezolucija sa selektivnom funkcijom za definitne klauzule. Valjana je i potpuna u smislu opovrgavanja za Hornove klauzule.

poput povezivanja, n-arne relacije, relacije između dijelova i cjeline (eng. *part-whole relations*), konstrukti višeg reda, kolektivni entiteti, pretpostavke i supsumpcije po postavci, intenzije i pogledi, cirkularne terminološke definicije, samoreferenciranje i dr.

- fleksibilnije definicije
- uniformna teorija dokaza
- integracija s logičkim programiranjem, objektno orijentiranim sustavima, funkcionalnim programiranjem i tehnologijama baza podataka
- omogućavanje inkrementalnog razvoja i implementacije

Za F-logiku vrijede jednake definicije kao i u logici 1. reda uz dodatna proširenja koja moraju omogućiti objektni pristup. Tu se prije svega misli na tri tipa *molekula*: Is-a molekula, podatkovnih i signaturnih molekula.

2.2.1. Sintaksa F-logike

Abeceda F-logike sastoji se od skupa objektnih konstruktora $\mathcal{F}$, beskonačnog skupa varijabli $\mathcal{V}$, pomoćnih simbola kao što su zagrade (i), $\rightarrow$, $\twoheadrightarrow$, $\bullet\!\!\rightarrow$, $\bullet\!\!\twoheadrightarrow$, $\Rightarrow$, $\Rrightarrow$, te logički simboli kao i u logici 1. reda, uključujući univerzalni i egzistencijalni kvantifikator. U F-logici, koja radi s klasama i objektima, atomarne formule su $\top, \bot,$
$P(t_1, t_2, \ldots, t_n), t_i = t_j$. Nakon toga definiramo pojam *molekule*.

Definicija 2.2.1 *Molekularna formula - molekula prema (Kifer, Lausen i Wu, 1994, str. 12) Molekula u F-logici je jedna od slijedećih izjava:*

i) "IS-A" tvrdnja oblika C::D ili oblika O:C gdje su C,D i O id-termi

ii) objektna molekula oblika O[lista izraza metoda odijeljena znakom ;].
Izrazi metoda može nenasljedivi podatkovni izraz, nasljedivi podatkovni izraz i signaturni izraz.

- *Nenasljedivi podatkovni izraz može biti u ove dvije forme:*
  - *nenasljedivi skalarni izraz ($k \geq 0$):*
    $SkalarnaMetoda@Q_1, \ldots, Q_k \rightarrow T$
  - *nenasljedivi skupovni izraz ($l, m \geq 0$):*
    $SkupovnaMetoda@R_1, \ldots, R_i \twoheadrightarrow \{S_1, \ldots, S_m\}$
- *Nasljedivi podatkovni izraz u dvije forme:*
  - *nasljedivi skalarni izraz ($k \geq 0$):*
    $SkalarnaMetoda@Q_1, \ldots, Q_k \bullet\!\!\rightarrow T$
  - *nasljedivi skupovni izraz ($l, m \geq 0$):*
    $SkupovnaMetoda@R_1, \ldots, R_i \bullet\!\!\twoheadrightarrow \{S_1, \ldots, S_m\}$
- *Signaturni izrazi u dva oblika:*

– *Skalarni signaturni izraz* ($n, r \geq 0$)*:*
$SkalarnaMetoda@V_1, \ldots, V_n \Rightarrow (A_1, \ldots, A_r)$
– *Skupovni signaturni izraz* ($s, t \geq 0$)*:*
$SkupovnaMetoda@W_1, \ldots, W_s \Rightarrow\!\!> (B_1, \ldots, B_t)$

F-logika razlikuje funkcijsko pridruživanje $\rightarrow$ i skupovno pridruživanje $\twoheadrightarrow$ pri čemu se u prvom slučaju atributu odnosno metodi pridružuje skalarna vrijednost, a u drugom slučaju skup određene kardinalnosti veće od 1. F-logika ne razlikuje objekte i vrijednosti. Tako je pridružena vrijednost nekom atributu zapravo pripadnik neke druge klase, na primjer klase String, Integer i slično. Ta klasa kojoj vrijednost pripada naziva se tip, i također se pridružuje atributu ali putem signature. Tipovi nisu ograničeni samo na predefinirane tipove, već to mogu biti i korisnički tipovi. Primjer je slijedeća molekula za oid† Student:
`Student[Ime=>\string, ProsjekOcjena=>\float, prijatelj =>> Osoba].`
Prema ovoj objektnoj molekuli objekt sa oid-om Student može imati za prijatelje skup osoba koje su individue i pripadaju klasi Osoba.
Izraz `O[atribut -> T]` pridružuje atributu vrijednost T, dok izraz `O[atribut ->> {T1, T2}]` atributu pridružuje skup vrijednosti. Jezici utemeljeni na F-logici poput *FLORA-2* ne prave razliku između skalarnih i skupovnih pridruživanja i uvijek koristi operator `->`. U F-logici atributu se mogu pridružiti vrijednosti koje mogu imati više tipova
`imovina=>(Nekretnina, Kuća)`, kao i naslijeđeni tipovi `kolege=>>Student(Osoba)`.
Specifično za primjer imovine, ona mora pripadati istovremeno objema klasama (konjukcija).
Atribut koji ima samo jedan tip (pripada samo jednoj klasi), naziva se *singleton*.
Atributi mogu biti i metode koje primaju neke argumente i vraćaju određene vrijednosti, na primjer `kupio@proizvod=>Račun` gdje se argument označava sa @.

Atom ili molekula koja ne sadrži varijable naziva se osnovnim (eng. *ground*) atomom ili molekulom. Atomi i molekule grade pravila na slijedeći način:

$$h \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

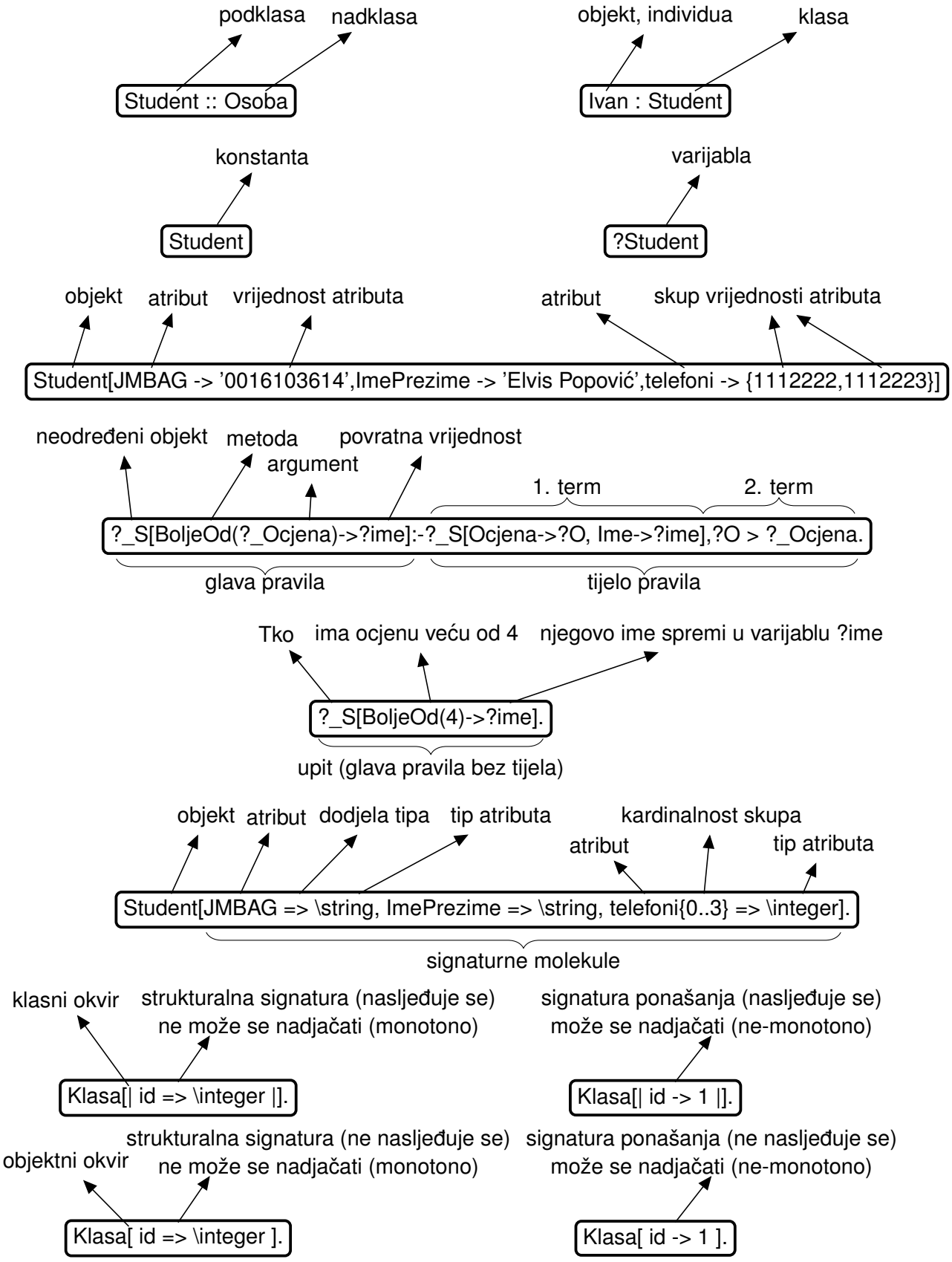
gdje su $h, b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$ slobodni atomi ili molekule koji pripadaju skupovima $H(r) = h$ (glava pravila), $B^+(r) = \{b_1, \ldots, b_m\}$ (pozitivni dio tijela pravila) i $B^-(r) = \{\neg c_1, \ldots, \neg c_n\}$ (negativni dio tijela pravila).

2.2.2. Generalno logičko pravilo

Pravila u F-logici razlikuju od Hornovih klauzula koje ne dozvoljavaju negaciju atoma u tijelu klauzule, a jednako tako i od programskih klauzula i logičkih programa zasnovanih na njima. Tako generalni logički program čini konačni skup generalnih pravila sa pozitivnim i negativnim pociljevima. Podciljevi su zapravo literali u tijelu klauzule oblika

$$p(X) \leftarrow a(X), \neg b(X)$$

†OID (eng. *object id*, akronim koji označava identifikator objekta)



Slika 1: Konstrukti u F-logici i Flora-2

Možemo reći da je Hornova klauzula specijalni slučaj generalnog pravila bez negativnih podciljeva (u primjeru je negativni podcilj $\neg b(X)$). Iako se može za generalno pravilo govoriti o klauzulama sa atomarnim podciljevima, ponekad se i tu čini razlika, pa se za klauzule sa atomarnim ciljevima govori o normalnim pravilima, dok se pojam generalnog pravila koristi za takva pravila koja u tijelu imaju složenije podciljeve kao u slijedećoj klauzuli: (Van Gelder, Ross i Schliph, 1991, str. 4)

$$w(X) \leftarrow m(X,Y), \neg(m(Y,Z), \neg w(Z))$$

U F-logici klauzula ima oblik $glava \leftarrow tijelo$ gdje je glava F-molekula a tijelo konjukcija F-molekula. Hornov program sa matematičkog stanovišta je jednak u F-logici kao i u logici 1. reda sve dok u tijelu klauzule nema negiranih molekula. Ako ima, tada govorimo o generaliziranom logičkom programu. Za generalizirani program više ne vrijede jednostavne poveznice sa logikom 1. reda, jer se gubi jedinstvenost minimalnog modela. U teoriji (Van Gelder, Ross i Schliph, 1991, str. 16) se govori o skupu kanonskih modela kao podskupu skupa svih minimalnih modela programa. Skup kanonskih modela nije jedinstveno dogovoren, ali za lokalno stratificirane probleme postoji konsenzus oko toga što je kanonski model. (Kifer, Lausen i Wu, 1994, str. 40)

F-logika, a tako i iz nje izvedeni jezici poput ErgoAI ili FLORA-e, omogućavaju i negativne atome u pravilima. Tako logički program može imati više takvih pravila koja nose sa sobom određene probleme. Osnovni problem jest nejedinstvenost minimalnog modela logičkog programa. Pod modelom smatramo interpretaciju u kojoj vrijede sve činjenice, sva pravila i sva ograničenja (restrikcije) u deduktivnoj bazi podataka ili logičkom programu. Minimalni model je najmanji skup takvih činjenica, pravila i ograničenja.

2.2.3. Stratifikacija

Negativni atomi u pravilima logičkog programa zahtijevaju konvenciju prilikom određivanja intendiranog, željenog jedinstvenog minimalnog modela. Jedan od mogućnosti određivanja takvog jedinstvenog minimalnog modela je i postupak stratifikacije. Stratificiranje je grupiranje pravila u slojeve odnosno *stratume* koji su međusobno usporedivi, odnosno može se odrediti koji je niži a koji viši. Ako su S_i slojevi, tada se može uspostaviti uređaj $S_1 < S_2 < \ldots < S_n$ tako da vrijedi (Maleković i Schatten, 2017, str. 184):

$$p \leftarrow \ldots q \ldots, p \in S_p, q \in S_q \Rightarrow S_p >= S_q \tag{2.1}$$

$$p \leftarrow \ldots \neg q \ldots, p \in S_p, q \in S_q \Rightarrow S_p > S_q \tag{2.2}$$

Stratifikacija se neće moći izvesti ako u programu postoje *negativni ciklusi*. Pravila logičkog primjera $p \leftarrow q$ uvijek možemo prikazati grafom gdje će atomi $p, q, \ldots$ činiti čvorove, usmjerenog grafa. Pojava ciklusa u takvom usmjerenom grafu značit će ulaske u beskonačne rekurzije i kao posljedicu toga nemogućnost zaključivanja. Promotrimo primjer (Maleković i Schatten, 2017,

str. 183):

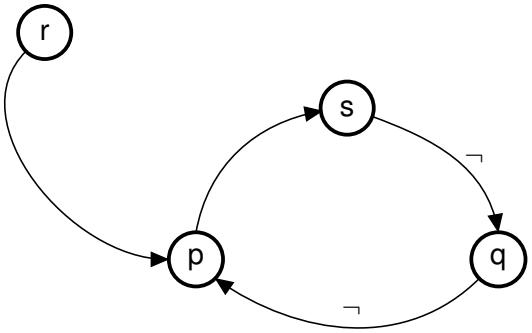
$$p \leftarrow r$$
$$p \leftarrow \neg q$$
$$q \leftarrow \neg s$$
$$s \leftarrow p$$

Ovaj skup pravila moguće je grafički prikazati pomoću *grafa zavisnosti* kao na slici 2. Kao što se vidi, u ciklusu (p s p q) znak $\neg$ se pojavljuje dva puta. To ukazuje na *negativni ciklus*. Negativni ciklus u deduktivnoj bazi podataka, odnosno u logičkom programu postoji ako su ispunjena dva uvjeta:

1. postoji pravilo $q \leftarrow \neg p$.
2. postoji put od q do p u pripadnom grafu zavisnosti

Ciklus će značiti da u primjeru logičkog programa iz q možemo doći u p preko luka sa negacijom $\neg$ (zbog pravila $p \leftarrow \neg q$), ali i obrnuto, iz p možemo doći u q posredno preko s (zbog dvaju pravila $s \leftarrow p$ i $q \leftarrow \neg s$). Ova dva puta zajedno čine ciklus. Logički program se u takvom ciklusu može beskonačno vrtjeti. U gornjem primjeru možemo primjetiti i da je moguće ovaj ciklus gledati prema pravilu $q \leftarrow \neg s$, jer postoji i put od q do s.

Γ_z :



Slika 2: Graf zavisnosti, prema izvoru: Maleković i Schatten (2017)

Ukoliko postoji ciklus u logičkom programu, takav se program neće moći stratificirati, odnosno neće se moći utvrditi jedinstveni minimalni model. Slijedeći algoritam omogućava određivanje postojanja ciklusa u logičkom programu, a ako takvih ciklusa nema, rezultirat će stratifikacijom logičkih pravila.

Algoritam 1 *Utvrđivanje negativnog ciklusa (Maleković i Schatten, 2017, str. 184)*
Postupak:

1. *Staviti sve relacije iz logičkog programa u stratum* S_1
2. *Ispitati pravila:*
   (a) *ako je* $p \leftarrow \ldots q \ldots$ *, te* $p \in S_i, q \in S_j, i \leq j$*, onda staviti* p *u* S_j

(b) ako je $p \leftarrow \ldots \neg q \ldots$ *, te* $p \in S_i, q \in S_j, i \leq j$*, onda staviti* p *u* S_{j+1}

3. *Ako se dobiveni skupovi relacija* $S_1 < S_2 < \ldots < S_k$ *ne mijenjaju pomoću točke 2, tada program* nema *negativnih ciklusa, i postignuta je stratifikacija* $S_1 < S_2 < \ldots < S_k$*.*

4. *Ako se ikada dođe do uvjeta da je neka relacija* $r \in S_j$ *te* $j > n$*, gdje je* n *broj relacija (pravila) u logičkom programu, tada postoji negativni ciklus i logički se program* ne može *stratificirati.*

U navedenom algoritmu, za stratume S_j, indeksi j označavaju rangove.

2.2.4. Dobro oblikovani model

Dva su pristupa pri određivanju kanonskog modela: stabilni model (eng. *stable model*) i dobro oblikovani model (eng. *well-founded model*). Za većinu logičkih programa te se dvije semantike podudaraju. Ipak postoje i protuprimjeri kada se one ne podudaraju. Pod dobro oblikovanom semantikom programa podrazumijeva se da svaki pozitivni literal označava svoj atom kao istinit, a svaki negativni literal svoj atom kao neistinit, te nedostajuće atome bez pridružene istinitosti. (Van Gelder, Ross i Schliph, 1991, str. 8)

Dobro oblikovani model odnosno dobro oblikovana semantika se za lokalno stratificirane probleme[‡] podudara sa savršenim modelom.

2.2.5. Rulelog i FLORA-2

Jedan od dijalekata F-logike je *Rulelog*, ekspresivni semantički jezik čija je svrha reprezentacija znanja i logičko zaključivanje u objektnom okruženju. Rulelog je komercijalno implementirala tvrtka *Coherent Knowledge* unutar platforme ErgoAI koja je namijenjena upravljanju složenim i promjenjivim logički strukturiranim informacijama. ErgoAI besplatno izdanje otvorenog koda nosi naziv Ergo Lite poznatiji pod imenom Flora-2[§].

Flora-2 je sustav za prikaz znanja i automatsko zaključivanje baziran na pravilima. Čine ga tri komponente:

- Logika okvira (eng. *Frame logic*) odnosno F-logika, objektna ekstenzija logike 1. reda
- HiLog sustav za logički jasno metaprogramiranje zasnovan na logici višeg reda
- Transakcijska logika koja omogućuje osvježavanje odnosno dinamičke promjene baza znanja

[‡] Przymusinski definira lokalno stratificirani logički program kao program u kojem se svakom atomu u njegovoj Herbrandovoj bazi može pridružiti rang, tako da niti jedan atom ne ovisi pozitivno o atomu većeg ranga, niti ovisi negativno o atomu jednakog ili većeg ranga u bilo kojem instanciranom pravilu. Savršeni model je model, kao što citiraju Van Gelder, Ross i Schliph (1991), sa minimiziranim brojem pozitivnih literala nižeg ranga naspram broja pozitivnih literala višeg ranga.

[§] FLORA je akronim na engleskom jeziku za **F**-**LO**gic t**RA**nslator, odnosno prevoditelj za F-logiku. Pri tome se misli na prevođenje sintakse bliske sintaksi F-logike na sintaksu jezika Prolog koji je u pozadini Flore, i koji ima ulogu sustava za logičko zaključivanje.

F-logika a jednako tako i Flora-2 kao njezin dijalekt, nalaze primjenu ne samo u inteligentnim informacijskim sustavima već i kao sredstvo za prikaz ontologija, čak i u okviru semantičkog weba koji ne zahtijeva veliku izražajnost.

2.3. Deskriptivne logike

Deskriptivne logike (DL) su odlučivi fragmenti logike prvog reda" (Horridge i dr., 2012). Odlučivost je važna u semantičkom webu bez obzira na to što su deskriptivne logike manje izražajne (ekspresivne) od logike 1. reda.

"Deskriptivne logike su zbirno ime formalizama za prikaz znanja, koji su usredotočeni na upravljanje osnovnim deskriptivnim vokabularom...odnosno konstrukte koji vjerodostojno odražavaju svoje intencijske ontologije i procese." (Balaban, 1995, str. 1)

I u deskriptivnim logikama postoje koncepti koji se podudaraju sa unarnim predikatima u logici prvog reda, uloge koje se podudaraju sa binarnim predikatima i individue koje se podudaraju sa konstantama u logici prvog reda.

Zadatak koji se postavlja pred deskriptivne logike je opis koncepata i uloga iz određene domene, njihova organizacija u hijerarhije odnosno taksonomije te njihova upotreba pri analitičkom opisu domene. Tako strukturirano znanje omogućuje postavljanje upita i dobivanje odgovora na njih. (Balaban, 1995)

2.3.1. Osnovni jezik $\mathcal{ALC}$

Temeljni jezik deskriptivnih logika je atributni jezik $\mathcal{AL}$, koji ne uključuje negaciju, uniju i potpuno egzistencijalno ograničenje. Sintaksa i semantika jezika $\mathcal{AL}$ dana je u tablici 1.

Tablica 1: Sintaksa i semantika jezika $\mathcal{AL}$

Konstruktor	**Sintaksa**	**Semantika**
Univerzalni koncept	$\top$	$\Delta^{\mathcal{I}}$
Dno koncept	$\bot$	$\emptyset$
Atomarna negacija	$(\neg A)$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
Presjek	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Restrikcija vrijednosti	$(\forall R.C)$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
Ograničena egzistencijalna kvantifikacija	$(\exists R.\top)$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\}$

(Prema izvoru: Ławrynowicz (2017))

Stoga je najmanji jezik koji omogućava zaključivanje $\mathcal{ALC}$ koji uključuje negaciju preko koje se mogu izraziti i unija i potpuno egzistencijalno ograničenje. Sintaksa jezika $\mathcal{ALC}$ definira se u dva koraka: (Lutz, 2016)

1. induktivno se definira skup logičkih formula koje označavaju koncepte
2. koncepti se stavljaju u međusobne relacije što se u logičkoj teoriji naziva *TBox*

Svaki koncept u deskriptivnim logikama može se prikazati i preko određene formule logike 1. reda (račun predikata) pri čemu je bar jedna varijabla slobodna. Tako za koncepte sa simbolima A, C i D vrijedi slijedeći odnos konstruktora koncepata i formula logike 1. reda: (Lutz, 2016)

$$\begin{aligned} A &\equiv A(x) \\ \neg C &\equiv \neg C(x) \\ C \sqcap D &\equiv C(x) \wedge D(x) \\ C \sqcup D &\equiv C(x) \vee D(x) \\ \exists r.C &\equiv \exists y(r(x,y) \wedge C(y)) \\ \forall r.C &\equiv \forall y(r(x,y) \rightarrow C(y)) \end{aligned}$$

Ovdje je r oznaka uloge a izraz $r(x,y)$ označava da x ispunjava ulogu za y. Tada je y r-sljedbenik od x. Uloge u deskriptivnoj logici odgovaraju binarnim predikatima u logici 1. reda. Koncepte se povezuje sa klasama (npr. u jeziku OWL), a uloge sa svojstvima. U deskriptivnoj logici, aksiomi su izjave koje povezuju uloge i koncepte. Konstante u logici 1. reda odgovaraju individuama.

Općenito se DL baza znanja definira kao trojka $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R} \rangle$ gdje su *T*, *A* i *R* TBox, ABox i RBox respektivno.(Sikos, 2019) "Baza znanja izražena u deskriptivnim logikama tradicionalno se sastoji od dviju komponenti: *TBox* i *ABox*." (De Giacomo i Lenzerini, 1996, str. 2). Općenito se baza znanja sastoji od općenitog, intenzijskog znanja vezanog uz problemsku domenu i ekstenzijskog znanja vezanog uz konkretan problem.

TBox sadrži intenzijsko znanje u obliku terminologija i sagrađeno je kroz deklaracije koje opisuju općenite osobine koncepata odnosno klasa. Takvo znanje često je organizirano u taksonomije pa je i to jedan od razloga zašto se ovo znanje označava slovom T.(Baader, McGuinness i Patel-Schneider, 2007, str. 17) TBox je kolekcija konceptualnih aksioma sadržavanja oblika $C \sqsubseteq D$ te aksioma konceptualnih ekvivalencija oblika. $C \equiv D$.(Sikos, 2019)

ABox sadrži ekstenzijsko znanje, asertivno ili konkretno znanje (eng. *assertions*), i to znanje se odnosi na individue neke domene. Takvo znanje se mijenja, za razliku od od intenzijskog znanja. (Baader, McGuinness i Patel-Schneider, 2007, str. 17).
ABox je konačna kolekcija aksioma oblika $C(a), R(a,b)$ gdje su a i b imena individua, C je neki koncept ili klasa, dok je R uloga, relacija ili svojstvo.(Sikos, 2019) Individualna konkretizacija može biti konkretizacija koncepta (objekt izveden iz klase C odnosno $C(a)$) konkretizacija uloge (konkretizacija relacije odnosno svojstva $R(a,b)$ i negativna konkretizacija uloge $\neg R(a,b)$) te jednakost individua($a = b$) i nejednakost individua ($a \neq b$).(Sikos, 2019) Konkretno znanje čine tvrdnje o pripadnosti individue nekom konceptu ili svojstvu u obliku osnovnih rečenica.(Baader, McGuinness i Patel-Schneider, 2007)

RBox odnosno terminološko znanje koje opisuje odnose uloga nije dozvoljen u $\mathcal{ALC}$

logici. Obično se ne navodi u literaturi, npr. Baader, McGuinness i Patel-Schneider (2007), za razliku od T-Boxa i A-Boxa, budući da pripada najekspresivnijim deskriptivnim logikama. (Sikos, 2019) RBox je konačna kolekcija generaliziranih aksioma koje opisuju inkluzije uloga odnosno hijerarhiju uloga oblika $R \sqsubseteq S$, njihovu ekvivalenciju ($R \equiv S$) kompoziciju uloga odnosno kompleksnu inkluziju oblika $R_1 \circ R_2 \sqsubseteq S$), razdvajanje uloga $Dis(R, S)$ te tranzitivnost aksioma oblika $R^+ \sqsubseteq R$ gdje je R^+ skup tranzitivnih uloga odnosno svojstava.(Sikos, 2019) Među aksiome RBoxa pripadaju i inverzne uloge $R \equiv S^-$ te druge osobine svojstava poput funkcijskih, simetričnih, refleksivnoh, antirefleksivnih i tako dalje.

2.3.2. Proširenja jezika $\mathcal{ALC}$

Osnovni jezik $\mathcal{ALC}$ može biti proširen novim izražajnim mogućnostima:

- $\mathcal{S}$ - $\mathcal{ALC}$ sa tranzitivnim ulogama ($\mathcal{ALC}_{\mathcal{R}^+}$), oznaka uvedena radi kraćeg pisanja
- $\mathcal{F}$ - funkcionalna svojstva
- $\mathcal{H}$ - hijerarhija uloga (podsvojstva i nadsvojstva)
- $\mathcal{R}$ - ograničeni aksiomi uključenja uloga; refleksivnost, ileksivnost i razdvojenost uloga
- $\mathcal{O}$ - nominali (pobrojane klase ograničenja vrijednosti objekata)
- $\mathcal{E}$ - puna egzistencijalna kvalifikacija
- $\mathcal{U}$ - unija koncepata
- $\mathcal{C}$ - složena negacija koncepata
- $\mathcal{I}$ - inverzna svojstva
- $\mathcal{N}$ - ograničenja kardinalnosti
- $\mathcal{Q}$ - okvalificirana ograničenja kardinalnosti
- $^{(\mathcal{D})}$- korištenje tipova podataka, podatkovnih vrijednosti i tipova

Primjerice, FaCT sustav za zaključivanje koristi $\mathcal{SHIQ}$ deskriptivnu logiku, Pellet koristi $\mathcal{SHIN}^{(\mathcal{D})}$, dok Protégé koristi $\mathcal{SHOIN}$ Sustavi za automatsko zaključivanje[¶] u semantičkom webu, obično su vezani uz deskriptivne logike, a jednako tako i jezici poput OWL-a (W3C-ovog standardiziranog jezika za web ontologije, eng. *Web Ontology Language*) i RDF-a (standardnog modela za opis i razmjenu web resursa, eng. *Resource Description Framework*).
Pored atributivnih jezika $\mathcal{AL}$ treba spomenuti i $\mathcal{FL}$, odnosno jezike bazirane na okvirima te $\mathcal{EL}$, egzistencijalne jezike. Svojstva ovih jezika kratko navodim u nastavku: (Ławrynowicz, 2017)

- atributni jezici $(\mathcal{AL})$ dozvoljavaju:
  - atomarnu negaciju

[¶]Zaključivanje je izvođenje činjenica koje nisu eksplicitno upisane u bazu znanja niti u ontologiji.

  - presjek klasa (koncepata)
  - univerzalna ograničenja
  - ograničene egzistencijalne kvantifikatore
- jezici okvira $(\mathcal{FL})$ dozvoljavaju:
  - presjek klasa (koncepata)
  - univerzalna ograničenja
  - ograničene egzistencijalne kvantifikatore
  - ograničenje uloga
- egzistencijalni jezici $(\mathcal{EL})$ dozvoljavaju:
  - presjek klasa (koncepata)
  - egzistencijalna ograničenja
- podjezici $(\mathcal{FL})$ i $(\mathcal{EL})$:
  - podjezici okvira bez ograničenja uloga $(\mathcal{FL}^{-})$
  - podjezici okvira bez ograničenja uloga i bez ograničenih egzistencijalnih kvantifikatora $(\mathcal{FL}_0)$
  - podjezici $(\mathcal{EL}^{++})$ - isto što i $\mathcal{ELRO}$

Deskriptivne logike karakterizira prilagodljivost pri zaključivanju, pri čemu sam sustav zaključivanja ima slijedeće zadatke Obitko (2007):

- zadovoljivost koncepta odnosno nekontradiktornost: može li postojati individua koja pripada tom konceptu
- supsumpcija koncepta: da li neki koncept podrazumijeva drugi koncept (je li opis prvog općenitiji od opisa drugog koncepta)
- konzistentnost *ABoxa* u odnosu na *TBox*: ne ruše li individue u ABoxu opis i aksiome TBoxa
- provjera individue: je li neka individua instanca koncepta (odnosno klase)
- dohvaćanje svih individua nekog koncepta
- realizacija individua: pronalaženje svih koncepata kojima neka individua pripada

Semantički web, RDFS i mali jezici poput OWL-a nisu jedini pristup dizajnu i manipulaciji bazama znanja, te radu s ontologijama. Općenito je ontologije moguće prikazati pomoću 4 vrste formalizama (Galba, 2016, str. 41):

- Okviri i logika prvog reda
- Deskriptivnih logika
- Tehnologija softverskog inženjerstva
- Tehnologija baza podataka

2.4. Semantički web i OWL

Uz formalnu semantiku RDF-a, OWL kao ontološki jezik je drugi bitan čimbenik za uspješno funkcioniranje semantičkog weba. Semantički web i jezici poput OWL zasnovani su na deskriptivnim logikama (DL) različitih razina izražajnosti. OWL je jezik koji omogućuje provjeru logičke konzistentnosti RDF tripleta tako što povezuje semantički web sa ontologijama. Budući da je nastao kasnije od RDF-a i RDFS-a, pri njegovom se stvaranju nastojala zadržati kompatibilnost sa već postojećim konstruktima prisutnima u RDFS shemama (klase, svojstva i mogućnost nasljeđivanja) koliko je god to moguće. RDF i RDFS tripleti su veće izražajnosti od bilo kojeg potencijalnog jezika sposobnog za podršku automatskom zaključivanju, kada se nastoje zadovoljiti kriteriji valjanosti, potpunosti i skalabilnosti.

Ontologije možemo smatrati formalizmom koji određuje kakve su tvrdnje prikazane RDF tripletima konzistentne a kakve nisu. Sam pojam formalizma odnosi se na mogućnost računala da čini takve provjere automatski. Budući da su brojni alati zasnovani upravo na OWL jeziku, a tu svakako treba spomenuti Protégé, nužno je opisati i tehnologiju semantičkog weba. To ću učiniti vrlo kratko, u mjeri koja je potrebna za lakše razumijevanje formalizama pri formuliranju uzoraka dizajna ontologija.

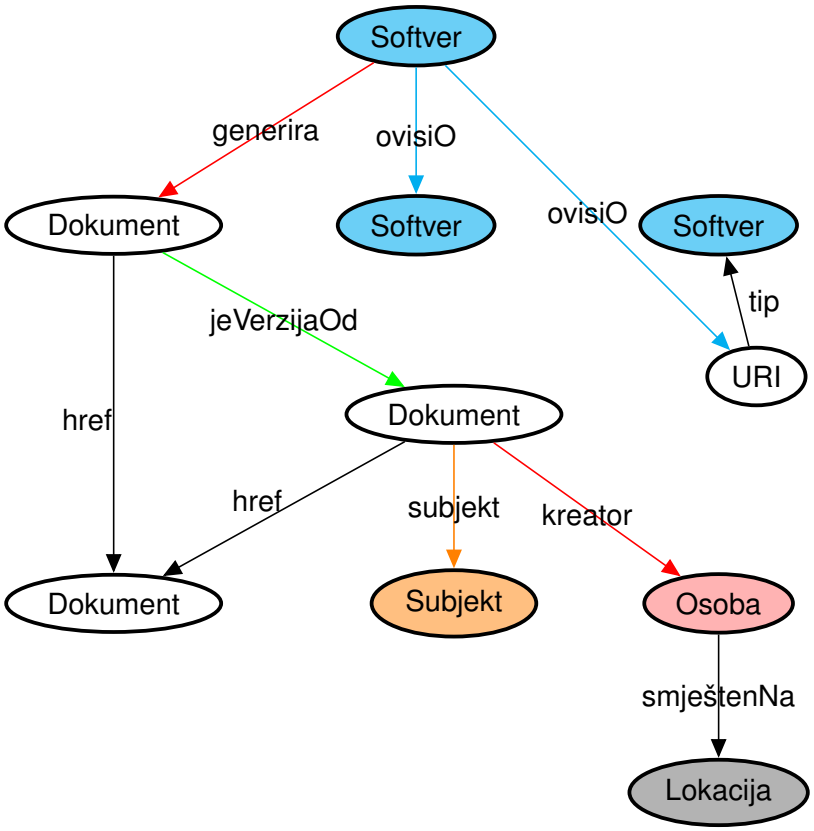
2.4.1. Principi semantičkog weba

Na seminaru o semantičkom webu 2001. godine (Koivunen i Miller, 2001), utvrđeni su slijedeći principi:

1. Sve može biti identificirano URI-jima: ljudi, mjesta, stvari... Svatko tko kontrolira dio imenskog prostora semantičkog weba može kreirati URI i pridružiti ga nečemu iz realnog svijeta bilo izravno, bilo posredno.
2. Resursi i poveznice mogu imati tipove koji pružaju dodatne informacije strojevima koji koriste koncepte povezane sa tim resursima i poveznicama (slika 3)
3. Tolerira se nepotpuna (parcijalna) informacija
4. Nema potrebe za apsolutnom istinom. Aplikacija će sama procijeniti istinitost određene informacije prema kontekstu i drugim dostupnim informacijama.
5. Podržana je evolucija. Mogu se dodavati nove informacije bez revidiranja starih, mogu se koristiti različiti vokabulari te efektivne kombinacije neovisnog djelovanja različitih grupacija koje stvaraju sadržaj. Predviđa se da semantički web ima alate koji se mogu koristiti za razjašnjavanje nedoumica i kontradikcija.
6. Koristi se minimalistički dizajn. Ovo znači da W3C standardizacije neće ići dalje od neophodnog.

2.4.2. XML, RDF i RDFS kao temelji semantičkog weba

Semantički web se razvijao paralelno, ali u sjeni world wide weba (često samo web, u daljem tekstu www), sa ciljem omogućavanja strojnog procesiranja i razumijevanja sadržaja



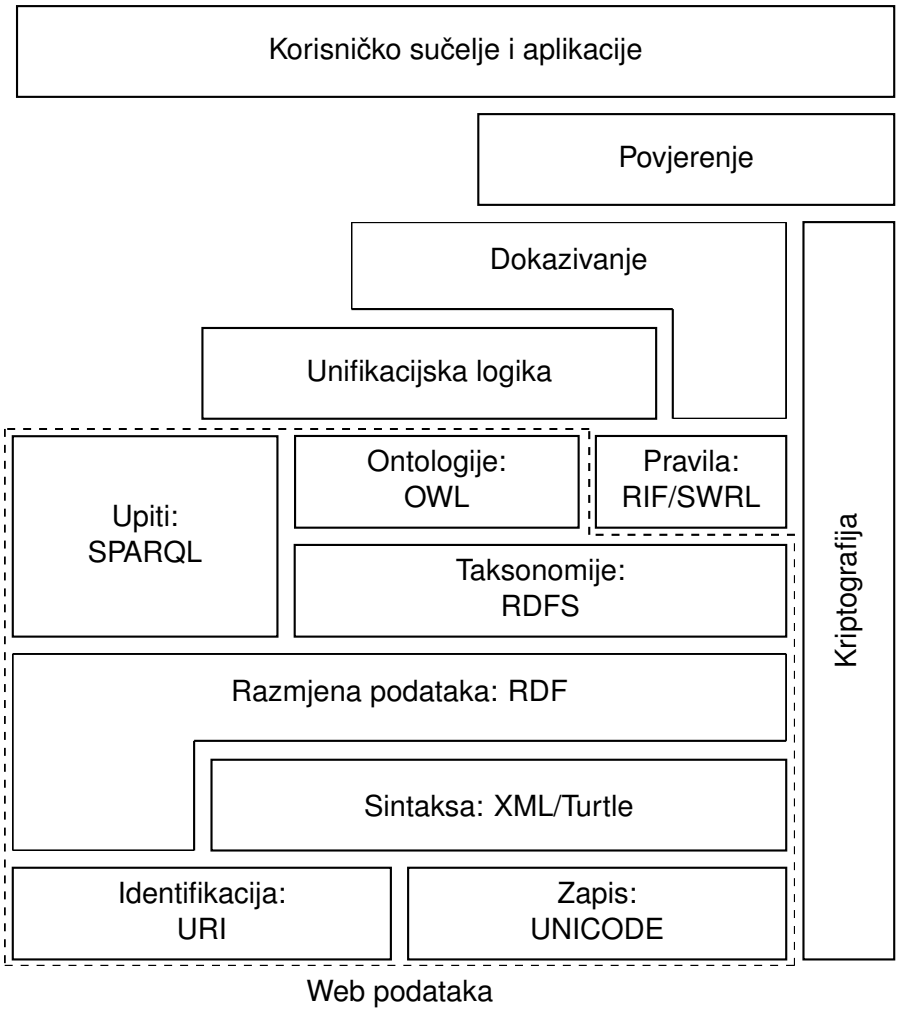
Slika 3: Tipovi resursa i poveznica prema konceptu semantičkog weba
Izrađeno prema izvoru Koivunen i Miller (2001)

na internetu bez intervencije čovjeka. I dok je www sustav raspodijeljenih hipertekstualnih dokumenata, semantički web je sustav raspodijeljenog znanja.

Semantički web razmjenjuje podatke a ne dokumente, što znači da svaki podatak ima svoj URI[‖] odnosno IRI[**] identifikator. Vizija semantičkog weba temeljila se na ponovnoj iskoristivosti podataka, njihovoj automatizaciji i integraciji uz upotrebu agenata koji bi podatke prikupljali iz različitih izvora, međusobno ih razmjenjivali i procesirali. Uz upotrebu jezika kao što su RDF/RDFS i OWL te upitnih jezika poput SPARQL, prikupljeno znanje bi se opisivalo i na osnovu njega stvarale ontologije.

Svrha ontologija je opis koncepata i njihovo povezivanje sa resursima na webu, što omogućava postavljanje upita vezanih uz te podatke kao i automatsko zaključivanje. Na slici 4 prikazani su slojevi semantičkog weba. Tu su prikazane i neke od spominjanih tehnologija poput RDF tripleta, OWL-a, upitnih jezika i sustava za automatsko zaključivanje. Temelj svih tehnologija semantičkog weba je XML, koji se u tom kontekstu koristi kao format za prijenos podataka. U višem sloju je okvir (eng. *framework*) namijenjen predstavljanju metapodataka odnosno opisu resursa, kojeg W3C konzorcij razvija pod nazivom RDF (eng. *Resource Description Framework*). U ovom podatkovnom modelu, podaci se predstavljaju u vidu trojki koje sačinjavaju subjekt, predikat i objekt. U smislu logike, RDF trojke su izjave, logičke rečenice, gdje je subjekt glavni resurs, predikat je svojstvo subjekta koje čini relaciju među resursima, a

[‖] URI je akronim za usklađeni identifikator resursa (eng. *Uniform Resource Identifier*)
[**] IRI je akronim za internacionalizirani identifikator resursa (eng. *Internationalized Resource Identifier*)

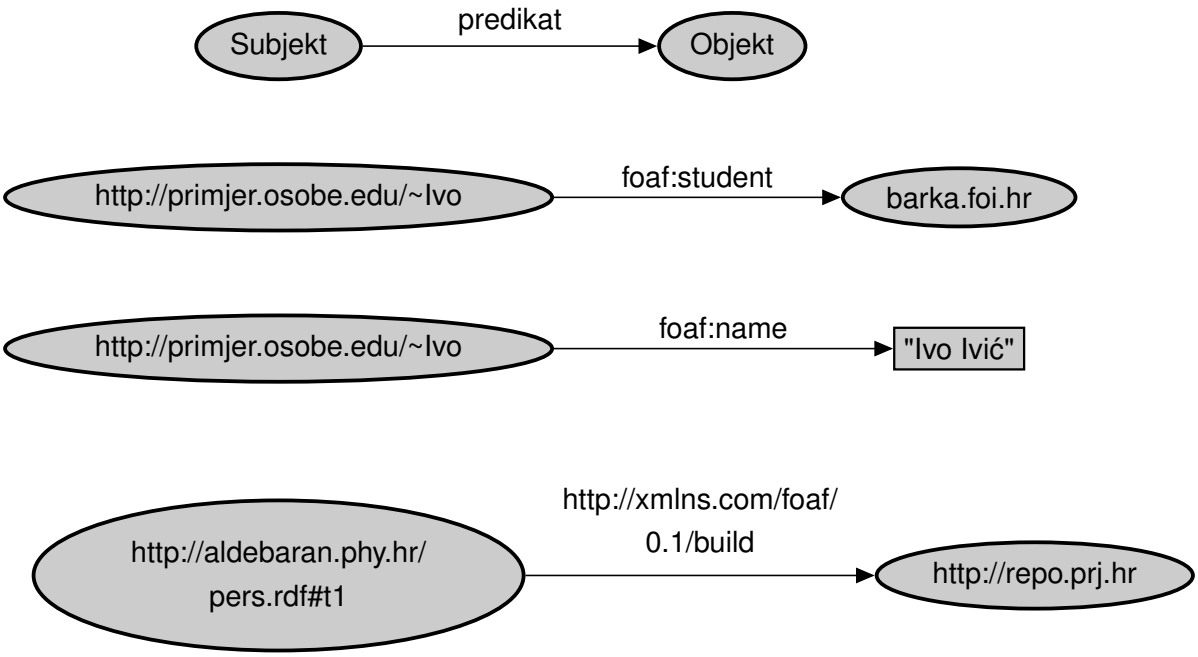


Slika 4: Slojevi semantičkog weba, prema izvoru: Hoyland i dr. (2014)

objekt vrijednost ili općenito neki resurs dodijeljen subjektu.

RDF je dakle samo model, pa se XML koristi kao jezik za njegovo predstavljanje prema preporukama W3C. Model je moguće predstaviti i na drugi način, npr. pomoću nešto elegantnijih N-triples ili TURTLE[††] jezika. (Galba, 2016, str. 16)

[††] Jezik za sažeto predstavljanje RDF trojki (eng. *Terse RDF Triple Language*)



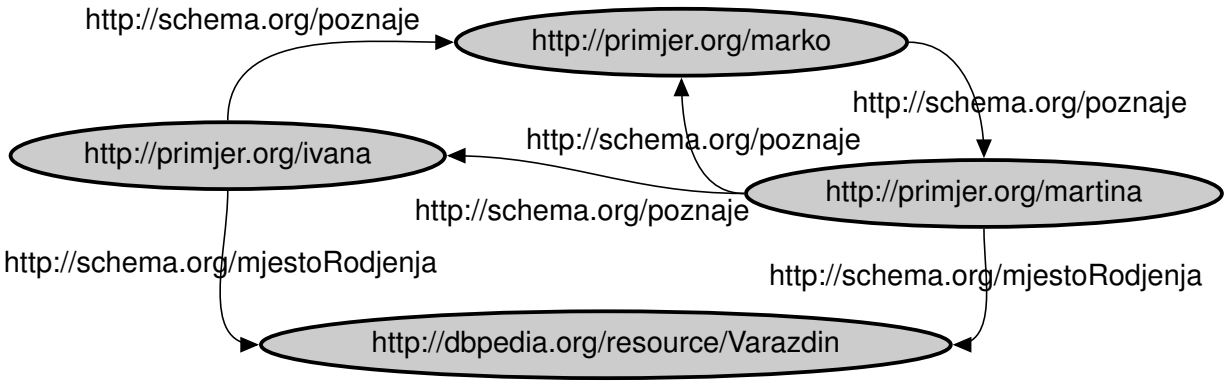
Slika 5: primjeri RDF trojki

Primjer TURTLE sintakse dat je u slijedećem primjeru:

```
<http://aldebaran.phy.hr/pers.rdf\#t1>
<http://xmlns.com/foaf/0.1/build> <http://repo.prj.hr>.
```

U ovom primjeru, osoba *t1* koja se vodi kao resurs na nekom poslužitelju sa adresom `http://aldebaran.phy.hr` ima svojstvo da je izgradila repozitorij projekata na `http://repo.prj.hr`. Pri tome je to svojstvo mogućnosti građenja preuzeto iz FOAF imenskog prostora koji se inače koristi za opis osoba na adresi `http://xmlns.com`.

Trojke mogu činiti složenije usmjerene aciklične grafove kao to je prikazano na slici 6. Ekvivalentni prikazi u N-Triple i Turtle sintaksi dani su u listinzima 2.1 (N-triple sintaksa), odnosno 2.2 (Turtle sintaksa).



Slika 6: primjer RDF grafa

Graf na slici 6 može se opisati i N-triple formalizmom kao u listingu 2.1.

Listing 2.1: N-triple prikaz usmjerenog acikličnog RDF grafa

```
<http://primjer.org/ivana> <http://schema.org/poznaje> <http://primjer.org/marko> .
<http://primjer.org/marko> <http://schema.org/poznaje> <http://primjer.org/martina> .
<http://primjer.org/martina> <http://schema.org/poznaje> <http://primjer.org/ivana> .
<http://primjer.org/martina> <http://schema.org/poznaje> <http://primjer.org/marko> .
<http://primjer.org/ivana> <http://schema.org/mjestoRodjenja> <http://dbpedia.org/resource/Varazdin> .
<http://primjer.org/martina> <http://schema.org/mjestoRodjenja> <http://dbpedia.org/resource/Varazdin> .
```

Listing 2.2: Turtle prikaz usmjerenog acikličnog RDF grafa

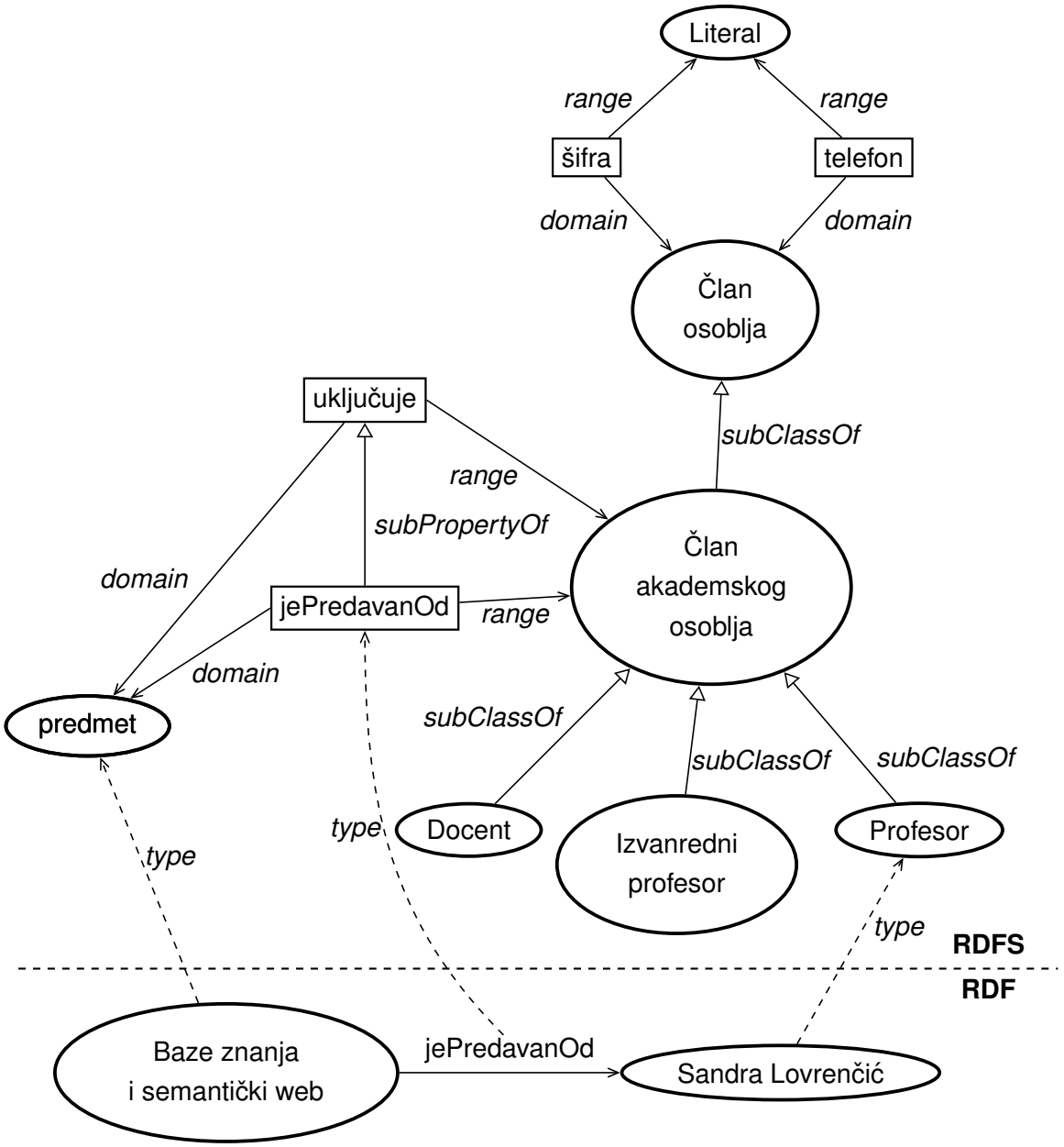
```
prefix: <http://primjer.org/>
prefix schema: <http://schema.org/>
prefix dbo: <http://dbpedia.org/ontology/>
prefix dbr: <http://dbpedia.org/resource/>
:ivana schema:mjestoRodjenja dbr:Varazdin ;
schema:poznaje :marko .
:marko schema:poznaje :martina .
:martina schema:mjestoRodjenja dbr:Varazdin ;
schema:poznaje :ivana ,
:marko .
```

Napredniji način izražavanja izjava o resursima je RDFS[‡‡] koja donosi klase i svojstva, njihove hijerarhije i mogućnost nasljeđivanja, što omogućuje bolje definiranje i klasifikaciju ontologije. RDFS se službeno naziva i RDF jezik za opis vokabulara. To je i dalje RDF dokument ali oblikovan u RDF/XML formatu koji koristi neke predefinirane klase poput `rdfs:Resource`, `rdf:Property`, `rdfs:Class`, `rdfs:Literal`, `rdf:Statement`, `rdfs:Container` i druge. Klase u RDFS označavaju koncepte kojima resursi pripadaju. Podklase su podrazumijevane od strane nadklasa, i svaki resurs koji pripada podklasi, automatski pripada i nadklasi.
RDFS se razlikuje od XML sheme (XSD) po slijedećim svojstvima:

- pretpostavka otvorenog svijeta (eng. *Open World Assumption - OWA*) naspram pretpostavke zatvorenog svijeta (eng. *Closed World Assumption - CWA*)
- RDFS se bavi opisivanjem resursa a ne validacijom zapisa podataka

Slika 7 prikazuje RDFS graf jednog modela domene koji se odnosi na predavače i predmete koje predaju.

[‡‡] RDFS je akronim koji se nadovezuje na RDF i označava RDF shemu



Slika 7: primjer RDFS grafa sa RDF dijelom prema Wollowski (2007)

XML/RDFS sintaksa koja kodira ontologiju prikazanu na slici /refslika:RDFS dana je u listingu 2.3. Možemo vidjeti kako se nasljeđivanje navodi unutar bloka određene klase.

Listing 2.3: Fragmenti RDFS kôd ontologije kao konceptualnog modela domene prema Wollowski (2007)

```
<rdf:RDF\\
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:ID="predavac">
<rdfs:subClassOf rdf:resource="#clanAkademskogOsoblja"/>
</rdfs:Class>
<rdfs:Class rdf:ID="clanAkademskogOsoblja">
<rdfs:subClassOf rdf:resource="#clanOsoblja"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="clanOsoblja">
</rdfs:Class>
<rdfs:Class rdf:ID="predmet">
</rdfs:Class>
<rdf:Property rdf:ID="ukljucuje">
<rdfs:domain rdf:resource="#predmet"/>
<rdfs:range rdf:resource="#predavac"/>
</rdf:Property>
<rdf:Property rdf:ID="jePredavanOd">
<rdfs:subPropertyOf rdf:resource="#ukljucuje"/>
</rdf:Property>
<rdf:Property rdf:ID="telefon">
<rdfs:domain rdf:resource="#clanOsoblja"/>
<rdfs:range rdf:resource="&rdf;Literal"/>
</rdf:Property>
</rdf:RDF>
---------------
<rdf:Description rdf:about="CIT1111">
<rdf:type rdf:resource="&uni;predmet"/>
<uni:imePredmeta>Baze znanja i semanticki web</uni:imePredmeta>
<uni:jePredavanOd rdf:resource="949318"/>
</rdf:Description>
<rdf:Description rdf:about="949318">
<rdf:type rdf:resource="&uni;predavac"/>
<uni:ime>Sandra Lovrencic</uni:ime>
<uni:titula>Profesor</uni:titula>
</rdf:Description>
```

U RDFS, a i u OWL profilima, posebno značenje ima pojam *domene*. Domena je klasa instanci koje smiju biti subjekti u izjavama koje uključuju određeni predikat. Sa druge strane je pojam *ranga*, koji definira klasu instanci koje smiju biti objekti u izjavama koje uključuju određeni predikat.

2.4.3. OWL - jezici ontologija

OWL je standard za prikaz i dijeljenje znanja, baziran na RDF-u i DAML (eng. *DARPA*[§§] *Agent Markup Language*). OWL i drugi ontološki jezici su prirodna nadogradnja na RDFS formalizam kako bi se omogućilo pisanje ontologija i formalno zaključivanje o samim resursima. Formalno zaključivanje je neophodno budući da je razvoj kvalitetnih ontologija vrlo težak posao. Upotrebom zaključivača provjeravamo konzistentnost ontologije i njezinu usklađenost sa intuitivnom predodžbom realnog svijeta a zaključivači nam pomažu da pronađemo ekvivalentne klase i kroz automatsku klasifikaciju bolje organiziramo taksonomije. Pored toga OWL omogućava uvoz odnosno referenciranje vanjskih ontologija (konstrukt *owl:imports*). OWL jezik dio je W3C standardiziranih tehnologija semantičkog weba, koje uključuju RDF, RDFS i SPARQL upite.

Mnoštvo alata i web infrastrukture je kreirano kao podrška OWL ontologijama i seman-

[§§] DARPA je akronim koji označava Agenciju za obrambene napredne istraživačke projekte (eng. *Defense Advanced Research Projects Agency*) Ministarstva obrane Sjedinjenih američkih država.

tičkom webu, kao na primjer open source Protégé (Sveučilište Stanford) i Swoop (Sveučilište u Marylandu), Neon Toolkit (NeOn Foundation), te komercijalni TopBraid Composer (TopQuadrant Inc.). Uz to su razvijeni zaključivači poput Pelleta, FaCT++, Racera, HermiTa, Quonto-a i KAON2 infrastrukture.

OWL jezici su ekspresivniji od RDFS jer omogućavaju izražavanje ekvivalencije identiteta kao i razlikovanje identiteta (sameAs, differentFrom, equivalentClass, equivalentProperty). Omogućavaju lokalizirane rangove i domene svojstava ovisne o klasi na koju se primjenjuju, egzistencijalna ograničenja i kardinalnost. (Wollowski, 2007) Sa stanovišta klasa, omogućavaju definiciju novih klasa kao presjek, uniju, komplement i razdvojenost postojećih klasa te kardinalna ograničenja. Sa stanovišta svojstava, omogućavaju definiciju objektnih i podatkovnih svojstava, pri čemu svojstva mogu biti tranzitivna, funkcionalna, simetrična, i inverzna. Omogućavaju i vrijednosna ograničenja.(W3C, 2009)

OWL jezici se sastoje od 2 osnovna jezika odnosno grupa profila. Viši profili uključuju sintaksu nižih profila (na primjer, OWL Full uključuje cjelokupan OWL DL vokabular ali sa manje ograničenja. OWL profili su navedeni u nastavku: (Karlsen, 2015)

- OWL-1 profili bazirani na DL $\mathcal{SHOIN}$
  - **OWL Full** - $\mathcal{SHOIN}$, najveća ekspresivnost. Neodlučivost[¶¶]. Klase mogu biti i individue i svojstva.
  - **OWL DL** - $\mathcal{SHOIN}$, visoka ekspresivnost. Odlučivost i potpunost. Omogućava izražavanje razdvojenosti, unija i komplementa klasa.
  - **OWL Lite** - $\mathcal{SHIF}$, niska ekspresivnost, ali i niska računalna kompleksnost. Uključuje sve RDFS osobine. Omogućava izražavanje ekvivalencije klasa i svojstava, te sameAs i differentFrom tvrdnje vezane uz identitet individua. Omogućava izražavanje inverznih, simetričnih, tranzitivnih i funkcionalnih svojstava, ograničenja nad svojstvima i kardinalnost (0 ili 1). Omogućava izražavanje presjeka klasa.
- OWL-2 profili bazirani na DL $\mathcal{SROIQ}$
  - **OWL 2 EL** - $\mathcal{EL}$, dobar za velike ontologije ali male količine podataka povezanih sa instancama (T-Box zaključivanje). Optimalna složenost računanja. Provjera zadovoljivosti u polinomnom vremenu.
  - **OWL 2 QL** Query language - limitirana izražajnost. Vrlo učinkovit vraćanje rezultata upita i manipulaciju velikim količinama podataka povezanih sa instancama. Skladištenje podataka i evaluacija upita može se delegirati sustavima za upravljanje relacijskim bazama podataka.
  - **OWL 2 RL** Rule language - $\mathcal{DLP}$, kompromis između skalabilnosti pri zaključivanju i izražajne snage (A-Box zaključivanje).

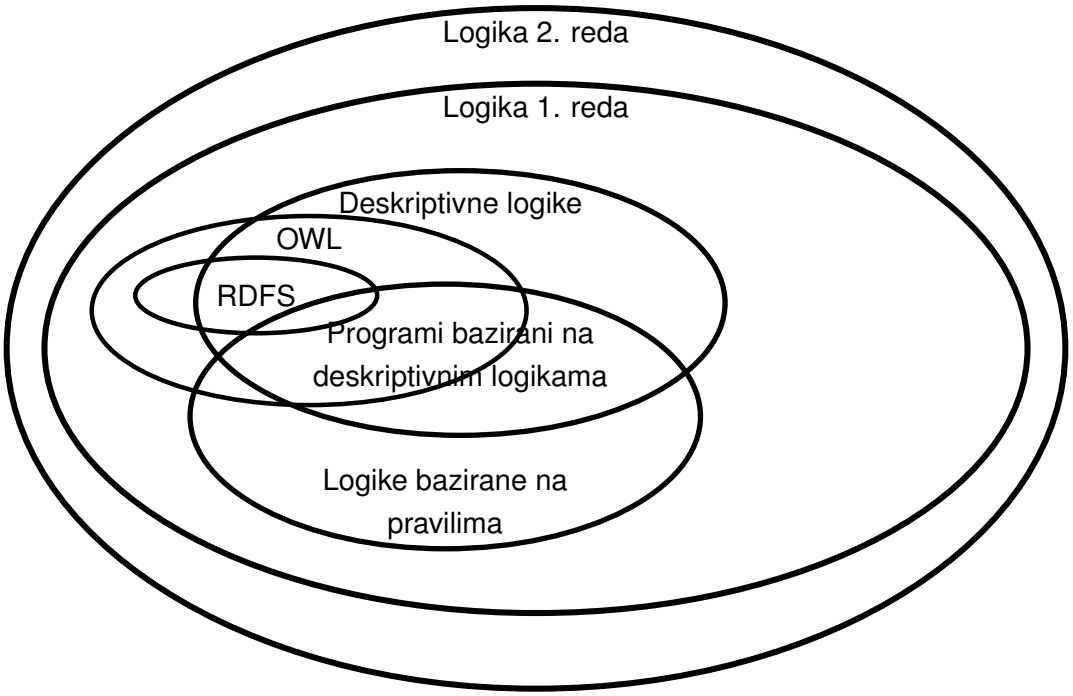
Bitan element OWL jezika je prilagođenost semantičkom webu. Kritični elementi pri tome su složenost i skalabilnost. Složenosti OWL jezika dani su u nastavku: (Zolin, 2013)

[¶¶]Ne mora dati rezultat u konačno mnogo koraka.

- OWL - 1 jezici, DL $\mathcal{SHOIN}$ - $NExp$
  - OWL Full - $NExp$
  - OWL DL - $NExp$
  - OWL Lite - Exp
- OWL - 2 jezici, DL $\mathcal{SROIQ}$ - $2NExp$
  - OWL 2 EL - P (polinomna složenost)
  - OWL 2 QL - AC^0 klasa složenosti*** s obzirom na količinu podataka (odgovara brzini odgovora na upite SUBP)
  - OWL 2 RL - P (polinomna složenost)

Smisao ovih proširenja je podizanje izražajnosti i sposobnosti zaključivanja semantičkog weba na razini deskriptivnih logika. Tako su OWL jezici (osim OWL Lite) bazirani na $\mathcal{SHOIN}^{(\mathcal{D})}$ (jednostavne hijerarhije uloga) a OWL-2 na $\mathcal{SROIQ}^{(\mathcal{D})}$ deskriptivnim logikama ($\mathcal{ALC}$ proširen tranzitivnim ulogama, odnosno $\mathcal{S}$, inverznim ulogama, kvalificiranim kardinalnim ograničenjima, R-Box i nominalima).(Horrocks, 2010)
Treba napomenuti ipak da deskriptivne logike imaju manju izražajnu snagu od RDF-a. Usporedba konstrukata OWL i DL dana je u tablici 2. Izražajnost OWL jezika i RDF sheme (RDFS) preklapa se sa izražajnošću deskriptivnih logika, ali je ipak znatno manja. Na slici 8 prikazan je Vennov dijagram izražajnosti jezika semantičkog weba unutar logika 1. i 2. reda. Smanjena izražajnost ontoloških jezika kompromis je zahtjevima za skalabilnost i odlučivost semantičkog weba.

***AC klase su definirane preko elektronskih logičkih sklopova u Booleovoj logici. Svaka AC^i klasa se sastoji od jezika koje mogu prepoznati Booleovi logički krugovi dubine $\mathcal{O}(log^i n)$ i polinomno brojnih OR i AND vrata sa neograničenim brojem ulaza. AC^0 klasa složenosti zbog stablaste strukture logičkih sklopova, približno odgovara *logaritamskoj* složenosti.



Slika 8: Izražajnost jezika semantičkog weba prema Gulay (2018)

Tablica 2: Korespondencija sintakse OWL jezika i deskriptivnih logika

Klasni konstruktori		
OWL konstruktor	**DL sintaksa**	**Primjer**
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Čovjek ⊓ Muško
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Liječnik ⊔ Odvjetnik
complementOf	$\neg C$	¬ Muško
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{ivan} ⊔ {marija}
allValuesFrom	$\forall P.C$	∀ imaDijete.Liječnik
someValuesFrom	$\exists P.C$	∃ imaDijete.Odvjetnik
maxCardinality	$\leq nP$	≤ 1 imaDijete
minCardinality	$\geq nP$	≥ 2 imaDijete
Ontološki aksiomi		
OWL sintaksa	**DL sintaksa**	**Primjer**
subclassOf	$C_1 \sqsubseteq C_2$	Čovjek ⊑ Životinja ⊓ Dvonožac
equivalentClass	$C_1 \equiv C_2$	Muškarac ≡ Čovjek ⊓ Muško
subPropertyOf	$P_1 \sqsubseteq P_2$	imaKčer ⊑ imaDijete
equivalentProperty	$P_1 \equiv P_2$	imaCijenu ≡ imaNovčanuVrijednost
transitiveProperty	$P^+ \sqsubseteq P$	jePredak$^+$ ⊑ jePredak
type	$a : C$	ivan:SretanOtac
property	$\langle a, b \rangle : R$	⟨Ivan,Marija⟩:imaDijete

(Prema izvoru: Horrocks (2010))

Klase, svojstva i individue definirane su formalizmom deskriptivnih logika, odnosno aksiomima, koristeći pretpostavku otvorenog svijeta (eng. *Open World Assumption*, OWA) što znači da nepostojanje činjenica u bazi znanja ne negira te činjenice. Drugim riječima, negacija se mora eksplicite navesti, inače se ne podrazumijeva.

OWL donosi predefinirane klase `owl:Thing`, ekvivalentnu klasu RDFS klasi `rdfs:Resource`, što je ishodišna klasa koja obuhvaća sve druge, i njoj pripadaju sve individue, i klasu `owl:Nothing`, praznu klasu koja ne može imati niti jednu individuu ukoliko je ontologija valjana (nezadovoljiva klasa). Klasa `owl:Nothing` je podklasa svake klase, dok je `owl:Thing` nadklasa svake klase.

2.4.4. Opis ontoloških jezika semantičkog weba (OWL)

OWL jezici zahtijevaju zaglavlja i prefikse na početku dokumenta koja uključuju i zaglavlja potrebna RDF Shemi i XML Shemi kao što je prikazano u listingu 2.4. Prva 4 reda u primjeru odnose se na imenski prostor konkretne ontologije i ontologija podrške konkretnoj ontologiji. U ovom primjeru dani su prefiksi "ontologija" i "podrska", no zapravo to mogu biti bilo kako nazvani imenski prostori potrebni za opis određene ontologije.

Listing 2.4: Zaglavlja OWL dokumenata prema W3C dokumentaciji

```
<rdf:RDF
xmlns ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/ontologija#"
xmlns:ont ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/ontologija#"
xml:base ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/ontologija#"
xmlns:podrska="http://www.w3.org/TR/2004/REC-owl-guide-20040210/podrska#"
xmlns:owl ="http://www.w3.org/2002/07/owl#"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"">
<owl:Ontology rdf:about="">
<rdfs:comment>Ontologija kao primjer</rdfs:comment>
<owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/podrska
    "/>
<rdfs:label>Ontologija</rdfs:label>
...
```

Zadnjih 4 reda u listingu 2.4 odnose se na zaglavlje određene ontologije, u kome mogu biti i uključenja drugih ontologija, oznake verzija i tako dalje. OWL prema (Hammar, 2017, str. 22) donosi slijedeće osobine:

- Ekvivalenciju klasa i svojstava - ako su dvije klase ili svojstva ekvivalentne, sve instance koje pripadaju jednoj, pripadaju i drugoj
- Razdvojene klase (eng. *disjoint With*) - ako su klase razdvojene, ne mogu postojati instance koje propadaju i jednoj i drugoj klasi
- Sličnost i različitost (eng. *same As* i *different From*) - individualna ekvivalentnost ili razdvojenost

- Inverzna, tranzitivna i funkcionalna svojstva - ovo nije moguće iskazati u RDFS
- Kardinalnost ograničenja svojstava - određuje se koliko je moguće iskoristiti određeni predikat za neki subjekt

Pored ovih značajki uvedene su i nove: ključevi, ulančavanje svojstava, ograničenje tipova podataka i tako dalje. U nastavku navodim važnije značajke sintakse OWL jezika koje su nadogradnja na XML, RDF i RDFS značajke *xsd:datatype*, *Class* (owl:Class je podklasa rdfs:Class), *Individual*, *rdfs:subClassOf*, *rdf:Property*, *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, *rdfs:label*, *rdfs:comment*, *rdfs:seeAlso* i *rdfs:isDefinedBy*:

- OWL Lite
  - Informacije zaglavlja
    - **Ontology** - oznaka informacija vezanih za konkretnu ontologiju
    - **imports** - adrese drugih ontologija kao resursa
  - Anotacije
    - **AnnotationProperty** - omogućava komentiranje ontologija, aksioma ili URI/IRI resursa
    - **OntologyProperty** - omogućava kontekst za definiranje korisničkih svojstava ontologije pored već postojećih *versionInfo*, *priorVersion*, *incompatibleWith* i *backwardCompatibleWith*
  - Presjek klasa
    - **intersectionOf** - opis nove klase i ograničenja kao presjek postojećih imenovanih klasa i ograničenja
  - Ograničenja svojstava
    - **Restriction** - kontekst ograničenja
    - **onProperty** - svojstvo koje se ograničava
    - **allValuesFrom** - ograničenje svojstva s obzirom na određenu klasu. Svojstvo s obrzirom na klasu ima lokalno ograničenje svog dosega.
    - **someValuesFrom** - ograničenje svojstva s obzirom na određenu klasu. Određena klasa može imati ograničenje nekog svojstva tako da je bar jedna vrijednost tog svojstva određenog tipa.
  - Karakteristike svojstava (objektnih i podatkovnih)
    - **ObjectProperty** - relacije između instanci dviju klasa
    - **DataProperty** - relacije između instanci klasa i RDF literala ili XML Shema tipova podataka
    - **inverseOf** - deklaracija inverznog svojstva. Ako je A u relaciji sa B preko svojstva P, onda je B u relaciji sa A preko inverznog svojstva P'. Npr. jeRoditeljOd i jeDijeteOd.

    - **TransitiveProperty** - deklaracija tranzitivnog svojstva. Ako je A u relaciji sa B preko tranzitivnog svojstva P, a B u relaciji sa C preko tranzitivnog svojstva P, onda je i A u relaciji sa C preko tog istog svojstva P, bez potrebe za eksplicitnim navođenjem.
    - **SymmetricProperty** - deklaracija simetričnog svojstva. Ako je A u relaciji sa B preko simetričnog svojstva P onda je i B u relaciji sa A preko tog svojstva.
    - **FunctionalProperty** - deklaracija funkcionalnog svojstva. Ima isto značenje kao i minimalna kardinalnost 0 i maksimalna kardinalnost 1.
    - **InverseFunctionalProperty** - deklaracija inverzno funkcionalnog svojstva. Inverzno svojstvo tog svojstva je tada funkcionalno svojstvo.
- Jednakost klasa, svojstava i individua
    - **equivalentClass** - izjava o ekvivalentnosti dviju klasa. Ekvivalentne klase imaju iste intance.
    - **equivalentProperty** - izjava o ekvivalentnim svojstvima. Ekvivalentna svojstva povezuju određenu individuu sa istim skupom drugih individua.
    - **sameAs** - izjava o identičnosti dviju individua. Individua može imati različita imena, i premda se vidi kao više individua, zaključivač zaključuje kao da je to jedinstvena individua.
    - **differentFrom** - izjava o različitosti određene individue od ostalih individua.
    - **AllDifferent** - za skup individua može se proglasiti da su sve one međusobno različite.
    - **distinctMembers** - funkcionira zajedno sa *AllDifferent* kao izjava da su svi članovi neke liste različiti i po parovima razdvojeni.
- Ograničena kardinalnost - omogućava izražavanje kardinalnosti 0 ili 1
    - **minCardinality** - izjava koja se odnosi na određeno svojstvo s obzirom na određenu klasu. Svaka instanca takve klase je u relaciji s barem tolikim brojem individua po tom svojstvu koliko iznosi vrijednost deklarirana pomoću *minCardinality*.
    - **maxCardinality** - isto kao za *minCardinality*, s tom razlikom što svaka instanca takve klase je u relaciji s najviše tolikim brojem individua po tom svojstvu koliko iznosi vrijednost *maxCardinality*. Ako je *maxCardinality* jednak 1, takvo se svojstvo naziva funkcionalno svojstvo.
    - **cardinality** - koristi se u OWL Lite za deklariranje jednake minimalne i maksimalne kardinalnosti koja može biti 0 ili 1. Na primjer, minCardinality=1 i maxCardinality=1.
- Verzioniranje
    - **versionInfo** - omogućava dodavanje oznake verzije za klasu, svojstvo ili ontologiju. Instanca owl:AnnotationProperty, i kao objekt sadrži znakovni niz koji opisuje verziju.
    - **priorVersion** - oznaka prethodne verzije. Također instanca owl:AnnotationProperty sa znakovnim nizom ali i referencom na drugu ontologiju.

        * **backwardCompatibleWith** - oznaka kompatibilnosti unazad.
        * **incompatibleWith** - oznaka kompatibilnosti.
        * **DeprecatedClass** - oznaka da je određena klasa zastarjela i da ju je bolje ne koristiti.
        * **DeprecatedProperty** - oznaka da je određeno svojstvo zastarjelo i da ju je bolje ne koristiti.

- OWL DL i OWL Full
  - Aksiomi klasa
    * **oneOf, dataRange** - definiranje pobrojane klase. Pobrojana klasa se definira tako da se pobroje sve individue koje joj pripadaju. Također je moguće klasu definirati kao raspon nekih numeričkih podataka.
    * **disjointWith** - deklarira razdvojenost klasa. Razdvojene klase ne mogu dijeliti istu individuu, odnosno, jedna individua ne može pripadati dvjema razdvojenim klasama. Zaključivači provjeravaju konzistentnost ontologija između ostalog i provjeravajući individue u razdvojenim klasama.
    * **equivalentClass** - za klasne izjave
    * **rdfs:subclassOf** - za klasne izjave
  - Booleanovske kombinacije klasnih izjava - OWL DL i OWL Full dozvoljavaju booleanske kombinacije klasa i ograničenja
    * **unionOf** - deklarira se da klasa sadrži individue koje pripadaju barem nekoj od klasa koje su navedene u uniji.
    * **complementOf** - deklarira se klasa koja sadrži individue koje ne pripadaju klasi navedenoj u komplementu.
    * **intersectionOf** - deklarira se klasa koja sadrži individue koje pripadaju svim klasama navedenim u presjeku.
  - Proizvoljna kardinalnost - omogućava izražavanje bilo koje cjelobrojne kardinalnosti, bilo kao točne kardinalnosti bilo kao raspona kardinalnosti.
    * **minCardinality** - najmanja dozvoljena kardinalnost nekog svojstva s obzirom na određenu klasu.
    * **maxCardinality** - najveća dozvoljena kardinalnost nekog svojstva s obzirom na određenu klasu.
    * **cardinality** - deklarira se kardinalnost koja je i najmanja i najveća, odnosno točno određena kardinalnost.
  - Filler
    * **hasValue** - odnosi se na svojstva, odnosno vrijednosti pridružene određenim svojstvima. Te vrijednosti su instance određene klase ili pak neke numeričke vrijednosti.

Kada se govori o uzorcima dizajna ontologija, najčešće se apstrahira razvojni jezik. To može biti neki objektno orijentirani jezik, jezik za modeliranje pa i grafički jezik (što u svojoj

suštini RDF i jest). Ipak, svoj pravi praktički smisao uzorci imaju u okviru semantičkog weba i jezika za razvoj ontologija u okviru semantičkog weba, odnosno neke W3C varijante jezika OWL jezika za prikaz znanja. Razloge možemo gledati sa povijesnog stanovišta, jer su se i ontologije a posljedično uzorci dizajna, razvijali u okviru semantičkog weba, RDFS i OWL.

2.4.5. Alati za zaključivanje

Semantički web nema jedinstveni sustav za automatsko zaključivanje. Primjena OWL jezika svodi se na pristup podacima pri čemu ontologije imaju ulogu opisa značenja tih podataka. Svi upitni jezici vezani uz OWL ontologije su konjuktivni nizovi čija je složenost izvršavanja velika, a postojeći algoritmi ne mogu istovremeno zadovoljiti zahtjeve valjanosti, potpunosti i skalabilnosti. Sustavi koji nastoje zadovoljiti potpunost i valjanost nisu skalabilni, a što je bitna odrednica semantičkog weba, budući da mu je ambicija povezati velike količine podataka. U tom trojstvu, uz uvjet skalabilnosti, može se žrtvovati izražajnost ontologija, odnosno valjanost ili pak potpunost. Najčešće se žrtvuje potpunost, jer se ona u semantičkom webu ni ne zahtijeva.(Stoilos, Grau i Horrcks, 2015)
Zbog toga postoje brojni sustavi za automatsko zaključivanje koji su kompromisna rješenja, svaki sa svojim prednostima i nedostacima, izražajnošću, skalabilnošću i odlučivošću, a svi se nastoje uskladiti sa postojećim standardnima vezanima uz infrastrukturu semantičkog weba: RDFS i OWL. U tablici 3 su prikazani neki od njih. Prema toj tablici, jedino F-OWL i Surnia podržavaju OWL Full, koji odgovara punoj izražajnosti RDF okvira. F-OWL je sustav za automatsko zaključivanje koji koristi sustav utemeljen na logici okvira, te je usmjeren na zaključivanje uz pomoć OWL ontologija. Koristi logiku višeg reda, što mu omogućava Flora-2 na kojoj je zasnovan, te napredne tehnike tabeliranja, koje preko Flora-2 preuzima iz Prologa XSB. To ga uvelike razlikuje od većine drugih sustava za automatsko zaključivanje.

Tablica 3: Tablica sustava za automatsko zaključivanje

Sustav	F-OWL	Racer	FaCT	Pellet	Hoolet	Surnia	Triple
Logika	Hornova, Okvira, Višeg reda	DL	DL	DL	Puna FOL	Puna FOL	Hornova
Podrška	OWL Full	OWL DL	OWL DL	OWL DL	OWL DL	OWL Full	RDF
Na bazi	XSB / Flora	Lisp	Lisp	Java	Vampire	Otter	XSB
XML tip podataka	Da	Da	Ne	Da	Ne	Ne	Ne
Odlučivost	Ne	Da	Da	Da	Ne	Ne	Da
Potpuna provjera konzistentnosti	Ne	Da (OWL-lite)	Da	Da (OWL-lite)	Ne	Ne	Ne
Sučelje	Java, GUI, Sučelje naredbenog retka	DIG, Java, GUI	DIG, Sučelje nared. retka	DIG, Java	Java	Python	Java
Upiti	Stil okvira, RDQL RDQL	Upitni jezik Racer		RDQL			Hornov logički stil
Poznata ograničenja	Slabo skaliranje		Nema podrške za A-Box		Slabo skaliranje	Slabo skaliranje	Podržava jedino RDF

(Prema izvoru: Zou, Finin i Chen (2004))

U F-OWL zaključivaču FLORA-2 pravila služe i za provjeru konzistentnosti OWL ontologija. Navodim primjer provjere ekvivalentnosti klasa. Ovaj mali Flora-2 program ima sličnu funkcionalnost kao i fragment F-OWL sustava za zaključivanje koji služi za određivanje ekvivalentnosti klasa. U listingu 2.5, prikazan je taj program koji provodi rezoniranje o ekvivalentnosti dviju klasa s obzirom na objektna svojstva, njihov tip i kardinalnost.

Listing 2.5: Ekvivalencija klasa (Flora-2)

```
//shema
Svojstvo[|nadsvojstvo=>Svojstvo, domena=>Klasa|].
ObjektnoSvojstvo::Svojstvo[|rang=>Klasa|].
Ogranicenje[|svojstvo=>Svojstvo, filler=>Klasa, tip=>\string, kardinalnost=>\integer
    |].
Klasa[|nadklasa=>Klasa|].

//klase u ontologiji
klasa1:Klasa.
klasa2:Klasa.
klasa3:Klasa.
//svojstva
imaS1:ObjektnoSvojstvo[domena->{klasa1,klasa2},rang->klasa3].
imaS2:ObjektnoSvojstvo[domena->{klasa1,klasa2},rang->klasa3].
//ogranicenje nad klasama
ogranicenjeO1:Ogranicenje[svojstvo->imaS1, filler->klasa3, tip->'some', kardinalnost
    ->1].
ogranicenjeO2:Ogranicenje[svojstvo->imaS1, filler->klasa3, tip->'some', kardinalnost
    ->1].
klasa1[nadklasa->ogranicenjeO1].
klasa2[nadklasa->ogranicenjeO2].

//pravila za ekvivalenciju
ekvivalentna(?_K2,?_K1):-ekvivalentna(?_K1,?_K2).
some(?_O):-?_O[tip->?_T,kardinalnost->?_Kard],(?_T='some';(?_T='min',?_Kard>0)).
ekvivalentna(?_K1,?_K2):-?_K1[nadklasa->?_O1],some(?_O1),?_O1[svojstvo->?_S],
?_K2[nadklasa->?_O2],some(?_O2),?_O2[svojstvo->?_S].
```

Danas su ontologije uglavnom utemeljene na deskriptivnim logikama koje imaju manju izražajnost pa se pribjegava posebnim trikovima, koji su čak i određeni uzorci, kako ćemo kasnije vidjeti da se iskaže višestruko pridruživanje. Primjerice, skupovno pridruživanje u F-logici nije moguće u deskriptivnim logikama, pa se koriste n-arne relacije. U nekoj projekciji budućeg razvoja, jezici deskriptivne logike vide se kao podskupovi općih deskriptivnih jezika (DFL, eng. *General Desctiption Language*) zasnovanih na logikama okvira. Svi konstrukti iz jezika deskriptivnih logika moći će se primjenjivati i u općim deskriptivnim jezicima. (Balaban, 1995, str. 15)

3. Uzorci dizajna

Budući da razvoj kvalitetne ontologije nije trivijalan, potrebno je oformiti uzorke dizajna ontologija kako bi se proces razvoja učinio jednostavnijim i bržim, a rješenje pouzdano i jednostavno za održavanje. Uzorci se nalaze u prostoru rješenja problema i povezani su sa kompetencijskim pitanjima u problemskom prostoru.(Gangemi, 2009)

Uzorke možemo promatrati na dva načina: kao predloške (eng. *templates*) i kao najbolje prakse odnosno preporuke opisa, kodiranja ili općenito oblikovanja određene problemske domene. Uzorci su odgovori na pitanja oblikovanja određenog rješenja, koja se često ponavljaju u različitim varijantama. Prema Websterovom rječniku, kako navode Presutti, Gangemi i David (2008), pojam uzorka ima skup u određenoj mjeri sličnih značenja:

a) preporučeni oblik (model) imitacije

b) nešto dizajnirano ili korišteno kao model prema kojem će se kreirati stvari

c) model kalupa

d) umjetnički, glazbeni, literalni ili mehanički oblik ili forma

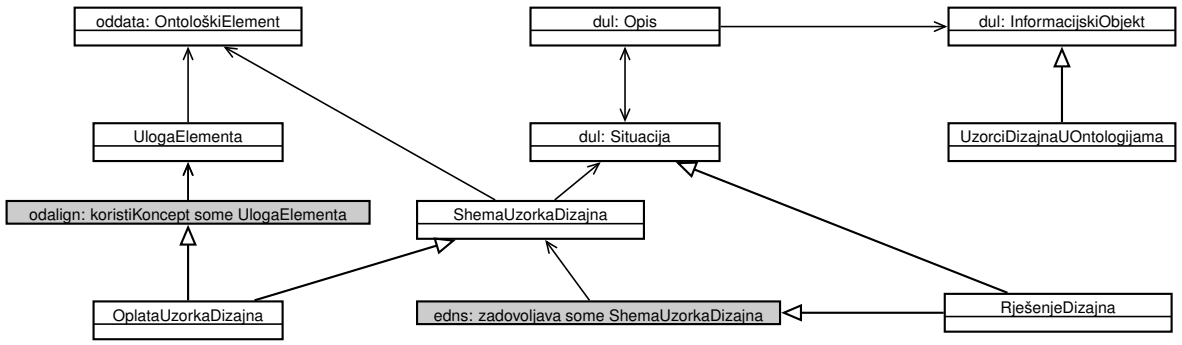
e) prirodna ili slučajna konfiguracija

f) uočljivo koherentni sustav utemeljen na željenoj povezanosti sastavnih komponenti

Uzorak dizajna ontologija se definira na slijedeći način:

Definicija 3.0.1 *Uzorak dizajna ontologija prema (Presutti, Gangemi i David, 2008, str. 18) Uzorak dizajna ontologija je model kao rješenje ponavljajućeg problema dizajna. To je informacijski objekt koji izražava shemu uzorka dizajna (odnosno oplatu dizajna). Takvu shemu zadovoljavaju jedino dizajnerska rješenja. Dizajnerska rješenja pružaju postavke za ontološke elemente koji igraju neke elementarne uloge u okviru takve sheme.*

Formalno je ovu definiciju moguće prikazati i grafički kao na slici 9:



Slika 9: Grafički prikaz definicije uzoraka dizajna ontologija (izvor: (Presutti, Gangemi i David, 2008, str. 18))

U dijagramu oznake *dul:*, *oddata:*, *edns:* i *odalign:* su imenski prostori vezani uz URI prefikse resursa semantičkog weba koje je koristio laboratorij za primijenjenu ontologiju (LOA, eng. *Laboratory for Applied Ontology*[*]) tijekom razvoja DOLCE Ultra-Lite ontologije:

- dul: `http://www.loa-cnr.it/ontologies/DUL.owl#`
- oddata: `http://www.loa-cnr.it/ontologies/OD/odData.owl#`
- odalign: `http://www.loa-cnr.it/ontologies/OD/odAlignment.owl#`
- edns: `http://www.loa.istc.cnr.it/ontologies/ExtendedDnS.owl#`

Definicije elemenata koji čine definiciju uzoraka dizajna ontologije slijede:
(Presutti, Gangemi i David, 2008, str. 16-18))

[*]Ovaj laboratorij je smješten u Trentu u Italiji, a web adresa je http://www.loa.istc.cnr.it/

Definicija 3.0.2 *Informacijski objekt*
Informacijski objekt je dio informacije o nekom entitetu, koji neki agent *može interpretirati. Kodiran je nekim sustavom kodiranja informacija i realiziran od strane nekog entiteta. Posljedično, informacijski objekt je ovisan o enkodiranju kao i o konkretnoj realizaciji.*

Definicija 3.0.3 *Opis*
Opis predstavlja konceptualizaciju te je ovisan o agentu i komunikacijskim mogućnostima. Može biti reifikacija kognitivnog konteksta ili logička n-arna relacija. Opisi koriste koncepte, izražavaju se preko informacijskih objekata i mogu biti zadovoljeni određenom situacijom.

Definicija 3.0.4 *Situacija*
Situacija je postavka za entitete i obično je konzistentna sa opisom.

Definicija 3.0.5 *Shema uzorka dizajna*
Shema uzorka dizajna je opis uzorka dizajna. Uključuje uloge, zadatke i parametre potrebne za rješavanje problema oblikovanja ontologija.

Definicija 3.0.6 *Oplata uzorka dizajna*
Oplata uzorka dizajna je takva shema uzorka dizajna koja opisuje čistu strukturu uzorka dizajna ontologije. Oplata identificira elemente koji se ugrađuju u ontologiju a koji su ponovno iskoristivi u uzorku dizajna.

Definicija 3.0.7 *Ontološki element*
Ontološki element je (identificirani) element ontologije. Ontološki se elementi također modeliraju metamodelom definicije ontologije i ontološkim vokabularom metapodataka.

U ontološke elemente ubrajamo OWL klase, OWL svojstva i OWL ograničenja.

Definicija 3.0.8 *Rješenje dizajna*
Rješenje dizajna je situacija koja uključuje samo formalne izraze (barem uzorke sadržaja ili logičke konstrukte), i njihove relacije.

Definicija 3.0.9 *Signatura uzorka*
Signaturu uzorka čini popis svi predikatnih naziva, odnosno naziva klasa i svojstava u OWL ontologiji. Ukoliko je ovaj popis prazan, uzorak je neovisan o sadržaju, odnosno domenski neovisan. Takvi uzorci ovise isključivo o logičkom formalizmu koji se koristi za njihovu reprezentaciju.

3.1. Katalog uzoraka dizajna

Svaki uzorak ima svoju katalošku oznaku i unos sa slijedećim mogućim stavkama: (Gangemi i Presutti, 2009)

- Naziv uzorka (eng. *Name*)
- Namjena i generički slučaj korištenja (eng. *Intent*)
- Pitanja kompetencije (eng. *Competency questions*)
- Alternativni naziv (eng. *Also known as* ili akronim *a.k.a.*) pod kojim je uzorak poznat
- Scenariji (eng. *Scenarios*) izraženi u prirodnom jeziku koji govore o primjerima zahtijeva koji se postavljaju pred uzorak
- Dijagrami (eng. *Diagrams*) najčešće u UML-u
- Elementi (eng. *Elements*) koji se najčešće svode na klase i relacije zajedno sa svojim ulogama unutar određenog uzorka
- Posljedice (eng. *Consequences*) koje govore o benefitima ali i kompromisima koje donosi korištenje određenog uzorka u određenom slučaju
- Poznati slučajevi korištenja (eng. *Known uses*) u konkretnim projektima, bilo da su oni softverski proizvodi ili ontologije
- Izvornik iz kojega je uzorak izvučen (eng. *Extracted from/Reengineered from*) opisuje referentni softverski proizvod ili ontologiju koja je izvorište tog uzorka
- Povezani uzorci (eng. *Related patterns*) čine skup uzoraka koji su blisko povezani sa promatranim uzorkom bilo da su njegova specijalizacija, generalizacija, kompozicija ili komponenta. Jednako tako to su i uzorci koji se najčešće koriste zajedno sa promatranim uzorkom.
- Gradivni blokovi (eng. *Building blocks*) specifično za ontologije označavaju URI reference prema drugim OWL datotekama koje sadržavaju njegove implementacije.

3.2. Uzorci dizajna u softverskom inženjerstvu

Glavni benefiti pri korištenju uzoraka dizajna u softverskom inženjerstvu svode se na izbjegavanje odabira neprikladnih modela koje je kasnije teško ispravljati i održavati. Kôd napisan prema uzorcima dizajna je čitljiviji i bolji u višestrukoj iskoristivosti.

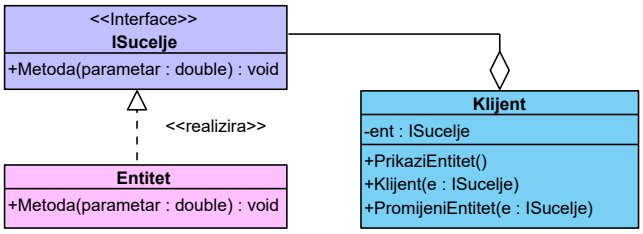
U softverskom inženjerstvu problemi dizajna su klasificirani i najbolje prakse su pretočene u skupove uzoraka koji su daleko manje podložni promjenama od uzoraka u ontologijama. Dva osnovna kataloga su sadržana u knjigama *Design Patterns: Elements of Reusable Object-Oriented Software* četvorice autora - Gamme, Johnsona, Vllisidesa i Boocha te *Pattern-Oriented Software Architecture* u 5 tomova grupe autora Buschmanna, Schmidt, Meuniera, Henneya i drugih. Prvi se katalog, prema četvorici autora kolokvijalno naziva Uzorcima četvorice autora (eng. *Gang of Four Patterns*, GOF) dok se drugi katalog, a zbog činjenice da je većina autora došla iz tvrtke Siemens često naziva i Siemensov katalog ili pak prema nazivu tih 5 tomova akronimom POSA uzorci (Arhitektura softvera utemeljena na uzorcima, eng.

Pattern-Oriented Softvare Architecture). Zbog obima POSA uzoraka, navest ćemo samo GOF uzorke.

3.2.1. Uzorci četvorice - GOF

Knjiga *Design Patterns: Elements of Reusable Object-Oriented Software* (1994) opisuje i katalogizira 23 klasična uzorka dizajna u softverskom inženjerstvu i temelj je svih uzoraka dizajna. Autori Erich Gamma, Ralph Johnson, John Vllisides i Grady Booch poznati su i kao grupa četvorice (eng. *Gang of Four*), pa se i katalog tih 23 uzoraka naziva *GOF uzorci*. U knjizi autori zastupaju upotrebu sučelja (eng. *interfaces*) te programiranja za sučelje a ne konkretnu implementaciju, a jednako tako i kompoziciju objekata naspram nasljeđivanja klasa. Kako smo vidjeli u primjeru Flora-2 implementacije F-OWL sustava za automatsko zaključivanje, sve ontološke klase implementirane su kao kompozicije objekata. Nasljeđivanje je bilo rezervirano za mehanizam samog F-OWL sustava i imalo je plitku hijerarhiju.

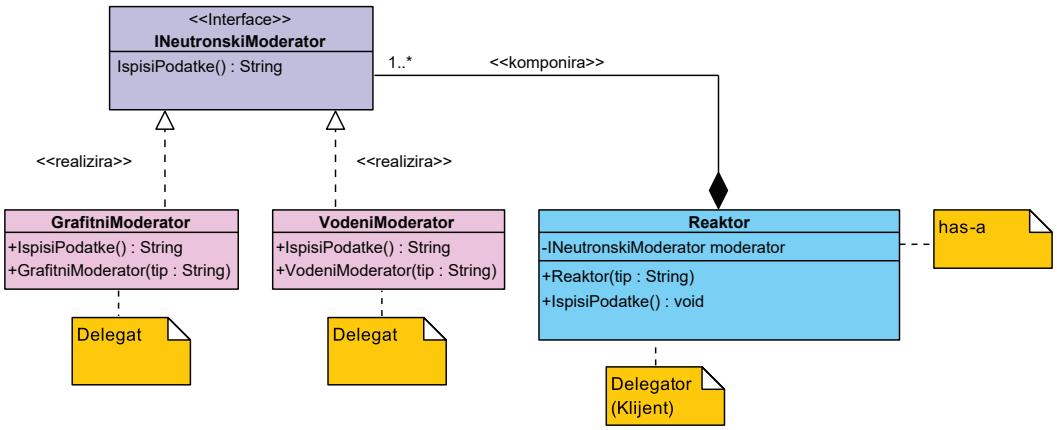
Kompozicija je posebni slučaj asocijacije objekata, kao što je to slučaj i sa agregacijom. Agregirani objekti nisu ovisni i objektu koji ih agregira, dok komponirani objekti jesu. Drugim riječima, životni ciklus komponiranih objekata strogo je vezan uz životni ciklus agregirajućeg objekta. Na primjer, član tima je agregiran u timu, može promijeniti tim a i postojati van tima. Sa druge strane, mozak je sastavni dio tijela i dijeli s tijelom životni ciklus. Usko povezan sa konceptom kompozicije je i koncept delegacije budući da kompozicija lesto zahtijeva delegiranje. Asocijacija, a posebno kompozicija, osnova je koncepta, ili prauzorka delegiranja. Prema (Gostischa-Franta, 2013), "delegacija je slična nasljeđivanju, izvedena ručno upotrebom kompozicije objekata". GOF ne svrstava delegiranje u uzorke, ali je dizajnerski koncept delegiranja osnova nekih GOF uzoraka poput *Visitora*, *Observera*, *Strategy* i *Event Listenera*. Također, GOF uzorak *Dekorator* vrlo je sličan delegiranju. Prema autorima GOF uzoraka, "delegacija je način stvaranja kompozicije koji je jednako dobar po pitanju ponovne iskoristivosti kao i nasljeđivanje" (Gamma i dr., 1997). Na slikama 11 i 12 prikazana su oba koncepta na istom primjeru. Programski kod u jeziku Java dan je u prilogu.



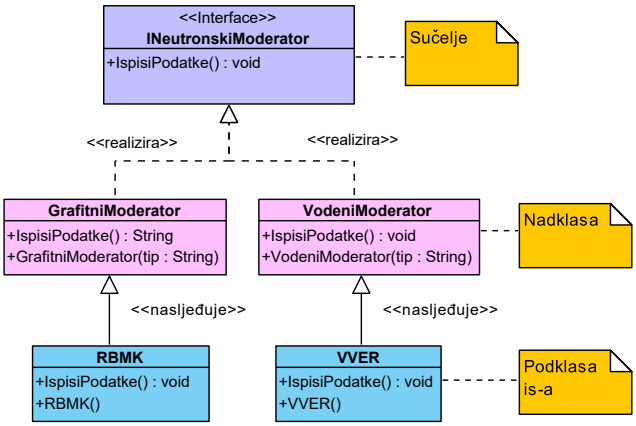
Slika 10: UML dijagram korištenja sučelja

U dijagramu, ali i kôdu, prikazana je i upotreba sučelja. Ukoliko je upotrijebi agregacija, i klijentu proslijedi objekt kao instanca klase koja realizira sučelje, klijent može pozivati metode tog objekta bez da je svjestan kojoj klasi on pripada. Time omogućava slaba povezanost klasa i veća fleksibilnost u dizajnu sustava što se ogleda u boljoj prenosivosti, proširivosti i promjenjivosti implementacije. Upotreba sučelja prikazana je i na slici 10, na maksimalno

pojednostavljenom primjeru koji koristi agregaciju. Objekt proslijeđen klijentu kreiran je izvana, i klijent zapravo ni ne zna kakvog je tipa proslijeđeni objekt. Jedino što po principu ugovorenog dizajna (eng. *design-by-contract*) zna jest da taj objekt ima implementiranu metodu "Metoda" budući da implementira sučelje. Svaki objekt, ma kako napisan, bit će prihvaćen od strane klijenta ispravno, ako odgovara na zahtjeve sučelja.



Slika 11: UML dijagram primjera delegiranja



Slika 12: UML dijagram primjera nasljeđivanja

Svi uzorci navedeni u "knjizi četvorice" mogu se smjestiti u tablicu prostora uzoraka dizajna.

Tablica 4: GOF uzorci dizajna

		Namjena		
		Kreiranje	Strukturiranje	Ponašanje
Područje	Klasa	Factory Method	Adapter	Interpreter
				Template Method
	Objekt	Abstract Factory	Adapter	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Facade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

(Prema izvoru: (Gamma i dr., 1997, str. 22))

Gang of Four uzorci dizajna dijele se, kao što se vidi u tablici na uzorke kreiranja, uzorke strukture i uzorke ponašanja. Uzorci se također dijele na klasne i objektne.

Uzorci kreiranja apstrahiraju kreiranje objekata odnosno inicijalizaciju sustava čime se postiže neovisnost o načinu kreiranja, sastavljanja i reprezentaciji objekata. Uzorci strukture pomažu prilikom sastavljanja objekata i klasa kako bi formirali složenije strukture. Uzorci ponašanja odnose se na algoritme i odgovornosti objekata. Opisuju uzorke objekata i klasa te njihovu međusobnu komunikaciju.

Klasni uzorci se bave vezama u hijerarhijskoj strukturi klasa, odnosno nasljeđivanjem. Takvi su uzorci uglavnom statični. Objektni uzorci se odnose na veze među objektima koje se mogu mijenjati tijekom izvršavanja, pa su ovi uzorci dinamičniji. Detaljnija razrada ovih uzoraka kao i navođenje cijelog kataloga, zahtijevala bi mnogo prostora, pa ću se ograničiti samo na kratki opis svakog od tih uzoraka, te ukoliko postoji poveznica sa ontološkim problemima tu ću poveznicu spomenuti.(Huston, 2014)

1. **Uzorci kreiranja**

   - **Singleton** - osigurava da klasa ima samo jednu instancu te omogućava globalni pristup toj instanci
   - **Builder** - odvaja složenu konstrukciju objekta od njegove reprezentacije. Isti proces konstrukcije može kreirati različite reprezentacije.
   - **Abstract factory (komplet - eng. *Kit*)** - definira sučelje za kreiranje obitelji povezanih ili ovisnih objekata bez specificiranja njihove konkretne klase.
   - **Factory method (virtualni konstruktor)** - definira sučelje za kreiranje objekata ali se na podklase prenosi odlučivanje koja će se klasa pri tome instancirati.

- **Prototype** - specificira vrste objekata koji se kreiraju koristeći prototipsku instancu (prototip). Kreiranje se svodi na kopiranje prototipa.

2. **Uzorci strukture**

   - **Adapter (omotač - eng. *Wrapper*)** - pretvara postojeće sučelje klase u korisniku potrebno sučelje. Omogućava zajednički rad klasa koje to inače ne mogu zbog prevelikih razlika u sučeljima
   - **Bridge (rukovatelj/tijelo - eng. *Handle/Body*)** - odvaja apstrakciju od implementacije. Omogućava da se oboje može neovisno mijenjati.
   - **Composite** - komponira objekte u stablastu strukturu predstavljajući dio/cjelina hijerarhijske strukture. Omogućava klijentima da individualne objekte ali i kompozicije objekata tretiraju jednako.
   - **Decorator (omotač)** - Pridružuje objektima nove odgovornosti dinamički. Fleksibilnija je alternativa nasljeđivanju klasa prilikom proširenja funkcionalnosti.
   - **Proxy (surogat)** - Pruža surogat ili nadomjestak za drugi objekt kako bi se omogućilo upravljanje pristupom do njega.
   - **Facade** - pruža novo sučelje umjesto skupa sučelja nekog podsustava. Definira sučelje na višoj razini kako bi se olakšalo korištenje podsustava
   - **Flyweight** - Koristi dijeljenje kako bi se omogućila podrška za veliki broj fino granuliranih objekata

3. **Uzorci ponašanja**

   - **Chain of Responsibility** - Izbjegava povezivanje pošiljatelja zahtijeva i njegovog primatelja omogućavajući većem broju od jednog objekta da obradi taj zahtjev. Zahtjev se prosljeđuje duž lanca povezanih objekata sve dok ga odgovarajući objekt ne obradi
   - **Command (akcija, transakcija - eng. *Action, Transaction*)** - Učahuruje zahtjev unutrar objekta čime se omogućava parametrizacija klijenata sa različitim zahtjevima, redovima čekanja ili dnevničkim zahtjevima (logovima). Omogućava operaciju vraćanja (eng. *undo*)
   - **Interpreter** - Za dani jezik, definira reprezentaciju njegove gramatike uključujući i interpretaciju koja koristi tu reprezentaciju kako bi se interpretirale rečenice izražene u tom jeziku.
   - **Iterator (kursor - eng. *Cursor*)** - Pruža način za sekvencijalni pristup elementima agregiranog objekta bez otkrivanja njegove unutarnje strukture
   - **Mediator** - Definira objekt koji učahuruje načine interakcije skupa objekata. Pruža slabu povezanost objekata izbjegavajući njihovo međusobno eksplicitno referenciranje, dajući korisniku slobodu neovisnog variranja njihovih interakcija.
   - **Memento (token)** - Ne narušavajući učahurivanje, obuhvaća i eksternalizira unutarnje stanje objekta, kako bi se objekt mogao kasnije vratiti u to stanje.

- **Observer** - Definira ovisnost među objektima "jedan prema više" tako da kada jedan promijeni stanje, svi ostali su informirani i ažurirani automatski
- **State (objekti za stanja - eng.Objects for States)** - Dozvoljava objektu promjenu ponašanja kada se promjeni njegovo unutarnje stanje. Objekt će se ponašati kao da je promijenio klasu.
- **Strategy (politika - eng. *Policy*)** - Određuje obitelj algoritama, učahuruje svakog od njih i omogućava njihovu razmjenu. Omogućava neovisne varijacije algoritama bez promjene njihove strukture.
- **Template Method** - Određuje kostur algoritma neke operacije, prenoseći neke njegove elemente na podklase. Omogućava podklasama redefiniranje pojedinih koraka algoritma bez cjelovite promjene algoritamske strukture.
- **Visitor** - Predstavlja operaciju koja se izvršava nad elementima objektne strukture. Visitor omogućava definiranje nove operacije bez promjene klasa elemenata nad kojima se primjenjuje.

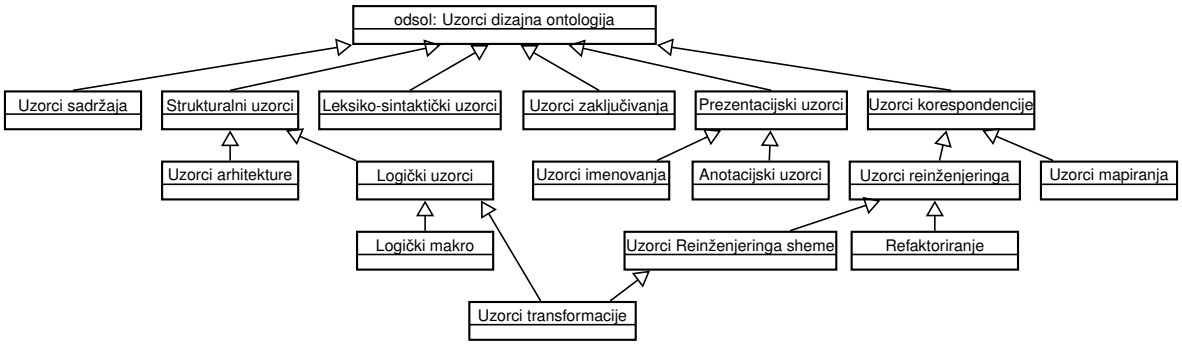
3.3. Uzorci dizajna ontologija

Prema Gangemi i Presutti (2009) uzorke dizajna ontologija možemo podijeliti u 6 obitelji:

1. strukturalni uzorci koji se dalje mogu podijeliti na
   - uzorke arhitekture
   - logičke uzorke
2. uzorci korespondencije
   - uzorke reinženjeringa
   - uzorci poravnanja
3. uzorci sadržaja
4. uzorci zaključivanja
5. prezentacijski uzorci
   - uzorci imenovanja
   - uzorci anotacije
6. leksiko-sintaktički uzorci

Ove obitelji imaju finije podjele, pa pored već navedenih tu su još i logički makroi, uzorci reinženjeringa sheme, uzorci refaktoriranja i uzorci transformacija. Njihovi detaljni odnosi prikazani su na slici 13 u vidu dijagrama klasa.

Iako uzorci mogu pripadati jednoj od tih 6 obitelji, u ovom trenutku su svi potpora životnom ciklusu uzoraka sadržaja. Uzorci sadržaja su male ontologije koje povezuju slučajeve



Slika 13: Obitelji uzoraka dizajna ontologija (izvor: Gangemi i Presutti (2009))

korištenja i dizajn rješenja. Kao takvi su idealne komponente za izgradnju takvih ontologija koje su valjane s obzirom na svoju domenu i postavljene zadatke. Kako bi OWL jezici mogli biti primijenjeni na sve uzorke koji su potrebni znanosti, osobito na OBO uzorke u biomedicinskim ontologijama, potrebna je visoka ekspresivnost OWL jezika poput $\mathcal{ALCHIN}$, $\mathcal{SHOIN}$, $\mathcal{SHOIQ}$ i $\mathcal{SROIQ}$ (Horridge i dr., 2012).

3.3.1. Osnovni koncepti

Kako bi se jednostavnije pratili shematski prikazi uzoraka, potrebno je prokomentirati osnovne koncepte od kojih se ti uzorci, ali i sami dijagrami sastoje. To je zapravo preslika dijela sintakse OWL jezika, no moguće ih je prikazati i u notaciji deskriptivnih logika. Koristit ću grafički oblik kakav se koristi za SourceForge katalog biomedicinskih uzoraka, a te sam uzorke i odabrao kao referentne, prema kojima ću uspoređivati sve druge kataloge uzoraka.

Jedan od osnovnih metakoncepata čini generalizacija odnosno specijalizacija, a sa stanovišta objektnog programiranja, to se može promatrati kao nasljeđivanje klasa. Klase su skupovi instanci koje dijele zajednička svojstva. Tako je podklasa poseban (specijalni) slučaj nadklase, ona sadrži odnosno nasljeđuje sva opća (generalna) svojstva nadklase, ali ih nadopunjava nekim svojim dodatnim, posebnim (specijalnim) svojstvima. Sve individue koje pripadaju podklasi, pripadaju i nadklasi. Obrnuto ne mora vrijediti.

Slika 14: Nasljeđivanje klasa

Sintaksa za ovaj osnovni koncept dana je u listingu:

Listing 3.1: Nasljeđivanje, RDF/XML serijalizacija

```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#Nadklasa"/>
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#Podklasa">
<rdfs:subClassOf rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/koncepti
    #Nadklasa"/>
```

```
</owl:Class>
```

Prikaz u DL sintaksi:

$$Podklasa \sqsubseteq Nadklasa$$

Pored nasljeđivanja drugi važni metakoncept je ekvivalentnost klasa. U poglavlju vezanom uz Flora-2 pokazao sam kako se modelira ekvivalentnost klasa, uz ilustraciju rada zaključivača. U OWL i alatima poput Protégé-a dvije se klase mogu učiniti ekvivalentnim postavljanjem aksioma o ekvivalentnim klasama, a jednako tako ih i zaključivač može prepoznati kao ekvivalentne ukoliko se dođe do zaključka da sve individue koje pripadaju jednoj klasi, pripadaju i drugoj klasi i obrnuto. Može se jednako tako reći da je svaka od njih podklasa one druge.

Slika 15: Ekvivalentnost klasa

Formalno se može i napisati za dvije ekvivalentne klase C i D:
$(C \sqsubseteq D) \sqcap (D \sqsubseteq C) \leftrightarrow (C \equiv D)$. Sintaksa za ekvivalentne klase prikazana je u listingu:

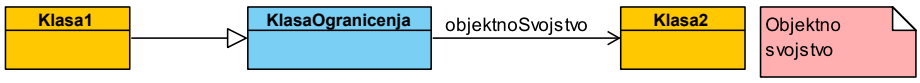
Listing 3.2: Ekvivalentnost klasa, RDF/XML serijalizacija

```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#Klasa1">
<owl:equivalentClass rdf:resource="http://www.semanticweb.org/
elvis/uzorci/2019/koncepti#Klasa2"/>
</owl:Class>
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#Klasa2"/>
```

Prikaz u DL sintaksi:

$$Klasa1 \equiv Klasa2$$

U OWL jezicima moguće je uvesti uvjete, odnosno ograničenja koje moraju zadovoljavati članovi neke klase. To se postiže klasom ograničenja, koja može biti anonimna klasa. Takva klasa nema ime niti URI/IRI resurs. Formalno, ograničena klasa (prikazana žuto u dijagramu 16) je potklasa klase svog ograničenja (prikazana plavo). U izjavi o ograničenju bitan element čini svojstvo, koje može biti objektno ili podatkovno. U ovom primjeru koristi se objektno svojstvo. Suprotno intuitivnom shvaćanju da objektna svojstva povezuju klase, ona zapravo povezuju objekte, instance klasa, odnosno individue koje pripadaju pojedinim klasama. Tako svojstvo može povezivati individue jedne klase sa nekim individuama druge klase (egzistencijalna kvantifikacija ograničenja svojstva - *some*), može ih povezivati isključivo sa individuama te klase i niti jedne druge (univerzalna kvantifikacija ograničenja svojstva - *all*), a moguće je specificirati i kardinalnosti (*min, exactly, max*) koje opisuju sa koliko je instanci druge klase moguće povezati individuu neke klase.



Slika 16: Objektno svojstvo

U slijedećem listingu prikazan OWL fragment koda (zapisan kao RDF/XML) sa egzistencijalno kvantificiranim ograničenjem svojstva i aksiomom *rdfs:subClassOf*:

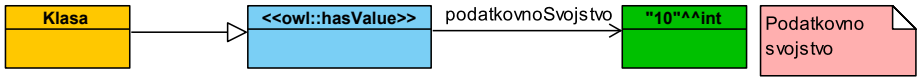
Listing 3.3: Ograničenje i objektno svojstvo, RDF/XML serijalizacija

```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/ogranicenje#
    Klasa1">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/
    ogranicenje#objektnoSvojstvo"/>
<owl:someValuesFrom rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/
    ogranicenje#Klasa2"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Prikaz u DL sintaksi za *some*, *all* i *exactly 1*:

$$Klasa1 \sqsubseteq (\exists objektnoSvojstvo.Klasa2)$$
$$Klasa1 \sqsubseteq (\forall objektnoSvojstvo.Klasa2)$$
$$Klasa1 \sqsubseteq (= 1\ objektnoSvojstvo.Klasa2)$$

Podatkovna svojstva povezuju objekte tj. instance klasa i literale tj. podatkovne vrijednosti. Pri tom se za podatkovne vrijednosti mogu koristiti predefinirani tipovi podataka iz XML sheme. Jednako tako mogu i konkretnim individuama pridružiti određene podatkovne vrijednosti.



Slika 17: Podatkovno svojstvo

Upotreba podatkovnog svojstva u izjavi o ograničenju pri kreiranju klase ograničenja:

Listing 3.4: Pridruživanje vrijednosti instancama klase

```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#Klasa">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    podatkovnoSvojstvo"/>
<owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">10</
    owl:hasValue>
</owl:Restriction>
```

```
</rdfs:subClassOf>
</owl:Class>
```

Upotreba podatkovnog svojstva za pridruživanje vrijednosti određenoj individui:

Listing 3.5: Pridruživanje vrijednosti individui

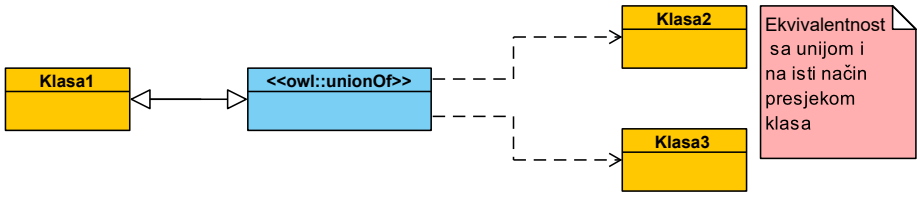
```
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
    koncepti#individua">
<podatkovnoSvojstvo rdf:datatype="http://www.w3.org/2001/XMLSchema#int">10</
    podatkovnoSvojstvo>
</owl:NamedIndividual>
```

U Protégéu, prilikom pridruživanja vrijednosti instancama klase, tip podatkovne vrijednosti se automatski određuje prema njezinom zapisu. Malo slovo f iza broja, kreirat će *xsd:float* tip, dok će upotreba zareda kreirati *xsd:decimal* tip. U Turtle zapisu za float tip će se koristiti oblik *"-10.5"^^xsd:float*, dok će u RDF/XML biti korišten *rdf:datatype="http://www.w3.org/2001/XMLSchema#float"* unutar *hasValue* oznake. Ovo je neovisno o pridruženom podatkovnom tipu samoj klasi individua. Prilikom pridruživanja vrijednosti određenoj individui, eksplicite se specificira podatkovni tip.

Formalni zapis tipa vrijednosti i samog iznosa vrijednosti koje se pridružuju instancama klase u DL sintaksi dan je u listingu:

$$Klasa1 \sqsubseteq (\exists podatkovnoSvojstvo.xsd : int)$$
$$Klasa1 \sqsubseteq (\exists podatkovnoSvojstvo.\{10\})$$
$$Klasa1 \sqsubseteq (\exists podatkovnoSvojstvo : 10)$$

Određena klasa se može poistovjetiti sa unijom ili presjekom drugih klasa. U slijedećem dijagramu *Klasa1* je ekvivalentna uniji klasa *Klasa2* i *Klasa3*. To znači da označava skup instanci koje pripadaju jednoj od klasa u uniji. Ako klase nisu razdvojene, mogu pripadati i objema klasama. Posebno predefinirano svojstvo *owl:disjointUnionOf* specificira takvo ograničenje kod kojeg individue mogu pripadati samo jednoj od klasa koje su u uniji, što je ekvivalentno uniji razdvojenih klasa.



Slika 18: Unija klasa

Zapis ekvivalencije jedne klase i unije više klasa (ne moraju biti nužno dvije) dat je u listingu:

Listing 3.6: Unija klasa

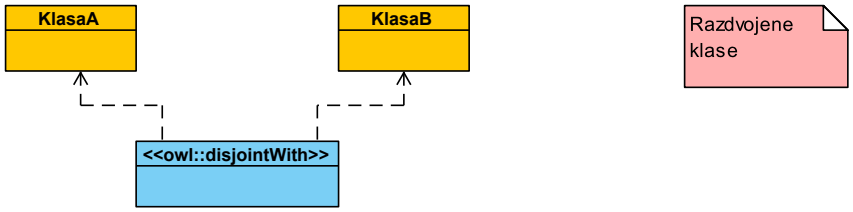
```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#Klasa1">
```

```
<owl:equivalentClass>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<rdf:Description rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    Klasa2"/>
<rdf:Description rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    Klasa3"/>
</owl:unionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
```

Zapis u DL sintaksi koristi disjunkciju odnosno logičko *ili* za uniju klasa (u Protégé-u **or**, u DL $\sqcup$):

$$Klasa1 \equiv (Klasa2 \sqcup Klasa3)$$

Na sličan način se koristi i aksiom o presjeku klasa (*owl:intersectionOf*), uz tu razliku što je presjek razdvojenih klasa uvijek prazan skup. Razdvojene klase se definiraju aksiomom *owl:disjointWith*. One ne mogu sadržavati takve individue koje pripadaju istovremeno više razdvojenih klasa. Ukoliko se definiraju dvije klase, tako da je jedna presjek razdvojenih klasa, a druga preko aksioma *owl:disjointUnionOf*, zaključivač će ih poistovjetiti, tj. učiniti ekvivalentnim klasama.



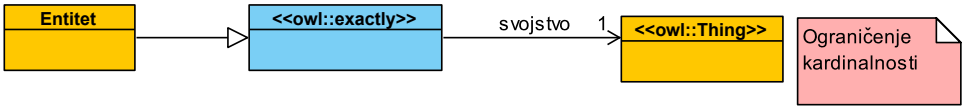
Slika 19: Razdvojene klase

Formalni zapis u sintaksi deskriptivnih logika koristi konjukciju odnosno logičko *i* (u Protégé-u **and**, u DL $\sqcap$) i tu konjukciju izjednačava sa antitautologijom:

$$KlasaA \sqcap KlasaB = \bot$$

Treba reći i da sama svojstva mogu također biti razdvojena (aksiom *propertyDisjointWith*), što znači da se ne mogu povezivati dvije individue sa više od jednog takvog razdvojenog svojstva. Pored toga, svojstva mogu biti također ograničena, i ta se ograničenja dijele na tri grupe: kvantificirana ograničenja (egzistencijalno *some* i univerzalno *all*), ograničenja kardinalnosti i ograničenje vrijednosti. U dijagramima na slici 20, odnosno slici 21, prikazano je ograničenje kardinalnosti na točno jedan. To znači da se individui iz klase *Entitet* mogu pridružiti samo jedna individua iz klase *owl:Thing* odnosno klase *Vrijednost* respektivno. U slučaju da pridružimo određenoj individui iz Klase *Entitet* više individua od broja određenog kardinalnošću, zaključivač će takve individue poistovjetiti. Ukoliko se postavi aksiom za te individue kao različite, zaključivač će čitavu ontologiju proglasiti nekonzistentnom, jer ne može poistovjetiti različite

individue, niti razriješiti narušenu kardinalnost. Funkcionalna svojstva su takva svojstva koja imaju maksimalnu kardinalnost 1.



Slika 20: Ograničenje kardinalnosti

Ukoliko se ne specificira klasa pripadnosti individue s kojom se povezuju individue u klasi *Entitet* kao u dijagramu na slici 20, tada se radi o općem ograničenju kardinalnosti svojstva. OWL sintaksa zapisana u XML/RDF serijalizaciji dana je u listingu 3.7:

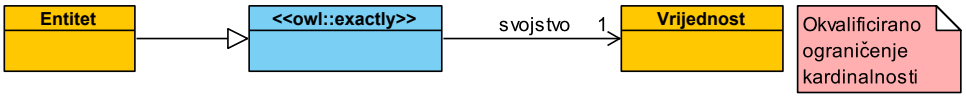
Listing 3.7: Ograničenje kardinalnosti objektnog svojstva

```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#Entitet">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    svojstvo"/>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
    1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

U Protégé-u se koristi opis *Entitet SubClass Of svojstvo exactly 1*. DL sintaksa je slijedeća:

$$Entitet \sqsubseteq (= 1\ svojstvo.\top)$$

Ukoliko se specificira klasa, kao u dijagramu 21, tada govorimo o okvalificiranom ograničenju kardinalnosti.



Slika 21: Okvalificirano ograničenje kardinalnosti

OWL zapis dan je u listingu 3.8:

Listing 3.8: Okvalificirano ograničenje kardinalnosti objektnog svojstva

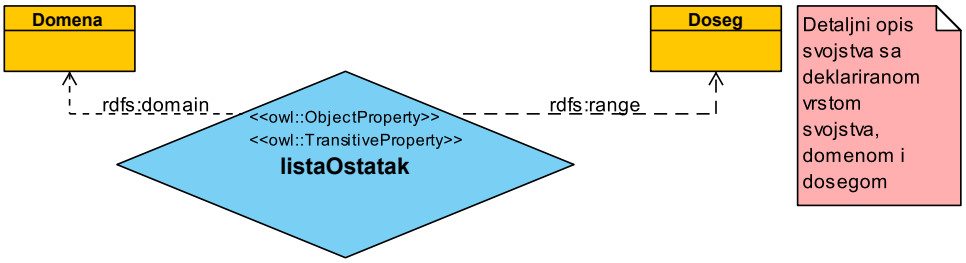
```
<owl:Class rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#Entitet">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    svojstvo"/>
<owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
    nonNegativeInteger">1</owl:qualifiedCardinality>
```

```
<owl:onClass rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    Vrijednost"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

U Protégé-u se koristi opis *Entitet SubClass Of svojstvo exactly 1 Vrijednost*. DL sintaksa je:

$$Entitet \sqsubseteq (= 1\ svojstvo.Vrijednost)$$

U uzorcima se često dodatno opisuju određena svojstva jer na samom grafu ne bi bilo dovoljno prostora za to. Svojstvo je označeno kao romb sa imenom, vrstom i tipom svojstva te definiranom domenom i dosegom svojstva. Svojstva po tipu mogu biti funkcionalna, inverzno funkcionalna, tranzitivna, simetrična, asimetrična, refleksivna i irefleksivna.



Slika 22: Opis svojstva

OWL zapis dan je u listingu:

Listing 3.9: Opis svojstvaa

```
<owl:ObjectProperty rdf:about="http://www.semanticweb.org/elvis/uzorci/2019/
koncepti#listaOstatak">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
<rdfs:domain>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
<owl:someValuesFrom rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/
    koncepti#Domena"/>
</owl:Restriction>
</rdfs:domain>
<rdfs:range>
<owl:Restriction>
<owl:onProperty rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/koncepti#
    listaOstatak"/>
<owl:someValuesFrom rdf:resource="http://www.semanticweb.org/elvis/uzorci/2019/
    koncepti#Doseg"/>
</owl:Restriction>
</rdfs:range>
</owl:ObjectProperty>
```

U DL sintaksi ovo se može izraziti na slijedeći način:

$$\exists listaOstatak.Doseg \sqsubseteq Domena$$
$$Domena \sqsubseteq \forall listaOstatak.Doseg$$
$$Tr(listaOstatak)$$

Baze znanja u novije vrijeme nalaze primjenu u sustavima utemeljenima na znanju, odnosno agentima. Višeagentni sustavi (VAS)[†], primjer su praktične primjene ontologija, i upravo zbog toga i izvrsna motivacija za njihov razvoj, budući da pripadaju danas popularnom području *umjetne inteligencije*. Zbog toga ću malo detaljnije opisati pojam **agenta** i **višeagentnog sustava**.

3.4. Analiza nekih kataloga uzoraka dizajna ontologija

Uzorci dizajna ontologija nisu tako dobro klasificirani i pročišćeni, kao što je to slučaj sa uzorcima u softverskom inženjerstvu. Postoje nekoliko izvora, odnosno javnih kataloga takvih uzoraka a tri najbolja primjera su:

- NeON projekt:
  `http://ontologydesignpatterns.org/wiki/Ontology_Design_Patterns_._org_%28ODP%29`
- Javni katalog uzoraka dizajna ontologija Univerziteta u Manchesteru:
  `http://www.gong.manchester.ac.uk/odp/html/`
- SourceForge katalog uzoraka dizajna ontologija namijenjen domeni znanja u biologiji:
  `http://odps.sourceforge.net/odp/html/index.html`
- Najbolje prakse semantičkog weba - W3C radna grupa za razvoj semantičkog weba
  `https://www.w3.org/2001/sw/BestPractices/`

Neću se baviti svim dostupnim katalozima, jer poneki sami obimom premašuju razuman prostor u ovom radu, već ću izabrati uzorke SourceForge katalog uzoraka otvorenih biomedicinskih ontologija, uzorke vezane uz agente te uzorke NeON projekta.

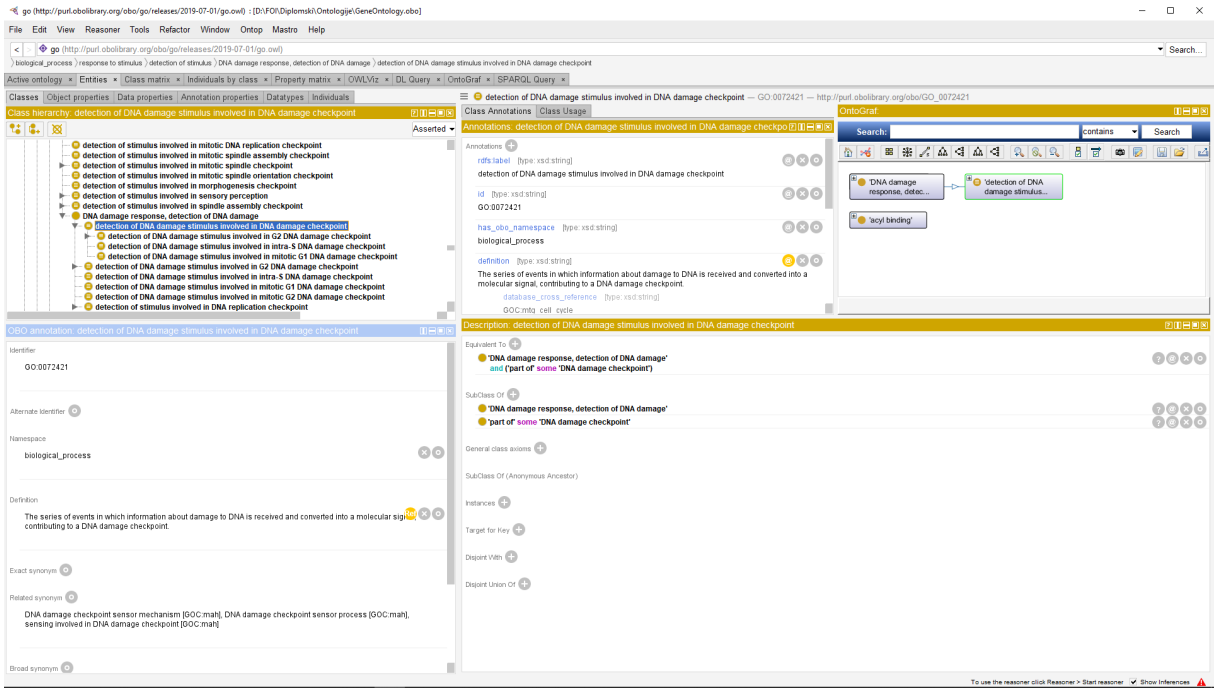
3.4.1. Otvorene biomedicinske ontologije i jezik OBO

BMC Bioinformatics grupa koja svoje uzorke dizajna za bio-ontologije objavljuje na SourceForge stranicama (`http://odps.sourceforge.net/odp/html/index.html`), koristi samo tri obitelji uzoraka: prošireni uzorci dizajna ontologija, uzorci dobre prakse i uzorci modeliranja domene. Njihovi su uzorci dobro razrađeni i temeljito dokumentirani. Pored OWL jezika za reprezentaciju znanja, koriste i OBO (eng. *Open Biomedical Ontologies*) Flat File Format koji je izvorno razvijen za ontologiju gena (eng. *Gene Ontology*, GO)(Golbreich i dr., 2019).

[†] U upotrebi je akronim MAS (eng. *Multi Agent System*)

Otvorene biološke i medicinske ontologije (OBO) je zajednica razvojnih inženjera dijele svoje napore kako bi stvorili kontrolirane vokabulare na području biomedicinskih znanosti koji su logički dobro oblikovani i sa znanstvenog stanovišta točni. Zajednica okuplja volontere koji svoj rad baziraju na konceptima otvorenog korištenja, zajedničkog razvoja, te nepreklapajućeg sadržaja sa striktnim opsegom, pri čemu se inzistira na zajedničkoj sintaksi i relacijama baziranima na kvalitetnim ontologijama. Zajednicu nadzire Operativni komitet koji uključuje uredništvo te grupe za tehničku i korisničku podršku. OBO Foundry koordinira svoj rad koristeći GitHub sustav za verzioniranje na adresi `https://github.com/OBOFoundry/OBOFoundry.github.io` te održava web stranice na `http://www.obofoundry.org/`.

OBO Flat File oblik zapisivanja se skraćeno naziva samo OBO, ali ga ne treba miješati sa repozitorijem OBO biomedicinskih ontologija. Alati poput Protégé-a mogu čitati OBO zapise, a imaju i prikaz OBO anotacija (slika 23).



Slika 23: Gene Ontology (GO) zapisana u OBO KR i otvorena u Protégé-u

OBO zapis koristi term oznake (*[Term]*) kao reprezentante koncepata odnosno klasa. Za objektna svojstva se koriste typedef oznake (*[Typedef]*). OBO *is-a* oznaka se u OWL-u prevodi kao *rdfs:subClassOf* ako se odnosi na klase odnosno *rdfs:subPropertyOf* ako se odnosi na svojstva. U OBO jeziku svojstva mogu biti:

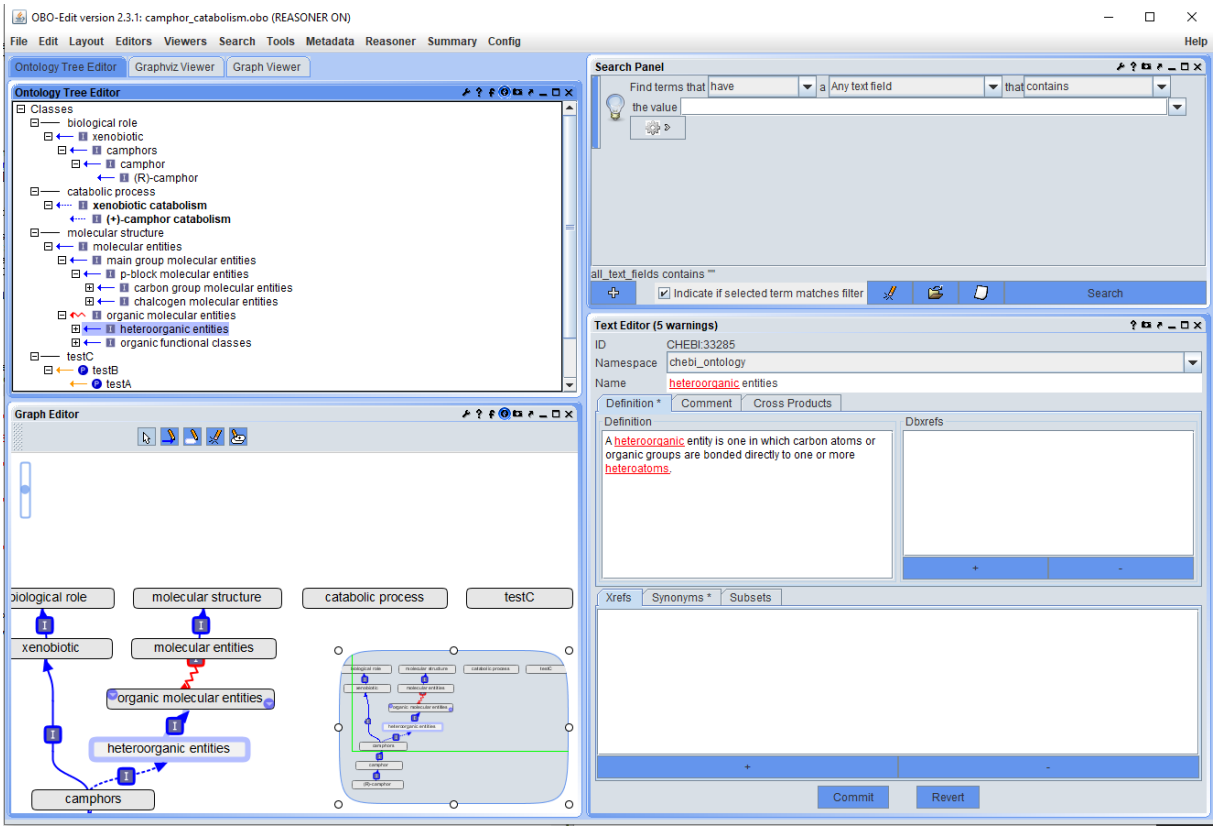
- is-a - podsvojstvo, kao i kod klasa
- ciklička - oznaka *is_cyclic*
- tranzitivna - oznaka *is_transitive*
- simetrična - oznaka *is_symmetric*

Razdvojene klase se označavaju oznakom *disjoint_from*, a domena i doseg svojstava oznakama *domain* i *range*. Primjer zapisa terma GO_0072421 iz ontologije gena dan je u listingu 3.10.

Listing 3.10: Primjer OBO terma

```
[Term]
id: GO:0072421
name: detection of DNA damage stimulus involved in DNA damage checkpoint
namespace: biological_process
def: "The series of events in which information about damage to DNA is received and
    converted into a molecular signal, contributing to a DNA damage checkpoint." [
    GOC:mtg_cell_cycle]
synonym: "DNA damage checkpoint sensor mechanism" RELATED [GOC:mah]
synonym: "DNA damage checkpoint sensor process" RELATED [GOC:mah]
synonym: "sensing involved in DNA damage checkpoint" RELATED [GOC:mah]
is_a: GO:0042769 ! DNA damage response, detection of DNA damage
intersection_of: GO:0042769 ! DNA damage response, detection of DNA damage
intersection_of: part_of GO:0000077 ! DNA damage checkpoint
relationship: part_of GO:0000077 ! DNA damage checkpoint
created_by: midori
creation_date: 2010-12-08T04:16:04Z
```

Ekvivalentni OWL zapis dan je u prilogu na stranici 142. Biološke ontologije predstavljaju veliki izazov klasičnim Tableau zaključivačima (FaCT++, Pellet i HermiT) zbog mnoštva cikličkih definicija u ontološkim termima. Posebno za OBO ontologije je na odjelu za bioinformatiku sveučilišta Berkeley razvijen OBO-Edit alat otvorenog kôda koji je opremljen i svojim zaključivačem. Projekt je financiran od strane Konzorcija za ontologiju gena (eng. *Gene Ontology Consortium*).



Slika 24: OBO-Edit uređivač razvijen na Berkeley Bioinformatics kao projekt otvorenog koda

3.4.2. SourceForge uzorci dizajna ontologija

Kao što je već ranije rečeno, uzorci dizajna ontologija na SourceForge stranicama izvrsno su dokumentirani. No jednako tako moguće je preuzeti arhivu koja sve uzorke zajedno sa kataloškim informacijama može prikazati u html i LaTeX obliku. Arhiva sadrži i generatore html i LaTeX dokumenata napisane u Javi. Svi uzorci su također izraženi u jeziku OWL, a katalozi u XML-u. Svaka grupa ima svoj katalog. I premda su sa projektnog stanovišta povezani sa određenom domenom, ovo su bazični uzorci vrlo široke primjene. Izvrsni su primjeri u edukativnom smislu, te se mogu koristiti na isti način kao i *Ontologija o pizzama* kao školski primjeri. Arhiva sadrži u ovom trenutku 17 uzoraka svrstanih u ranije spomenute 3 grupe:

- Prošireni uzorci dizajna ontologija - 3 uzorka
  - Iznimka (eng. *Exception*), modeliranje posebnih slučajeva bez razbijanja hijerarhije klasa odnosno narušavanja konzistentnosti modela unesene ontologije.
  - n-arna objektna relacija (eng. *Nary Relationship*), izražavanje odnosa s više od jednog elementa. OWL ne dozvoljava povezivanje nekog entiteta sa dva ili više drugih entiteta izravno.
  - n-arna podatkovna relacija (eng. *Nary DataType Relationship*), reprezentira podatkovnu vrijednost sa više od jednog stanovišta odnosno komponenti.
- Uzorci dobre prakse - 9 uzoraka

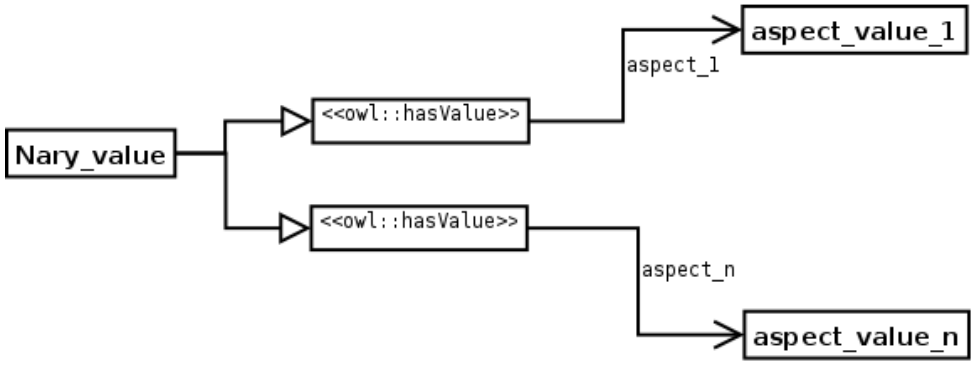
  - Entitet-Svojstvo-Vrijednost (eng. *Entity-Feature-Value*)
  - Izbornik (eng. *Selector*)
  - Normalizacija (eng. *Normalisation*)
  - Ontologija više razine (eng. *Upper Level Ontology*)
  - Zatvaranje (eng. *Closure*)
  - Entitet-Kakvoća (eng. *Entity-Quality*)
  - Particija vrijednosti (eng. *Value Partition*)
  - Entitet-Svojstvo-Kakvoća (eng. *Entity-Property-Quality*)
  - Definirani opis klase (eng. *Defined Class Description*)

- Uzorci modeliranja domene - 5 uzoraka
  - Međudjelovanje ovisno o ulozi učesnika(eng. *Interactor Role Interaction*)
  - Slijed (eng. *Sequence*)
  - Niz povezanih svojstava (eng. *Composite Property Chain*)
  - Popis (eng. *List*)
  - Prilagođeni uzorak Strukture-Entiteta-Dijela (eng. *Adapted SEP*)

Katalog uzoraka sadrži naziv, klasifikaciju, motivaciju, cilj, elemente, opis implementacije, posljedice, povezane uzorke, reference i URI adresu te grafički strukturalni prikaz i prikaz jednog primjera koji koristi određeni uzorak. Slični elementi kataloga su prisutni i u katalozima uzoraka dizajna u softverskom inženjerstvu.

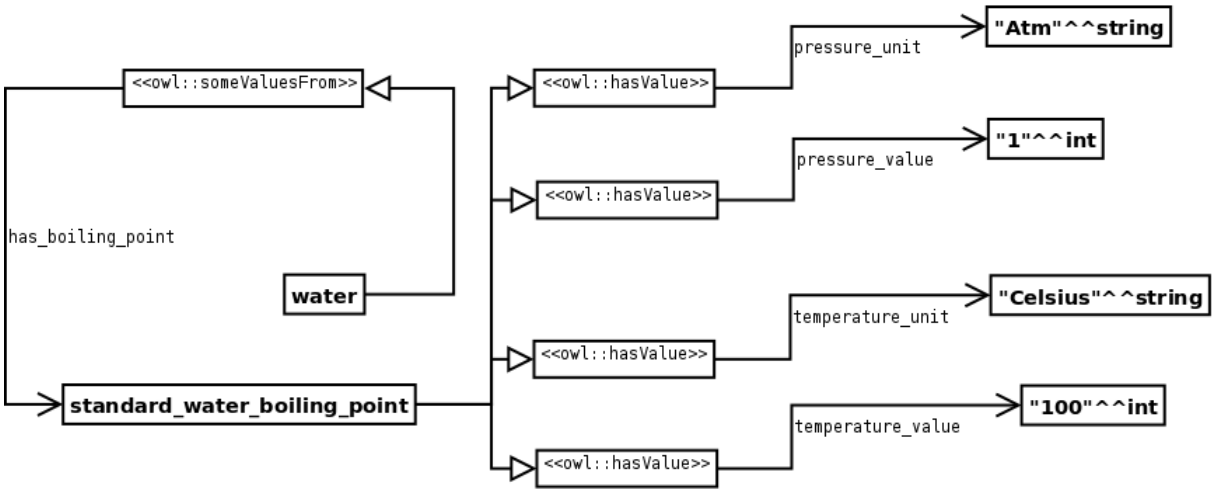
Primjer stavke u katalogu za uzorak Nary DataType Relationskip (katalog je dostupan i on line na `http://odps.sourceforge.net/odp/html/index.html`):

Naziv: Nary DataType Relationship (n-arna podatkovna relacija)
Klasifikacija: Proširenje
Motivacija: Numeričke vrijednosti mogu imati različita stanovišta.
Cilj: Prikazati podatkovnu vrijednost sa više od jednog stanovišta.
Struktura:



-

Primjer:



-

Elementi: Izvorna vrijednost je reificirana (dekomponirana) na sva potrebna podatkovna svojstva i vrijednosti.
Implementacija: U prvom koraku, izabire se podatkovna vrijednost koja će se reificirati i kreira se njegova klasa. Klasi se dodaje ograničenje te sva potrebna podatkovna svojstva i ograničenja reificirane klase.
Rezultat: Nakon reifikacije, vrijednost različitih aspekata se prikazuje u ontologiji.
Reference:

- `http://www.cs.man.ac.uk/~stevensr/menupages/ontologies.php`
- Bijan Parsia and Michael Smith. Quantities in OWL. OWLed 2008 EU

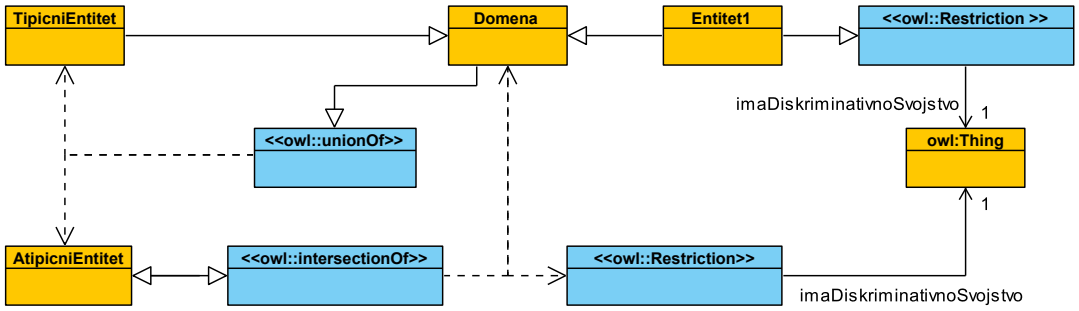
URL: `http://odps.sourceforge.net/odp/owl/Extension_ODP/Nary_DataType_Relationship.owl`

Bez navođenja elemenata kataloga svakog od 17 uzoraka zbog preopsežnosti i redundantnosti, opisat ću te uzorke u kraćem obliku. Sve uzorke izvodim u Protégéu, uz shematski prikaz prilagođen konkretnoj izvedbi, a OWL fragmente zapisa u Turtle obliku navodim u prilogu. Izvedeni uzorci nisu kopija OWL izvedbe iz repozitorija na SourceForge stranicama već su izvedeni kao minimalistički primjer bez kontekstualnih klasa ali i dalje usklađeni sa strukturom navedenom u katalogu.

3.4.2.1. Prošireni uzorci dizajna ontologija

Iako postoje strogo formalni postupci vrednovanja ontologija poput *OntoClean* metode koja koristi o domeni neovisna metasvojstva klasa poput identiteta, jedinstva, rigidnosti i ovisnosti, za sustavno razvijanje konzistentnih, potpunih i neredundantnih taksonomija, ponekad

je potrebno omogućiti i iznimke. Ovo je važno ukoliko se ne želi narušiti postojeća kanonska norma, osobito po pitanju postojećih klasifikacija, a što zahtijeva uklapanje novog sadržaja. Tada se koristi uzorak iznimke. Rezultat sa stanovišta logike nije konzistentan, stvara neintuitivne strukture koje je teško održavati, ali može barem privremeno zadovoljiti potrebe.

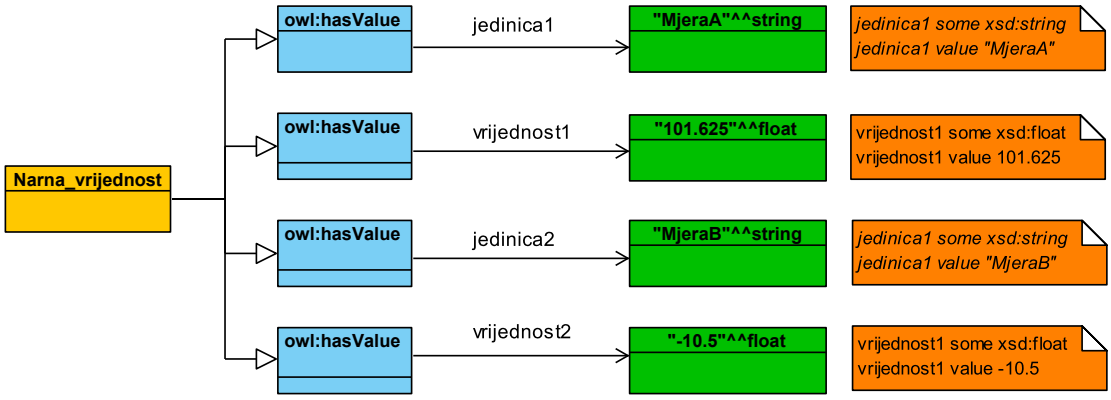


Slika 25: Uzorak iznimke

U uzorku iznimke stvaraju se razdvojene klase tipičnih i atipičnih elemenata, te se uvodi diskriminativno svojstvo pridruženo iznimkama (u dijagramu klasa *Entitet1*). Koristi se i aksiom pokrivanja prema kojem svi elementi moraju pripadati jednoj od tih klasa (*TipicniEntitet* ili *AtipicniEntitet*), dok je klasa atipičnih entiteta presjek svih entiteta (*Domena*) i ograničenja po diskriminativnom svojstvu. Drugim riječima klasa atipičnih entiteta sadrži takve entitete za koje vrijedi diskriminativno svojstvo. Bez obzira što iznimke nisu klasificirane u hijerarhiju atipičnih entiteta, zaključivač će na osnovu diskriminativnog svojstva od unesene hijerarhije izgraditi ispravnu zaključenu hijerarhiju sa ispravno klasificiranim tipičnim i atipičnim entitetima na svakoj razini. Potrebna DL ekspresivnost: $\mathcal{ALCN}$

U fizici, ali i općenito prirodnim znanostima, fizikalne veličine se izražavaju umnoškom brojne veličine i jedinice mjere koja može biti jednostavna ili složena. Na primjer, vrijednost Planckove konstante se može iskazati kao $h = 6,62607 \cdot 10^{-34} J \cdot s$ gdje je $J \cdot s$ ili češće Js oznaka za složenu mjernu jedinicu - *Joule sekundu*. Vrlo često se u rezultatima mjerenja iskazuju i slučajne pogreške mjerenja i to apsolutne, relativne i standardne pogreške uz aritmetičku sredinu kod neovisnih veličina, te srednja kvadratna pogreška uz najvjerojatniju vrijednost veličine kod posredno određenih (izračunatih) fizikalnih veličina. Primjer takvog iskaza je NIST-ova[‡] vrijednost Planckove konstante $h = (6,62606983 \pm 0,00000022) \cdot 10^{-34} J \cdot s$, gdje je iznos $0,00000022 \cdot 10^{-34}$ standardna pogreška mjerenja i odgovara navedenoj relativnoj standardnoj pogrešci od $34 \cdot 10^{-9}$ (Haddad i dr., 2016), ili kraće $h = 6,62606983(22) \cdot 10^{-34} J \cdot s$.

[‡]NIST je akronim za Nacionalni institut za standarde i tehnologiju (eng. *National Institute of Standards and Technology*). NIST je laboratorij za fiziku i neregulatorna agencija Ministarstva trgovine Sjedinjenih Američkih Država. Različite izmjerene fizikalne veličine mogu se pronaći na njihovoj stranici https://physics.nist.gov/cuu/Constants/index.html .



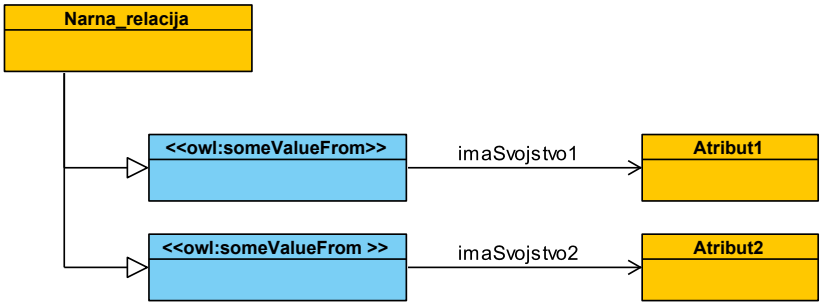
Slika 26: Uzorak n-arne podatkovne relacije

Očito nije dovoljno određenim veličinama pridružiti samo jednu podatkovnu vrijednost. Jednako tako, neke veličine mogu imati više aspekata, odnosno pogleda. Vrelište vode nema jednaku temperaturu pri svim tlakovima. Tako veličini mora biti pridružena i jedna ili više vrijednosti pri kojima je osnovna vrijednost ispravna. Kako bi se omogućilo pridruživanje više podatkovnih vrijednosti određenim entitetima, koristi se uzorak *n-arnih podatkovnih relacija*. Ovdje treba naglasiti da u F-logici n-arne relacije nisu potrebne, budući je tamo dozvoljeno skupovno pridruživanje. Sve vrijednosti se mogu pridružiti kao jedinstven skup. OWL tom ograničenju pribjegava upotrebom ovog uzorka. Potrebna DL ekspresivnost: $\mathcal{ALEF}^{(\mathcal{D})}$

Pored n-arnih podatkovnih relacija postoji i uzorak n-arnih objektnih relacija. Ovdje određenom elementu nije pridruženo više literala tj. podatkovnih vrijednosti, nego više drugih elemenata koji pripadaju različitim klasama. Ako želimo iskazati "X imaSvojstvo Y specifičnosti Z", to je nemoguće opisati trojkama doslovno jer je svaka izjava binarna relacija. Sama svojstva u RDF-u i OWL-u su binarna. Uvodi se nova klasa S pa se iskaz oblikuje kao "X imaSvojstvo S koje imaNaziv Y i imaSpecifičnost Z". Ovakav postupak posrednog povezivanja jedne individue sa više drugih individua se naziva *reifikacija*[§]. Prilikom reifikacije, relacije se zamjenjuju entitetima koji omogućavaju više dodatnih informacija za razliku od relacija. Na primjer "X pripada Y" se može zamijeniti entitetom *Pripadnost* koji onda može povezivati individuu *x* na način "pripadnost1 imaOsobu x" i individuu *y* na način "pripadnost1 imaDruštvo y". Na ovaj način se omogućuje višestruka pripadnost dodavanjem npr. "pripadnost1 imaDruštvo z". Sada je individua *x* povezana i sa *y* i sa *z* što kod binarnih relacija nije moguće. Reifikacija se često koristi za opise vjerovanja u druge izjave. Nova klasa *Vjerovanje* tako može imati objekt *vjerovanje1* koji onda ima subjekt, predikat i objekt.

Jednako se tako može koristiti i navođenje. Na primjer: "X tvrdi da Y ima svojstvo Z" se može predstaviti novim entitetom *Tvrdnja* i onda reći za objekt x "x imaTvrdnju tvrdnja1", "tvrdnja1 ima subjekt y", "tvrdnja1 ima predikat imaSvojstvo" i "tvrdnja1 imaObjekt Z".

[§]U engleskom jeziku riječ *reify* ima značenje konkretizacije nekog apstraktnog pojma, opredmećivanja.

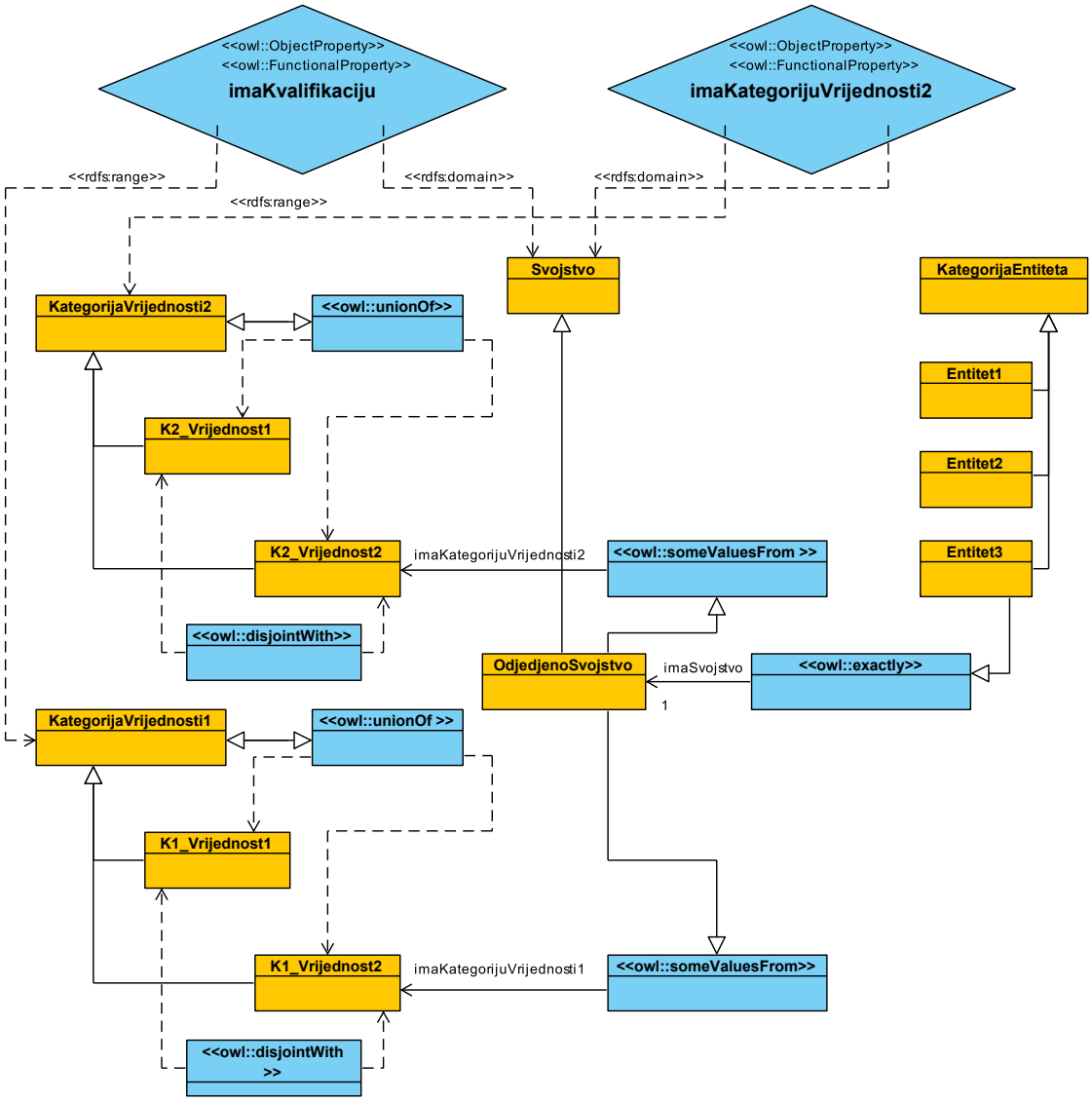


Slika 27: Uzorak n-arne objektne relacije

Prema ovom dijagramu, uzorku se reificira nova klasa (*Narna_relacija* u dijagramu) a elementi se povežu s njom odgovarajućim relacijama. Sada je moguće preko tog uzorka svakom entitetu omogućiti po volji veliki broj relacija kao zamjenu za jednu n-arnu relaciju. Potrebna DL ekspresivnost: $\mathcal{EL}^{++}$ (Horridge i dr., 2012), odnosno $\mathcal{ALC}$ prema Protégéovom mjeraču ekspresivnosti.

3.4.2.2. Uzorci dobre prakse

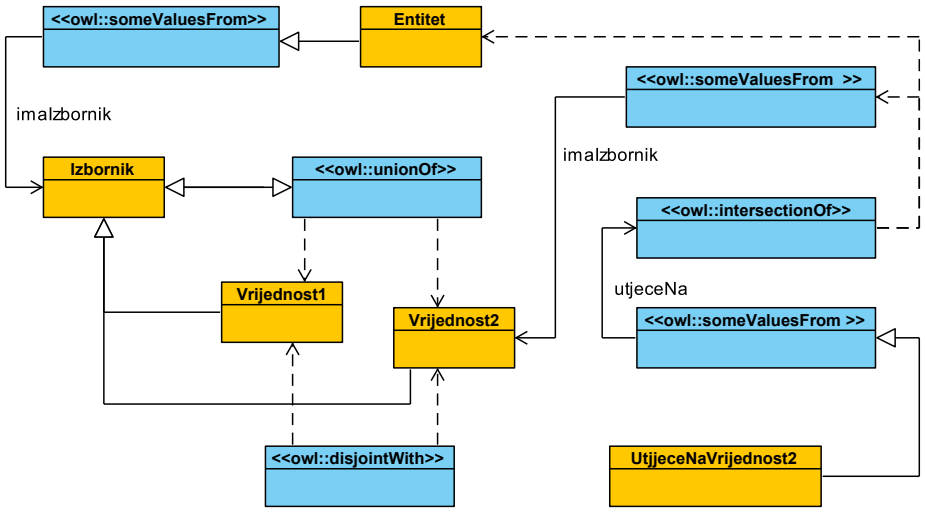
Uzorak **entiteta-svojstva-vrijednosti** (eng. emphEntity-feature-value) pridružuje entitetima svojstva koja za svaki svoj aspekt imaju particiju vrijednosti i određeno objektno svojstvo. Ovo je složeni uzorak koji povezuje uzorke particije vrijednosti i uzorak n-arnih relacija.



Slika 28: Uzorak entiteta-svojstva-vrijednosti

Uzorak omogućava pravilno razdvajanje svojstava entiteta i aspekata vrijednosti. Popratni efekt korištenja ovog uzorka je njegova kompleksnost. Ontologije koje imaju mnogo aspekata a svaku kvalifikaciju, teško je održavati. Potrebna DL ekspresivnost: $\mathcal{ALCQ}$

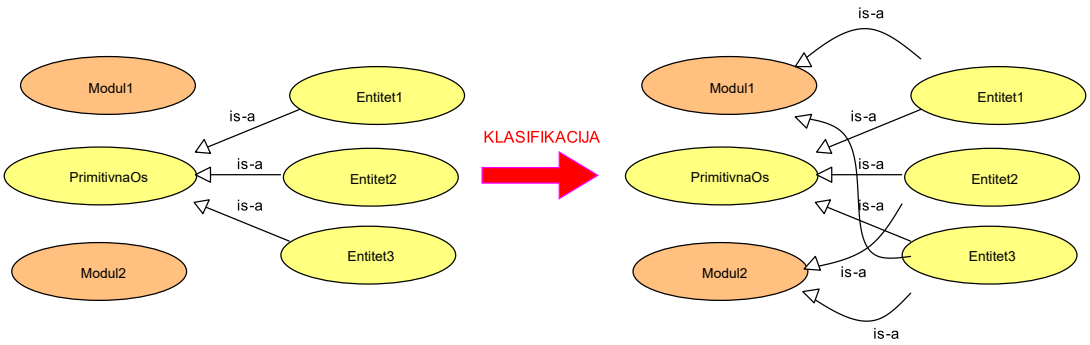
Uzorak **izbornika** obično se koristi u raznim simetrijama kako bi se smanjila veličina ontologija, budući da se simetrični entiteti ne moraju redundantno opisivati. Uzorak se obično ugrađuje u već postojeće ontologije i dodatno precizira svojstvo vezano uz simetriju. Primjerice, ontologija ne mora imati poseban opis za lijevu i desnu ruku, već samo za ruku.



Slika 29: Uzorak izbornika

U dijagramu na slici 29 Entitet koji može imati vrijednosti iz *Vrijednost1* ili *Vrijednost2* što odgovara primjeru lijeve ili desne ruke. *Utječe naVrijednost2* ne bitna informacija koja se odnosi na entitet, a tek se izbornikom specificira na koji se entitet prema vrijednosti misli. Potrebna DL ekspresivnost: $\mathcal{ALHCF}$ (Horridge i dr., 2012) odnosno $\mathcal{ALCF}$ za konkretnu izvedbu u Protégéu.

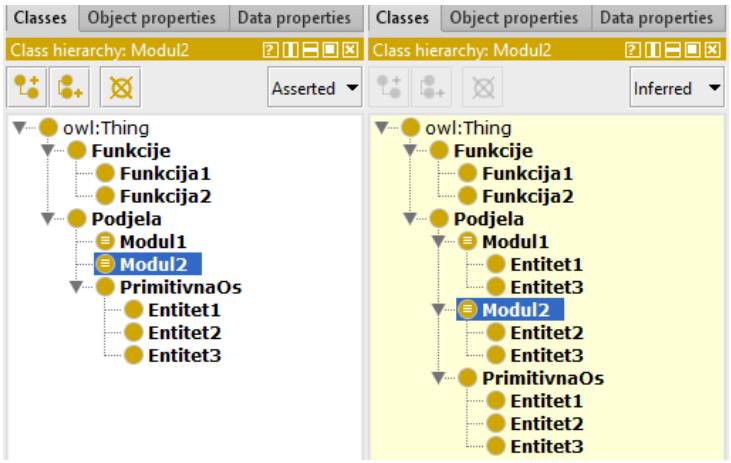
Normalizacija je uzorak koji olakšava modeliranje ontologija sa višestrukim nasljeđivanjem, odnosno višestrukim hijerarhijama. Slično kao i kod uzorka iznimke i ovdje se nastoji olakšati izgradnju i održavanje ontologija. Unesena višestruka hijerarhija zahtijevala bi mnogo posla pri svakoj promjeni klasa, a sve to je povezano sa velikom vjerojatnošću da se u ručnom postupku naprave pogreške. Ukoliko je hijerarhija sagrađena na implicitnoj semantici, drugi zaključivači ili razvojni inženjeri teško mogu vidjeti zbog čega su klase povezane u takvu hijerarhiju. Stoga se pored odnosa klasa - potklasa koriste i ograničenja. Zaključivač će na osnovu tih ograničenja stvoriti ispravnu višestruku hijerarhiju klasa.



Slika 30: Uzorak normalizacije

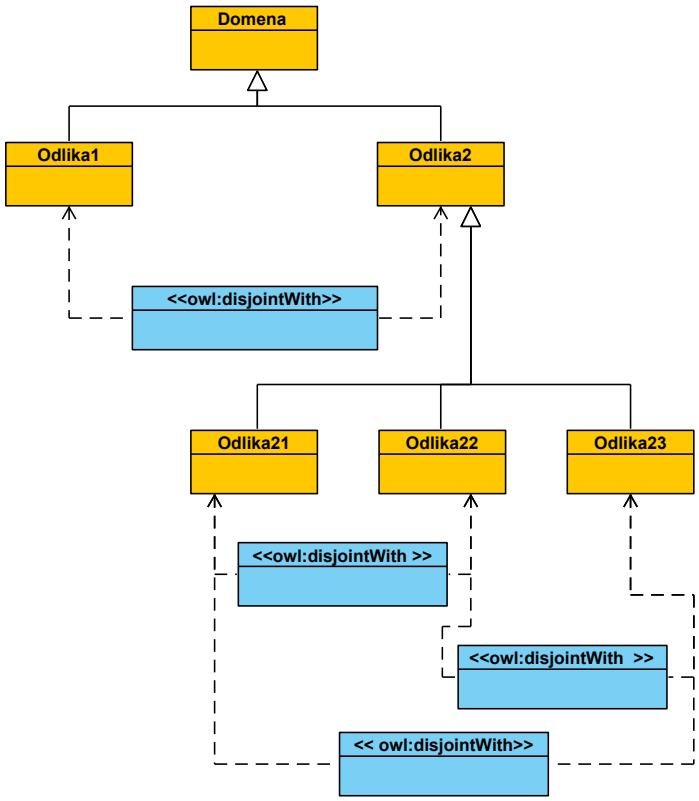
U alatu Protégé zaključivač će uključenjem opcije Inferred presložiti unesenu u zaključenu hijerarhiju. Na slici 31 *Modul1* sadrži instance koje obavljaju funkciju *Funkcija1* a *Modul2*

instance koje obavljaju funkciju *Funkcija2*. Entiteti će pripadati pojedinom modulu ako obavljaju jednu od funkcija ili obje. Na desnoj strani je zaključivač stvorio novu hijerarhiju, po kojoj *Entitet1* pripada *Modulu1*, *Entitet2* pripada *Modulu2*, dok *Entitet3* koji obavlja obje funkcije pripada objema entitetima. Potrebna DL ekspresivnost: $\mathcal{ALE}$



Slika 31: Automatska klasifikacija u alatu Protégé

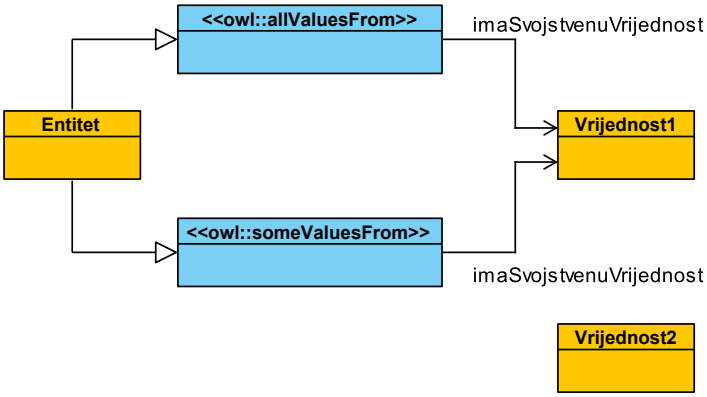
Ontologije više razine često sadrže većinu koncepata potrebnih pri izgradnji novih ontologija niže razine. Kako se ne bi ponovno kreirali, takvi koncepti se mogu preuzeti iz viših ontologija. Pri integraciji ontologija koristi se sličan pristup kao i pri nasljeđivanju klasa. Klase u ovom uzorku predstavljaju konceptualne ontologije.



Slika 32: Uzorak ontologije više razine

Uzorak **ontologija više razine** (eng. *Upper level ontology*) sve klase a time i ontologije iste razine naslijeđene od ontologija više razine, smatra razdvojenima. Popratni efekt korištenja ovog uzorka vodi na kontroverze, budući da ontologije više razine ne moraju pratiti specifično viđenje domenskog znanja potrebno ontologijama niže razine. Potrebna DL ekspresivnost: $\mathcal{ALC}$

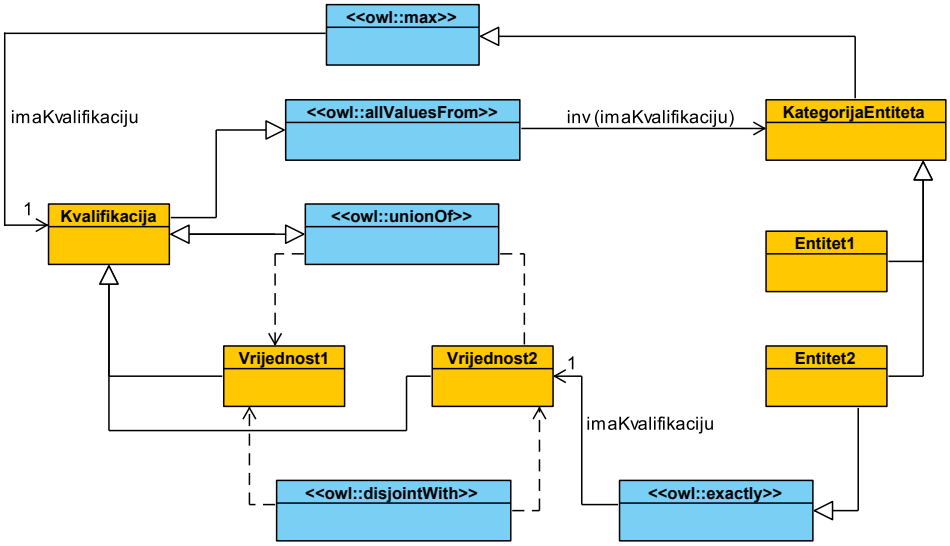
OWL koristi pretpostavku otvorenog svijeta. To znači da ukoliko neka informacija nije poznata, odnosno nije sadržava u bazi znanja, ona se ne proglašava neistinitom. Kada u ontologiji instancama neke klase pridružimo neke instance druge klase, to ne znači da im nisu pridružene i instance neke treće klase, budući da to nije specificirano.



Slika 33: Uzorak zatvaranja

Kako bismo zaključivaču dozvoliti da isključi bilo kakve poveznice sa instancama drugih klasa osim navedene, provodimo zatvaranje. Drugim riječima, osnovnom ograničenju (*owl:someValuesFrom*) dodajemo još jedno, po kojem su instance nekog entiteta povezane samo (isključivo) sa instancama drugog entiteta (*owl:allValuesFrom*). Potrebna DL ekspresivnost: $\mathcal{ALC}$

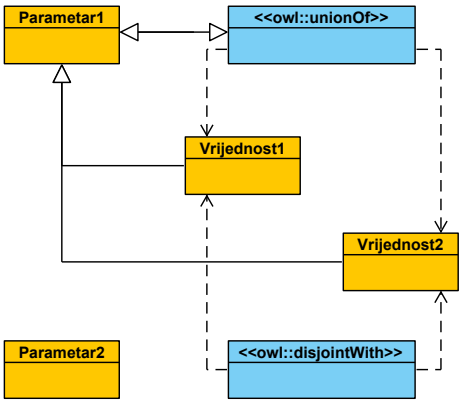
Uzorak **entiteta-kvalifikacije** (eng. *Entity-Quality*) pridružuje entitetima kvalifikacije koje se sastoje od razdvojenih vrijednosti. Ovisno od kardinalnosti svojstva *imaKvalifikaciju* koja može biti 1 ili najviše 1, entiteti mogu imati obaveznu ili neobaveznu kvalifikaciju. Kvalifikacije koje su po svojoj prirodi neovisne, ovim se uzorkom čine ovisnima.



Slika 34: Uzorak entiteta-kvalifikacije

Inverzno svojstvo *inv (imaKvalifikaciju)* tj. *jeKvalifikacijaOd* onemogućava pridruživanje nekih drugih entiteta kvalifikacijama. Svaki entitet povezan je sa samo jednom kvalifikacijskom vrijednošću.
Popratni učinak upotrebe ovog uzorka je zahtjevno modeliranje hijerarhije podkvalifikacija i nemogućnost korištenja za višeaspektne kvalifikacije. Potrebna DL ekspresivnost: $\mathcal{ALCIQ}$

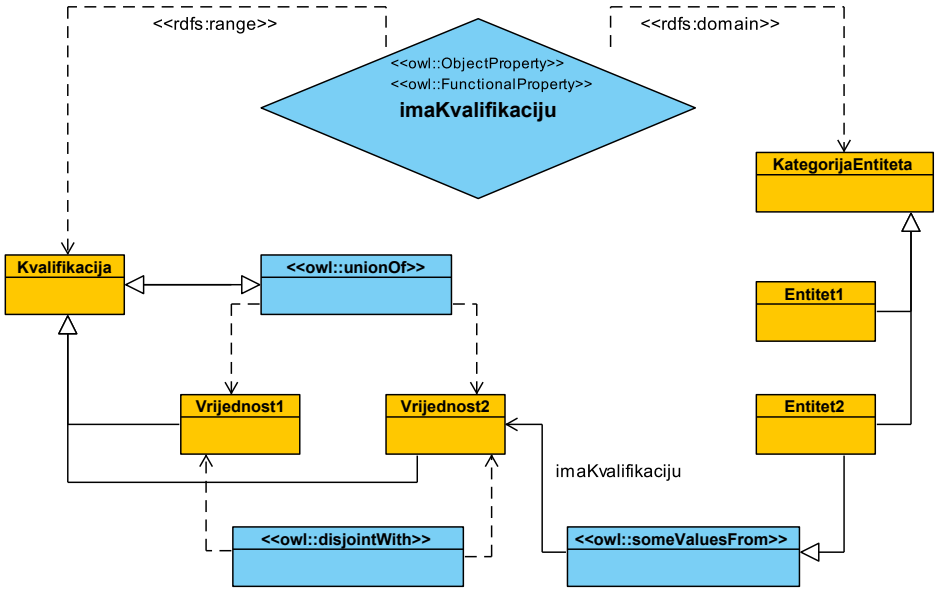
Uzorak **particija vrijednosti** (eng. *Value partition*) jedan je od važnih uzoraka, i ima vrlo široku primjenu. Sličan je prebrojivim klasama koje rade sa individuama, no ovaj uzorak je primarno namijenjen particiji klasa. Ukoliko neki parametar može imati nekoliko vrijednosti, i te su vrijednosti razdvojene (individue ne mogu istovremeno imati više od jedne vrijednosti) i zatvorene (individue mogu imati samo te vrijednosti), tada je potrebno koristiti ovaj uzorak.



Slika 35: Uzorak particije vrijednosti

U uzorku vrijednosti su razdvojene i naslijeđene od klase *Parametar1*. Istovremeno, klasa *Parametar1* je ekvivalentna uniji svih vrijednosti. Potrebna DL ekspresivnost: $\mathcal{ALCF}$ (Horridge i dr., 2012), odnosno $\mathcal{ALC}$ za konkretnu izvedbu u Protégéu.

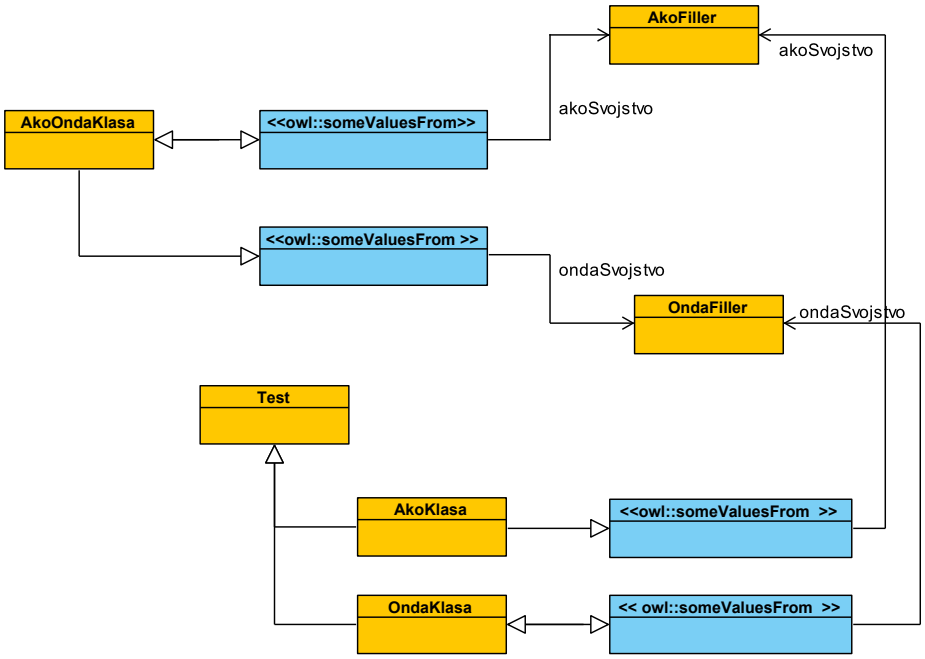
Uzorak **entiteta-svojstva-kvalifikacije** (eng. *Entity-property-quality*) sličan je uzorku *entiteta-kvalifikacije* s tom razlikom što nema stroge povezanosti kvalifikacije i kategorije entiteta. Poveznica je samo preko domene i dosega koji je određen kategorijom entiteta i kvalifikacijom. Uzorak profinjuje neovisne entitete slično uzorku **izbornika**.



Slika 36: Uzorak entitet-svojstvo-kvalifikacija

Uzorak omogućava točnu specifikaciju koje kvalifikacije su povezane sa kojim entitetom. Uzorak se ne može koristiti kod višeaspektnih entiteta. Potrebna DL ekspresivnost: $\mathcal{ALCF}$

Opis definirane klase (eng. *Defined class description*) je uzorak koji omogućava kreiranje *ako-onda* struktura pomoću OWL DL ekspresivnosti. Prema dijagramu na slici 37 a na osnovu promatranog uzorka, ako klasa *AkoOndaKlasa* zadovoljava uvjete određene izjavom koju čine *akoSvojstvo* i *AkoFiller*, tada ona ima *ondaSvojstvo* vezano uz određeni *OndaFiller*. Dodatne klase u testu slijede osnovni uzorak. Svaka klasa koja se nadoveže na uzorak mora ispunjavati ako uvjet da bi za nju vrijedilo onda svojstvo. Razlog tome je što će zaključivač svaku klasu koja zadovoljava uvjet poistovjetiti sa *AkoOndaKlasom*, a ona je u zaključenoj hijerarhiji podklasa ograničenja *ondaSvojstvo some OndaFiller*, pa će i nova klasa biti podklasa tog ograničenja, tj. imat će *OndaFiller* atribut. Potrebna DL ekspresivnost: $\mathcal{EL}^{++}$ (Horridge i dr., 2012), odnosno $\mathcal{ALE}$ za konkretnu izvedbu u Protégu.

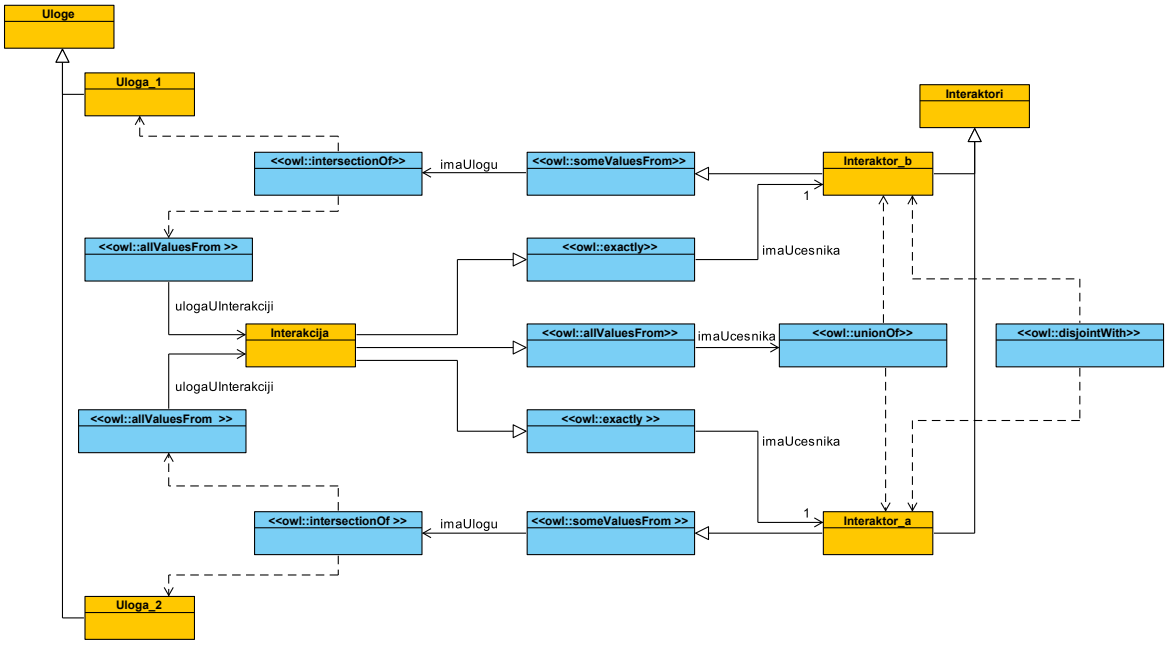


Slika 37: Uzorak opisa definirane klase

3.4.2.3. Uzorci modeliranja domene

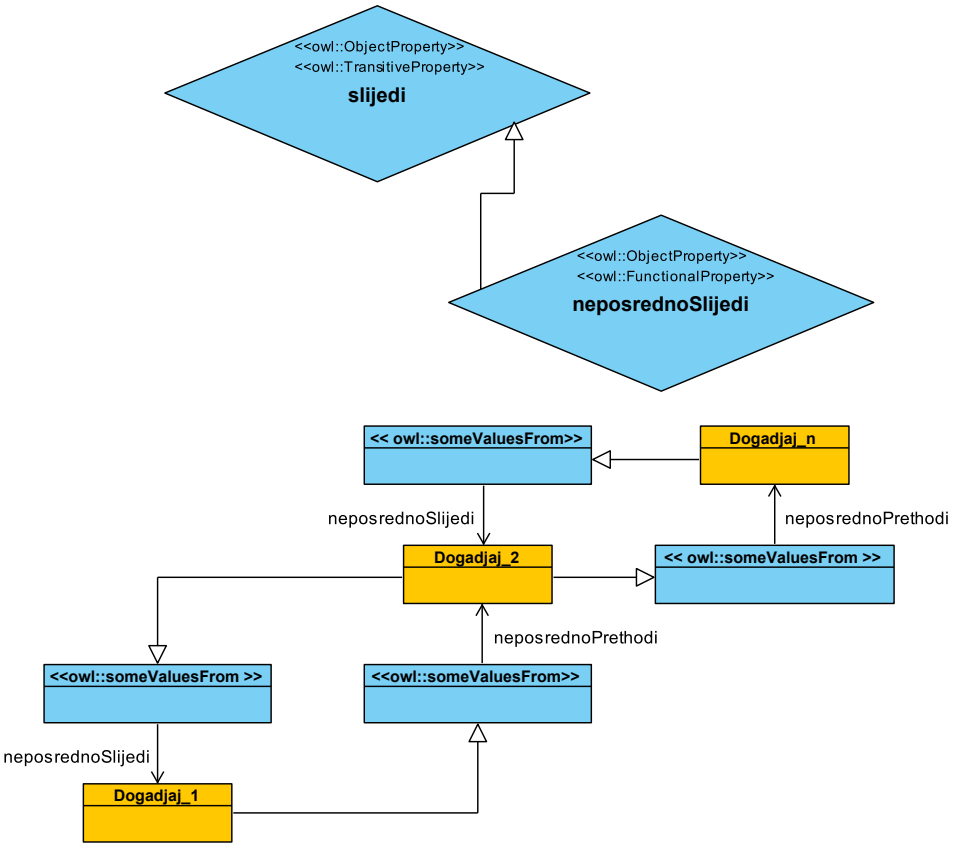
Domenskih uzoraka obično ima veliki broj. No u slučaju biomedicinskih uzoraka svedeni su samo na njih 5. Prvi među njima je uzorak **međudjelovanja ovisnog o ulozi sudionika** (eng. *Interactor Role Interaction*), a u biomedicinskim istraživanjima često se koristi za modeliranje interakcije proteina. Ipak, uzorak je moguće primijeniti i u drugim domenama, uključujući i poslovno okruženje i informacijske znanosti gdje interakcija često ovisi o ulozi korisnika. Tu se prije svega misli na različita prava pristupa aplikacijama, bazama podataka i slično.

Uzorak omogućava modeliranje interakcija, pri čemu sudionici imaju različite uloge i međusobno su razdvojeni. Interakcija je zadužena i za aksiome zatvaranja prema sudionicima. Tako određena interakcija ima određeni skup sudionika, i niti jedan drugi sudionik ne može sudjelovati u toj interakciji. Svaki sudionik ima ulogu koja je sastavljena od opće uloge i određene interakcije, pa je to zapravo pravo na interakciju po podređenoj ulozi. Potrebna DL ekspresivnost: $\mathcal{ALCQ}$



Slika 38: Uzorak međudjelovanja ovisnog o ulozi sudionika

U složenim procesima događaji su često povezani u sljedove. Da bi se dogodio određeni događaj treba se dogoditi prethodni događaj, da bi nakon njega uslijedio neki drugi događaj. U uzorku **slijeda** (eng. *Sequence*) nije određen redoslijed svih događanja, već se samo navodi što nekom događaju neposredno prethodi i što mu od događaja neposredno slijedi.



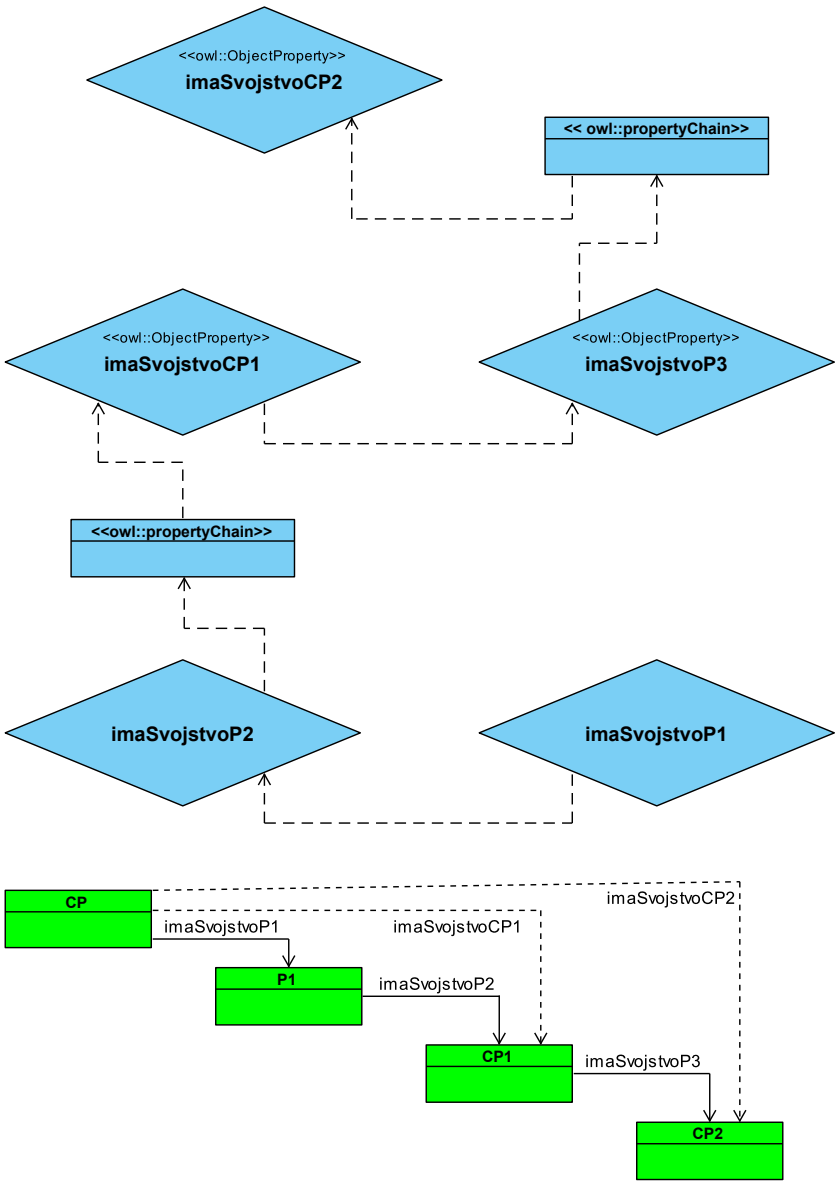
Slika 39: Uzorak slijeda (sekvence)

Ovakav uzorak, zbog tranzitivnog svojstva *slijedi* mogu koristiti upiti, kako bi dobili informaciju što se u nekom trenutku događa, što mu sve slijedi, a jednako tako i za dobivanje neposrednih prethodnika i sljedbenika. Uzorak nije pogodan za traženje posrednih prethodnika. Potrebna DL ekspresivnost: $\mathcal{SHF}$

Kompozicije svojstava vrlo su važan element OWL izražajnosti, i često se primjenjuju u ontologijama. Uzorak **niza povezanih svojstava** daje dobar primjer upotrebe kompozicije svojstava kako bi se uspostavio lanac odnosno niz povezanih svojstava. Ovo je korisno u modeliranju obiteljskih odnosa i uzastopnih modifikacija u nekim procesima. Kompoziciju svojstava nalazimo i u ontologiji o pizzama, gdje neka pizzerija prodaje pizzu koja ima neku zemlju porijekla. Ova dva svojstva se ulančavaju u jedno svojstvo - *imaAsortimanIz* na slijedeći način:

$$prodajePizzu \circ imaZemljuPorijekla \;\; subPropertyOf: \;\; imaAsortimanIz$$

Prema navedenom uzorku, dva lanca se spajaju u jedan, i to čini zaključivač.



Slika 40: Uzorak niza povezanih svojstava

Prva karika, odnosno kompozicija svojstava komponira svojstva *imaSvojstvoP1* i *imaSvojstvoP2* u novo svojstvo *imaSvojstvoCP1*. Druga karika odnosno kompozicija, komponira *imaSvojstvoP3* i ranije komponirano svojstvo *imaSvojstvoCP1* u novo svojstvo *imaSvojstvoCP2*. Nastaje lanac kompozicija

$$imaSvojstvoP1 \circ imaSvojstvoP2 \circ imaSvojstvoP3 \sqsubseteq imaSvojstvoCP2$$

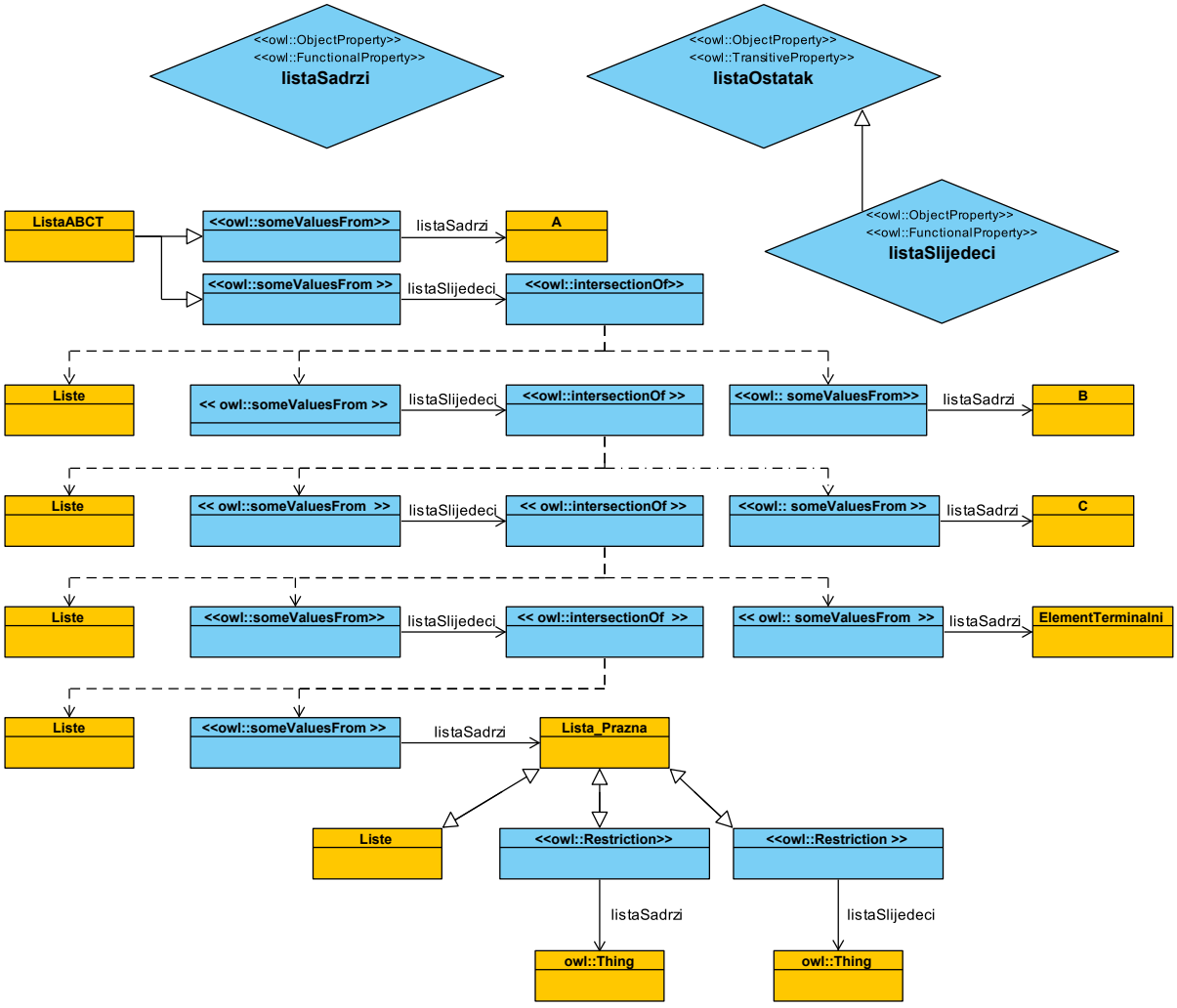
odnosno u formalizmu logike 1. reda

$$\forall x \forall w(\exists z((\exists y(imaSvojstvoC1(x,y) \wedge imaSvojstvoC2(y,z)) \rightarrow imaSvojstvoCP1(x,z)) \wedge imaSvojstvoC3(z,w)) \rightarrow imaSvojstvoCP2(z,w))$$

Način kako su individue povezane tim svojstvima i na kraju ulančanje u niz povezanih svojstava,

prikazan je na donjem dijelu dijagrama na slici 40. Potrebna DL ekspresivnost: $\mathcal{EL}^{++}$ (Horridge i dr., 2012), odnosno $\mathcal{ALCRO}$ za konkretnu izvedbu u Protégéu.

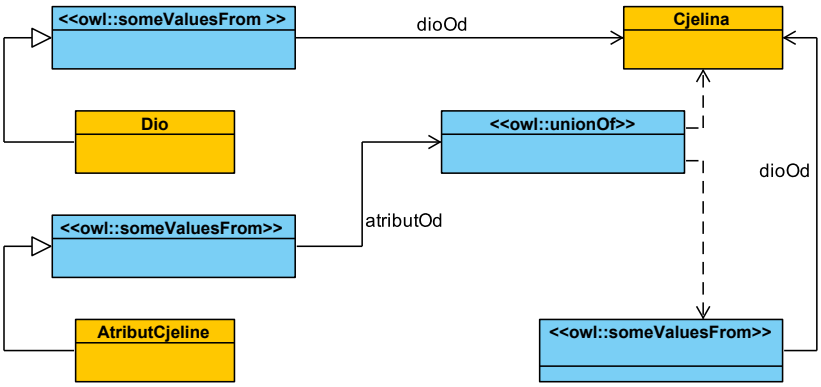
Vezane liste su jedna od najbitnijih struktura podataka, a nalazi primjenu i u ontologijama. Uzorak **liste** (eng. *List*) omogućava stvaranje vezanih listi odnosno uređene grupe elemenata. Elementi se u listi mogu ponavljati, pa su to zapravo uređeni multiskupovi. Zadnji element liste se često naziva i terminalni član. Na ovakve liste se mogu primjenjivati različiti upiti, primjerice, pronalaženje liste koja ima dva uzastopna elementa *X* ili lista koja ima terminalni član *Y* i tako dalje. U molekularnoj biologiji, osobito ontologiji gena, liste su nezaobilazne, jer se u njih mogu upisati elementi koji čine određene gene. Liste omogućavaju izgradnju moćnih klasifikacijskih sustava.



Slika 41: Uzorak vezane liste

Tri se svojstva koriste za izgradnju liste: *listaSadrzi*, *listaOstatak* i svojstvo koje ga nasljeđuje - *listaSlijedeći*. Protégé sadrži čarobnjaka za izgradnju vezanih listi. Problem kod lista je što su kompleksne i teške za održavanje, a ako su dugačke, mogu značajno usporiti zaključivače. Potrebna DL ekspresivnost: $\mathcal{SHN}$

Prilagođeni uzorak strukture-entiteta-dijela (eng. *Adapted SEP*, gdje je SEP akronim za *structure-entire-part*) služi za izgradnju *partonomija* odnosno hijerarhija dio-cjelina. Omogućava izgradnju struktura u kojima se zna u kojem djelu cjeline se nalazi neki još manji dio, odnosno modeliranje selektivne tranzitivne propagacije.



Slika 42: Prilagođeni uzorak strukture-cjeline-dijela (adapted SEP)

Uzorak je posebno koristan kada treba zaključiti lokaciju nekog dijela u velikoj strukturi. Potrebna DL ekspresivnost: $\mathcal{S}$

3.4.3. Uzorci prema katalogu NeOn projekta

Portal NeOn projekta navodi i prihvaća raniju podjelu uzoraka dizajna. Ipak, u svrhu klasifikacije i evaluacije uzoraka koje generira zajednica, koristi ponešto reduciranu podjelu:

- Uzorci sadržaja
- Uzorci reinženjeringa
- Uzorci poravnanja
- Logički uzorci
- Uzorci arhitekture
- Leksičko-sintaktički uzorci

Ovoj podjeli nedostaju uzorci imenovanja i anotacije (zbirno su to prezentacijski uzorci), te uzorci zaključivanja.
Prema tvrdnjama uredništva portala, uzorci se zaprimaju od korisnika u uvrštavaju u kataloge. Na žalost, same stranice nisu dovoljno održavane pa trenutno službeni katalozi nisu popunjeni. Usprkos tome, neslužbeni uzorci jesu oblikovani kao kataloški sadržaji. Uzorke je moguće pronaći on-line na adresi
`http://ontologydesignpatterns.org/wiki/OPTypes`.
Jednako tako, u ponešto sređenijem obliku se mogu naći u PDF *dokumentu* na adresi
`http://neon-project.org/deliverables/WP2/NeOn_2008_D2.5.1.pdf`.

3.4.3.1. Oblik kataloga NeOn uzoraka

NeON nije samo katalog i projekt, to je i metodologija (Ławrynowicz i dr., 2018, str. 1) Ova metodologija nema strogi workflow, već pruža: (Suárez-Figueroa, Gomez-Perez i dr., 2012, str. 1)

- rječnik procesa i aktivnosti povezanih sa razvojem ontologija
- dva ontološka modela životnog ciklusa
- skup metodoloških vodilja za različite procese i aktivnosti opisane:
  - funkcionalno, pojmovima cilja, ulaza, izlaza i relevantnih ograničenja
  - proceduralno, u smislu specifikacije workflowa
  - empirijski, kroz niz ilustrativnih primjera

Katalog NeOn projekta ima slične elemente kao i ranije spominjani standardni katalog. Ipak, ima i neke specifičnosti. Jedna od njih je i tko je autor uzorka odnosno, tko ga je dostavio uredništvu NeOn projekta. Pored toga je tu i jedan scenarij upotrebe uzorka kao i adresa implementacije najčešće zapisana u OWL obliku. Nemaju sve obitelji uzoraka sve navedene elemente, budući da su uzorci u pojedinim obiteljima specifični. Uzorci poravnanja najčešće su vrlo kratki, i njihov OWL kod je naveden izravno u katalogu. Sa druge strane uzorci sadržaja mogu biti vrlo opsežni. Često su to male ontologije koje se nalaze na posebnim adresama ili u repozitorijima sustava za verzioniranje poput githuba. Elementi kataloga za sve obitelji uzoraka uz varijacije su:

- Naziv uzorka, i alternativni naziv
- Autor uzorka i ponekad odvojeno tko je uzorak poslao uredništvu
- Jezik (osobito za leksičko-sintaktičke uzorke), u smislu prirodnog jezika, na primjer španjolskog ili talijanskog
- Svrha uzorka
- Domena uzorka, na primjer općenito, agrokultura, ribarstvo, informacijske znanosti, pravo, fizika, biologija, IoT, menadžment, softver,
- Pitanja kompetencije
- Opis rješenje
- Ponovno iskoristivi OWL gradivni blok - adresa, odnosno resurs uzorka, ili adresa projekta u sustavu za verzioniranje
- Posljedice
- Scenarij upotrebe uzorka

- Poznata upotreba - primjeri gdje se u poznatim projektima upotrebljava taj uzorak
- Web reference i druge reference
- Primjer korištenja (OWL datoteke)
- Mjesto iz kojeg je uzorak izdvojen, i odakle je dobiven reinženjeringom
- Komponente - za složene uzorke često se navode jednostavniji uzorci koji ga čine.
- Čega je specijalizacija
- Povezani uzorci

3.4.3.2. Uzorci sadržaja

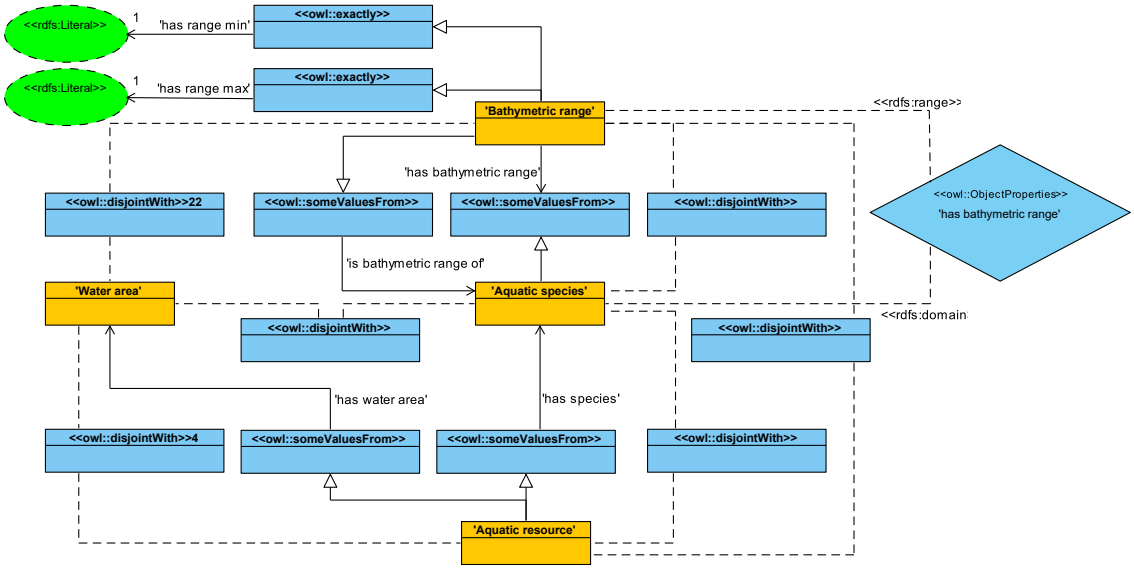
Ovih uzoraka u ima najviše - čak 157 od ukupno 223. Grupirani su prema domenskim područjima, od općih, preko fizikalnih, softverskih, organizacijskih, dio-cjelina, arhivskih, pravnih, procesiranja događaja i jezičnih pa do uzoraka u igrama, ribolovu, zemljoradnji i ekologiji. Ovi su uzorci neovisni o promjeni signature ukoliko se čuva odnos u taksonomijama prema dolje ¶, odnosno oni su *invarijantni na signaturne polimorfizme koji čuvaju redoslijed u taksonomijama prema dolje*. (Presutti i Gangemi, 2007, str. 34)
Značenje ovoga može se ilustrirati primjerom: ako neki *Agent imaUlogu Uloga*, onda vrijedi i da *Osoba imaUlogu Zaposlenik*, signatura je promijenjena, ali je sačuvan redoslijed u taksonomijama prema dolje jer je *Osoba* vrsta (kind-of) *Agenta*, a *Zaposlenik* je vrsta *Uloge* (primjer iz Presutti i Gangemi (2007)).

Ovi uzorci najčešće se nastali ekstrakcijom iz gotovih ontologija. Odlikuje ih različita složenost, od jednostavnih kao na slici 43, pa do izrazito kompleksnih kao na slici 44. Za neke se uzorke može reći i da su male ontologije, kao što je uzorak konačnog stanja detektora (eng. *Detector final state*) i ovdje priloženi uzorak trajektorija (eng. *Trajectory*). Jednako tako, neki uzorci su opće primjenjivi bez obzira na naziv koji nose, kao što su različiti uzorci uloga, lista ili cijena, dok je primjena nekih ograničena na usko, domenski specifično područje, kao na primjer uzorak događaja kao teme reportaže (eng. *NewsReportingEvent*). U uzorku trajektorije vidimo još jednostavniji uzorke zatvaranja. Ovaj uzorak je već opisan u podpoglavlju otvorenih biomedicinskih ontologija (OBO) i zasniva se na povezivanju individua klasa preko egzistencijalnog i univerzalnog svojstva istovremeno. U konkretnom slučaju, individue klase *Trajectory* su povezane svojstvom *hasSegment* jedino sa individuama iz klase *Segment*. Jednako tako u uzorku batimetrije‖ vrsta vidimo uzorak n-arne relacije jer se klasa *'Aquatic resource'* povezuje sa klasama *'Water area'* i *'Aquatic species'* preko odgovarajućih svojstava.

Već prema nazivima uzoraka može se vidjeti da ih je vrlo malo opće namjene, što bi i trebao biti njihov smisao. Na ovaj način pobrojani, bez konačne recenzije, zapravo su samo ideje kako bi se mogle graditi određene ontologije. Pri tome postoji velika redundantnost: ista struktura uzorka u okvirima različitih domenskih problema, se drugačije naziva i time smatra različitim uzorcima. Tu se ogleda nezrelost kataloga, osobito po tome što on može na ovaj način beskrajno rasti, budući da su domenski problemi neiscrpni. Kako ćemo kasnije vidjeti, u OBO ontologijama uzoraka ima malo, i oni su sa logičkog stanovišta odlično razrađeni, i premda daleko jednostavniji, često sadrže sintaktičku kompleksnost.

¶Signatura uzorka je skup imena predikata, odnosno u OWL jezicima, skup klasa i svojstava.

‖Batimetrija (grč. *bathus*, dubina) je znanost o mjerenju dubina mora, rijeka i jezera te njihove dubinske kartografije.
Prema portalu strukovnog nazivlja "Struna" (`http://struna.ihjj.hr/naziv/batimetrija/1712/`), predloženi hrvatski naziv je *dubinomjerstvo*.



Slika 43: Uzorak batimetrije vrsta, vlastiti dijagram

Uzorak batimetrije vrsta jedan je od brojnih uzoraka Alda Gangemija koji se odnose na oceanografiju i ribarstvo, a izvučeni su iz odgovarajućih ontologija nastalih za potrebe oceanografskih i drugih srodnih istraživanja. Uzorke takve vrste potpisuje i Eva Blomquist pa su vjerojatno dio istog projekta. To su uzorci poput *SpeciesConditions*, *SpeciesConservation*, *SpeciesEat*, *SpeciesHabitat*, *SpeciesName* i tako dalje. U ovom uzorku batimetrije vrsta, grupa vodenih organizama se povezuje sa određenim područjem i rasponom dubine (batimetrijskim rasponom). U grupi se mogu razvrstati organizmi prema vrstama, i svaka vrsta ima određeni karakteristični dubinski raspon. Sam raspon ima svoju minimalnu i maksimalnu dubinu izraženu pomoću dva literala. Taj dio ovog složenog uzorka je zapravo uzorak N-arne podatkovne relacije, spomenut u OBO katalogu. Kompetencijska pitanja ovog uzorka, prema NeOn katalogu su:

- Koje vrste imaju koji tipični batimetrijski raspon s obzirom na koje područje?
- Gdje se s obzirom na određeno vodeno područje neka vrsta može pronaći?

Posljedice korištenja uzorka: Iako uzorak opisuje gdje se neka vrsta nalazi s obzirom na područje i dubinu, ne postoji izravna veza između područja i dubine. Ne može se zaključiti postoji li uopće područje koje ima toliku dubinu kao što navodi neki batimetrijski raspon.

Uzorak trajektorije izabrao sam jer sadrži veliku izražajnu raznolikost. Tu imamo općenite klasne aksiome, kompozicije svojstava, inverzna i negirana svojstva i tako dalje. Prema ontološkoj metrici, potrebna DL izražajnost za oblikovanje ovog uzorka je $\mathcal{ALCRIF}$. Uzorak pripada velikom projektu na području geo-znanstvenih istraživanja. Autor je Adila Krishnadi, koji potpisuje i uzorak konačnog stanja detektora na području fizike. Tu su još uzorci poput uzorka aktivnosti, modificirane krizne situacije, igre šaha i transformacije materijala. Svi ovi uzorci mogu se pronaći na github-u na adresi
`https://github.com/cogan-shimizu-wsu/Logician/tree/master/resources`.

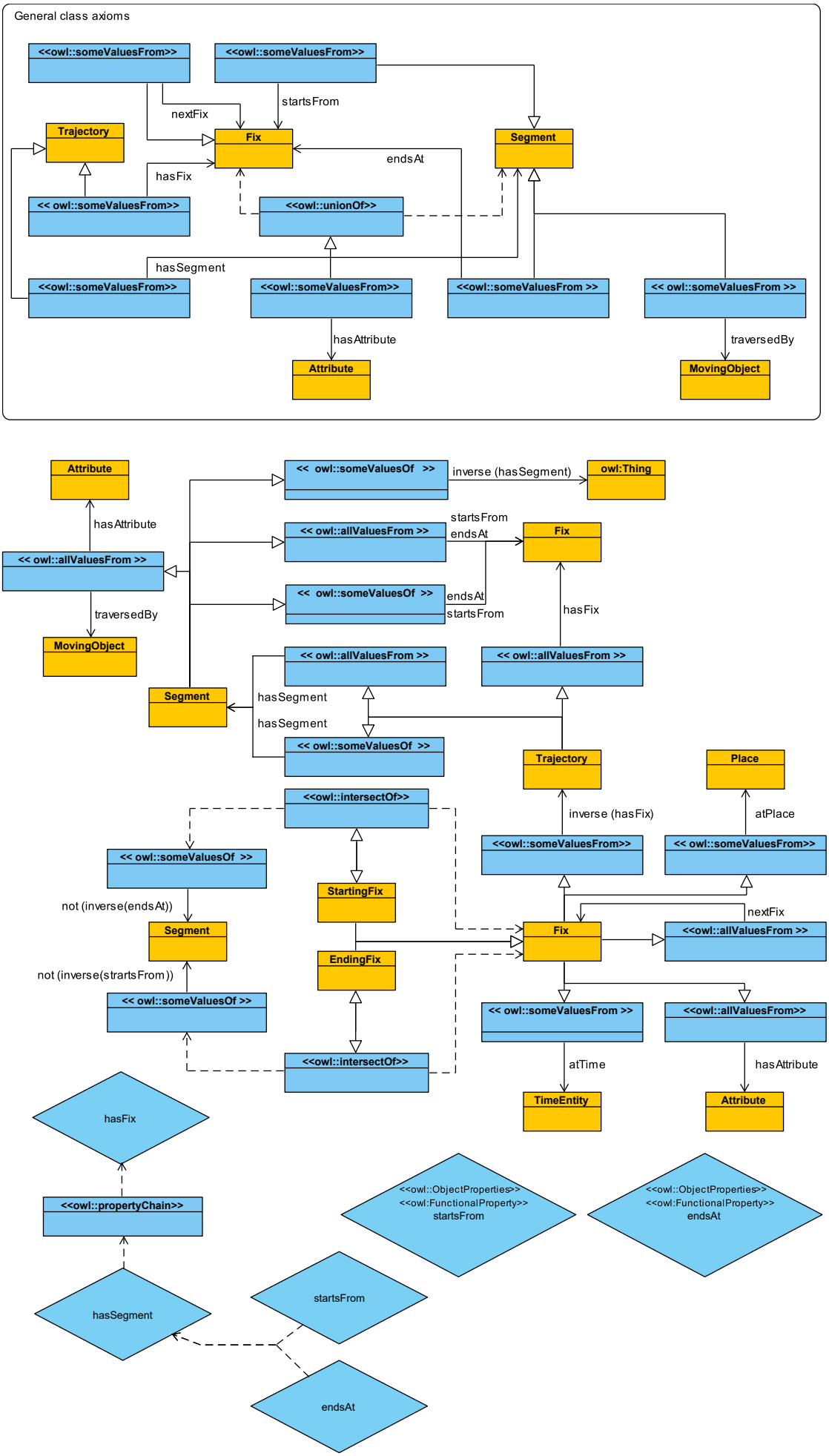
Navedene adrese u on-line katalogu NeOn portala nisu ispravne. Pored geoznanosti, uzorak ima i generalnu primjenu.

Prema katalogu, uzorak modelira trajektoriju, putanju sačinjenu od niza prostorno-vremenskih točaka u kojima se može nalaziti neki objekt. Pitanja/zadaci kompetencije su slijedeća:

- Prikaži pticu koja se zaustavlja na koordinatama (x, y)!
- Prikaži pticu koja se giba po tlu brzinom od 0,4 m/s!
- Prikaži trajektoriju rijeke koja teče kroz nacionalni park!
- Gdje su luke u kojima je brod na oceanskom krstarenju A3321 pristao, nakon što je napustio Woods Hole?
- Ispiši vremena i mjesta koja predstavljaju prostorno-vremensku ekstrapolaciju događaja na svjetskom prvenstvu u šahu 1991. godine!

Mogući scenariji:

- Majkov put prema GeoVoCamp 2012, od njegove kuće, nastao integracijom položaja izmjerenih pomoću GPS-a, informacija o vozilu, i osobnih informacija.
- Let tukana kao rezultat promatranja istraživača zabilježenih u MoveBanku.
- Godine 1990, svjetsko prvenstvo u šahu se događalo na dvije lokacije u dva različita vremena



Slika 44: Uzorak trajektorije, vlastiti dijagram

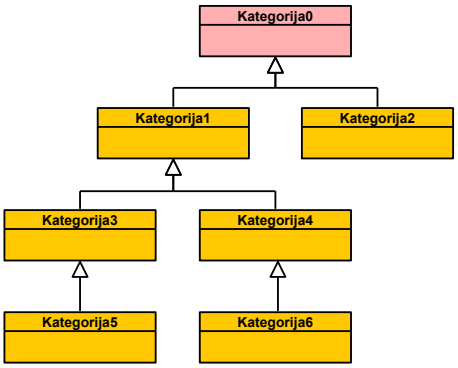
Dokument NeOn projekta D5.1.1v3 (Suárez-Figueroa, Brockmans i Gangemi, 2007) navodi uzorke sadržaja od kojih su neki kasnije objašnjeni u podpoglavlju o OBO biomedicinskim ontologijama, pa i ovdje samo nabrajamo zajedno sa identifikacijskom oznakom:

- Modeliranje sudjelovanja (participacije) CP-PA-01
- Modeliranje opisa i situacije CP-DS-01
- Modeliranje uloga i zadataka CP-RT-01
- Modeliranje plana i izvršenja CP-PE-01
- Jednostavna relacija dio-cjelina CP-PW-01 i hijerarhija klasa tipa dio-cjelina CP-PW-02

3.4.3.3. Uzorci reinženjeringa

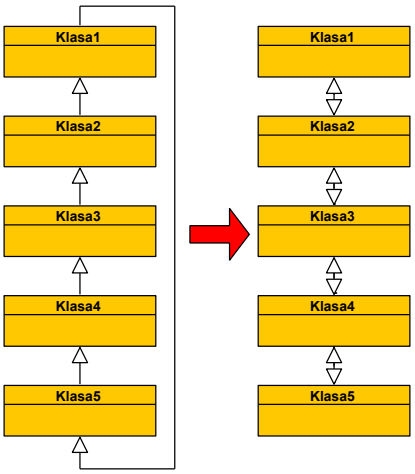
Uzorci reinženjeringa zajedno sa uzorcima poravnanja (mapiranja) ubrajaju se u uzorke korespondencije. Oni nisu dijelovi većih ontologija i ne prikazuju dio nekog određenog problema. To su transformacijski algoritmi i pravila pomoću kojih se kreiraju nove ontologije iz neontoloških izvora, na primjer zapisa u tablicama, ili se postojeće ontologije transformiraju u drugi oblik. U NeOn uzorcima najveći broj uzoraka se bavi klasifikacijom, odnosno stvaranjem hijerarhija klasa na osnovu različitih zapisa u relacijama/tablicama. Primjer jednog takvog uzorka je *Klasifikacijska shema - model liste susjedstva - prema taksonomiji*.

Oznaka kategorije	Ime kategorije	Oznaka kategorije roditelja
1	Kategorija1	NULL
2	Kategorija2	NULL
3	Kategorija3	1
4	Kategorija4	1
5	Kategorija5	3
6	Kategorija6	4



Slika 45: Klasifikacijska shema - model susjednosti

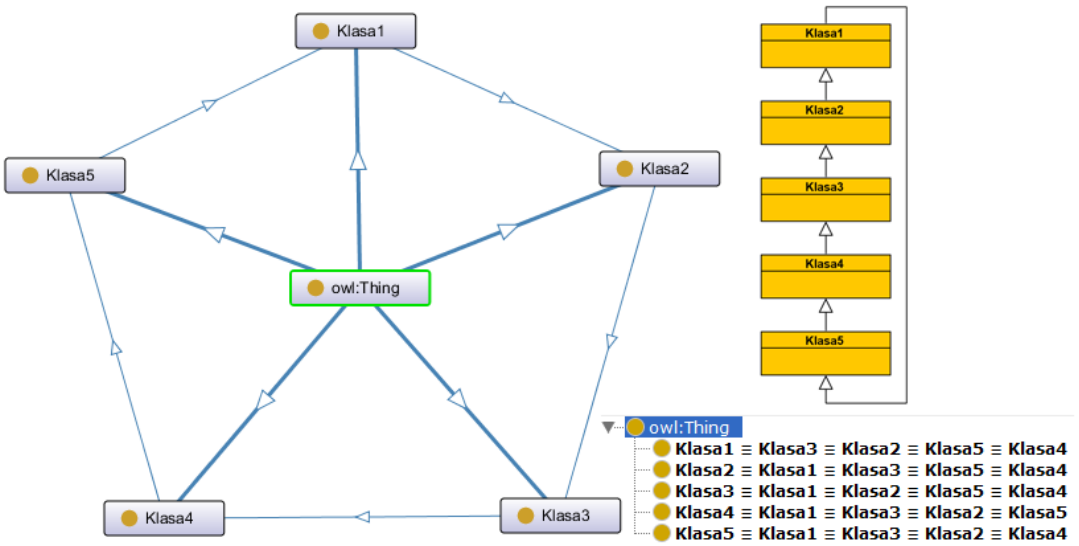
Ovaj uzorak od modela zapisa u tablici, kreira hijerarhiju klasa. Ako za neku klasu u tablici nije navedeno tko su joj roditelji, tj. nema roditelja pa je unos *NULL*, tada se kreira klasa više hijerarhije na koju se ovakve klase usmjeravaju. Budući da je u ontologijama dozvoljeno višestruko nasljeđivanje, ne radi se o hijerarhijskom stablu nego grafu. Upravo zato ima smisla govoriti o listi susjedstva, što je blisko povezano sa matricom susjedstva iz teorije grafova. Autor uzorka je Boris Villazón Terrazas.



Slika 46: Uzorak reinženjeringa cikličkih klasa

Pored ovog uzorka postoje i druge varijante koje se razlikuju jedino po tome kako se zapisuje povezanost koncepata u tablici. Tako uzorak *Klasifikacijske sheme - model enumeracije staze - prema taksonomiji* ima stupac enumeracije staza. Klasa enumeracije 1, 2, ... nema roditelja. Klase 11, 12, 13... imaju roditelja sa enumeracijom 1, dok klase 21, 22, 23... imaju roditelja sa enumeracijom 2. Dalja se hijerarhija označava dodavanjem nove znamenke, npr. 111, 112 itd.

Cikličnost hijerarhija klasa ponekad je teško prepoznati. Zaključivači omogućavaju rearanžiranje hijerarhija, i ta je transformacija također uzorak reinženjeringa. Na slici 47 prikazano je kako se svaki ciklički lanac nasljeđivanja pretvara u kolekciju ekvivalentnih klasa.



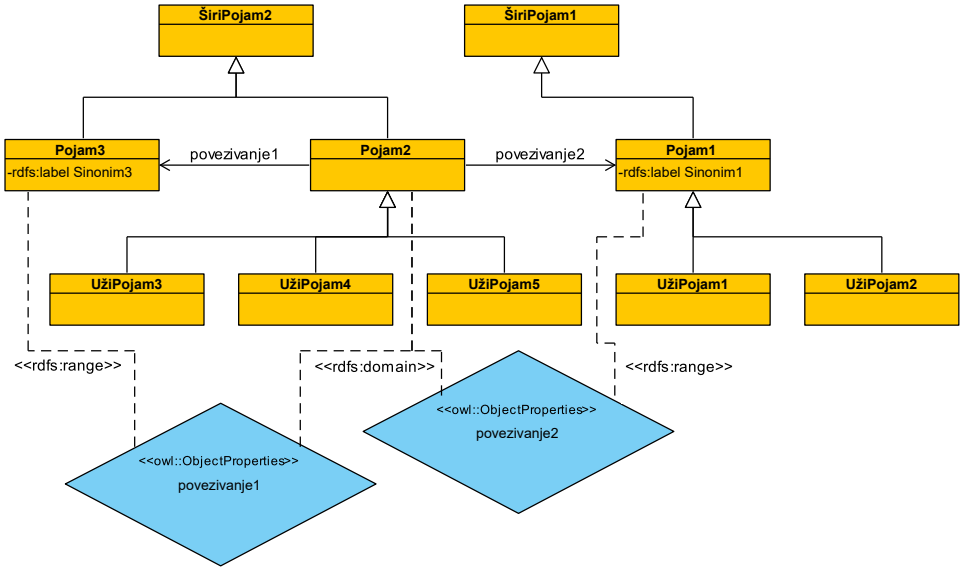
Slika 47: Cikličke klase u Protégéu

Treba napomenuti da alati za razvoj ontologija poput Protégé-a automatski transformiraju cijeli ciklički lanac u niz pravila o ekvivalentnim klasama kao što je prikazano na slici 47. Postoji više načina kako se može kreirati ciklički lanac. Najjednostavnije je klasu u lancu na najnižoj razini postaviti pravilom kao nadklasu najvišoj klasi. Moguće je to učiniti i pomoću ge-

neralnog aksioma o klasama. Što god da se učini, alat će odmah poistovjetiti klase u lancu, bez obzira je li uključen zaključivač ili nije, te je li odabrana unesena ili zaključena hijerarhija. Autor uzorka je Olaf Noppens.

Pojmovno bazirani - baziran na zapisu - tezaurus - prema laganoj ontologiji uzorak na osnovu stupaca u tablici uspostavlja hijerarhiju klasa i njihovo međusobno povezivanje objektnim svojstvima (povezuju se zapravo individue tih klasa). Jednako tako im se dodaju i odgovarajući atributi sinonima. Na slici 48, prikazano je kako se gradi ontologija na osnovu zapisa u tablici.

Pojam	Širi pojam	Uži pojam	Povezani pojam	Sinonim
Pojam1	ŠiriPojam1	UžiPojam1 UžiPojam2		Sinonim1
Pojam2	ŠiriPojam2	UžiPojam3 UžiPojam4 UžiPojam5	Pojam1 Pojam3 Pojam4	
Pojam3	ŠiriPojam2			Sinonim3



Slika 48: Pojmovno bazirani - baziran na zapis - tezaurus model

Tezaurus, kao i leksikon, može biti ontologija. Ovaj uzorak pretvara neontološki model u ontologiju. Svaki pojam iz tezaurusa se mapira na neku klasu u ontologiji. Na osnovu stupca povezanih pojmova, grade se odgovarajuća svojstva koja povezuju klase. Autor ovog uzorka je Boris Villazón Terrazas.

3.4.3.4. Uzorci poravnanja

Kada se gradi ontologija od uzoraka ili se povezuju heterogene ontologije u jednu veću cjelinu, potrebno je uskladiti njihove elemente na ispravan način. Odgovarajuće klase se moraju povezati na odgovarajući način. Taj zadatak često nije trivijalan i čini ontološke medijacijske tehnike. Postoje tri osnovne relacije koje se odnose na poravnanje: ((Presutti i Gangemi, 2007, str.))

- Ekvivalencija
- Sadržavanje
- Preklapanje

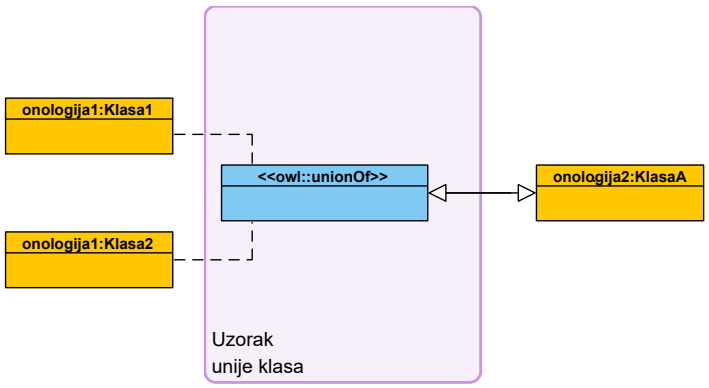
Uzorci koji te postupke olakšavaju, odnosno uspostavljaju korespondencije između heterogenih ontologija i uzoraka smanjujući mogućnost pojavljivanja pogrešaka, nazivaju se uzorci poravnanja. François Scharffe navodi dva razloga nepodudarnosti ontologija prema Kleinu Scharffe (2008):

1. nepodudarnost na razini ontologija - nastaje zbog različitog kodiranja ontoloških koncepata. Ove se nepodudarnosti mogu dalje prema Visseru podijeliti na:
    - Konceptualne nepodudarnosti - nastaju zbog različite konceptualizacije unutar iste domene i dijele se na tri grupe:
        - Nepodudarnosti dosega (eng. *scope mismatch*) - klase imaju jednaka preklapanja u svojim ekstenzijama ali te ekstenzije nisu jednake
        - Relacijske nepodudarnosti - relacije među klasama su različito modelirane u dvije ontologije koje se nastoje uskladiti
        - Različita pokrivenost modelom i granularnost - dvije ontologije pokrivaju različita područja domene, ili su modeli različite razine detaljnosti
    - Izražajno nepodudaranje - nisu povezana sa ontologijama već sa različitim tumačenjima entiteta. Dijele se na:
        - Različite paradigme, na primjer vrijeme izraženo kao interval u jednoj ontologiji i vrijeme kao točka u drugoj
        - Opis koncepata - različite konvencije imenovanja. Primjer: knjiga i podklasa znanost u jednoj ontologiji, te knjiga i podklasa znanstvena_knjiga u drugoj ontologiji
    - Terminološko nepodudaranje - odnosi se na imena entiteta:
        - Sinonimi - različite riječi sa istim značenjem, na primjer tuga i žalost
        - Homonimi - iste riječi sa različitim značenjem, na primjer grad Rijeka i rijeka
        - Hiponimi - izraz u jednoj ontologiji je manje općenit nego u drugoj ontologiji, na primjer hrast i drvo
        - Hipernimi - izraz u jednoj ontologiji je općenitiji od izraza u drugoj, na primjer cvijet i ruža
    - Nepodudaranje kodiranja - čest kod integracije podataka iz različitih izvora, i povezan je sa tipovima atributa
        - Kodiranje prezentacije - primjer je nepodudaranje mjernih jedinica u dvije ontologije
        - Tip podataka - različiti tipovi podataka, na primjer integer i float

– Nedostatak podataka - jedna od ontologija nije jednako instancirana. Može se dogoditi i kada dvije ontologije nemaju jednako strogo definirana ograničenja poput kardinalnosti. Iako će zaključivač povezati dvije ekvivalentne klase, njihove instance se mogu razlikovati. Takve instance neće imati određena svojstva u obje ontologije.

2. neusklađenosti na razini jezika - iako su RDFS i OWL najčešći jezici, odnosno formalizmi za prikaz ontologija, postoje i drugi jezici poput WSML jezika (jezik za modeliranje web servisa, eng. Web Services Modelling language). Različiti jezici i formalizmi mogu imati različitu izražajnost pa je tome potrebno posvetiti pažnju prilikom prevođenju na zajednički jezik.

Jednostavan uzorak unije klasa povezuje par klasa iz jedne ontologije sa jednom klasom iz druge ontologije na način da uspostavlja ekvivalenciju unije para klasa u jednoj ontologiji i klase u drugoj ontologiji. Dijagram tog uzorka prikazan je na slici 49.



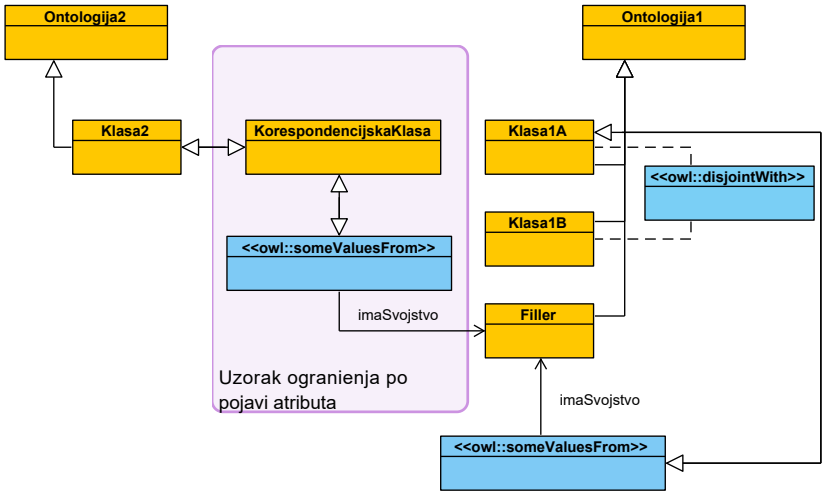
Slika 49: Uzorci poravnanja - unija klasa

Ovaj uzorak potpisuje, odnosno šalje na NeOn portal uzoraka dizajna François Scharffe, a u katalogu kao primjer daje slijedeći kod Scharffe (2008):

Listing 3.11: Poravnanje u RDF/XML

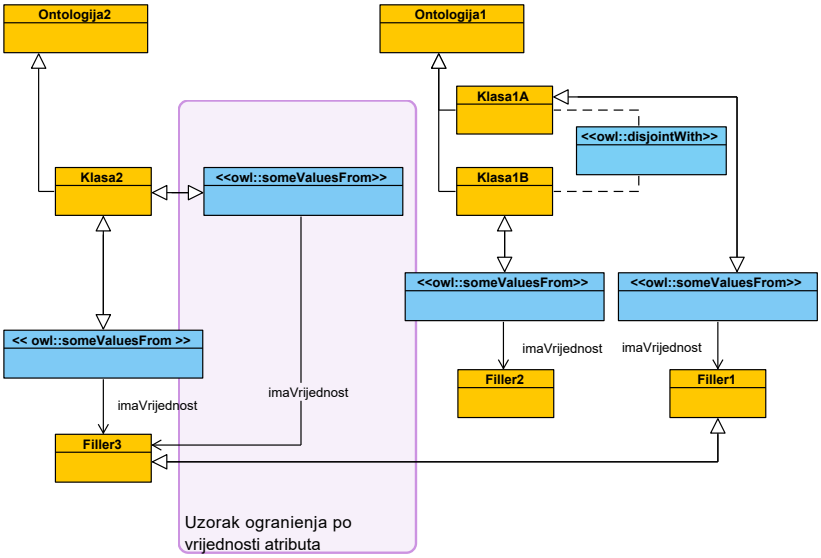
```
<Cell>
<entity1>
<Class>
<or rdf:ParseType="Collection">
<Class rdf:about="O1:PersonBornInCanada"/>
<Class rdf:about="O1:PersonWithCanadianParent"/>
</or>
</Class>
</entity1>
<entity2>
<Class rdf:about="O2:CanadianCitizenByBirth"/>
</entity2>
<relation>equivalence</relation>
</Cell>
```

Uzorak poravnanja po pojavi atributa povezuje klasu u jednoj ontologiji (ili uzorku) sa odgovarajućom klasom u drugoj ontologiji (ili uzorku), ukoliko prva klasa ima odgovarajuće ograničenje po svojstvu, odnosno atribut. Uzorak ne precizira moraju li klase biti ekvivalentne, tj. dvostranost poveznice. Pošiljatelj uzorka na NeOn portal je François Scharffe.



Slika 50: Poravnanje sa klasom po pojavi atributa

U izvedbi prikazanoj dijagramom na slici 50, odgovarajuće klase u različitim ontologijama je postavljaju kao ekvivalentne klase. Kao što je ranije rečeno, nisu nužna oba smjera. Ovaj je slučaj složeniji, i zahtijeva da klasa unutar druge ontologije bude ekvivalentna svom ograničenju. Između ontologija je ugrađena korespondencijska klasa radi jasnijeg razdvajanja elemenata i izbjegavanja dodavanja novih svojstava i ograničenja samim ontologijama koje se povezuju. Poravnanje je moguće obaviti i na jednostavniji način, direktnim dodavanje ekvivalencije klasi *Klasa2*. Jednako tako, ukoliko se ne inzistira na dvosmjernosti, moguće je povezivanje ugraditi i u generalni aksiom klasa $imaSvojstvo\ some\ Filler\ SubClassOf Klasa2$. Na sličan se način mogu riješiti i poravnanja po tipu i vrijednosti atributa, domeni i dosegu relacije, te putanji atributa. Na slici 51 je prikazan dijagram poravnanja po vrijednosti atributa pri čemu je poravnanje dvostrano.



Slika 51: Poravnanje sa klasom po vrijednosti atributa

Uz ove uzorke, katalog navodi uzorak ekvivalencije klasa. U ovom modelu je i to ugrađeno, budući je klasa *Filler3* postavljena kao ekvivalentna klasa klasi *Filler1*. Zbog jednostavnosti nije prikazana ekvivalencija svojstava *imaVrijednost1* i *imaVrijednost2* za svaku ontologiju respektivno.

3.4.3.5. Logički uzorci

Logički uzorci pripadaju uzorcima strukture, zajedno sa uzorcima arhitekture. Imaju praznu signaturu, uz izuzetak klase *owl:Thing*.(Presutti i Gangemi, 2007)
Dijele se na logičke makroe koji komponiraju OWL DL konstrukte i transformacijske uzorke koji transformiraju logičke izraze iz jednog jezika u drugi. Logički uzorci dizajna imaju zadatak riješiti probleme ekspresivnosti kao i detektirati pogreške u ontologijama, osobito po pitanju logičkih anti-uzoraka (eng. *Logical Antipatterns* - LAP). Neovisni su o domenskim problemima, slično kao i uzorci reinženjeringa s kojima su povezani preko uzoraka transformacije. Uzorci transformacije pripadaju i logičkim uzorcima i uzorcima reinženjeringa. Pored njih, logičkim uzorcima pripadaju i logički makroi. Neki od tih uzoraka poput n-arnih relacija, listi, normalizacija i particije, prikazani su kasnije u uzorcima otvorenih biomedicinskih ontologija, pa ih ovdje preskačem.

Logički uzorci nisu namijenjeni samo kreiranju logički valjanih i provjerenih elemenata većih ontologija, nego i detekciji loših rješenja, takozvanih anti-uzoraka (eng. *antipattern*). Jedan od anti-uzoraka je i uzorak "jedinstvenost je usamljenost" (eng. *Onlyness is loneless*), koji se označava akronimom *OIL*. Ovaj se anti-uzorak može pojaviti u više varijanti, kao što je

prikazano aksiomima deskriptivne logike (Roussey i Zamazal, 2013):

$$C_3 \sqsubseteq \forall r.C_1;\ C_3 \sqsubseteq \forall r.C_2; \qquad Disj(C_1, C_2); \tag{3.1}$$

$$C_3 \equiv \forall r.C_1;\ C_3 \sqsubseteq \forall r.C_2; \qquad Disj(C_1, C_2); \tag{3.2}$$

$$C_3 \equiv \forall r.C_1;\ C_3 \equiv \forall r.C_2; \qquad Disj(C_1, C_2); \tag{3.3}$$

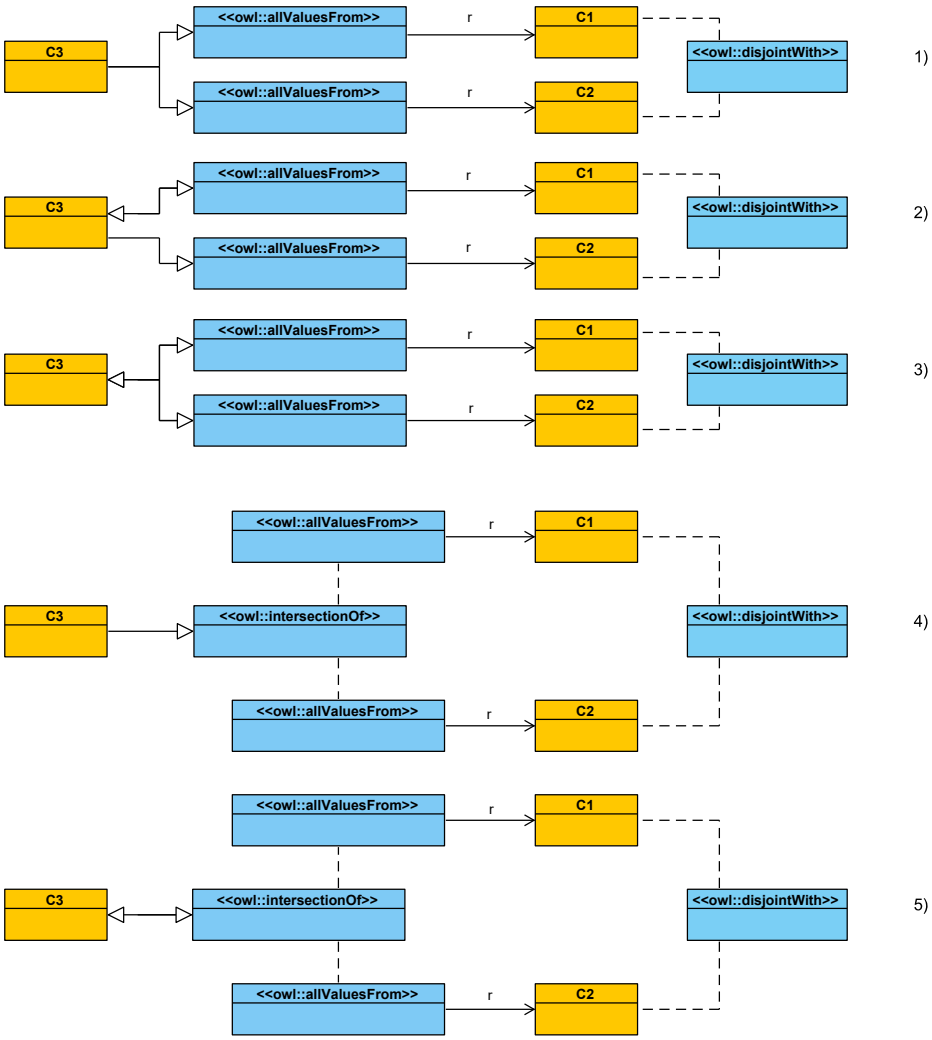
$$C_3 \sqsubseteq \forall r.C_1 \sqcap \forall r.C_2; \qquad Disj(C_1, C_2); \tag{3.4}$$

$$C_3 \equiv \forall r.C_1 \sqcap \forall r.C_2; \qquad Disj(C_1, C_2); \tag{3.5}$$

Ovaj anti-uzorak spominje se i u ontologiji o pizzama, gdje se tvrdi kako je pogrešno postaviti pravila:

$$imaNadjev\ only\ NadjevSir$$
$$imaNadjev\ only\ NadjevPovrće$$

Ovo je slučaj *4)* i ukoliko postoji bar jedna individua na koju bi se svojstvo r moralo odnositi, zaključivač će javiti da je ontologija nekonzistentna. Već i iz samo dijagrama je jasno zašto je tome tako:



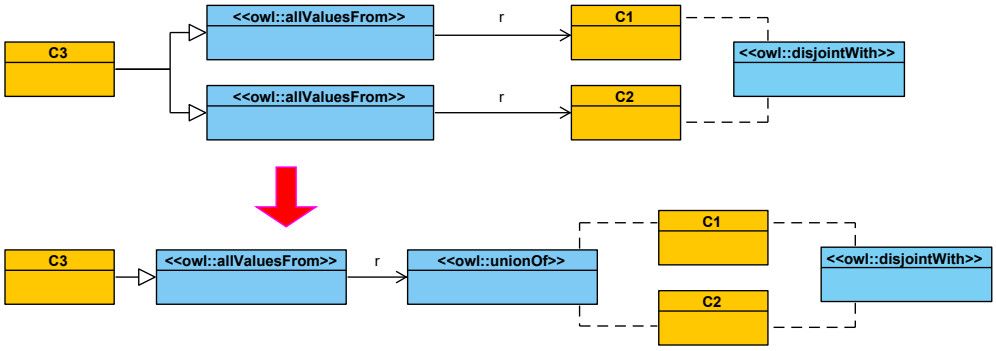
Slika 52: Anti-uzorak "jedinstvenost je usamljenost"

Sve individue klase *C3* povezane su samo sa individuama klase *C1*. Ukoliko se doda novo ograničenje da su sve individue povezane samo sa individuama klase *C2*, zaključivač će pretpostaviti da postoji neki presjek (*owl::intersectionOf*) klasa *C1* i *C2*. Sve individue jedne od klasa smjestit će i u drugu klasu. Ako je dodano i pravilo da su klase *C1* i *C2* razdvojene, takav presjek ne može postojati i zaključivač će označiti takvu ontologiju nekonzistentnom.

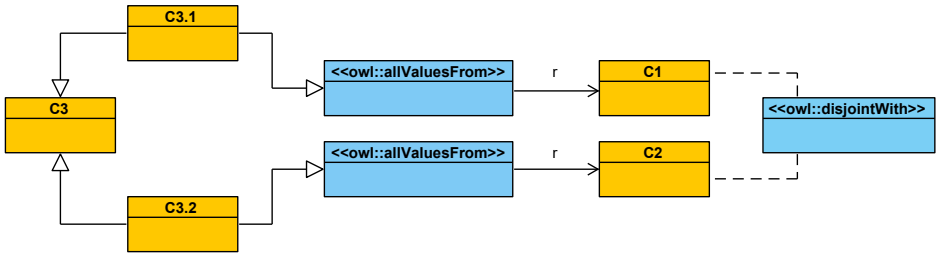
U uzorcima NeOn kataloga Catherine Roussey i Oscar Corcho prilažu *OIL* anti-uzorak, uz opise kako do njega može doći uz moguća rješenja problema. Tako za *OIL* uzorak daju pretpostavku da korisnik prvo kreira univerzalno ograničenje po svojstvu r za jednu klasu, a onda, zaboravivši taj aksiom, kreira drugo univerzalno ograničenje po svojstvu r za drugu klasu. Rješenja koje autori nude:

- Pretvoriti dva univerzalna ograničenja u jedno prema disjunkciji klasa *C1* i *C2* (slika 53)
- Ukloniti razdvojenost klasa *C1* i *C2*
- Kreirati dvije podklase od *C3* i pridružiti njihove individue uz univerzalno ograničenje po svojstvu *r* klasama *C1* i *C2* respektivno (slika 54)

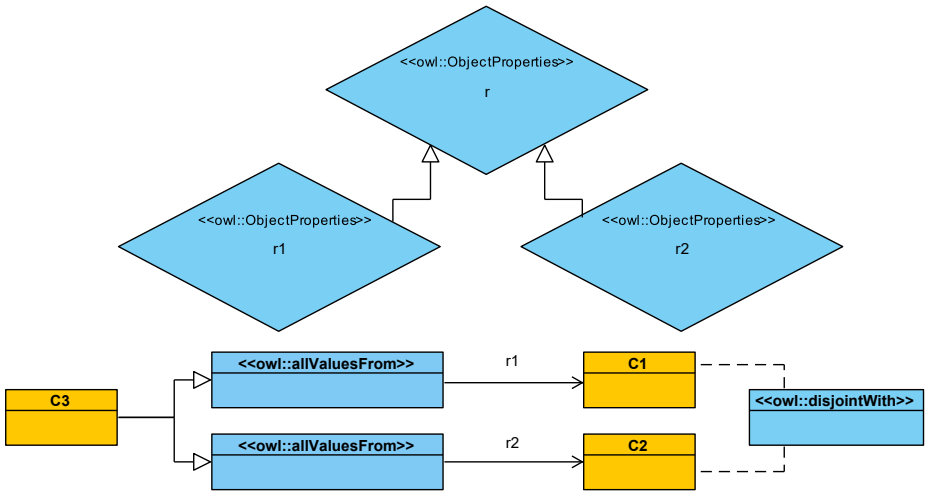
- Kreirati novu klasu *C4* koja je ekvivalentna *C1 or C2*. Povezati individue klase *C3* sa individuama *C4* uz univerzalno ograničenje po svojstvu r.
- Kreirati dva podsvojstva od r. Pomoću ta dva podsvojstva povezati individue klase *C3* sa individuama klasa *C1* i *C2* (slika 55)



Slika 53: Mogućnost razrješenja problema OIL - ujedinjenje dva univerzalna ograničenja



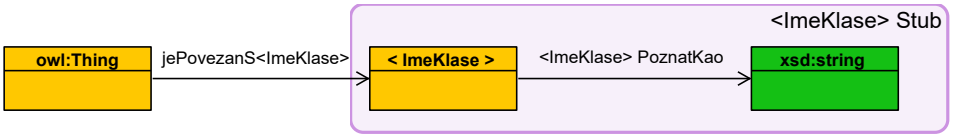
Slika 54: Mogućnost razrješenja problema OIL - razdvajanje ishodišne klase



Slika 55: Mogućnost razrješenja problema OIL - razdvajanje svojstva r na dva podsvojstva

Jedan jednostavan metauzorak koji pripada logičkim uzorcima je uzorak kontrolirane zamjene (eng. *stub*). On kao takav je samo nadomjestak (eng. *placeholder*) za buduća proširenja

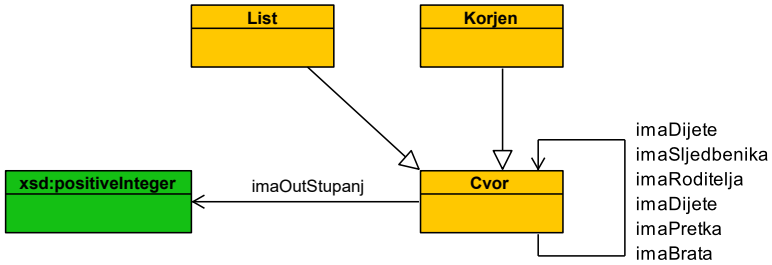
i dodavanje detalja bez potrebe za inicijalnom visokom granulacijom koja je često nepoželjna. Autori uzorka su Adila Krishnadi i Paul Hitzler.



Slika 56: Metauzorak kontrolirane zamjene (stub)

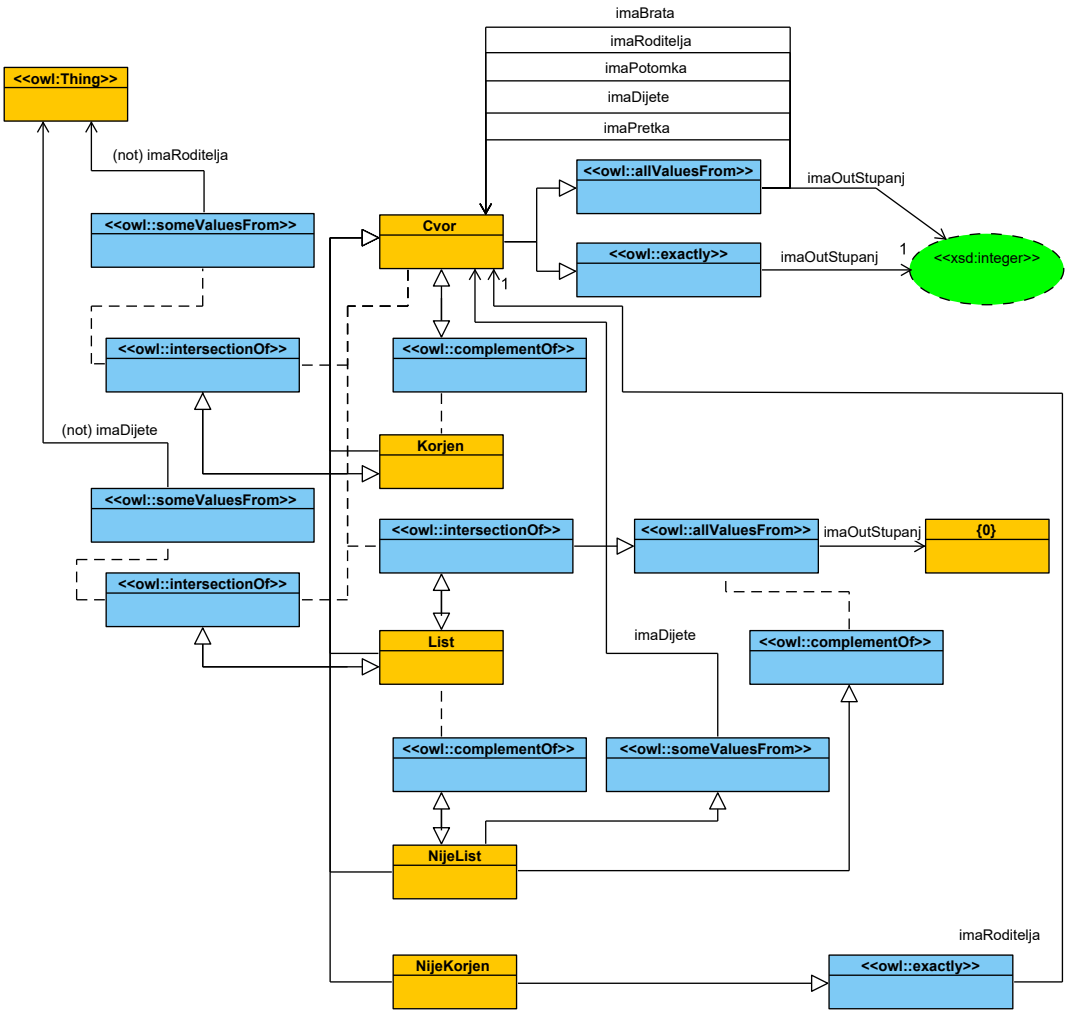
Ovaj uzorak ima dva varijabilna svojstva i jednu varijabilnu klasu. Prilikom upotrebe metauzorka ova se metasvojstva i metaklase instanciraju u konkretna svojstva i klase.

Uzorak stabla je izuzetno zanimljiv uzorak. Omogućava oblikovanje struktura podataka koje imaju središnje mjesto u organizaciji znanja. Autori koji su poslali uzorak, a to su David Carral, Pascal Hitzler, Hilmar Lapp i Sebastian Rudolph bavili su se problemima ekspresivnosti OWL-a i ugradili saznanja u svoje rješenje ovog uzorka.



Slika 57: Uzorak stabla

Adila Krishnadi na gihub repozitoriju (`https://github.com/cogan-shimizu-wsu/Logician/tree/master/resources`) ima detaljnije razrađen uzorak stabla u kojem postoje posebna ograničenja za čvorove stabla ovisno kakva im je uloga: korjen, list, čvor koji nije list i čvor koji nije korjen. Osim podatkovnog svojstva *imaOutStupanj*, tu su i objektna svojstva poput *jeBrid*, *imaPretka*, *imaRoditelja*, *imaNasljednika*, *imaDijete* i *imaBrata*.



Slika 58: Uzorak stabla - detaljniji

Dokument NeOn projekta D5.1.1v3 (Suárez-Figueroa, Brockmans i Gangemi, 2007) navodi logičke uzorke od kojih su neki kasnije objašnjeni u potpoglavlju o OBO biomedicinskim ontologijama, pa i ovdje samo nabrajamo zajedno sa identifikacijskom oznakom:

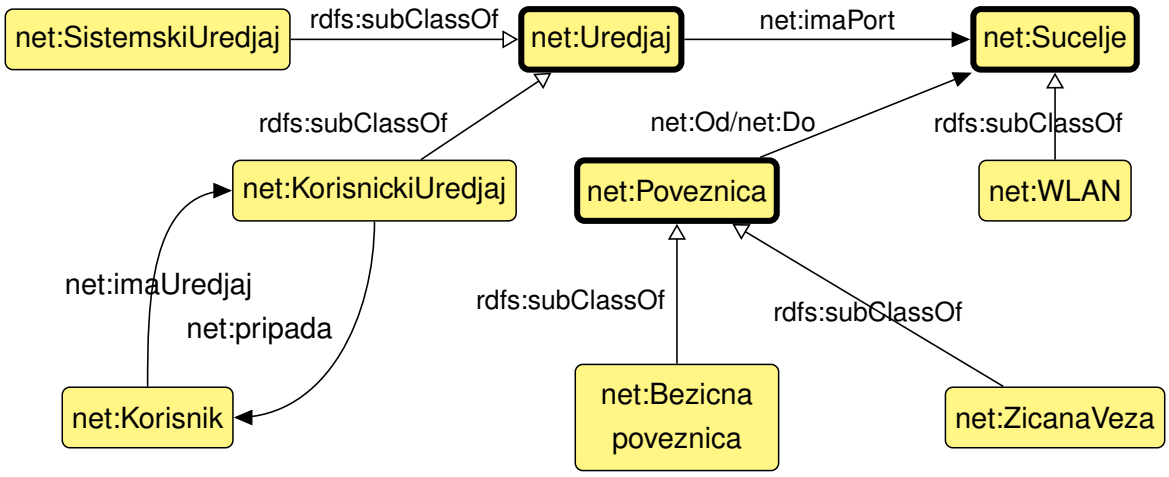
- Modeliranje primitivne klase LP-PC-01
- Modeliranje definirane klase LP-DC-01
- Modeliranje subClassOf relacije LP-SC-01
- Modeliranje višestrukog nasljeđivanja LP-MI-01
- Modeliranje objektnog svojstva LP-OP-01
- Modeliranje SubPropertyOf relacije LP-SP-01
- Modeliranje podatkovnog svojstva LP-DP-01
- Modeliranje egzistencijalnog ograničenja LP-ER-01
- Modeliranje univerzalnog ograničenja LP-UR-01

- Modeliranje unije relacija LP-UO-01
- Modeliranje individue LP-In-01
- Modeliranje razdvojenih klasa LP-Di-01
- Modeliranje iscrpnih klasa LP-EC-01
- Modeliranje N-arnih relacija LP-NR-01 i liste argumenata u relacijama LP-NR-02
- Modeliranje specificiranih vrijednosti: skup individua LP-SV-01 i vrijednosti kao podklasa LP-SV-02

3.4.3.6. Uzorci arhitekture

Uzorci arhitekture opisuju okvirni izgled ontologija sukladno određenim zahtjevima kao što su kompleksnost, računalni resursi, ekspresivnost i slično. Podklasa su uzoraka strukture, kao i logički uzorci. I dok su logički uzorci kompozicije logičkih konstrukata, uzorci arhitekture su kompozicije logičkih uzoraka. (Presutti i Gangemi, 2007, str. 19)
Uzorci arhitekture usporedivi su sa uzorcima solidne softverske arhitekture, sa ciljem rješavanja problema učinkovite interakcije među entitetima.

Domena ovog uzorka su telekomunikacijske mreže (prema NeOn katalogu). Autor uzorka Qianru Zhou, a ovaj uzorak je nastao na temelju čestih dizajnerskih problema u dizajnu telekomunikacijskih mreža.



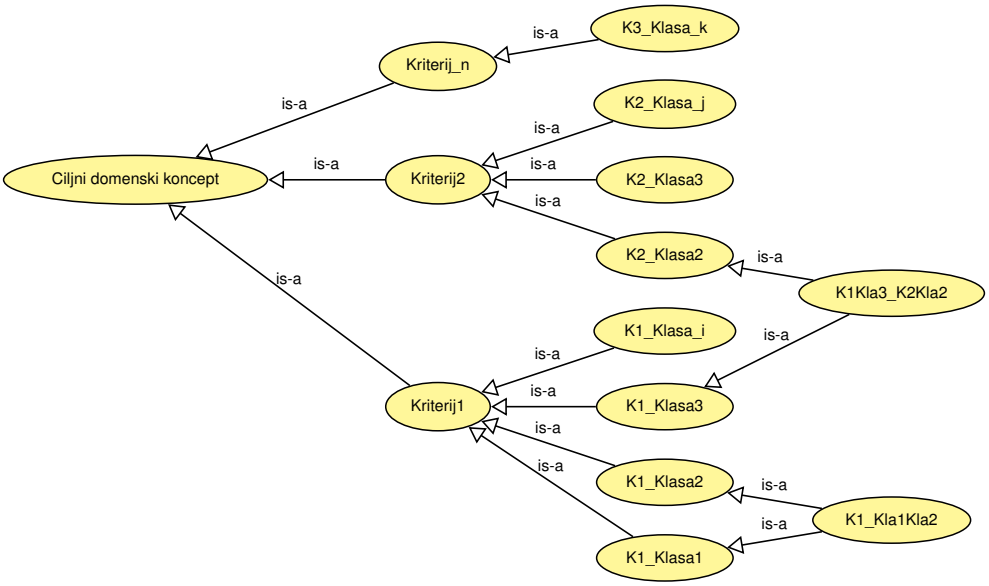
Slika 59: Uzorak Uređaja - sučelja - poveznica (Device - Interface - Link) prema dijagramu Qianru Zhoua

Pored ovog uzorka pripremi NeOn kataloga se nalazi i uzorak nasljeđivanja pogleda (eng. *View inheritance*). To je jedan usmjereni graf nasljeđivanja koji dozvoljava višestruko nasljeđivanje. Koristi se za modeliranje domenskih problema sa kriterijima na više razina, pri čemu postoji mogućnost njihovog preklapanja. U katalogu je dan primjer analize kvarova sa tri razine kriterija, gdje su prve dvije razine u hijerarhiji stabla a treća s njima čini matricu incidencije. Prilikom povezivanja klasa, ovaj će uzorak pomoći da se odredi najbolji način kako

formirati graf višestrukog nasljeđivanja. Budući da konačni rezultat nije stablasta hijerarhija, bit će poželjno koristiti aksiome, kako bi zaključivač mogao formirati ispravnu strukturu.

Smisao uzorka je prikaz koncepata unutar neke domene, koji su prekompleksni u smislu definicije, te višestrukih kriterija klasifikacije njihove apstrakcije. Uzorak u primjeru ima 4 sloja apstrakcije (slika 60):

- Ciljni domenski koncept koji treba prikazati
- Sloj kriterija apstrakcije koji ne moraju nužno biti razdvojeni
- Sloj profinjenja svakog kriterija apstrakcije - također ne moraju biti razdvojeni
- Sloj kombinacija različitih klasa profinjenja. Ove klase mogu višestruko nasljeđivati klase profinjenja iz različitih klasa kriterija



Slika 60: Uzorak nasljeđivanja pogleda (prema Benedicto Rodriguez-Castru i Hughu Glaseru)

Posljedice upotrebe ovih uzorka su: višestruko nasljeđivanje klasa unutar nekog kriterija i klasa različitih kriterija.
Referenca navedena u katalogu: Meyer, B.: Object-Oriented Software Construction (Book/CD-ROM) (2nd Edition). Prentice Hall PTR (March 2000) Dokument NeOn projekta D5.1.1v3 (Suárez-Figueroa, Brockmans i Gangemi, 2007) navodi uzorke arhitekture:

- Taksonomija AP-TX-01
- Lagana ontologija AP-LW-01
- Modularna arhitektura AP-MD-01

3.4.3.7. Leksičko-sintaktički uzorci

"Leksičko-sintaktički uzorci dizajna se definiraju kao lingvističke strukture ili sheme koje se sastoje od različitih vrsta riječi koje slijede specifični redoslijed, dozvoljavajući generalizaciju i ekstrakciju određenih zaključaka povezanih sa njihovim značenjem." (Presutti i Gangemi, 2007, str. 29)

Leksičko sintaktički uzorci ne opisuju probleme unutar neke domene, već se bave ekspresivnošću jezika koji se koristi za opis modela i njegovom sintaksom. U NeOn katalogu koristi se formalizam u prirodnom jeziku (engleskom) te formalizam leksičko-sintaktičkih uzoraka (LSP). Ovo je izvedenica BNF notacije** prilagođena uzorcima dizajna ontologija. Primjer nasljeđivanja klase prikazuje se na slijedeći način:

$$NP < podklasa > je\ NP < nadklasa >$$

Ovdje *NP* označava imenični izraz (eng. *Noun Phrase*).

Tablica 5: Formalizam leksičko-sintaktičkih uzoraka

Kratice u opisima leksičko-sintaktičkih uzoraka i njihovo značenje	
CD	Kardinalni broj
CATV	Klasifikacijski izrazi {classify in/into, comprise in, contain in, compose of, group in/into, divide in/into, fall in/into, belong to}
CN	Ime klase: generičko ime za tip klase sa prijedlogom {class of, group of, type of, member of, subclass of, category, ...}
NEG	Negativno (eng. *no, not*)
NP	Imenični izraz (eng. *Noun Phrase*)
PARA	Paralingvistički simbol, kao što je dvotočka (:)
PREP	Prijedlog (eng. *Preposition*)
RPRO	Odnosna zamjenica (eng. *Relative pronoun*)
VB	Glagol (eng. *Verb*), osnovni oblik
Ostali simboli	
()	Grupiranje elemenata (dva ili više)
*	Ponavljanje
[]	Neobavezni element
¬	Element koji se ne bi trebao pojavljivati na toj poziciji u uzorku

(Prema izvoru: (Presutti i Gangemi, 2007))

Jedan od jednostavnijih primjera je leksičko-sintaktički uzorak (u NeOn dokumentaciji

**BNF je akronim za eng. *Backus-Naur form*, ponekad i *Backus normal form*, Backus-Naurova (normalna) forma. BNF je metasintaksa odnosno notacijska tehnika za izražavanje kontekstno neovisnih gramatika, odnosno formalnih opisa formalnih jezika. Koristi se za službeno specificiranje sintakse u korisničkim priručnicima, literaturi koja opisuje programske jezike i njihov formalizam, te teoriju programskih jezika. Primjer u SQL jeziku je naredba DELETE za brisanje jednog ili više zapisa u tablici, koja se u BNF izražava kao <brisanje retka: pretraženo> ::= DELETE FROM <tablica> [WHERE <uvjet pretraživanja>]

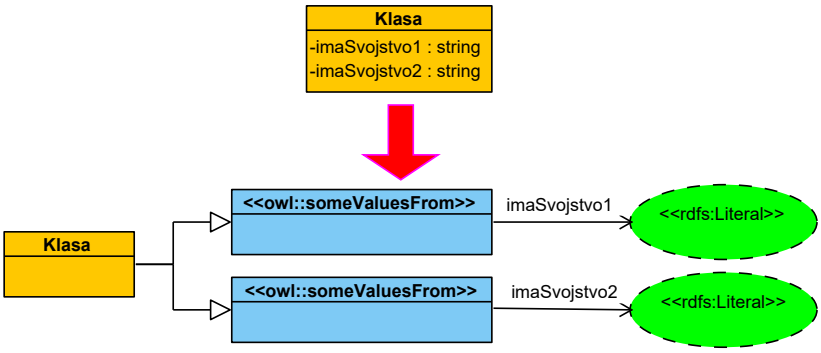
pod identifikacijskom oznakom LSP-DP-EN) koji korespondira sa uzorkom podatkovnog svojstva. Zadnja dva slova označavaju engleski jezik. Uzorak podatkovnih svojstava koji se po NeOn katalogu vodi pod oznakom LP-DP-01 (Suárez-Figueroa, Brockmans i Gangemi (2007)) prikazan je na slici 61, kao i na slici 17 za podatkovno svojstvo među osnovnim konceptima na strani 47.

U prirodnom jeziku (NL, engl. *Natural language*) podatkovna svojstva klase mogu se iskazati na slijedeći način: *"Svojstva klase Klasa su svojstvo1, svojstvo2 ... i svojstvoN."*
U LSP formalizmu ovo se općenito prikazuje na način:

```
Property/characteristic/attribute/features of NP<class> be
[PARA] [(NP<property>,)* and] NP<property>
```

Drugi iskaz može u prirodnom jeziku biti: *"Svi pripadnici klase Klasa su takvi, takvi ... i takvi."* U LSP formalizmu ovo se prikazuje ovako:

```
NP<class> be [(AP<property>,)*] and AP<property>
```



Slika 61: Logički uzorak podatkovnih svojstava (LP-DP-01)

Kao autore, NeOn navodi Elenu Montiel-Ponsoda, Guadalupe Aguado de Cea, Mari Carmen Suárez-Figueroa i Asunción Gómez-Pérez.

Na sličan način povezuje i uzorak razdvojenih klasa (LP-Di-01) sa iskazom u LSP formalizmu. Ovaj uzorak potpisuju isti autori, a vodi se pod oznakom LSP-Di-EN.
U prirodnom jeziku iskaz "agenti klase A se razlikuju od agenata klase B" se može formalno iskazati u LSP formalizmu:

```
NP<class> differ/be different/be differentiate from NP<class>
```

Za višestruko nasljeđivanje (LSP-MI-EN) LSP formalizam će biti

```
NP<subclass> [can]be NP<superclass> and NP<superclass>
```

dok će za objektno svojstvo i univerzalno ograničenje (LSP-OP-UR-EN) biti

```
NP<class> VB NEG(be/have/CATV/PART) just/only/exclusively NP<class>.
```

3.4.4. Uzorci dizajna primijenjeni na agente

3.4.4.1. Sustavi utemeljeni na znanju - agenti

Baze znanja u novije vrijeme nalaze primjenu u sustavima utemeljenima na znanju, odnosno agentima. Višeagentni sustavi (VAS)[††], primjer su praktične primjene ontologija, i upravo zbog toga i izvrsna motivacija za njihov razvoj, budući da pripadaju danas popularnom području *umjetne inteligencije*. Zbog toga ću opisati pojam **agenta** i **višeagentnog sustava** te navesti neke uzorke koji se mogu koristiti u oblikovanju višeagentnog sustava. "Agent je sve što može percipirati svoju okolinu putem određenih senzora i djelovati na tu istu okolinu preko određenih efektora". (Russell i Norvig, 1995, str. 31) Računala su u današnje vrijeme sposobna obavljati zadatke bez ljudske intervencije. Pri tome su i sami sustavi koji te zadatke obavljaju distribuirani i konkurentni a cjelokupno računarstvo svodi se na proces interakcije u kojem računalni sustavi djeluju neovisno, ali tako da zastupaju interese svojih vlasnika. Ovakvi sustavi se ponašaju poput društvene mreže u kojoj postoji pregovaranje, dogovaranje, konkurencija i parcijalni interesi. Društvenost agenata vodi do pojma *višeagentnog sustava* kojeg Glavic (2006) definira na slijedeći način:

> "Višeagentni sustav je slabo povezan skup entiteta namijenjenih rješavanju određenog problema (agenata) koji rade zajedno kako bi pronašli rješenje problema koji nadilazi sposobnosti rješavanja i znanje svakog od njih pojedinačno."
> Agenti se često susreću sa problemom pribavljanja potpunih informacija. U pristupačnom okruženju, agent može pribaviti sve njemu potrebne informacije kako bi donio odluku o daljnjim akcijama. No takva okruženja su rijetka. Sve češće se agenti implementiraju u nepristupačnim okruženjima, a današnja nastojanja da se usavrši tehnologija samovozećih automobila upravo je primjer agenata u nepristupačnom okruženju.

Agente treba razlikovati od objekata u objektnoj programerskoj paradigmi koji su najčešće pasivni i djeluju tek kad se to od njih zatraži. Agenti su sa druge strane reaktivni, proaktivni, sposobni su za društveno ponašanje i učenje. U smislu samog mehanizma kako su implementirani, to su dretve i procesi, sposobni komunicirati sa drugim takvim dretvama i procesima, bilo da se to čini nekim međuprocesnim sustavima za razmjenu podataka, bilo komunikacijom preko web utičnica (eng. *sockets*) odnosno mrežne infrastrukture.

Agenti su sposobni sami donositi odluke o svojim akcijama, ali jednako tako agente karakterizira i *znanje* o okolini. Agenti sa sposobnošću deduktivnog rezoniranja koriste logiku za opis teorije po kojoj se poduzima najbolja akciju u nekoj situaciji. Kada agent komunicira sa drugim agentima, također će koristiti konstrukte zapisane u određenoj ontologiji. Svi agenti koji razmjenjuju poruke, moraju poznavati tu ontologiju. To agentima omogućava razumijevanje poruke. Ovo se osobito odnosi na agente koji pripadaju otvorenim višeagentnim sustavima, koje razvijaju različiti timovi i različite organizacije. Kako bi agenti znali kontekst, unutar poruke mora biti navedena i ontologija. Često višeagentni sustavi nisu otvoreni, pa je vokabular potreban za razumijevanje poruka hardkodiran u samom programskom kodu agenta. Takve je agente teško integrirati, pa je daleko bolje rješenje eksplicitno navesti ontologiju (Obitko, 2007).

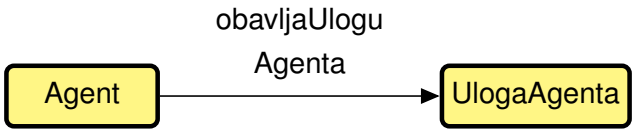
[††] U upotrebi je akronim MAS (eng. *Multi Agent System*)

Ovi uzorci preuzeti su iz tehničkog izvještaja GeoLink projekta, koji je derivat dostavljene dokumentacije OceanLink projekta, on-line platforme za razmjenu podataka i rezultata istraživanja znanstvenika na područjima vezanima uz oceanografiju. (Krishnadhi, Hu i Arko, 2015)

OceanLink projekt podupire tehnologije semantičkog weba, rudarenja podataka na webu i dijeljenje ideja preko weba (eng. *crowdsourcing*) kako bi se olakšala istraživanja, omogućila kolaboracija i ocjenjivanje doprinosa znanstvenika u istraživanjima. Projekt podupire Nacionalna znanstvena fondacija (eng. *National Science Foundation*, NSF), neovisna federalna agencija Sjedinjenih američkih država.

Uzorci navedeni ovdje dio su uzoraka primijenjenih za razvoj ontologije OceanLink projekta, i u tim uzorcima vrlo često nalazimo klase koje konceptualiziraju agente. To su sve uzorci sadržaja, i odnose se na konkretnu organizacijsku domenu u kojoj participiraju agenti. Od svih njih, izabrani su oni najzanimljiviji koji ilustriraju formalizme u logici prvog reda i u deskriptivnoj logici. Koristim i njihov pojednostavljen način grafičkog predočavanja.

3.4.4.2. Uzorak agenta



Slika 62: Uzorak agenta (prema Krishnadhi, Hu i Arko (2015))

Formalno, ovo je jednostavan uzorak u kojem objektno svojstvo sa određenom domenom i dosegom povezuje dvije klase. U konkretnoj primjeni povezuje klase agenata i njihovih uloga. Formalni opis u sintaksi deskriptivnih logika dan je slijedećim aksiomima:

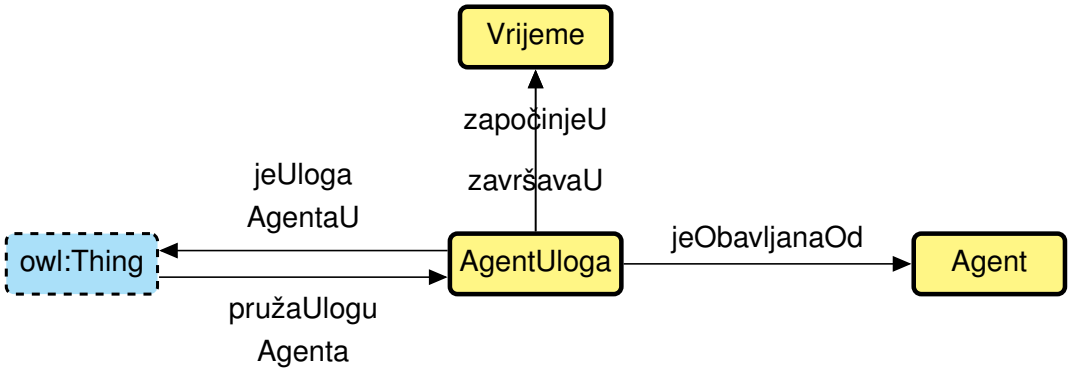
$$Agent \sqcap UlogaAgenta \sqsubseteq \bot$$
$$\exists obavljaUloguAgenta.UlogaAgenta \sqsubseteq Agent$$
$$Agent \sqsubseteq \forall obavljaUloguAgenta.UlogaAgenta$$

U ovom skupu aksioma, prvi govori o razdvojenosti klasa $Agent$ i $UlogaAgenta$. Ne može postojati individua koja pripada objema klasama. Druga dva retka se odnose na domenu i rang svojstva $obavljaUloguAgenta$. U deskriptivnim logikama oblik
$\exists obavljaUlogu.UlogaAgenta$ odgovara formuli
$\exists y(obavljaUlogu(x,y) \wedge UlogaAgenta(y))$ u logici prvog reda[‡‡], dok oblik
$\forall obavljaUlogu.UlogaAgenta$ odgovara formuli
$\forall y(obavljaUlogu(x,y) \rightarrow UlogaAgenta(y))$ u logici prvog reda.
Prema aksiomima sadržavanja oblik
$\exists obavljaUlogu.UlogaAgenta \sqsubseteq Agent$ odgovara formuli

[‡‡] U deskriptivnim logikama, binarne relacije kao što je $obavljaUlogu(x,y)$ su uloge u formalnom smislu, odnosno terminološko znanje koje dodatno opisuje strukturu domene odnosno njezinu konceptualnu shemu.

$\forall x(\exists y(obavljaUlogu(x,y) \wedge UlogaAgenta(y)) \rightarrow Agent(x))$.
Ako postoji uloga koju netko obavlja i ta je uloga $UlogaAgenta$, tada je taj netko agent.
Svatko tko obavlja ulogu agenta je agent. Agent je domena svojstva $obavljaUloguAgenta$.
Na sličan način oblik
$Agent \sqsubseteq \forall obavljaUlogu.UlogaAgenta$ odgovara formuli
$\forall x(Agent(x) \rightarrow (\forall y(obavljaUlogu(x,y) \rightarrow UlogaAgenta(y))))$.
Agent pripada onima koji kada obavljaju neku ulogu, svaka je takva uloga $UlogaAgenta$.
Svaki onaj tko je agent, može obavljati samo ulogu agenta. $UlogaAgenta$ je doseg svojstva $obavljaUloguAgenta$.

3.4.4.3. Uloga agenta



Slika 63: Uzorak uloge agenta (prema Krishnadhi, Hu i Arko (2015))

Ovaj uzorak opisuje ulogu koja ima temporalnu odrednicu: ona traje u vremenu. Formalni opis:

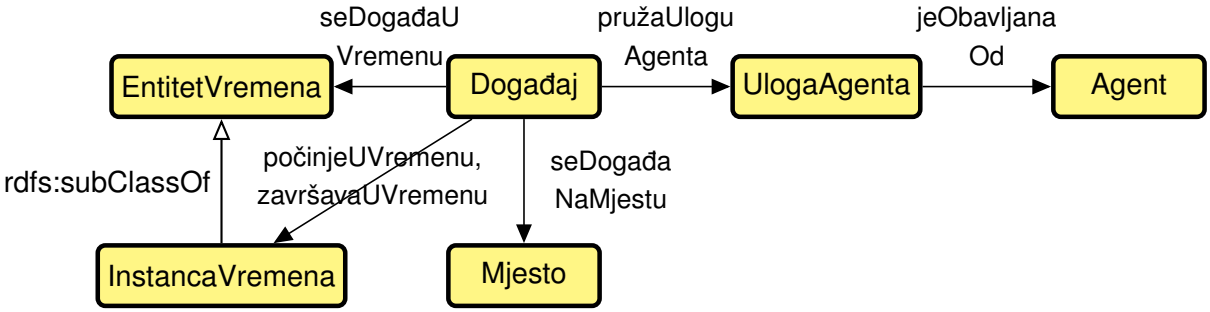
$$UlogaAgenta \sqsubseteq (= 1\ jeObavljanaOd.Agent) \sqcap (= 1\ jeUlogaAgentaU.\top) \sqcap$$
$$\sqcap (= 1\ započinjeU.Vrijeme) \sqcap (= 1\ završavaU.Vrijeme)$$
$$pružaUloguAgenta \equiv jeUlogaAgentaU^{-}$$

Uloge $pružaUloguAgenta$ i $jeUlogaAgentaU$ su inverzne uloge. uz to definiramo i aksiome zaključivanja, odnosno domene i dosege svojstava:

$$\exists jeObavljanaOd.Agent \sqsubseteq UlogaAgenta$$
$$UlogaAgenta \sqsubseteq \forall jeObavljanaOd.Agent$$
$$\exists započinjeU.Vrijeme \sqsubseteq UlogaAgenta$$
$$UlogaAgenta \sqsubseteq \forall započinjeU.Vrijeme$$
$$\exists završavaU.Vrijeme \sqsubseteq UlogaAgenta$$
$$UlogaAgenta \sqsubseteq \forall završavaU.Vrijeme$$
$$\exists jeUlogaAgentaU.\top \sqsubseteq UlogaAgenta$$
$$\top \sqsubseteq \forall obavljaUloguAgenta.UlogaAgenta$$

Sve klase su razdvojene:
$alldisjoint(UlogaAgenta, Agent, Vrijeme)$

3.4.4.4. Događaj



Slika 64: Uzorak događaja (prema Krishnadhi, Hu i Arko (2015))

Događaj se pojavljuje u nekom vremenu na nekom prostoru i može pružati neku ulogu koju obavlja agent. Formalni opis u sintaksi deskriptivnih logika:

$$Događaj \sqsubseteq \exists seDogađaNaMjestu.Mjesto \sqcap \exists seDogađaUVremenu.EntitetVremena$$

$$\begin{aligned} počinjeUVremenu &\sqsubseteq seDogađaUVremenu \\ završavaUVremenu &\sqsubseteq seDogađaUVremenu \\ InstancaVremena &\sqsubseteq EntitetVremena \end{aligned}$$

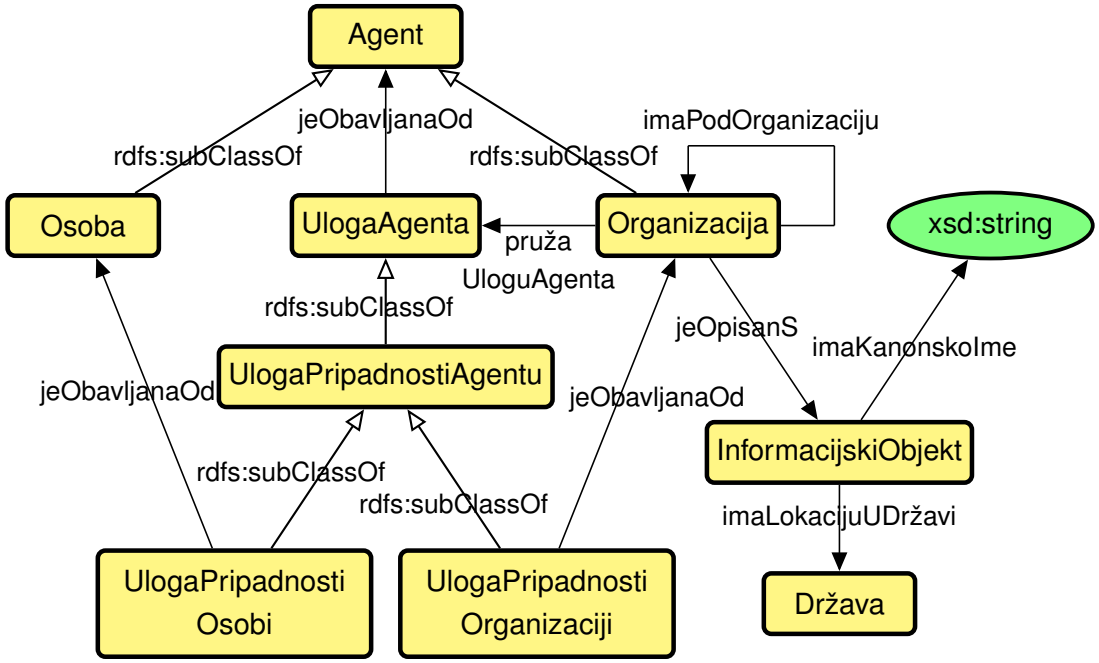
$$UlogaAgenta \sqsubseteq (= 1\ jeObavljanaOd.Agent)$$

$$\begin{aligned} \exists seDogađaNaMjestu.Mjesto &\sqsubseteq Događaj \\ Događaj &\sqsubseteq \forall seDogađaNaMjestu.Mjesto \\ \exists seDogađaUVremenu.EntitetVremena &\sqsubseteq Događaj \\ Događaj &\sqsubseteq \forall seDogađaUVremenu.EntitetVremena \\ \exists započinjeUVremenu.InstancaVremena &\sqsubseteq Događaj \\ Događaj &\sqsubseteq \forall započinjeUVremenu.InstancaVremena \\ \exists završavaUVremenu.InstancaVremena &\sqsubseteq Događaj \\ Događaj &\sqsubseteq \forall završavaUVremenu.InstancaVremena \\ \exists pružaUlogu Agenta.UlogaAgenta &\sqsubseteq Događaj \\ Događaj &\sqsubseteq \forall pružaUlogu Agenta.UlogaAgenta \\ \exists jeObavljanaOd.Agent &\sqsubseteq UlogaAgenta \\ UlogaAgenta &\sqsubseteq \forall jeObavljanaOd.Agent \end{aligned}$$

Razdvajanje klasa:
$alldisjoint(Događaj, Mjesto, EntitetVremena, UlogaAgenta, Agent)$

3.4.4.5. Organizacija

Ovo je nešto složeniji uzorak koji daje kontekst samom agentu. Organizacija pruža agentu ulogu, a agent je sa druge strane povezan sa osobnim i organizacijskim ulogama.



Slika 65: Uzorak organizacije (prema Krishnadhi, Hu i Arko (2015))

$Organizacija$ i $Osoba$ su lokalno definirani i podklase su lokalno definiranog $Agenta$. Posebne klase $UlogaPripadnostiOsobi$ i $UlogaPripadnostiOrganizaciji$ mogu biti obavljane isključivo od strane $Osobe$ odnosno $Organizacije$ respektivno. Organizaciji je pridružen $InformacijskiObjekt$ koji sadrži zapis o kanonskom imenu i državi u kojoj ona djeluje. Formalni opis u sintaksi deskriptivnih logika razdijeljen je naviše dijelova. Odnos organizacije i informacijskog objekta je u odnosu 1:1. Svaki informacijski objekt ima točno jedno kanonsko ime.

$$Organizacija \sqsubseteq Agent$$
$$Osoba \sqsubseteq Agent$$
$$Organizacija \sqsubseteq (= 1\ jeOpisanS.InformacijskiObjekt)$$
$$InformacijskiObjekt \sqsubseteq (= 1\ jeOpisanS^{-}.Organizacija)$$
$$InformacijskiObjekt \sqsubseteq (= 1\ imaKanonskoIme.xsd : string)$$
$$imaPodOrganizaciju \circ imaPodOrganizaciju \sqsubseteq imaPodOrganizaciju$$

Zadnji aksiom ukazuje na tranzitivno svojstvo $imaPodOrganizaciju$. Ovo znači da je podorganizacija podorganizacije neke organizacije, ujedno i podorganizacija te organizacije. U sof-

tverskom inženjerstvu ovakvo rekurzivno svojstvo povezujemo sa uzorkom *Composite* koji se koristi prilikom prikaza hijerarhije "dio-cjelina", pri čemu se jednostavni i složeni objekti uniformno tretiraju. U ontologijama odnosno DL sintaksi to su tranzitivna svojstva kao proširenja jezika $\mathcal{ALC}$.
U nastavku je opisana hijerarhija uloga:

$$UlogaPripadnostiAgentu \sqsubseteq UlogaAgenta$$
$$UlogaPripadnostiOsobi \sqsubseteq UlogaPripadnostiAgentu$$
$$UlogaPripadnostiOsobi \sqsubseteq (= 1\ jeObavljanaOd.Osoba)$$
$$UlogaPripadnostiOrganizaciji \sqsubseteq UlogaPripadnostiAgentu$$
$$UlogaPripadnostiOrganizaciji \sqsubseteq (= 1\ jeObavljanaOd.Organizacija)$$

Daljnja se pravila odnose na domene i dosege:

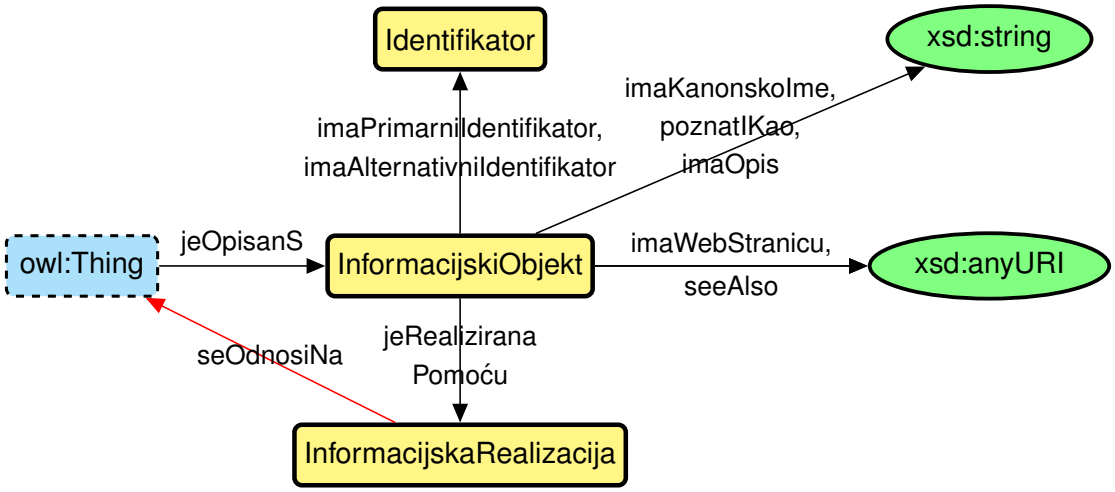
$$UlogaPripadnostiOsobi \sqsubseteq \forall jeObavljanaOd.Osoba$$
$$UlogaPripadnostiOrganizaciji \sqsubseteq \forall jeObavljanaOd.Organizacija$$
$$\exists jeObavljanaOd.Agent \sqsubseteq UlogaAgenta$$
$$UlogaAgenta \sqsubseteq \forall jeObavljanaOd.Agent$$
$$\exists omogućavaUloguAgenta.UlogaAgenta \sqsubseteq Organizacija$$
$$Organizacija \sqsubseteq \forall omogućavaUloguAgenta.UlogaAgenta$$
$$\exists imaPodOrganizaciju.Organizacija \sqsubseteq Organizacija$$
$$Organizacija \sqsubseteq \forall imaPodOrganizaciju.Organizacija$$
$$\exists jeOpisanS.InformacijskiObjekt \sqsubseteq Organizacija$$
$$Organizacija \sqsubseteq \forall jeOpisanS.InformacijskiObjekt$$
$$\exists imaLokacijuUDržavi.Država \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaLokacijuUDržavi.Država$$
$$\exists imaKanonskoIme.xsd : string \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaKanonskoIme.xsd : string$$

Razdvojene klase:

$$alldisjoint(Agent, UlogaAgenta, InformacijskiObjekt, Država)$$
$$Organizacija \sqcap Osoba \sqsubseteq \bot$$
$$UlogaPripadnostiOsobi \sqcap UlogaPripadnostiOrganizaciji \sqsubseteq \bot$$

3.4.4.6. Informacijski objekt

U uzorku organizacije, pojavljuje se Informacijski objekt koji entitetima pridružuje državu i kanonsko ime. Generalno, informacijski objekt može pridruživati i druge informacije.



Slika 66: Uzorak informacijskog objekta (prema Krishnadhi, Hu i Arko (2015))

Ovaj uzorak modelira dodatne informacije pridružene nekom informacijskom objektu kakav postoji i u DOLCE ontologiji, poput identifikatora, URI-ja web stranice, kanonskog i alternativnog imena, opisa, te logičke povezanosti sa nekim drugim sadržajem. Jednako tako, informacijski se objekt može pridružiti svemu kao opis, pa tako i agentu ili bilo kojem entitetu kojeg agent koristi u komunikaciji s drugim agentima. Njegove su realizacije različiti formati dokumenata ili strukture podataka u bazi. U uzorku se može primijetiti n-arna podatkovna relacija, gdje Informacijski objekt ima kanonsko ime i web stranicu. No jednako tako ima i objektnu n-arnu relaciju jer ima identifikatore kao objekte iz klase *Identifikator* i realizaciju iz klase *InformacijskaRealizacija* kao što je prikazano u OBO uzorcima SourceForge kataloga.

Aksiomi informacijskog objekta se formalno mogu zapisati na slijedeći način:

$$\top \sqsubseteq (\leq 1\ jeOpisanPomoću.InformacijskiObjekt)$$
$$InformacijskiObjekt \sqsubseteq (= 1\ jeOpisanS^{-}.\top)$$
$$InformacijskiObjekt \sqsubseteq \neg\exists jeOpisanS.InformacijskiObjekt$$
$$InformacijskaRealizacija \sqsubseteq \neg\exists jeOpisanS.InformacijskiObjekt$$
$$Identifikator \sqsubseteq \neg\exists jeOpisanS.InformacijskiObjekt$$
$$jeRealiziranaPomoću^{-} \circ jeOpisanS^{-} \sqsubseteq seOdnosiNa$$

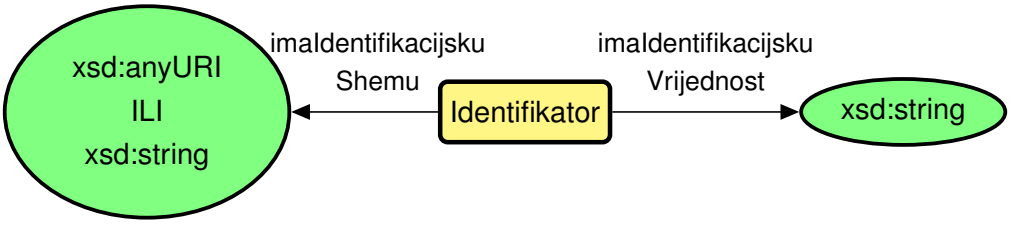
Aksiomi povezani sa domenama i dosezima slijede:

$$doseg(jeOpisanS) \sqsubseteq InformacijskiObjekt$$
$$domena(seOdnosiNa) \sqsubseteq InformacijskaRealizacija$$
$$\exists jeRealiziranaPomoću.InformacijskaRealizacija \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall jeRealiziranaPomoću.InformacijskaRealizacija$$
$$\exists imaKanonskoIme.xsd : string \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaKanonskoIme.xsd : string$$
$$\exists poznatIKao.xsd : string \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall poznatIKao.xsd : string$$
$$\exists imaOpis.xsd : string \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaOpis.xsd : string$$
$$\exists imaWebStranicu.xsd : anyURI \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaWebStranicu.xsd : anyURI$$
$$\exists seeAlso.xsd : anyURI \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall seeAlso.xsd : anyURI$$
$$\exists imaAlternativniIdentifikator \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaAlternativniIdentifikator$$
$$\exists imaPrimarniIdentifikator \sqsubseteq InformacijskiObjekt$$
$$InformacijskiObjekt \sqsubseteq \forall imaPrimarniIdentifikator$$

Razdvajanje klasa:
$alldisjoint(InformacijskiObjekt, InformacijskaRealizacija, Identifikator)$

3.4.4.7. Identifikator

Prema temeljnim postavkama i principima semantičkog weba, svaki bitan podatak trebao bi imati svoj URI identifikator, i sve može biti referencirano na semantičkom webu korištenjem raznih identifikatora. Uzorak *Identifikator* povezuje neki kataloški identifikator (na primjer međunarodni standardni knjižni broj (ISBN, eng. *International Standard Book Number*)) zapisan kao niz znakova, sa URI identifikatorom. Na taj se način bilo kojem objektu u fizičkom svijetu može pridružiti URI na semantičkom webu.



Slika 67: Uzorak identifikatora (prema Krishnadhi, Hu i Arko (2015))

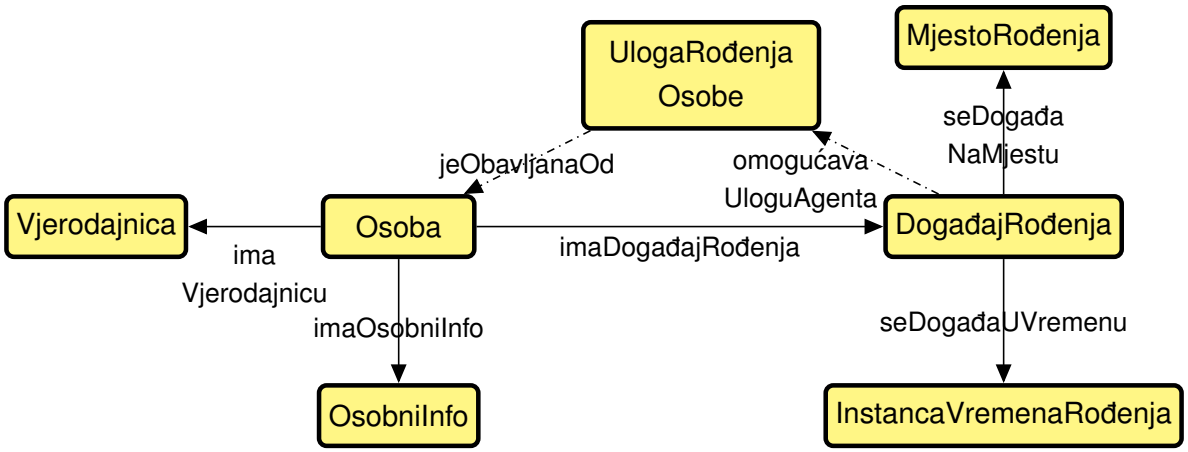
Ovaj jednostavni uzorak ima slijedeću aksiomatizaciju:

$$Identifikator \sqsubseteq (= 1\ imaIdentifikacijskuVrijednost.xsd : string)$$
$$Identifikator \sqsubseteq (\leq 1\ imaIdentifikacijskuShemu.(xsd : string \sqcup xsd : anyURI))$$
$$\exists imaIdentifikacijskuVrijednost.xsd : string \sqsubseteq Identifikator$$
$$Identifikator \sqsubseteq \forall imaIdentifikacijskuVrijednost.xsd : string$$
$$\exists imaIdentifikacijskuShemu.(xsd : string \sqcup xsd : anyURI)) \sqsubseteq Identifikator$$
$$Identifikator \sqsubseteq \forall imaIdentifikacijskuShemu.(xsd : string \sqcup xsd : anyURI))$$

Budući da u uzorku sudjeluje samo jedna klasa, nema, kao u uzorcima ranije, potrebe za razdvajanjem klasa. Identifikator je povezan n-arnom podatkovnom relacijom sa određenim identifikatorom i identifikacijskom vrijednošću.

3.4.4.8. Osoba

Još jedan zanimljiv uzorak iz GeoLinkove jezgrene ontologije sa stanovišta logičkog formalizma je uzorak osobe.



Slika 68: Uzorak osobe (prema Krishnadhi, Hu i Arko (2015))

U ovom uzorku bitan dio je Vjerodajnica (eng. *Credential*), budući da se u dinamičkim uvjetima pretpostavlja da osoba većinu atributa, pa čak i svoje ime ne mora zadržati stalnim. Prema modelu, svaka osoba je agent i ovaj uzorak se može poravnati sa uzorcima $Agent$ i $UlogaAgenta$. Osobi je pridružen događaj rođenja, i atributi povezani s tim događajem kao što su vrijeme i mjesto rođenja su konstantni. $DogađajRođenja$ instancira ulogu $UlogaAgenta$ koja je instanca uloge $UlogaRođenjaOsobe$. Crtkane poveznice za svojstva $jeObavljanaOd$ i $omogućavaUloguAgenta$ označavaju neodređenu domenu i doseg takvih svojstava.

Aksiomatizacija:

$$\begin{aligned} Osoba &\sqsubseteq (=1\ imaDogađajRođenja.DogađajRođenja) \\ DogađajRođenja &\sqsubseteq (=1\ imaDogađajRođenja^{-}.Osoba) \\ DogađajRođenja &\sqsubseteq (=1\ seDogađaNaMjestu.MjestoRođenja) \\ &\quad \sqcap (=1\ seDogađaUVremenu.InstancaVremenaRođenja) \end{aligned}$$

Ovi aksiomi označavaju da je svaka osoba rođena na jednom mjestu u jednom vremenskom trenutku. $Događajrođenja$ povezan je sa ulogom $UlogaRođenjaOsobe$ odnosnom 1:1, a taj odnos omogućava svojstvo $omogućavaUloguAgenta$. Ulogu $UlogaRođenjaOsobe$ obavlja točno jedna osoba.

Zanimljivi dio ovog uzorka je i što može poslužiti kao primjer razlike izražajnosti logike 1. reda, deskriptivnih logika i samog OWL-2 jezika. Da bi se opisalo kako je osoba rođena prema događaju rođenja, gore navedeni aksiomi uz određivanje domene i dosega svojstava $omogućavaUloguAgenta$ i $imaDogađajRođenja$ trebaju biti dopunjeni pravilom koje se u formalizmu logike 1. reda zapisuje kao:

$$UlogaRođenjaOsobe(x), omoguávaUloguAgenta(y,x), imaDogađajRođenja(z,y) \rightarrow jeObavljanaOd(x,z)$$

U deskriptivnim logikama ovo se pravilo može zapisati na slijedeći način

$$\begin{aligned} UlogaRođenjaOsobe &\equiv \exists R_{UlogaRođenjaOsobe}.Self \\ R_{UlogaRođenjaOsobe} \circ omogućavaUloguAgenta^{-} \circ imaDogađajRođenja^{-} &\sqsubseteq jeObavljanaOd \end{aligned}$$

Kako se navodi u Krishnadhi, Hu i Arko (2015), ova ulančana svojstva vode na kršenje ograničenja regularnosti za objektna svojstva u OWL-2. Zato se uzorak zapisuje na sigurniji način.

Aksiomi vezani za domene i dosege za svojstva slijede:

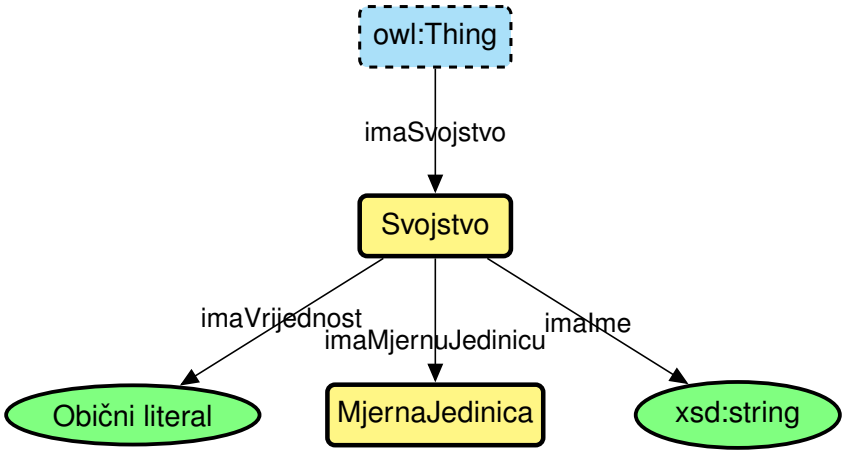
$$\exists imaDogađajRođenja.DogađajRođenja \sqsubseteq Osoba$$
$$Osoba \sqsubseteq \forall imaDogađajRođenja.DogađajRođenja$$
$$\exists imaOsobniInfo.OsobniInfo \sqsubseteq Osoba$$
$$Osoba \sqsubseteq \forall imaOsobniInfo.OsobniInfo$$
$$\exists imaVjerodajnicu.Vjerodajnica \sqsubseteq Osoba$$
$$Osoba \sqsubseteq \forall imaVjerodajnicu.Vjerodajnica$$
$$\exists seDogađaNaMjestu.MjestoRođenja \sqsubseteq DogađajRođenja$$
$$DogađajRođenja \sqsubseteq \forall seDogađaNaMjestu.MjestoRođenja$$
$$\exists seDogađaUVremenu.InstancaVremenaRođenja \sqsubseteq DogađajRođenja$$
$$DogađajRođenja \sqsubseteq \forall seDogađaUVremenu.InstancaVremenaRođenja$$
$$UlogaRođenjaOsobe \sqsubseteq \forall seObavljaOd.Osoba$$
$$\exists omogućavaUlogu Agenta.UlogaRođenjaOsobe \sqsubseteq DogađajRođenja$$

Razdvojene klase:

$$alldisjoint(Osoba, Vjerodajnica, OsobniInfo, DogađajRođenja, UlogaRođenjaOsobe,$$
$$InstancaVremenaRođenja, MjestoRođenja)$$

3.4.4.9. Uzorak vrijednosti svojstva

Ovaj uzorak je jedan od najvažnijih uzoraka u eksperimentalnom radu, i može se primijeniti bilo gdje u znanstvenim istraživanjima, gdje je potrebno iskazati vrijednost nekog svojstva zajedno sa mjernom jedinicom. Dakako, postoje slični uzorci i u DOLCE ontologijama, koji su još više prilagođeni mjerama, ali ovaj jednostavan uzorak je dobra ilustracija kako je moguće oblikovati mjerene vrijednosti u nekom modelu. Uzorak se može pridružiti bilo kojoj mjerenoj pojavi, i zato je povezan sa OWL klasom $owl : Thing$. Za razliku od informacijskog objekta, kojem je sličan, bliskiji je fizikalnim svojstvima, gdje je bitno znati određene skalarne vrijednosti zajedno sa mjernim jedinicama na koje se odnose. Informacijski objekt može sadržavati bilo kakve opisne informacije poput alternativnog imena, URI-ja, i sličnog. Svojstva u formalno-filozofskom smislu treba razlikovati od relacija. Relacije uspoređuju više objekata, i obično su n-arne, dok je svojstvo unarno (na primjer: za brzinu svjetlosti je $3 \cdot 10^8 m/s$ svojstvo, dok je usporedba "Ahil je brži od kornjače" relacija (binarna), kao što je i relacija "Ahil je brži od kornjače i puža" (ternarna relacija)). Agenti često razmjenjuju podatke koji sadrže razne mjerene vrijednosti. Upravo zato je ovaj uzorak bitan i pri dizajnu ontologija koje će koristiti agenti u svojoj komunikaciji.



Slika 69: Uzorak vrijednosti svojstva (prema Krishnadhi, Hu i Arko (2015))

Svojstvo se povezuje sa samo jednom stvari, i ne može se povezivati sa drugim svojstvom. Aksiomatizacija:

$$\top \sqsubseteq (forall ima Svojstvo.Svojstvo)$$
$$Svojstvo \sqsubseteq (= 1\ imaSvojstvo^{-}.\top) \sqcap \neg \exists imaSvojstvo^{-}.Svojstvo$$

Domena i doseg svojstva $imaMjernuJedinicu$:

$$\exists imaMjernuJedinicu.MjernaJedinica \sqsubseteq Svojstvo$$
$$Svojstvo \sqsubseteq \forall imaMjernuJedinicu.MjernaJedinica$$
$$\exists imaIme.xsd : string \sqsubseteq Svojstvo$$
$$Svojstvo \sqsubseteq \forall imaIme.xsd : string$$
$$dom(imaVrijednost) \sqsubseteq Vrijednost$$

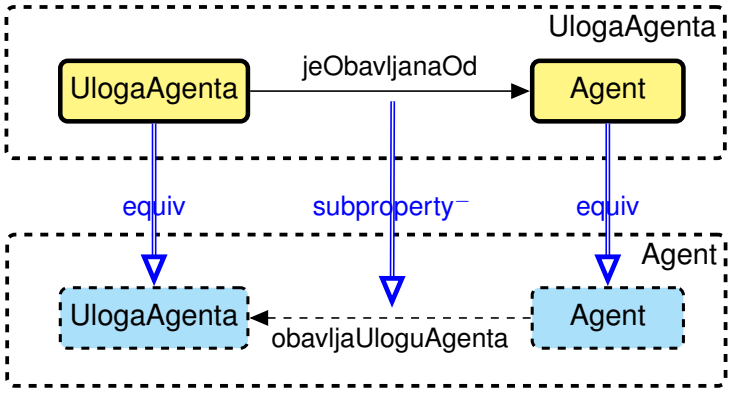
Uzorak karakterizira kombinacija n-arnih podatkovnih i vrijednosnih relacija u isto vrijeme. Ime i vrijednost su literali povezani podatkovnom relacijom, a mjerna jedinica je ovdje objekt iz klase mjernih jedinica. Nije loše mjerne jedinice zapravo imati uređene kao klase, pa čak i ontologije, budući da su one standardizirane (na primjer SI sustav), a mogu i međusobno biti u određenim relacijama.

Uzoraka u okviru OceanLinkovih projekta ima još mnogo, i navoditi ih sve ne bi imalo smisla, budući da je ovdje bitno ilustrirati samo način kako se oni prikazuju u sintaksi deskriptivnih logika i jezika OWL, te kako u njima sudjeluje koncept agenta. Stoga ću se u nastavku osvrnuti još samo na problem poravnanja uzoraka.

3.4.4.10. Poravnavanje uzoraka

Budući da se klase i svojstva mogu nasljeđivati, odnosno mogu svrstavati u određene hijerarhije, moguće je i elemente jednog uzorka poravnati u odnosu specijalizacija/generalizacija

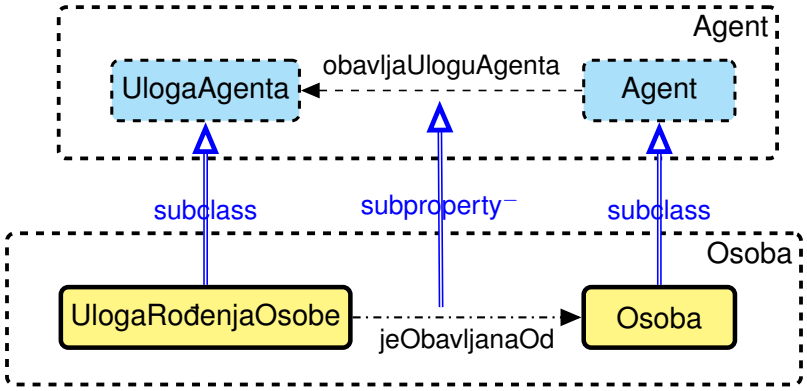
ili ekvivalencija sa elementima drugog uzorka. Katalog sadrži i naputke za poravnanje uzoraka, što bi također odgovaralo uzorcima poravnanja odnosno mapiranja, što je podklasa uzoraka korespondencije. Navodim nekoliko jednostavnih primjera: poravnavanje uzorka $Agent$ i $UlogaAgenta$ (slika 70) i poravnanje uzorka $Agent$ i $Osoba$ (slika 71) .



Slika 70: Poravnanje uzorka agenta i uzorka uloge agenta (prema Krishnadhi, Hu i Arko (2015))

Aksiomatika:

$$UlogaAgenta \equiv UlogaAgenta$$
$$Agent \equiv Agent$$
$$jeObavljanaOd \sqsubseteq obavljaUloguAgenta^{-}$$



Slika 71: Poravnanje uzorka agenta i uzorka osobe (prema Krishnadhi, Hu i Arko (2015))

Aksiomatika:

$$Osoba \sqsubseteq Agent$$
$$UlogaRođenjaOsobe \sqsubseteq UlogaAgenta$$
$$jeObavljanaOd \sqsubseteq obavljaUloguAgenta^{-}$$

3.5. Primjena uzoraka u praksi

Uzorci dizajna u softverskom inženjerstvu vrijedan su metodološki alat u profesionalnom razvoju softvera koji nosi mnoge benefite poput koordinacije procesa razvoja, jednostavnije klasifikacije fragmenta dizajna te mogućnosti stvaranja točaka preklapanja u modulima dizajna, uz čiju pomoć će se takvi moduli moći sigurno ugrađivati u različita okruženja.(Cline, 1996, str. 47)

Ipak, neka istraživanja pokazuju da unatoč benefitima, primjena uzoraka dizajna u softverskom inženjerstvu i nije uvijek idealna, pa da je često ugrožena evolutivnost, te povećana zahtjevnost prilikom održavanja takvog softvera, posebno ako je dizajn prepun isprepletenih uzoraka.(Khomh i Guéhéneuc, 2008)5. Uzorci, ako se koriste neumjereno, često narušavaju eleganciju dizajna, čineći ga kompleksnim za razumijevanje. Izrada modula koji će se uklapati po određenim točkama često je znatno skuplja od uobičajenog dizajna, pa zajednica nevoljko prihvata takva nefunkcionalna svojstva softvera.(Cline, 1996, str. 48)

Sa druge strane u turbulentnim poslovnim okruženjima kada je prilagodljivost softverskih sustava imperativ, uzorci dizajna nalaze svoju svrhu. (Cline, 1996, str. 48)

Uzorci dizajna u softverskom inženjerstvu već su duže vrijeme u primjeni i premda su postigli su određenu zrelost, još uvijek postoje dvojbe oko segmenata njihove primjene. U usporedbi s njima, uzorci dizajna ontologija nisu dosegli tu razinu niti zrelosti niti primjene, pa je potrebno procijeniti njihovu raširenost i primjenjivost u konkretnim projektima. Analizirajući NeON i OBO kataloge, mogli smo vidjeti da su prisutni u znanstveno-istraživačkim projektima koji se odnose na oceanografiju, biomedicinska istraživanja, višeagentne sustave i umjetnu inteligenciju. Jedan od najboljih primjera je Geolink projekt semantike i povezanih podataka za geoznanstvena istraživanja povezan sa projektom *EarthCube*. Geolink koristi kolekcije standardnih protokola, formata i vokabulara sa svrhom razmjene prikupljenih podataka i pružanja usluga istraživačima sa područja geoznanosti, računalnih znanosti, bibliotekarstva itd.(GeoLink, 2019).

Njihov repozitorij podataka uključuje podatke sa terenskih istraživanja, laboratorijske analize, članke u znanstvenim publikacijama, prezentacije sa konferencija, izvještaje i disertacije i drugo. (GeoLink, 2019).

U razvoju infrastrukture semantičkog weba, kao popratni rezultat je nastao i čitav katalog uzoraka dizajna ontologija, uglavnom povezanih sa geoznanstvenom domenom. Neke od ovih uzoraka sam spomenuo analizirajući agente i uzorke povezane s njima. Geolinkovi suradnici i tvorci Geolinkove modularne oceanografske ontologije poput Adila Krishnadija i Yingjie Hua objavljuju uzorke dizajna i u NeON katalogu. Primjer su uzorci sadržaja poput Trajektorije, Jezgre događaja i Relativne povezanosti (Krishnadi) te LCA uzorka i Uzorka za zaključivanje o aktivnostima (Hu).

3.5.1. Upotreba uzoraka dizajna ontologija u znanstvenim projektima

U slučaju bio-informatike, pokazuje se da su ontologije ključni faktor upravljanja znanjem, budući da bio-ontologije moraju predstavljati znanje na vjeran i robustan način. (Aranguren i dr., 2008, str. 1)

Bio ontologije implementiraju različite jezike za prikaz znanja poput OWL-a ali i OBO-a. OBO uzorci su ujedno i temelj analize u ovom radu, budući da nisu opsežni, a pokrivaju elementarne konstrukte u razvoju ontologija poput listi, stabala, n-arnih relacija i tako dalje. Jedna od aplikativnih ontologija iz područja bioinformacijskih znanosti je i *Cell Cycle Ontology*, vršna (top-level) ontologija koja obuhvaća podatke iz brojnih izvora poput:

- GO (ontologija gena)
- RO (ontologija odnosa tj. relacija)
- IntAct (izvor podataka povezanih sa molekularnim međudjelovanjima)
- NCBI taksonomija (baza podataka Nacionalnog centra za biotehnološke informacije)
- UnitProt (univerzalni izvor proteinskih sekvenci)

Ontologiju je u OWL obliku moguće preuzeti na adresi
`https://www.ebi.ac.uk/ols/ontologies/cco`
Ontologija gena također je izgrađena na OBO uzorcima, i može se preuzeti na adresi
`http://geneontology.org/docs/download-ontology/#go_obo_and_owl`
Uključuje aksiome iz drugih ontologija poput *ChEBI* ontologije kemijskih entiteta od interesa bioznanstvenicima (eng. *Chemical Entities of Biological Interest*), ontologije stanica i ontologije anatomskih struktura životinja Uberon. Vršne ontologije su korisne za razvoj prakse dobrog modeliranja, ali sa druge strane, otežavaju specifične zahtjeve i poglede pojedinih grana istraživanja.(Aranguren i dr., 2008)

U znanstvenim krugovima i projektima očito se uzorci aktivno koriste pri izradi vokabulara baza znanja, ali se prema popisima imena i u NeON katalogu i u drugim katalozima i projektima može uočiti da se time bavi uski krug stručnjaka poput Alda Gangemija, Valentine Presutti, Eve Blomqvist, Adila Krishnadija, Pascala Hitzlera i drugih.

3.5.2. Prednosti upotrebe uzoraka dizajna

Prema mišljenju autora koji se bave ontologijama, a što je još uvijek predmet istraživanja, prednosti korištenja uzoraka dizajna u razvoju ontologija mogu se klasificirati na slijedeći način: (Aranguren i dr., 2008)

- Dizajn
  - Bogato i granularno modeliranje - ontologije imaju bogatiju aksiomatizaciju koja omogućava modeliranje domenskog znanja na finijoj razini, osobito u biološkim istraživanjima gdje ontologije produbljuju sama znanja koja prikazuju
  - Semantičko učahurivanje - kompleksna semantika se može ugraditi u uzorak i time pojednostaviti njezino korištenje. Dobro formiran uzorak neće zahtijevati od korisnika da prati kompleksnu semantiku, vjerujući u provjereni uzorak. Primjer ovoga možemo naći u uzorku opisa definirane klase (OBO SourceFoge, slika 37, str. 67), kao i u uzorku vezane liste iz ontologije gena (slika 41, str. 71).

  - Robusnost i modularnost
  - Bolje i brže zaključivanje - bogatija aksiomatika koja pomaže u zaključivanju lakše se postiže upotrebom uzoraka
  - Poravnanje - povezivanje ontologija je složen posao. Tu uzorci imaju veliku ulogu jer je za njih ne samo definirano poravnanje i često navedeno u katalogu (kao na primjer kod GeoLinkovih uzoraka (slika 70, str. 107), već postoje i posebni uzorci poravnanja.

- Implementacija
  - Fokusiran razvoj - razvojni inženjer se može posvetiti detaljima domenskog problema
  - Upotreba alata - postoje programski alati koji automatski grade dijelove ontologija koristeći određene uzorke, pri čemu pružaju i savjet kako razraditi pojedini dio ontologije
  - Brzo prototipiranje - uzorci omogućavaju brzu izradu prototipa koji olakšava daljnje odlučivanje o razvoju
  - Reinženjering - uzorci olakšavaju refaktoriranje ontologija
  - Bolja komunikacija - kao i kod softverskog inženjerstva, razvojni inženjeri lakše komuniciraju ako govore jezikom uzoraka. Uzorci imaju definirana imena i značenje, pa se smanjuje potreba za detaljnim elaboriranjem nekog rješenja
  - Dokumentirano modeliranje - koristeći uzorke, jednostavnije je u dokumenti opisati određeno rješenje, budući se koristi terminologija uzoraka
  - Olakšano razumijevanje prihvaćanje novih mogućnosti jezika za prikaz znanja. Jezici poput OWL-a nadograđuju se novim izražajnim elementima, a uzorci koji se s njima također razvijaju, olakšavaju dizajnerima ontologija učenje kako te nove mogućnosti koristiti.

3.5.3. Usporedba kataloga

U radu analiziram više kataloga: katalog otvorenih biomedicinskih ontologija, GeoLinkov katalog uzoraka povezan u ovom radu sa agentima i katalog NeON projekta. Svaki od tih kataloga ima neke svoje specifičnosti. Tako je NeON katalog najopsežniji, i usuglašen sa podjelom obitelji uzoraka dizajna kao što je predloženo u literaturi (Gangemi i Presutti, 2009, str. 5).

Sadrži i uzorke koji nisu vezani za problemsku domenu i sam sadržaj ontologija, kao što su logički i leksičko-sintaktički uzorci. Ipak, u segmentu uzoraka sadržaja, vlada popriličan nered. Uzoraka ima ogroman broj i teško ih je povezivati. Ostali segmenti su dizajneru ontologija od velike pomoći.

Katalog biomedicinskih ontologija je mali katalog, i uzorci se odnose na osnovne elemente ontologija. Bez obzira što je ovaj katalog namijenjen biomedicinskim ontologijama kao i ontologijama gena, svi uzorci su univerzalno primjenjivi. Velik dio njih se nalazi u Ontologiji o

pizzama poput n-arnih relacija, zatvaranja, particije klasa i niza povezanih svojstava. Uzorak vezane liste, potreban za pretraživanje sekvenci genoma, može se koristiti i za druge svrhe, kao što je na primjer pretraživanje uzoraka u slikama, elektronskih sklopova i slično.

GeoLinkov katalog izvorno namijenjen konceptualizaciji eko sustava, može se povezati sa uzorcima arhitekture poslovnih sustava Martina Fowlera, a univerzalno je primjenjiv u ontologijama koje prikazuju konceptualizaciju organizacija, poslovnih okruženja i temporalnih baza podataka. Ovo je također opsežan katalog, koji sadrži i neke vrlo kompleksne uzorke kao što je uzorak *Krstarenja kao niza događaja: trajektorija i uloge agenta* (Krishnadhi, Hu i Arko, 2015, str. 57).
Katalog sadrži opis uzoraka i grafički i pomoću DL formalizma.

4. Softverski alati za razvoj ontologija

Premda RDF sintaksa omogućava veliku izražajnost te se ontologije mogu barem u teoriji razvijati i korištenjem običnog uređivača teksta, za svaki složeniji zadatak potrebni su posebni alati za kreiranje, implementaciju i održavanje ontologija. Prilikom kreiranja ontologija, često prvi rezultati nisu zadovoljavajući, pa su potrebne prepravke i korekcije. Ovo je iterativni proces koji postupno vodi ka sve kvalitetnijem modelu realne domene. Alati moraju omogućiti jednostavno nadopunjavanje i mijenjanje hijerarhije klasa i svojstava, ugradnju i promjenu ograničenja i aksioma te provjeru konzistentnosti ontologija. Razvoj ontologije može uključivati slijedeće faze prema Kapoor i Sharma (2010):

- Odabir problemske domene i područja interesa unutar problemske domene
- Razmatranje ponovne iskoristivosti - korištenje uzoraka
- Pronalaženje važnih termina
- Definiranje klasa i hijerarhija klasa
- Definiranje svojstava klasa i ograničenja
- Kreiranje instanci klasa (individua)

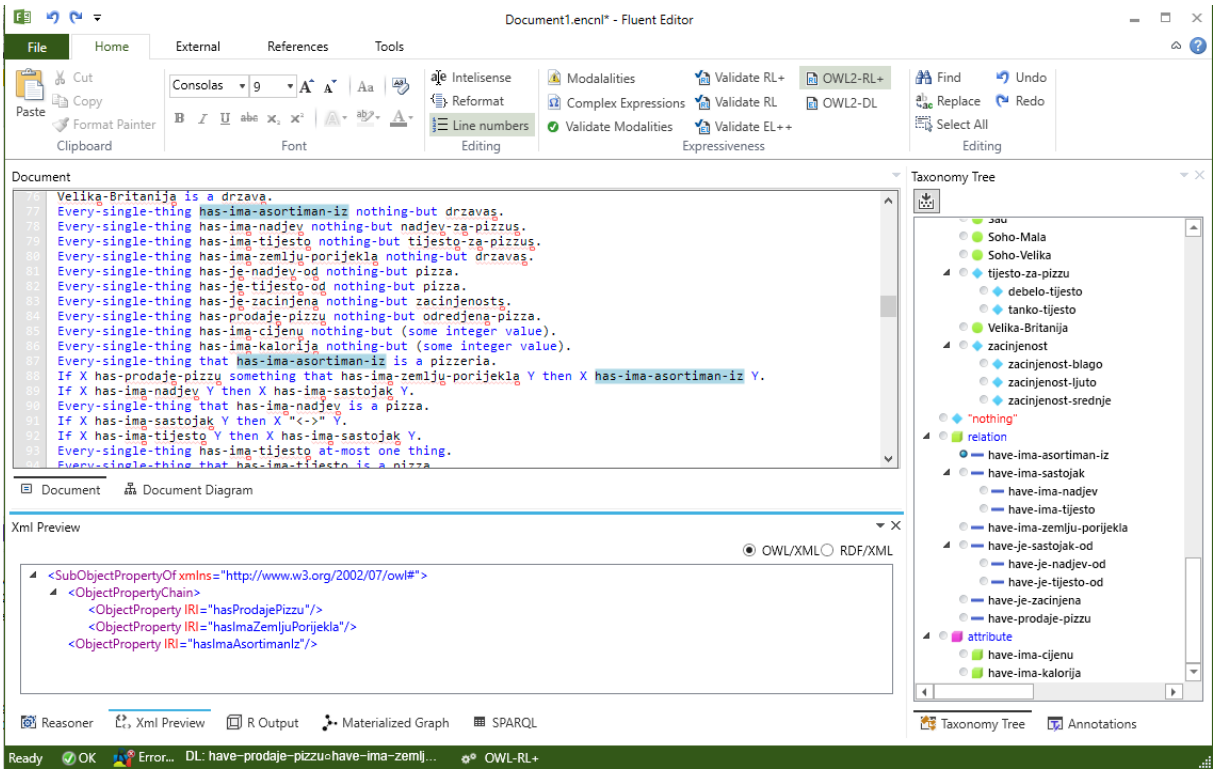
4.1. Alati općenito

Ukratko ću nabrojati neke od dostupnih alata za razvoj ontologija:

- Protégé - besplatni uređivač ontologija i baza znanja razvijen na odjelu za biomedicinska i informatička istraživanja sveučilišta Stanford u suradnji sa sveučilištem u Manchesteru.
- TopBraidComposer - komercijalni proizvod tvrtke TopQuadrant koji nudi napredne mogućnosti modeliranja semantičkog weba uz konverziju podataka i integrirano grafičko sučelje. U potpunosti je usklađen sa W3C standardima, prema kojima omogućava razvoj, održavanje i testiranje modela i baza znanja.
- NeON toolkit - softversko okruženje za razvoj ontologija, nastalo u sklopu NeOn Projekta. Kao open-source aplikacija može se besplatno preuzeti sa `neon-tolkit.org` stranica, zajedno sa 45 plug-inova koji pokrivaju različite aktivnosti prilikom inženjerskog razvoja ontologija.
- SWOOP - OWL uređivač ontologija razvijen na sveučilištu Maryland koji u današnje vrijeme održava Clark & Parsia LLC zajedno sa IBM Watson Research centrom te sveučilištem u Manchesteru. Ovo je vrlo jednostavni pretraživač i uređivač ontologija, daleko skromnijih mogućnosti u odnosu na Protégé ili TopBraid Composer.
- Fluent Editor - moćan uređivač ontologija koji je sposoban manipulirati složenim ontologijama koristeći prirodni jezik što je alternativa OWL uređivačima utemeljenima na XML-u.

Ovo intuitivno sučelje koristi jezik Ontorion Controlled Natural Language (OCNL), koji je potpuno prilagođen OWL-2, RDF i SWRL sintaksi, a jednako tako i upitnom jeziku SPARQL (slika 72).

- OBO Editor - razvijen za potrebe uređivanja genetskih ontologija odnosno kotroliranih vokabulara u okviru američkog projekta otvorenih biomedicinskih ontologija.



Slika 72: Fluent Editor na primjeru Ontologije o pizzama

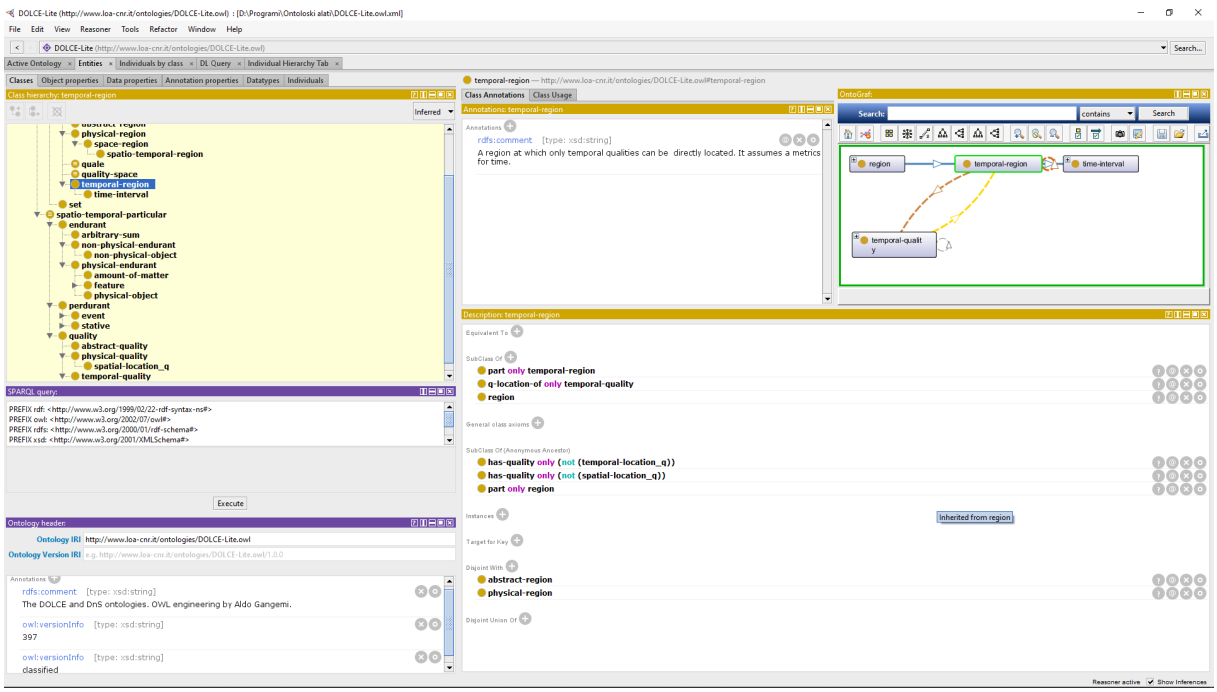
Detaljno ću opisati alat Protégé u kojem sam i izveo projektne ontologije, te prikazati dvije pedagoške ontologije: Ontologiju o pizzama i DOLCE ontologiju. Pri razvoju obiju ontologija koristili su se uzorci dizajna.

4.2. Razvoj i analiza ontologija u alatu Protégé

Protégé je napisan je u jeziku Java (Java Swing grafički API) i po svojoj strukturi je softverski okvir sa velikom podrškom u pluginovima i čak 300 000 registriranih korisnika, pa se može smatrati vodećim alatom za razvoj ontologija. Omogućava jednostavno kreiranje, modificiranje, dijeljenje ontologija, njihovo postavljanje na web te kolaborativni rad u smislu pregleda i uređivanja. U potpunost je usklađen sa najnovijim W3C standardima OWL 2 jezika, te podržava deduktivne klasifikatore za validaciju modela i izdvajanje novih informacija iz postojećih ontologija. Protégé podržava RDF/XML, Turtle, OWL/XML, OBO, Mancherster OWL, JSON-DL i LaTeX sintaksu pri čitanju i zapisivanju ontologija. Opremljen je alatima za grafički prikaz ontologija u vidu grafova poput OWL Viz (baziran na GraphViz softveru za vizualizaciju koji se mora instalirati posebno) i OntoGrapha, te tabličnih prikaza poput Class matrixa i Property matrixa.

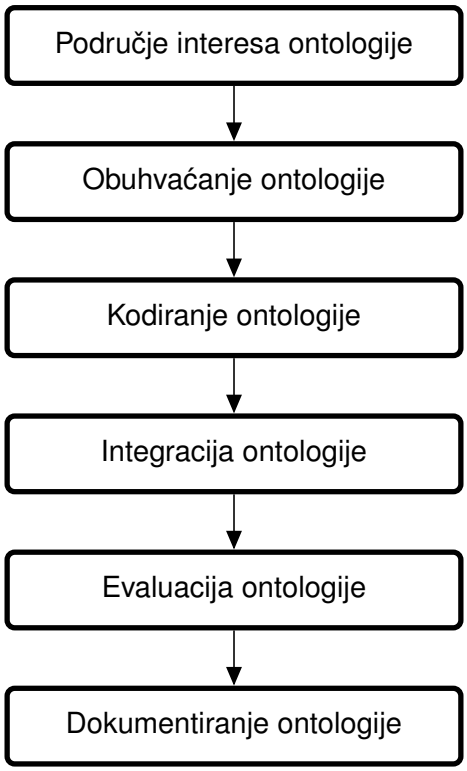
Ima potpuno prilagodljivo korisničko sučelje, pri čemu su boje editora povezane sa vrstom ontoloških entiteta:

- OWL klase i DL upiti - prljavo žuta, definicije i hijerarhija klasa te DL upiti sa filtriranjem nadklasa, ekvivalentnih klasa, podklasa i instanci
- svojstva - definicije i hijerarhija svojstava.
  - objektna svojstva - zeleno plava boja. Mogu bili funkcionalna, inverzno funkcionalna, tranzitivna, simetrična, asimetrična, refleksivna i irefleksivna.
  - podatkovna svojstva - zelena boja. Mogu biti funkcionalna svojstva.
  - anotacijska svojstva - narančasta.
- tipovi podataka - vinsko crvena
- individue - ljubičasta
- metapodaci - plavo ljubičasta, ontologije, SPARQL upiti i zaglavlja ontologija



Slika 73: Protégé na primjeru DOLCE Lite ontologije

Razvoj ontologije zahtijeva određenu metodologiju. Kao i u mnogim područjima softverskog inženjerstva i ovdje je proces razvoja podložan improvizacijama. Ipak moguće je dati smjernice ka metodološkom pristupu koji je preduvjet za inženjersku praksu. Prema jednoj takvoj metodološkoj smjernici faze razvoja ontologija prikazane su na slici 74.



Slika 74: Metodologija konstrukcije ontologije Kapoor i Sharma (2010)

4.2.1. Ontologija o pizzama

Ontologija o pizzama je vježbovni projekt (eng. *tutorial*) razvoja ontologija, razvijen od strane BioHealth Informatics Group iz Manchestera. Demonstrira većinu mogućnosti ontoloških alata, pri čemu je razumljiva korisniku koji tek uči kreirati ontologije. U primjeni je duže vrijeme i stoga je detaljno testirana i bliska korisnicima ontoloških alata, osobito korisnicima Protégéa za koje je i osmišljena. Cjelokupna ontologija dostupna je u izvornom kodu na adresi `https://protege.stanford.edu/ontologies/pizza/pizza.owl` .
Pojednostavljena ontologija o pizzama, kakva je upotrebljavana na vježbama iz kolegija *Baze znanja i semantički web*, a na kakvu se referiram u ovom radu, dostupna je u mom GitHub repozitoriju na adresi:
`https://github.com/elvispopovic/Ontologije`.
Potonja ontologija iako ima manje klasa tj. vrsta pizza, uključuje podatkovna svojstva vezana uz cijenu i kalorije pojedine pizze, kojih u originalnoj ontologiji nema. Druga suštinska razlika je što ova ontologija uključuje i ulančana svojstva *imaZemljuPorijekla* i *prodajePizzu* u novo svojstvo *imaAsortimanIz*. Ove dopune čine i dva uzorka dizajna: n-arnu podatkovnu relaciju i niz povezanih svojstava. Elementi ontologije o pizzama daju presjek uobičajenih postupaka i koncepata pri razvoju ontologija. Slijedeći vježbu mogu se izvući slijedeći koncepti:

- kreiranje hijerarhija klasa u samom stablu klasa ili u posebnom editoru koristeći mogućnost predefiniranja prefiksa i sufiksa
- deklariranje razdvojenih klasa

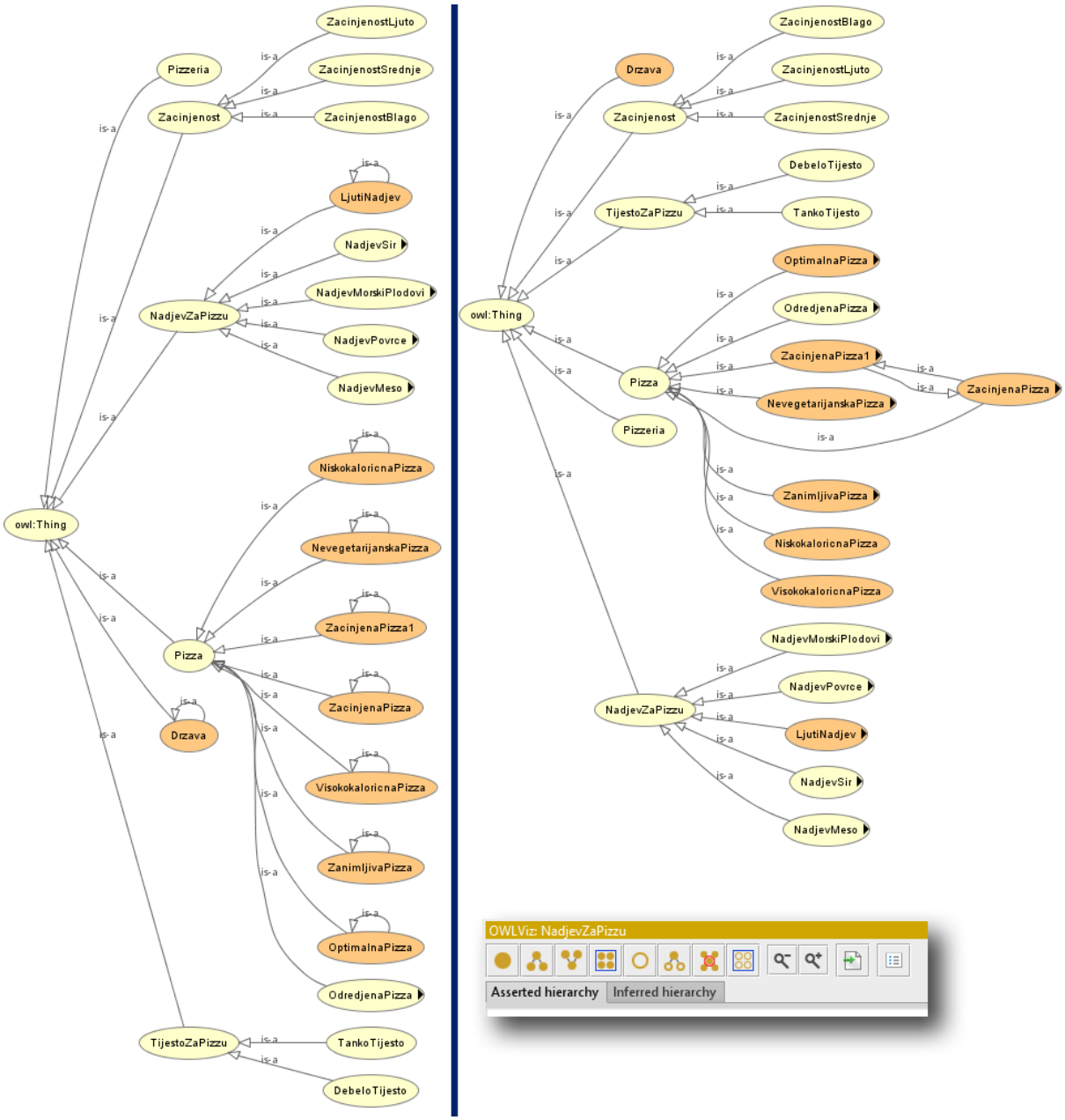
- kreiranje objektnih svojstava također u samom stablu ili posebnom editoru koristeći mogućnost predefiniranja prefiksa i sufiksa
- deklariranje karakteristike objektnih svojstava (funkcionalno, inverzno funkcionalno, tranzitivno, simetrično itd.)
- deklariranje domene i dosega objektnih svojstava
- dodavanje ograničenja svojstvima - okvalificirana ograničenja (egzistencijalna i univerzalna), ograničenje kardinalnosti (minimum, maksimum, jednakost), ograničenje vrijednosti
- upotreba zaključivača za provjeru konzistentnosti i automatsku klasifikaciju
- korištenje aksioma zatvaranja
- kreiranje individua odnosno instanci klasa
- definiranje prebrojivih klasa i rad sa anonimnim klasama
- deklariranje ekvivalentnih klasa
- rad sa podatkovnim svojstvima, deklaracija funkcionalnih podatkovnih svojstava
- korištenje particije vrijednosti i aksioma pokrivanja
- definiranje komplementarnih klasa
- kreiranje nizova povezanih svojstava
- DL i SPARQL upiti

Ontologija o pizzama obuhvaća hijerarhije pizza, vrsta tijesta za pizzu, nadjeva, pizzerija, kao i različitih svojstava koje pizze, tijesta, nadjevi itd. mogu imati. Na slici 75 možemo vidjeti grafički prikaz ontologije, gdje je prikazana hijerarhija klasa onako kako je unesena prilikom kreiranja ontologije, te hijerarhija nastala zaključivanjem i klasifikacijom FaCT++ zaključivača. Prikaz je nastao u alatu Protégé uz upotrebu OWLViz plugina. Referirajući se na OBO uzorke, u ontologiji o pizzama mogu se pronaći n-arne objektne relacije (na primjer pizza Americana ima bar jedan nadjev s kobasicama i ima bar jedan nadjev od mozzarele) i n-arne podatkovne relacije (na primjer svaka pizza ima cijenu i ima određen broj kalorija). Ovo su prošireni uzorci dizajna, jer nadilaze ograničenja OWL jezika koji nema skupovno pridruživanje.

Iz područja uzoraka dobre prakse moguće je pronaći uzorak particije vrijednosti (na primjer, pizza Margherita ima neki od nadjeva od mozzarele i neki od nadjeva s rajčicom ali su to i jedini nadjevi koje može imati pri čemu su ti nadjevi razdvojene klase) te uzorak zatvaranja (pizza Quattro formaggi ima 4 nadjeva sira, ali ne može imati niti jedan drugi nadjev osim nadjeva od sira). Pored toga, upotrebom zaključivača mijenja se hijerarhija pizza, pa tako pizze Margherita i Soho zaključivač smješta u vegetarijanske pizze, a pizze Americana u nevegetarijanske. Ovo je uzorak normalizacije iz OBO uzoraka dobre prakse.

Od uzoraka modeliranja domene, može se uočiti već ranije spomenuti uzorak niza povezanih svojstava oblika:

$$prodajePizzu \circ imaZemljuPorijekla \sqsubseteq imaAsortimanIz$$



Slika 75: Unešena i zaključena hijerarhija klasa ontologije o pizzama - OWLViz u Protégéu

4.2.2. DOLCE ontologija

DOLCE je deskriptivna ontologija za lingvistički i kognitivni inženjering (eng. *Descriptive Ontology for Linguistic and Cognitive Engineering*) koju jer razvio ISTC-CNR* Laboratorij

*ISTC je akronim koji označava Institut za kognitivne znanosti i tehnologije (it. *Istituto di Scienze e Technologie della Cognizione*) sa rimskom adresom, koji je osnovan 2001. na nasljeđu bivšeg Instituta za psihologiju, zatim

za primijenjenu ontologiju. Izvorno je ova ontologija stvorena kao dio WonderWeb projekta financiranog u okviru programa informacijskih društvenih tehnologija europske komisije, i taj projekt je završio 2004. godine. DOLCE je zamišljena kao prvi WFOL (eng. *WonderWeb Foundational Ontologies Library*) modul WonderWeb projekta[†]. Bila je utemeljena je na logici prvog reda, ali su je kasnije Aldo Gangemi i suradnici prilagodili jeziku OWL. Tako su nastala dva modela:

- DOLCE Lite - osnovna prilagodba DOLCE ontologije OWL-2 jeziku. Napravljene su brojne prilagodbe zbog smanjene ekspresivnosti deskriptivnih logika. Na primjer, problemi relacija sa temporalnim argumentima u logici prvog reda koje za posljedicu imaju ternarne relacije, riješeni su uklanjanjem tih argumenata. Za razliku od izvorne DOLCE ontologije, ne podrazumijeva modalnost, temporalno indeksiranje i kompoziciju relacija. Ima 37 klasa i 70 objektnih svojstava te 541 aksiom od toga 356 logičkih aksioma (broj aksioma varira od verzije do verzije).

- DOLCE+DnS Ultralite (DUL) - pojednostavljena prilagodba u kojoj su imena klasa intuitivnija, prostorne i vremenske relacije pojednostavljene, jednostavniji konstrukti aksiomatizacija, korištenje uzoraka u arhitekturi ontologije odnosno modularnost (uzorci sadržaja). Olakšana primjena prilikom fizikalnih i sociooloških kontekstualizacija koja kao rezultat daje lagane ontologije. Ipak, ova ontologija se ne čini prema metrikama manja od DOLCE Lite: ima 76 klasa, 197 objektnih svojstava, 5 podatkovnih svojstava te čak 1493 aksioma od čega 581 logičkih.

  Neki uzorci dizajna koji se nalaze u NeON katalogu izvučeni su upravo iz DOLCE Ultralite ontologije i prirađeni u literaturi. (Gangemi i Presutti, 2009, str. 12) i citeppPresutti|200861. Ovi se uzorci razlikuju od OBO uzoraka na osnovu kojih sam analizirao *Ontologiju o pizzama*, i ima ih mnogo. Podijeljeni su na generalne uzorke, dijelove i kolekcije, semiotske uzorke, količine i dimenzije, uzorke participacije, uzorke organizacije, menadžmenta i zakazivanja (eng. *Organization, Management, and Scheduling*),

  Generalne uzorke čine Tip entiteta (eng. *Types of entities*, Opis (eng. *Desctiption*), Situacija (eng. *Situation*), Klasifikacija (eng. *Classification*), N-arna klasifikacija (eng. *N-ary Classification*), Opis i situacija (eng. *Description and Situation*) i Objektna uloga (eng. *Object Role*).

  Dijelove i kolekcije čine uzorak Dio od (eng. *Part of*), Čimbenici (eng. *Constituency*) i Entitet kolekcije (eng. *Collection entity*).

  Semiotske uzorke čine Intenzija Ekstentija (eng. *Intension Extension*) i Informacijska realizacija (eng. *Information Realization*).

  Količine i dimenzije čine Područje (eng. *Region*) i Parametar (eng. *Parameter*).

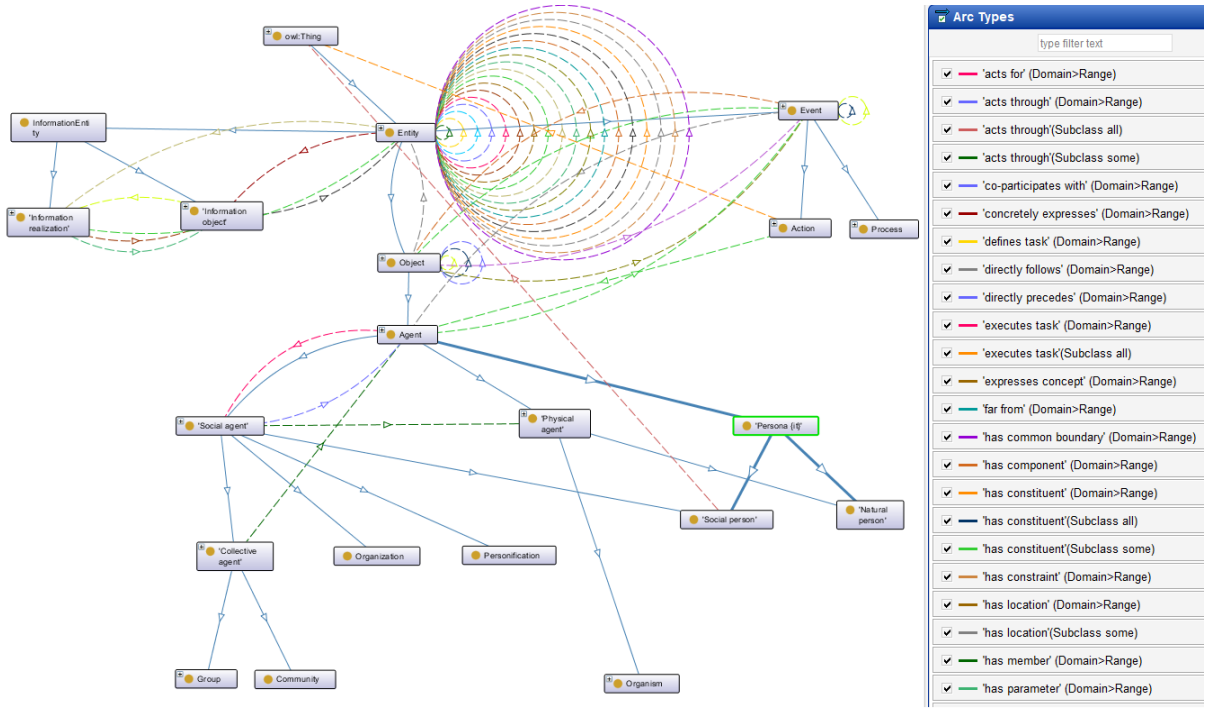
Instituta za fonetiku i dijalektologiju u Padovi i nekih grupa sa Biomedicinskih tehnologija u Rimu, te Instituta za sistemska istraživanja i bio-inženjering (Istituto di Sistemistica e Bioingegneria - LADSEB) iz Padove i Grupe za elektroniku poluvodiča u Rimu. CNR je akronim za Nacionalno istraživačko vijeće (it. *Consiglio Nazionale delle Richerche*).

[†]Ostali moduli su Temeljna formalna ontologija (eng. *Basic Formal Ontology*, BFO) i Ontologija referenci visoke razine usredotočena na objekte (eng. *Object-Centered High-level Reference ontology*, OCHRE).

Uzorke participacije čine Participacija (eng. *Participation*). U literaturi (Presutti, Gangemi i David, 2008, str. 89) se navode i drugi uzorci ali nisu nastali reinženjeringom DOLCE Ultralite ontologije.

Među uzorke organizacije, menadžmenta i zakazivanja pripadaju Prednost (eng. *Precedence*) i Uloga prema zadatku (eng. *Task role*).

DOLCE ontologija pripada višim (eng. *upper-level*) ontologijama, koje se također nazivaju i temeljnim (eng. *foundational*) ontologijama. Ovakve ontologije sadrže domenski neovisne elemente poput prostora, vremena, objekata i procesa. Vrlo su korisne u ontološkim integracijama, jer pružaju kostur na koji se mogu naslanjati odnosno na temelju kojih se mogu karakterizirati novi koncepti. Više ontologije su ponovno iskoristive, djeljive i zbog slabe aksiomatizacije lako proširive.



Slika 76: DOLCE+DnS Ultralite fragment centriran oko agenata u OntoGraf prikazu

5. Primjer izrade ontologija zasnovanih na uzorcima dizajna

Kako bi se ilustrirao razvoj ontologija uz upotrebu uzoraka dizajna ontologija, kreirane su dvije ontologije. Prva od njih opisuje odnose među uzorcima dizajna "Gang of Four", a druga je nastala na temelju projektnog zadatka na temu ekološkog zbrinjavanja otpada na kolegiju "Uzorci dizajna". Obje ontologije izrađene su u alatu Protégé, a njihova konzistentnost je provjerena upotrebom FaCT++ zaključivača. Obje izvedbe imaju individue pridružene određenim klasama te nekoliko testnih klasa nastalih na temelju DL upita.

5.1. Ontologija uzoraka dizajna "Gang of Four"

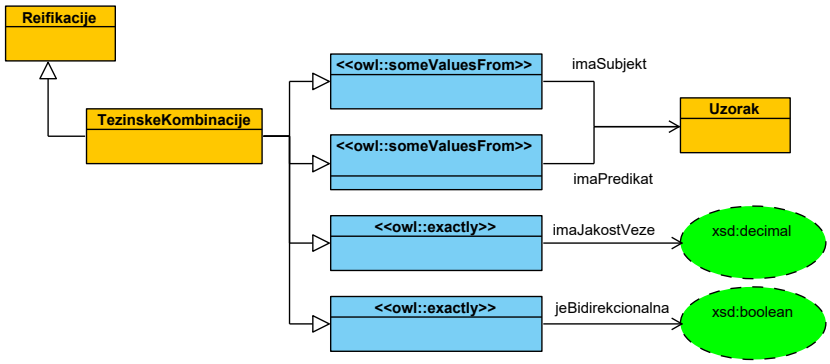
Prilikom korištenja uzoraka dizajna u softverskom inženjerstvu, često je potrebno znati i kako su uzorci međusobno povezani, osobito ako razvojni inženjer nema dovoljno iskustva sa njihovom upotrebom. U katalozima uzoraka opisane su suradnje, odnosno poveznice među uzorcima koje se odnose na mogućnost zajedničke upotrebe uzoraka. Ipak, u svrhu boljeg iskorištenja potencijala uzoraka, javlja se potreba i za automatskom klasifikacijom. Javljaju se slijedeća pitanja: (Zimmer, 1995)

- Kako su uzorci povezani i kakva je karakteristika te povezanosti?
- Kada i na koji način dva uzorka rješavaju isti problem?
- Je li moguće kombinirati dva uzorka?
- Kakvi su kriteriji klasifikacije uzoraka u određene kategorije?

Uzorci četvorice autora (GOF) svrstani su u određene kategorije prema području i namjeni na slici 4 (strana 42). Kratki opis uzoraka naveden je u popratnom tekstu. Na slici 79 prikazane su poveznice među uzorcima, koje se klasificiraju u tri grupe:

- X koristi Y (asimetrična veza)
- X je sličan Y (simetrična veza)
- X se može kombinirati sa Y (simetrična veza)

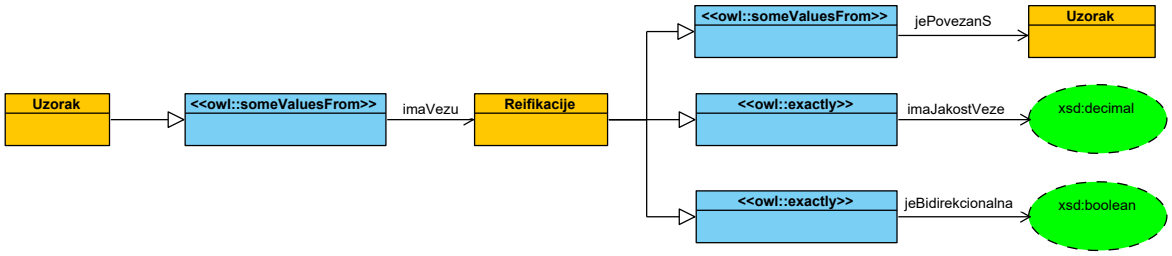
Automatski sustavi klasifikacije omogućavaju i proširenje modela na neizrazitu povezanost, gdje se vezama među uzorcima pridružuju težinske vrijednosti. Budući da su relacije binarne, i ne sadrže dodatne parametre, ovaj se problem može riješiti reifikacijama kako je opisano u OBO uzorku n-arne objektne relacije:



Slika 77: Fragment ontologije - reifikacija težinske kombinacije uzoraka

Na slici 77 prikazana je reifikacija veze među dvama GOF uzorcima, gdje je relacija zamijenjena entitetom *TezinskaKombinacija*. Pojedine individue ove klase mogu opisivati pojedine veze među uzorcima. Svakoj vezi dodijeljena je i težinska vrijednost kao i oznaka dvosmjernosti (bidirekcionalnost). Zbog upotrebe egzistencijalnog ograničenja svojstva, istu vezu se može više upotrebljavati. Određena težinska kombinacija ima bar jedan subjekt i bar jedan predikat, pri čemu su ti subjekti i predikati iz klase GOF uzoraka.

Ovakva reifikacija pridružuje se primjerice klasi *Model* pomoću svojstva *imaTezinskuKombinacijuUzoraka*. Riječima se to može iskazati da je prema modelu težina mogućnosti kombiniranja uzorka primjerice *Factory_method* i *Template_method* $w = 0,8$. Poveznice su individue iz klase *TezinskaKombinacija*.



Slika 78: Fragment ontologije - reifikacija težinske kombinacije uzoraka, drugi način

Pored ovog načina moguć je i drugi način, prikazan na slici 78. Sada je reifikacija povezana sa određenim uzorkom. Smisao ovog načina je da uzorak ima vezu za koju vrijedi da ga povezuje sa nekim drugim uzorkom iz klase uzoraka, da ima jakost vezanja kao vrijednost nekog težinskog faktora, što je u ovom slučaju literal, te da može ali i ne mora biti bidirekcionalna. U ovom rješenju dio od reifikacije prema povezanom uzorku i literalima potpuno je jednak GeoLinkovom uzorku vrijednosti dvojstva (slika 69, str. 106). Sa stanovišta težinske vrijednosti i bidirekcionalnog svojstva, oba načina se mogu povezati sa OBO uzorkom n-arnih podatkovnih relacija (slika 26, str. 59). Budući da u prvom načinu imamo i subjekt i predikat, i oba su objekti klase uzorak, to je n-arna objektna relacija kao na slici 27, str. 60.

Upotreba poveznica sa težinskim faktorima omogućava bolju klasifikaciju veza među uzorcima, lakši izbor uzoraka i smanjenje grešaka u izboru uzoraka (Konecki, Orehovački i

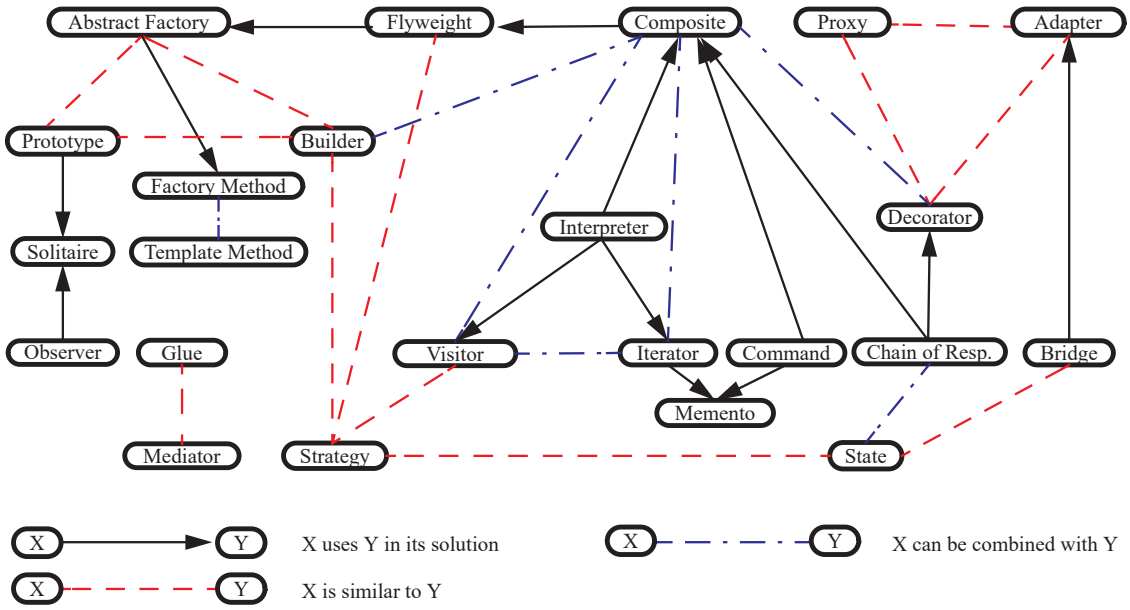
Kermek, 2009).
Tipovi veza u ovom slučaju se svode na slijedeće: (Konecki, Orehovački i Kermek, 2009)

- X može koristiti Y u rješenje prema stupnju težine Z
- X je sličan Y na razini stupnja Z
- X se može kombinirati sa Y prema stupnju težine Z

Među GOF uzorcima, nalaze se i neki koji su promijenili ime. To su *Singleton* (*Solitaire*), *Iterator* (*Walker*), *Facade* (*Glue*) i *Decorator* (*Wrapper*). Takvi uzorci imaju svojstvo *imaStariNaziv* i *imaNoviNaziv*. Parovi individua bi se u okviru OWL jezika mogli izjednačiti, no tada bi se u pretraživanju pojavljivali kao parovi. Stoga to nije tako učinjeno. Umjesto toga, uvijek su povezani pomoću starih imena. Ako je potrebno pretraživati i nova imena, koristi se uzorak kompozicije svojstava kako bi se imena pretvorila u stare nazive, te nakon usporedbe, ovisno o potrebi pretvorila opet u nove. U DL formalizmu opisana su svojstva:

$$imaNoviNaziv \equiv imaStariNaziv^{-}$$
$$koristi \equiv jeKoristenOd^{-}$$
$$imaStariNaziv \circ jeKoristenOd \circ imaNoviNaziv \sqsubseteq jeKoristenOdNovog$$
$$jeKoristenOd \circ imaNoviNaziv \sqsubseteq jeKoristenOdNovog1$$
$$jeKoristenOdNovog1 \sqsubseteq jeKoristenOdNovog$$
$$jeSlican \equiv jeSlican^{-}$$
$$imaStariNaziv \circ jeSlican \circ imaNoviNaziv \sqsubseteq jeSlicanNovom$$
$$jeSlican \circ imaNoviNaziv \sqsubseteq jeSlicanNovom1$$
$$jeSlicanNovom1 \sqsubseteq jeSlicanNovom$$
$$mozeSeKombiniratiSa \equiv mozeSeKombiniratiSa^{-}$$
$$imaStariNaziv \circ mozeSeKombiniratiSa \circ imaNoviNaziv \sqsubseteq mozeSeKombiniratiSaNovim$$
$$mozeSeKombiniratiSa \circ imaNoviNaziv \sqsubseteq mozeSeKombiniratiSaNovim1$$
$$mozeSeKombiniratiSaNovim1 \sqsubseteq mozeSeKombiniratiSaNovim$$

Primjer povezanosti uzorka *Visitor* te korištenje svojstva *mozeSeKombiniratiSa* dan je na slici 80. Na istoj slici su prikazane i kompozicije svojstava, odnosno lanci *jeKoristenOdNovog* i *mozeSeKombiniratiSaNovim*. Ovakav sustav je neovisan o tome ima li neki uzorak staro i novo ime ili nema.



Slika 79: Povezanost uzoraka dizajna "Gang of Four". Preuzeto iz Zimmer (1995)

Nije potrebno uspostavljati simetrične veze sa svake strane. Ukoliko je uzorak *X* sličan uzorku *Y* onda će se moći pronaći i da je uzorak *Y* sličan uzorku *X* nakon uključenja zaključivača, te u DL upitima. U ontologiju su uključene i testne klase, odnosno slijedeći DL upiti:

$$MnogoGaKoriste \sqsubseteq (> 1\ jeKoristenOd.Uzorci)$$
$$MnogoSeKombinira \sqsubseteq (> 1\ mozeSeKombiniratiSa.Uzorci)$$
$$MnogoSlicnosti \sqsubseteq (> 2\ jeSlican.Uzorci)$$
$$SlicniSuBuilderu \sqsubseteq (jeSlican)^{-} \ni Builder$$

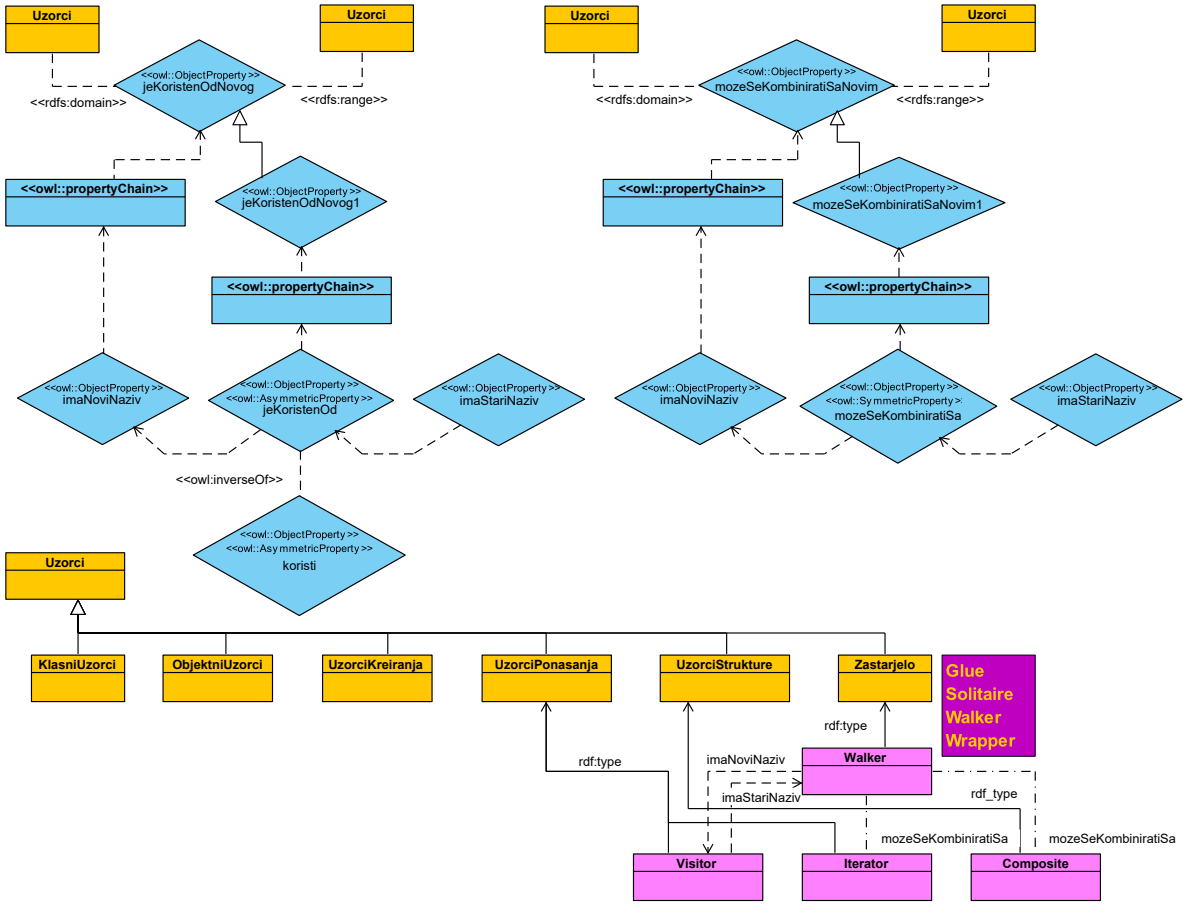
Uzorak dizajna koji omogućava pretraživanje povezanosti i među uzorcima sa starim i novim imenima je OBO uzorak niza povezanih svojstava (slika 40, str. 70). Formalno se za poveznicu sličnosti uzoraka može napisati:

$$imaStariNaziv \circ jeSlican \circ imaNoviNaziv \sqsubseteq jeSlicanNovom$$

Ovdje au ulančana čak 3 objektna svojstva. Može se dogoditi da uzorak nema stari naziv. To su uzorci koji nisu promijenili ime. Kako bi se i to riješilo, dodano je novo svojstvo $jeSlican1$ kao podsvojstvo svojstva $jeSlicanNovom$ oblika:

$$jeSlican \circ imaNoviNaziv \sqsubseteq jeSlican1$$
$$jeSlican1 \sqsubseteq jeSlicanNovom$$

Grafički prikaz hijerarhija svojstava i klasa, te ulančavanja prikazan je na slici 80.

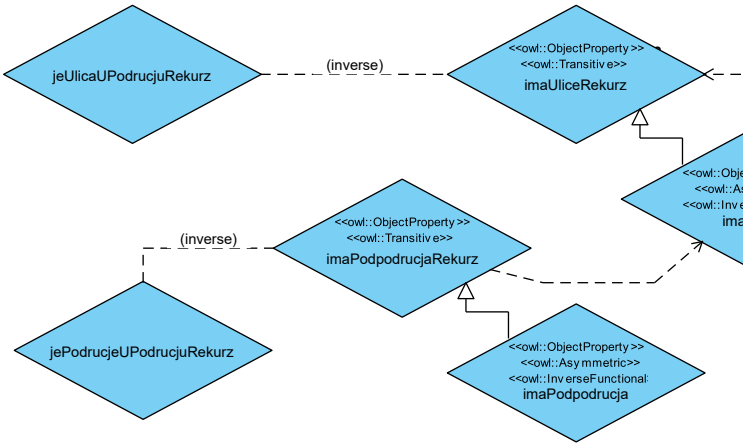


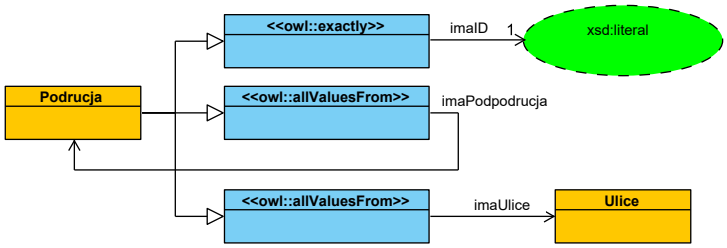
Slika 80: Ontologija uzoraka dizajna "Gang of Four"

Prema mjeraču metrike za ontologije u Protégéu, potrebna ekspresivnost za ovu ontologiju je $\mathcal{ALCROIQ}^{(\mathcal{D})}$.

5.2. Ontologija projekta ekološkog zbrinjavanja otpada

Ova ontologija nastala je na temelju projektnih zadataka na kolegiju "Uzorci dizajna" koji zbog svoje složenosti omogućavaju implementaciju raznovrsnih uzoraka. Jedan od složenijih elemenata je organizacija strukture područja i ulica koje se u područjima nalaze. Potrebno je omogućiti rekurzivno pretraživanje tih područja, te dolazak do samih ulica koje su listovi stablaste strukture. Ovdje je to riješeno kompozicijama svojstava (područje $\circ$ ulica), tranzitivnim svojstvima i izgradnjom stablaste strukture individua prema uzorku stabla. Kako bi se podaci iz tablice na slici 81 organizirali u hijerarhiju, koristi se klasifikacijska shema za model susjednosti opisan u uzorcima reinženjeringa na stani 79. Treba napomenuti da se ovdje ipak radi o individuama a ne o klasama, pa ovo nije klasifikacijska shema. No način prikazan na slici 45 jednako je primjenjiv i na individue.





pID	Dijelovi	Dio od
p1	EZO Anglomeracija Varaždin	NULL
p2	Varaždin grad	p1
p3	Varaždin okolica	p1
p4	Centar	p2
p5	Banfica	p2
p6	Đurek	p2
p7	Hraščica	p3
p8	Jalkovec	p3
uID	**Ulice**	
u1	Pavlinska	p4
u2	Alojzija Stepinca	p4
u3	Petra Krešimira IV	p4
u4	Zrinskih i Frankopana	p6
u5	Krešimira Filića	p6
u6	Tina Ujevića	p6
u7	Ruđera Boškovića	p5
u8	Dravska	p5
u11	Varaždinska	p7
u12	Braće Radića	p8
u13	Dopunska ulica	p1

Slika 81: EZO ontologija - područja i ulice

Područja su povezana sa vrijednostima uz upotrenu n-arnih veza, pa tako svako područje kao individua ima svoju identifikacijsku oznaku a može imati i podpodručja te ulice:

$$Podrucja \sqsubseteq (= 1\ imaID.xsd : literal)$$
$$Podrucja \sqsubseteq (\forall imaPodpodrucja.Podrucja)$$
$$Podrucja \sqsubseteq (\forall imaUlice.Ulice)$$

Područja ne mogu imati nikakve druge podatke osim prostornih obilježja.

Za jednu individuu, na primjer *Varazdin_grad* pridruživanje vrijednosti može se formalno izraziti na slijedeći način:

$$Podrucja \ni Varazdin\_grad$$
$$\langle Varazdin\_grad, "p2"\hat{}\hat{}xsd : literal\rangle \in imaID$$
$$\langle Varazdin\_grad, Centar\rangle \in imaPodpodrucja$$
$$\langle Varazdin\_grad, Djurek\rangle \in imaPodpodrucja$$
$$\langle Varazdin\_grad, Banfica\rangle \in imaPodpodrucja$$
$$\langle Varazdin\_grad, Dopunska\_ulica\rangle \in imaUlice$$

Rekurzivna svojstva i rekurzivne povezanosti nalazimo u Geolinkovom uzorku organizacije

(slika 65, str. 99). U tom uzorku organizacija je povezana sama sa sobom, što omogućava izgradnju hijerarhija organizacija i podorganizacija. Svojstva moraju biti tranzitivna kako bi se moglo pretraživati po dubini. Uz sve ovo, potrebno je omogućiti po želji rekurzivno i nerekurzivno pretraživanje. Zbog toga se kreira tranzitivno svojstvo i netranzitivno podsvojstvo. Po želji je tada moguće birati jedno ili drugo. Na primjer, ako želimo pretražiti samo podpodručja nekog područja, ali ne i daljnja podpodručja u dubinu, koristit ćemo naslijeđeno, netranzitivno svojstvo. U suprotnom ćemo koristiti tranzitivno nadsvojstvo. Ovakvu strukturu nalazimo u OBO uzorku slijeda odnosno sekvence (39, str. 69).
Pretraživanje podpodručja pomoću DL upita za *Varazdin_grad* formalno se definira kao:

$$RekurzivnaPodpodrucja\_Varazdin\_grad \equiv jePodrucjeUPodrucjuRekurz \ni Varazdin\_grad$$

Problem nastaje kada nema homogenosti u hijerarhijskoj strukturi. U mom primjeru, dok god se pretražuju područja, radi se o homogenom slučaju, budući da područja sadrže druga područja kao podpodručja. No ako želimo pronaći i ulice u takvoj hijerarhiji, ovakav način neće pomoći. Ulice su listovi stabla, pojedini objekti, odnosno individue pripadaju drugačijim klasama (imamo klase područja i ulica), i potrebno je omogućiti da se uvijek pretražuju i područja i ulice. Tek ako se ne pronađe ulica, ide se dalje u dubinu. Zato se za pretraživanje ulica gradi kompozicija svojstava. Slično kao i kod područja, imamo hijerarhiju tranzitivnog svojstva i njegovog netranzitivnog podsvojstva. Netranzitivno svojstvo pretražuje samo ulice. Ako neko područje ima takvu ulicu, zaključivat će ju naći.

Kod tranzitivnog svojstva situacija je drugačija. Potrebno je pretražiti ne samo ulice nego i svojstva, budući da se ulica možda nalazi u nekom podpodručju. Gradi se kompozicija

$$imaPodpodrucjeRekurz \circ imaUlice \sqsubseteq imaUliceRekurz$$
$$imaUlice \sqsubseteq imaUliceRekurz$$

Ovo je kombinacija uzorka niza povezanih svojstava (komponiraju se ulice i područja) i uzorka slijeda (pretražuju se područja rekurzivno). Zapravo je oblik NeON uzorka stabla (slika 57, str. 89).

Ulice pored prostornih podataka mogu imati i neke druge podatke poput broj mjesta, udio malih, srednjih i velikih korisnika. Analogno za ulice formalno se može napisati:

$$Ulice \sqsubseteq (= 1\ imaID.xsd : literal)$$
$$Ulice \sqsubseteq (\exists imaUdioMali.xsd : integer)$$
$$Ulice \sqsubseteq (\exists imaUdioSrednji.xsd : integer)$$
$$Ulice \sqsubseteq (\exists imaUdioVeliki.xsd : integer)$$

Zbog povezanosti preko inverznog svojstva *imaUlice*, nije potrebno specificirati kojem području ulica pripada odnosno *jeUlicaUPodrucju*. Zaključivač će to sam zaključiti, analizirajući koje sve ulice pripadaju određenim područjima, pa se mogu postavljati izravni i rekurzivni upiti koristeći

to svojstvo iako nije nigdje navedeno.

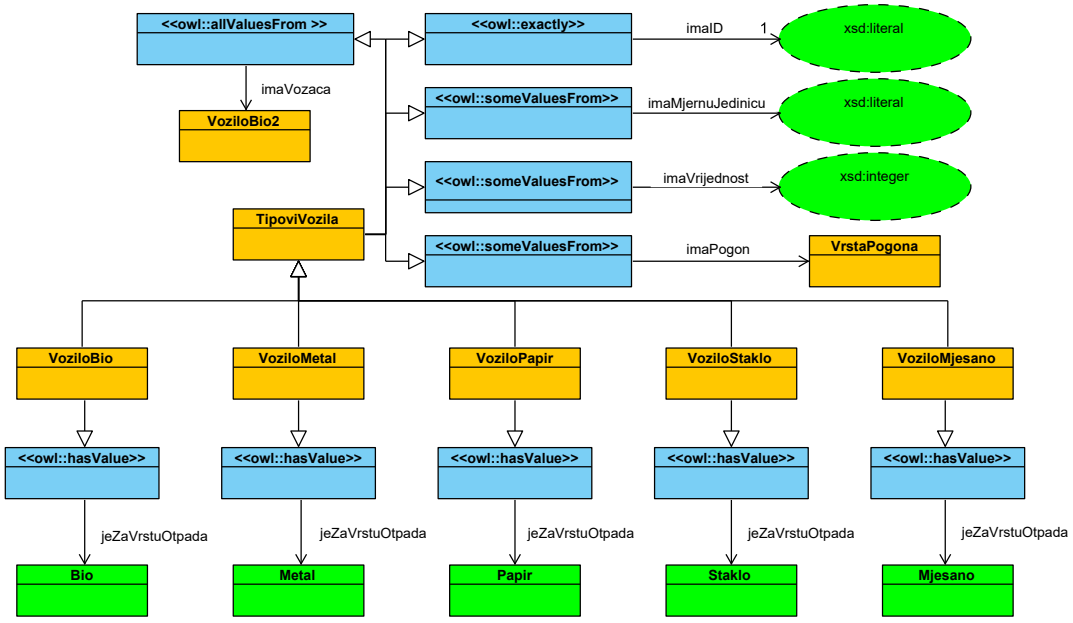
$$Nadpodrucja\_Banfice \equiv imaPodpodrucja \ni Banfica$$
$$Nadpodrucja\_BanficeRekurz \equiv imaPodpodrucjaRekurz \ni Banfica$$

Analogno područjima, mogu se dobiti i sve ulice u područjima i podpodručjima za neko područje:

$$RekurzivneUlice\_Varazdin\_grad \equiv jeUlicaUPodrucjuRekurz \ni Varazdin\_grad$$

DL upiti će dati ispravne odgovore, no ukoliko se kreiraju klase na osnovu tih upita, može se dogoditi da alat ne prikaže sve individue. Razlog zašto je tome tako je zaključena hijerarhija. Zaključivač može svrstati neku drugu klasu kao podklasu klase upita. U tom slučaju sve individue koje pripadaju podklasama, podrazumijevaju se i u nadklasama, ali se ne prikazuju. Preporuka je uključiti prikazivanje zaključene hijerarhije i pregledati individue svih podklasa. Ovo je osobito bitno ukoliko se u ontologiju ugrađuje mnoštvo upitnih klasa. DL upiti rade uobičajeno.

Osim područja i ulica, ontologija opisuje i osobine spremnika i vozila. Spremnici su klasificirani prema vrsti otpada i tipu (kanta ili kontejner). Vozila nisu klasificirana prema tipu otpada. I vozila i spremnici, koristeći uzorak n-arnih relacija povezani su sa svojim podacima.



Slika 82: EZO ontologija - vozila

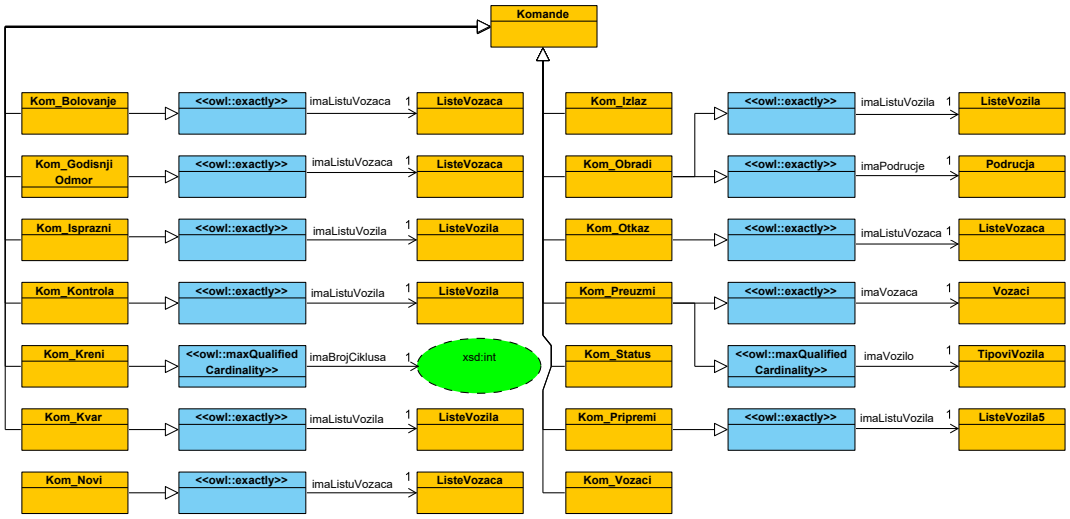
Spremnici pored količine otpada prikazane pomoću mjerne jedinice i vrijednosti (n-arna relacija) imaju i tip (kanta ili kontejner) te vrijednosti koliko korisnika ga dijeli. Zbog sličnosti ovaj dijagram ne prikazujem.

Posljednji dio ontologije su komande dispečera. Ove komande imaju svoje parametre,

i mogu se odnostiti na vozače, vozila, područja i broj ciklusa. Najčešće imaju kao parametre liste vozila i liste vozača pa su ugrađeni entiteti za takve liste.

$$ListaVozila \sqsubseteq (= 1\ imaClanova.xsd : integer)$$
$$ListaVozila \sqsubseteq (\exists imaVozilo.TipoviVozila)$$
$$ListaVozaca \sqsubseteq (= 1\ imaClanova.xsd : integer)$$
$$ListaVozaca \sqsubseteq (\exists imaVozaca.Vozaci)$$

Prema modelu, postoji 14 komandi i odnose se na određene operacije poput zadavanje područja koja vozila moraju obraditi, pripremanje liste vozila, kretanje u odvoz sa određenim brojem ciklusa, obrade kvara, pražnjenje i kontrole liste vozila, prikaz statusa i vozača. Tu su još komande za uvođenje novih vozača, preuzimanje vozila, otkazivanje rada i bolovanje liste vozača. Prikaz komandi i povezanosti sa parametrima prikazana je na slici 83.



Slika 83: EZO ontologija - komande dispečera

U ontologiju su ugrađeni upiti koji omogućavaju dobivanje vozila i vozača koji su kao parametri pridruženi određenim komandama. Komande sa specifičnim parametrima su individue. Za pretraživanje listi vozača među parametrima komande koristi se svojstvo *komandaVozaci*.

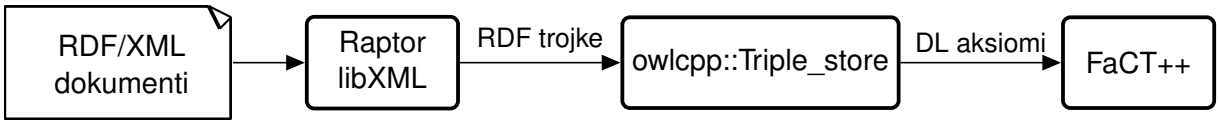
$$imaListuVozaca \circ imaVozaca \sqsubseteq vozaciKomanda$$
$$imaVozaca \subseteq imaListuVozaca$$
$$imaListuVozaca \equiv komandaVozaci^{-}$$

Ekvivalentno za pretraživanje vozila:

$$imaListuVozila \circ imaVozilo \sqsubseteq vozilaKomanda$$
$$imaVozilo \subseteq imaListuVozila$$
$$imaListuVozila \equiv komandaVozila^{-}$$

5.3. Integracija baza znanja u aplikacije

Baze znanja same za sebe nemaju prevelikog smisla. U okviru semantičkog weba ontologije imaju ulogu nacrta, unutrašnjeg tijela znanja (Obitko, 2007), ali to ne znači da one trebaju biti ograničene samo na semantički web. I druge vrste aplikacija, čak i stolnih, mogu koristiti baze znanja, postavljati upite i dobavljati rezultate na osnovu zaključaka koji proizlaze iz podataka i aksioma zapisanih u bazu znanja ili mrežu povezanih baza znanja. I sami alati za izradu ontologija su takve aplikacije.



Slika 84: Dijagram toka OWLCPP biblioteke, prema izvoru: Levin i Cowell (2015)

Kako bi se omogučio pristup OWL-u iz različitih jezika softverskog inženjerstva koriste se različite biblioteke, od kojih ovdje izdvajam *owlcpp* biblioteku otvorenog kôda koja omogućava pristup OWL ontologijama iz jezika *C++* i *python*. Druge slične biblioteke poput *Jene* i *OWL API* povezuju OWL i jezik *Java*. (Levin i Cowell, 2015)

Primjer C++ programa kao i popratne Boost OWL_JAM datoteke dajem u prilogu 1.1 str. 143.

6. Zaključak

Izrada ontologija pored domenskog znanja zahtijeva specifična znanja iz informacijskih znanosti i računarstva koja uključuju informacijsko programsko inženjerstvo, logičko programiranje, baze znanja, višeagentne sustave, algoritme, osobito rekurzivne ali i lingvistiku, leksikografiju i teoriju sustava. Zbog takve multidisciplinarnosti, otežano je pronalaženje potrebnog stručnog kadra koji bi ih mogao razvijati. Upravo zbog toga se ontologije pojavljuju najčešće u okviru znanstvenih projekata i služi znanstvenicima za klasifikaciju prikupljenih znanja. Uzorci dizajna nastali su iz težnje da se razvoj ontologija olakša i približi širem sloju razvojnih inženjera, a da se znanstvenike rastereti od logičkih problema koji se pojavljuju pri dizajnu ontologija.

U razradi dviju ontologija, koristeći uzorke i ideje nastale njihovom analizom, a tu se prije svega misli na n-arne relacije i kompozicije svojstava, pokazao sam koliko je jednostavno manipulirati podacima u listama i stablima, na identičnom problemu kakav sam u sklopu projektnih zadataka morao izvesti u jeziku Java koristeći uzorke softverskog inženjerstva (tu se prije svega misli na GOF uzorke). Tako su se razjasnile i neke nejasnoće koje nastaju zbog logičkih pogrešaka poput anti-uzorka OIL (onlyness is loneliness), kao i problem prikaza individua u zaključenim hijerarhijama kada zaključivač na nižoj razini od klase upita ugradi klase nekih drugih upita i time sakrije sve individue koje se trebaju prikazati.

U radu sam izradio veliki broj dijagrama potpuno samostalno, povezujući ih sa sintaksom deskriptivnih logika, što može u značajnoj mjeri olakšati opis ontologija, budući da se ukida potreba za ispisom RDF, RDFS i OWL kôda. Testiranjem uz upotrebu zaključivača i upita, pokazao sam da takve ontologije zaista obavljaju svoju funkciju, eksperimentirajući istovremeno i sa unesenim hijerarhijama, aksiomima i ograničenjima. Alat Protégé se pokazao kao izuzetno zrela platforma, pružajući velike mogućnosti pri analizi i grafičkom prikazivanju elemenata kao i cijelih ontologija. Ipak, zbog oslanjanja na dijagramsku tehniku koju koristi OBO, u radu nisam koristio njegove alate poput Ontografa i OWLViz-a. Treba napomenuti da iskusnom razvojnom inženjeru ovi alati omogućavaju velike mogućnosti tijekom testiranja ontologija, a primjenjivi su i za generiranje konačnih dijagrama visoke kvalitete koji će se koristiti u dokumentaciji.

Ovaj rad je pored toga i koristan u smislu da je započeta realizacija te dane smjernice kako bi se mogle kreirati veze sa težinskim faktorima za uzorke GOF. To bi bio logični nastavak objavljen u stručnom članku koji je naveden u literaturi (Konecki, Orehovački i Kermek, 2009). Dakako, ovo nije do kraja realizirano iz razloga što nisam raspolagao sa konkretnim težinskim vrijednostima za poveznice GOF uzoraka, za što bi bilo potrebno provesti nova istraživanja u kojima bi se ti faktori utvrdili. Jednako tako, ostaje i pitanje utvrđivanja klasifikacijskih kriterija koji bi koristili težinske faktore poveznica.

I na kraju, dajem svoje mišljenje o uzorcima dizajna ontologija. Uzorci se očito intenzivno koriste u projektima znanstvenih organizacija jer je znanstvenicima potrebno imati dobro organizirane informacije i baze znanja. Prilikom razvoja ontologija, često se može dogoditi da se nađemo pred logičkim problemima koje ne znamo riješiti. Uzorci su uvijek tu kao pomoć odnosno ideja kako se određeni problemi trebaju rješavati. Same ontologije smatram kralježnicom umjetne inteligencije, pri čemu učenje, odnosno treniranje neuronskih mreža nema smisla

ako ne postoji i konceptualizacija problema. Ontologije je daleko jednostavnije graditi ukoliko poznajemo bar najosnovniji skup uzoraka kakav je primjerice katalog uzoraka otvorenih biomedicinskih ontologija. Sa uzorcima ne treba pretjerivati. Oni su pomoć u rješavanju problema ukoliko se koriste na ispravan način i ne narušavaju eleganciju rješenja problema.

Popis literature

Aranguren, M. E. i dr. (2008). „Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology". *BMC Bioinformatics*. Dostupno na https://doi.org/10.1186/1471-2105-9-S5-S1 17. 9. 2019.

Baader, F., McGuinness, D. L. i Patel-Schneider, P. F. (2007). „The Description Logic Handbook: Theory, implementation, and applications". Dostupno na http://www.naturalthinker.net/trl/texts/Logic/Baader%20(et%20al)%20(ed.)%20-%20The%20Description%20Logic%20Handbook%20-%20Theory,\%20Implementation%20and%20Applications%20(2003).pdf 27. 5. 2019.

Balaban, M. (1995). *The F-logic Approach for Description Languages*. Dostupno na https://www.cs.bgu.ac.il/~mira/DFL.pdf 6. 5. 2019.

Busch, J. A. (2015). „What's SKOS, What's not, Why and What Should be Done About It". Dostupno na http://nkos.slis.kent.edu/ASIST2015/ASISTBusch-SKOS.pdf 23.4.2019.

Cline, M. (1996). „The Pros and Cons of Adopting and Applying Design Patterns in the Real World". *Commun. ACM* 39. Dostupno na https://www.researchgate.net/publication/220424606_The_Pros_and_Cons_of_Adopting_and_Applying_Design_Patterns_in_the_Real_World 17.9.2019., str. 47–49. DOI: 10.1145/236156.236167.

Čačić, V., Paradžik, P. i Vuković, M. (2014). „Logičko programiranje". Dostupno na https://hrcak.srce.hr/133048 9. 5. 2019.

Čubrilo, M. (1985). *Matematička logika za ekspertne sisteme*. Informator, Zagreb.

De Giacomo, G. i Lenzerini, M. (1996). „TBox and ABox Reasoning in Expressive Description Logics". Sv. 1996, str. 37–48.

Falbo, R. A. i dr. (2014). „Ontology Patterns: Clarifying Concepts and Terminology". Dostupno na http://ceur-ws.org/Vol-1188/paper_11.pdf 23. 4. 2019.

Galba, T. (2016). „Algoritam transformacije ontologije u strukturu taksonomije za evidencijsko-zaključivanje". Dostupno na https://dr.nsk.hr/islandora/object/etfos%3A1728 23. 4. 2019. Disertacija. Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek.

Gamma, E. i dr. (1997). *Design Patterns: Elements of Reusable Object-Oriented Software*. Dostupno na https://epdf.pub/design-patterns-elements-of-reusable-object-oriented-software.html 23.06.2019. Addison-Wesley Professional.

Gangemi, A. (2009). „Best practices in ontology design". Dostupno na http://sssw.org/2009/slides/SSSW09-AG-slides.pdf 31.07.2019.

Gangemi, A. i Presutti, V. (2009). „Ontology Design Patterns". Dostupno na https://www.academia.edu/991536/Ontology_design_patterns 23.4.2019.

— (2010). „Best practices in ontology design - SSSW09-AG-slides.pdf". Dostupno na http://sssw.org/2009/slides/SSSW09-AG-slides.pdf 23.4.2019.

GeoLink (2019). „Semantics and Linked Data for the Geosciences". Dostupno na http://sssw.org/2009/slides/SSSW09-AG-slides.pdf 23.4.2019.

Glavic, M. (2006). „Agents and Multi-Agent Systems: A Short Introduction for Power Engineers". Dostupno na http://www.montefiore.ulg.ac.be/~glavic/MAS-Intro_Tech_report.pdf 26. 4. 2019.

Golbreich, C. i dr. (2019). „OBO and OWL: Leveraging Semantic WebTechnologies for the Life Sciences". Dostupno na https://www.cs.ox.ac.uk/people/boris.motik/pubs/ghhms07obo-and-owl.pdf 03.08.2019.

Gostischa-Franta, E. (2013). „Best Practice Software Engineering - Delegation". *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Dostupno na http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/delegation.html 23.06.2019.

Gruber, T. (2009). „Ontology". Dostupno na http://tomgruber.org/writing/ontology-definition-2007.htm 24. 4. 2019.

Guarino, N. (1995). „Ontologies and Knowledge Bases. Towards a Terminological Clarification". Dostupno na https://www.researchgate.net/publication/220041941_Ontologies_and_knowledge_bases_towards_a_terminological_clarification 27. 5. 2019., str. 25–32.

Guarino, N., Oberle, D. i Staab, S. (2009). „What Is an Ontology?": Dostupno na https://www.researchgate.net/publication/226279556_What_Is_an_Ontology 27. 5. 2019., str. 1–17. DOI: 10.1007/978-3-540-92673-3_0.

Gulay, U. (2018). „Stream Reasoning on Expressive Logics". Dostupno na https://arxiv.org/pdf/1808.04738.pdf 15. 7. 2019.

Haddad, D. E. i dr. (2016). „Invited Article: A precise instrument to determine the Planck constant, and the future kilogram - 1.4953825". Dostupno na https://aip.scitation.org/doi/pdf/10.1063/1.4953825?class=pdf 15.08.2019.

Hammar, K. (2017). „Content Ontology Design Patterns: Qualities, Methods, and Tools". Dostupno na https://karlhammar.com/downloads/hammar2017content.pdf 23. 4. 2019. Disertacija. Linköping Studies in Science i Technology.

Horridge, M. i dr. (2012). „Ontology Design PatternLanguage Expressivity Requirements". Dostupno na http://ontologydesignpatterns.org/wiki/images/0/01/WOP_paper_3.pdf 03.08.2019.

Horrocks, I. (2010). „Description Logic: A Formal Foundation for Ontology Languages and Tools". Dostupno na http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/download/Horrocks_Ian_pt2.pdf 10. 7. 2019.

Hoyland, C. A. i dr. (2014). „The RQ-Tech methodology: a new paradigm for conceptualizing strategic enterprise architectures". *Journal of Management Analytics* 1. Dostupno na https://www.researchgate.net/publication/273266393_The_RQ-Tech_Methodology_A_New_Paradigm_for_Conceptualizing_Strategic_Enterprise_

Architectures 2. 6. 2019., str. 55–77. URL: http://dx.doi.org/10.1080/23270012.2014.889912.

Huston, V. (2014). „Design Patterns: An index of GoF and POSA Patterns". *TekSourcery*. Dostupno na http://teksourcery.com/design-patterns-an-index-of-gof-and-posa-patterns/ 27.06.2019.

Kapoor, B. i Sharma, S. (2010). „A Comparative Study Ontology Building Tools for Semantic Web Applications". *International journal of Web & Semantic Technology (IJWesT)* 1.3. Dostupno na https://www.researchgate.net/publication/45825700_A_Comparative_Study_Ontology_Building_Tools_for_Semantic_Web_Applications 18. 7. 2019.

Karlsen, L. H. (2015). „Description Logic 1: Syntax and Semantics". Dostupno na https://www.uio.no/studier/emner/matnat/ifi/INF3170/h15/undervisningsmateriale/dl1.pdf 12.7.2019.

Khomh, F. i Guéhéneuc, Y.-G. (2008). „Do Design Patterns Impact Software Quality Positively?": Dostupno na https://www.researchgate.net/publication/4330216_Do_Design_Patterns_Impact_Software_Quality_Positively 17. 9. 2019., str. 274–278. ISBN: 978-1-4244-2157-2. DOI: 10.1109/CSMR.2008.4493325.

Kifer, M., Lausen, G. i Wu, J. (1994). *Logical Foundation of Object-Oriented and Frame-Based Languages*. Dostupno na https://www.academia.edu/5396567/Logical_foundations_of_object-oriented_and_frame-based_languages 8. 5. 2019.

Koivunen, M.-R. i Miller, E. (2001). „W3C Semantic Web Activity". Dostupno na https://www.w3.org/2001/12/semweb-fin/w3csw 29.7.2019.

Konecki, M., Orehovački, T. i Kermek, D. (2009). „Design Patterns – education and classification challenge". Dostupno na https://www.researchgate.net/publication/224930182_Design_Patterns_-_education_and_classification_challenge, 30.08.2019.

Krishnadhi, A., Hu, Y. i Arko, R. (2015). *GeoLink Core Ontology Design Patterns*. Dostupno na http://schema.geolink.org/patterns/core/main-pattern-collections.pdf 25.7.2019.

Ławrynowicz, A. (2017). „Semantic Data Mining: An Ontology-based Approach". Dostupno na https://www.researchgate.net/publication/319346347_Semantic_Data_Mining_An_Ontology-based_Approach 12. 7. 2019.

Ławrynowicz, A. i dr. (2018). „Discovery of Emerging Design Patterns in Ontologies Using Tree Mining". Dostupno na https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6261490/ 12. 7. 2019.

Levin, M. K. i Cowell, L. G. (2015). „owlcpp: a C++ library for working with OWL ontologies". *Journal of Biomedical Semantics*. Dostupno na https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4574266/, 2.09.2019.

Lutz, C. (2016). „Complexity and Expressive Power ofOntology-Mediated Queries". Dostupno na http://drops.dagstuhl.de/opus/volltexte/2016/5703/pdf/3.pdf 27. 5. 2019.

Maleković, M. i Schatten, M. (2017). *Teorija i primjena baza podataka*. Fakultet organizacije i informatike, Varaždin.

Marchetti, A. i dr. (2009). „Formalizing Knowledge by Ontologies: OWL and KIF". Dostupno na http://puma.isti.cnr.it/rmydownload.php?filename=cnr.iit/cnr.iit/2008-TR-007/2008-TR-007.pdf 24. 4. 2019.

Martínez, D., Taboada, M. i Mira, J. (2003). „Knowledge Base Development". Sv. 2774. Dostupno na https://www.researchgate.net/publication/221019104_Knowledge_Base_Development 27. 5. 2019., str. 1373–1380. DOI: 10.1007/978-3-540-45226-3_186.

Naykhanova, L. i Naykhanova, I. (2018). „Conceptual model of knowledge base system". *Journal of Physics: Conference Series* 1015. Dostupno na https://www.researchgate.net/publication/325279239_Conceptual_model_of_knowledge_base_system 27. 5. 2019., str. 032097. DOI: 10.1088/1742-6596/1015/3/032097.

Noy, N. F. i McGuinness, D. L. (2014). „Ontology Development 101: A Guide to Creating Your First Ontology". Dostupno na https://protege.stanford.edu/publications/ontology_development/ontology101.pdf 23. 4. 2019.

Obitko, M. (2007a). *Ontologies - Introduction to ontologies and semantic web - tutorial*. Dostupno na https://www.obitko.com/tutorials/ontologies-semantic-web/ontologies.html 25. 5. 2019.

— (2007b). *Reasoning - Introduction to ontologies and semantic web - tutorial*. Dostupno na https://www.obitko.com/tutorials/ontologies-semantic-web/reasoning.html 24. 5. 2019.

Presutti, V. i Gangemi, A. (2007). „D2.5.1: A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies". Dostupno na http://neon-project.org/deliverables/WP2/NeOn_2008_D2.5.1.pdf 25.08.2019.

Presutti, V., Gangemi, A. i David, S. (2008). „A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies". Dostupno na http://neon-project.org/deliverables/WP2/NeOn_2008_D2.5.1.pdf 27. 5. 2019.

Roussey, C. i Zamazal, O. (2013). „Antipattern Detection: How to Debug an Ontology without a Reasoner". Dostupno na https://www.ida.liu.se/~patla00/conferences/WoDOOM13/papers/WoDOOM_4.pdf, 22.08.2019.

Russell, S. J. i Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Dostupno na https://people.eecs.berkeley.edu/~russell/aima1e/chapter02.pdf 5. 5. 2019. Prentice Hill, Englewood Cliffs, New Jersey.

Scharffe, F. (2008). „Design Patterns for Ontology Mediation". Dostupno na https://www.academia.edu/2868056/Design_Patterns_for_Ontology_Mediation, 20.08.2019.

Sehring, H.-W. i dr. (2006). „Pattern Repositories for Software Engineering Education". Dostupno na https://www.researchgate.net/publication/221278077_Pattern_Repositories_for_Software_Engineering_Education, 10.9.2019., str. 40–54.

Sikos, L. F. (2019). *Knowledge base*. Dostupno na https://www.lesliesikos.com/knowledge-base/ 15. 9. 2019.

Stoilos, G., Grau, B. C. i Horrcks, I. (2015). „How Incomplete is Your Semantic Web Reasoner?": Dostupno na https://www.cs.ox.ac.uk/files/4542/StCH10a.pdf 26. 5. 2019.

Suárez-Figueroa, M. C., Brockmans, S. i Gangemi, A. (2007). „D 5.1.1 NeOn Modelling Components". Dostupno na http://neon-project.org/deliverables/WP5/NeOn_2007_D5.1.1v3.pdf 25.08.2019.

Suárez-Figueroa, M. C., Gomez-Perez, A. i dr. (2012). *Ontology Engineering in a Networked World*. Dostupno na https://www.researchgate.net/publication/321614296_Ontology_Engineering_in_a_Networked_World 25.08.2019. ISBN: 978-3-642-24793-4. DOI: 10.1007/978-3-642-24794-1.

Van Gelder, A., Ross, K. A. i Schliph, J. S. (1991). „The Well-Founded Semantics for General Logic Programs". *Journal of the ACM* 38. Dostupno na http://www.cse.unsw.edu.au/~cs4415/2010/resources/gelder91wellfounded.pdf 8.5.2019., str. 620–650.

Veljak, L. (2013). „Ontologija i metafizika". *Logos* 2. Dostupno na http://www.ffzg.unizg.hr/filoz/wp-content/uploads/2011/09/Lino-Veljak-Ontologija-i-metafizika.pdf 23.4.2019., str. 9–19.

Vuković, M. (2007). „Matematička logika 1". Dostupno na http://www.mathos.unios.hr/logika/Logika_skripta.pdf 8. 5. 2019.

W3C (2009). „OWL 2 Web Ontology LanguageDocument Overview". Dostupno na https://www.w3.org/TR/2004, 19.7.2019.

Wollowski, M. (2007). „RDFS Example". Dostupno na https://www.rose-hulman.edu/~wollowsk/classes/csse481/schedule/day29/RDFSExample.pdf 8.7.2019.

Zimmer, W. (1995). „Relationships between Design Patterns". Dostupno na http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.8779&rep=rep1&type=pdf, 30.08.2019.

Zolin, E. (2013). „Description Logic Complexity Navigator". Dostupno na http://www.cs.man.ac.uk/~ezolin/dl/ 12.7.2019.

Zou, Y., Finin, T. i Chen, H. (2004). „F-OWL: an Inference Engine for the Semantic Web". Dostupno na https://www.researchgate.net/publication/221106072_F-OWL_An_Inference_Engine_for_Semantic_Web 25. 5. 2019.

Popis slika

Popis tablica

1. Prilozi

1.1. OWL zapis OBO Terma GO_0072421 iz ontologije gena

```
<owl:Class rdf:about="http://purl.obolibrary.org/obo/GO_0072421">
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/GO_0042769"/>
<owl:Restriction>
<owl:onProperty rdf:resource="http://purl.obolibrary.org/obo/BFO_0000050"/>
<owl:someValuesFrom rdf:resource="http://purl.obolibrary.org/obo/GO_0000077"/>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="http://purl.obolibrary.org/obo/GO_0042769"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="http://purl.obolibrary.org/obo/BFO_0000050"/>
<owl:someValuesFrom rdf:resource="http://purl.obolibrary.org/obo/GO_0000077"/>
</owl:Restriction>
</rdfs:subClassOf>
<obo:IAO_0000115 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The series of events in which information about damage
      to DNA is received and converted into a molecular signal, contributing to a DNA damage checkpoint.</obo:IAO_0000115>
<oboInOwl:created_by rdf:datatype="http://www.w3.org/2001/XMLSchema#string">midori</oboInOwl:created_by>
<oboInOwl:creation_date rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2010-12-08T04:16:04Z</oboInOwl:creation_date>
<oboInOwl:hasOBONamespace rdf:datatype="http://www.w3.org/2001/XMLSchema#string">biological_process</oboInOwl:hasOBONamespace>
<oboInOwl:hasRelatedSynonym rdf:datatype="http://www.w3.org/2001/XMLSchema#string">DNA damage checkpoint sensor mechanism</
    oboInOwl:hasRelatedSynonym>
<oboInOwl:hasRelatedSynonym rdf:datatype="http://www.w3.org/2001/XMLSchema#string">DNA damage checkpoint sensor process</
    oboInOwl:hasRelatedSynonym>
<oboInOwl:hasRelatedSynonym rdf:datatype="http://www.w3.org/2001/XMLSchema#string">sensing involved in DNA damage checkpoint</
    oboInOwl:hasRelatedSynonym>
<oboInOwl:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GO:0072421</oboInOwl:id>
<rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">detection of DNA damage stimulus involved in DNA damage
    checkpoint</rdfs:label>
</owl:Class>
<owl:Axiom>
<owl:annotatedSource rdf:resource="http://purl.obolibrary.org/obo/GO_0072421"/>
<owl:annotatedProperty rdf:resource="http://purl.obolibrary.org/obo/IAO_0000115"/>
<owl:annotatedTarget rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The series of events in which information about
    damage to DNA is received and converted into a molecular signal, contributing to a DNA damage checkpoint.</
    owl:annotatedTarget>
<oboInOwl:hasDbXref rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GOC:mtg_cell_cycle</oboInOwl:hasDbXref>
</owl:Axiom>
<owl:Axiom>
<owl:annotatedSource rdf:resource="http://purl.obolibrary.org/obo/GO_0072421"/>
<owl:annotatedProperty rdf:resource="http://www.geneontology.org/formats/oboInOwl#hasRelatedSynonym"/>
<owl:annotatedTarget rdf:datatype="http://www.w3.org/2001/XMLSchema#string">DNA damage checkpoint sensor mechanism</
    owl:annotatedTarget>
<oboInOwl:hasDbXref rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GOC:mah</oboInOwl:hasDbXref>
</owl:Axiom>
<owl:Axiom>
<owl:annotatedSource rdf:resource="http://purl.obolibrary.org/obo/GO_0072421"/>
<owl:annotatedProperty rdf:resource="http://www.geneontology.org/formats/oboInOwl#hasRelatedSynonym"/>
<owl:annotatedTarget rdf:datatype="http://www.w3.org/2001/XMLSchema#string">DNA damage checkpoint sensor process</
    owl:annotatedTarget>
<oboInOwl:hasDbXref rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GOC:mah</oboInOwl:hasDbXref>
</owl:Axiom>
<owl:Axiom>
<owl:annotatedSource rdf:resource="http://purl.obolibrary.org/obo/GO_0072421"/>
<owl:annotatedProperty rdf:resource="http://www.geneontology.org/formats/oboInOwl#hasRelatedSynonym"/>
<owl:annotatedTarget rdf:datatype="http://www.w3.org/2001/XMLSchema#string">sensing involved in DNA damage checkpoint</
    owl:annotatedTarget>
<oboInOwl:hasDbXref rdf:datatype="http://www.w3.org/2001/XMLSchema#string">GOC:mah</oboInOwl:hasDbXref>
</owl:Axiom>
```

1.2. Primjer povezivanja C++ programa i OWL ontologija

Listing 1.1: C++ program - primjer

```
#include <iostream>
#include <string>
#include "boost/filesystem.hpp"
#include "boost/foreach.hpp"
#include "boost/program_options.hpp"
#include "boost/range.hpp"
#include "factpp/Kernel.hpp"
#include "owlcpp/rdf/triple_store.hpp"
#include "owlcpp/io/input.hpp"
#include "owlcpp/io/catalog.hpp"
#include "owlcpp/logic/triple_to_fact.hpp"

using namespace std;
namespace oc = owlcpp;
namespace bpo = boost::program_options;
namespace bfs = boost::filesystem;

void ispisiTrojku(string t1, string t2, string t3)
{
cout << '\"' << t1 << "\"\t\"" << t2 << "\"\t\"" << t3 << "\"\t" << endl;
}
void ispisiEntitet(string s)
{
cout << '\"' << s << "\"" << endl;
}

int main(int argc, char* argv[])
{
/* parsiranje opcija i parametara */
bpo::options_description od;
od.add_options()
("pomoc,h", "Poruka pomoci")
("datoteka", bpo::value<string>(), "Unos OWL datoteke")
("include,i", bpo::value<vector<string> >()->zero_tokens()->composing(), "Ukljucivanje")
("trojke,t", bpo::bool_switch(), "Ispis trojki")
("klase,k", bpo::bool_switch(), "Ispis klasa")
("svojstva,s", bpo::bool_switch(), "Ispis svojstava")
("individuals,n", bpo::bool_switch(), "Ispis ogranicenja")
("dijagnoza,d", bpo::bool_switch(), "Provjera gresaka u zakljucivanju nakon svakog aksioma ")
("labavo,l", bpo::bool_switch(), "Labavo, ne-striktno parsiranje")
("uspjeh,u", bpo::bool_switch(), "Vraca 1 ako ontologija nije konzistentna");
bpo::positional_options_description pod;
pod.add("datoteka", -1);
bpo::variables_map vm;
store(bpo::command_line_parser(argc, argv).options(od).positional(pod).run(), vm);
notify(vm);

if( !vm.count("datoteka") || vm.count("pomoc") )
{
cout << "Program za rad sa OWL ontologijama." << endl;
cout << "Koristenje:" << endl;
cout << "OntoProg [-i[staza]] [-c] <ontologija.owl>" << endl;
cout << od << endl;
return !vm.count("pomoc");
}
/* Trojke */
oc::Triple_store trojke;
const bfs::path datoteka = vm["datoteka"].as<string>();
try
{
if( vm.count("include") )
{
oc::Catalog katalog;
vector<string> const& vin = vm["include"].as<vector<string> >();
if( vin.empty() )
add(katalog, datoteka.parent_path(), true);
else
BOOST_FOREACH(string const& p, vin) add(katalog, p, true);
}
else
load_file(datoteka, trojke);
if( vm["trojke"].as<bool>() )
{
cout << "Ucitano je " << trojke.map_triple().size() << " trojki, " << trojke.map_node().size()
<< " cvorova, " << trojke.map_ns().size() << " imenskih IRI-a." << endl;
BOOST_FOREACH( owlcpp::Triple const& t, trojke.map_triple() )
ispisiTrojku(to_string(t.subj_, trojke), to_string(t.pred_, trojke), to_string(t.obj_, trojke));
```

```
}
if( vm["svojstva"].as<bool>() )
{
owlcpp::Triple_store::query_b<0,1,1,0>::range r = trojke.find_triple( owlcpp::any,
owlcpp::terms::rdf_type::id(), owlcpp::terms::owl_ObjectProperty::id(), owlcpp::any);
cout << "Pronadjeno je " << r.size() << " svojstava." << endl;
BOOST_FOREACH( owlcpp::Triple const& t, r)
ispisiEntitet(to_string(t.subj_, trojke));
}
if( vm["klase"].as<bool>() )
{
owlcpp::Triple_store::query_b<0,1,1,0>::range r = trojke.find_triple( owlcpp::any,
owlcpp::terms::rdf_type::id(), owlcpp::terms::owl_Class::id(), owlcpp::any);
cout << "Pronadjeno je " << r.size() << " klasa." << endl;
BOOST_FOREACH( owlcpp::Triple const& t, r )
{
ispisiEntitet(to_string(t.subj_, trojke));
}
}
if( vm["individuals"].as<bool>() )
{
owlcpp::Triple_store::query_b<0,1,1,0>::range r = trojke.find_triple( owlcpp::any,
owlcpp::terms::rdf_type::id(), owlcpp::terms::owl_NamedIndividual::id(), owlcpp::any);
cout << "Pronadjeno je " << r.size() << " individua." << endl;
BOOST_FOREACH( owlcpp::Triple const& t, r )
ispisiEntitet(to_string(t.subj_, trojke));
}
}
catch(...)
{
cerr << "Greska prilikom ucitavanja trojki iz datoteke." << endl;
cout << "Poruka boosta: " << boost::current_exception_diagnostic_information() << endl;
return 1;
}
/* Zakljucivac FaCT++ */
try
{
ReasoningKernel kernel;
const std::size_t n = submit(trojke, kernel, vm["labavo"].as<bool>(), vm["dijagnoza"].as<bool>());
cout << "Generirano je " << n << " aksioma." << endl;
const bool konzistentna = kernel.isKBConsistent();
if(konzistentna)
cout << "Ontologija je konzistentna." << endl;
else
cout << "Ontologija nije konzistentna." << endl;
return vm["uspjeh"].as<bool>() && ! konzistentna;
}
catch(...)
{
cout << "Greska prilikom upotrebe zakljucivaca." << endl;
cout << "Poruka boosta: " << boost::current_exception_diagnostic_information() << endl;
return 1;
}
}
```

Listing 1.2: Pomoćni jamroot za povezivanje sa boost bootstrapom

```
use-project /owlcpp : $(OWLCPP[1]) ;

exe OntoProg
: #sources
Razvoj.cpp
: #requirements
<link>static
<library>/owlcpp//rdf
<library>/owlcpp//io
<library>/owlcpp//logic
<library>/boost/program_options
;
```