Matematički fakultet
Univerzitet u Beogradu

Semantički veb – primena u automatskom upravljanju uređajima

Master rad

Mentor:
Prof. dr. Gordana Pavlović-Lažetić

Autor:
Marija Bogdanović

Beograd , 2011.

Sadržaj

1 Semantički veb – uvod

1.1 Motivacija i potreba za uvodjenjem tehnologija semantičkog veb-a

Rastuća popularnost veba (*WWW - World Wide Web*) je promenila način razmišljanja ljudi o računarima. Prvobitni računari su korišćeni za izvršavanje numeričkih operacija. Sa rastućim brojem personalnih računara, postoji sve više aplikacija za obradu teksta, tabela i igranje igara. Rastuća popularnost mobilnih uređaja, kao što su: prenosni računari (*engl. notebook*), PDA (*engl. personal digital assistant*) i mobilni telefoni sa mogućnošću povezivanja na Internet čak i sa javnih mesta, promenila je pogled na računare kao na ulaznu tačku globalnog informacionog prostora veba. Veb je uspeo da preraste svoje tehnološke okvire i postane kulturni i društveni fenomen koji je prisutan u svim sferama života. Ljudi koriste veb za posao, zabavu, informisanje, za kupovinu i prodaju, druženje i sl. Veb je jedini slobodan medij na kome mogu da se izraze praktično svi koji imaju nešto da kažu. Ova sloboda donosi i problem kako iz velikog broja informacija koje postoje na vebu izdvojiti one koje su nam potrebne, jer često ne postoji mehanizam za proveru verodostojnosti informacija. Većina aktivnosti na vebu je podržana softverskim alatima koji uspostavljaju vezu među dokumentima, ali neophodni su efikasniji alati za pretragu. Potraga za željenom informacijom često je vrlo dugotrajna aktivnost, a razlog za to je što rezultati pretrage koja je izvršena jednim softverskim alatom nisu automatski dostupni drugim softverskim alatima. Pretraživači su često izolovane aplikacije koje nemaju komunikaciju sa drugim aplikacijama i alatima. Postoje alati koji mogu preuzimati tekstove i razdeliti ih na delove, raditi proveru pravopisa, brojati njihove reči, ali kada je u pitanju tumačenje rečenica i odabir korisne informacije za korisnike mogućnosti softverskih alata su još uvek male. Jedan pristup rešavanju ovog problema je da se veb sadržaj koristi na način na koji je trenutno predstavljen, u obliku HTML(*HyperText Markup Language*) dokumenta, i da se razviju tehnike pretrage bazirane na veštačkoj inteligenciji i računarskoj lingvististici. Pristup alternativan ovom je da se veb sadržaj predstavi u obliku koji je jednostavnije mašinski obraditi korišćenjem neke od inteligentnih tehnika. Ovaj pristup je poznat kao *semantički veb*. Semantički veb nije zamišljen kao nova globalna mreža sa sopstvenim izvorima informacija, paralelno sa postojećim vebom, već će se postupno razvijati na njegovoj osnovi, kao nadogradnja postojećeg veba.

Korišćenje veba je prošireno alatima koji omogućavaju interaktivno učešće korisnika. Takvi alati su poznati pod nazivom Web 2.0. Deo tih Web 2.0 alata uključuje vikije (*engl. wiki*), zbirke veb stranica koje omogućavaju korisnicima dodavanje novih sadržaja putem korisničkih aplikacija. Viki biblioteke omogućavaju stvaranje baze znanja kroz saradnju svih korisnika veba jer one daju skoro potpunu slobodu za dodavanje i promenu informacija bez ograničenja vlasništva sadržaja, pristupa i redosleda rada. Viki biblioteke se koriste za razne svrhe, uključujući i razvoj znanja dajući doprinos širokom spektru aktivnosti korisnika. Najpoznatiji

primer viki bibilioteke je opšte poznata i široko prihvaćena i korišćena *Wikipedia*[1]. Viki sistemi su prepoznati kao nešto što definitivno može imati koristi od korišćenja semantičkih veb tehnologija, a glavna ideja je da se struktura vikija proširi na nivo viši od same navigacije. To može biti postignuto obogaćivanjem veza (*engl. link*) između viki stranica strukturiranim tekstom i semantičkim referencama koje se odnose na model znanja koji je sačuvan u vikijima. Na ovaj način ove dodatne informacije mogu biti iskorišćene za određenu konteksno zavisnu prezentaciju stranica, napredne upite, i proveru doslednosti sadržaja.

1.2 Osnovna ideja i definicija

Semantički veb je nastao kao dopuna postojećeg veba u potrazi za efikasnijim rešenjima za pronalaženje informacija. Ovim je omogućena bolja saradnja između računara i korisnika. Osnovna ideja semantičkog veba je da se sve informacije koje se pojavljuju na vebu označe posebnim oznakama čime bi se dobilo to da računari mogu automatizovano da povezuju podatke jednog tipa (npr. fotografija sa mestom, datumom i vremenom nastanka) sa podacima nekog drugog tipa (npr. meteorološki izveštaj za period kada je nastala fotografija). Sve je više istraživackih napora da se poveća efikasnost pretraživanja na vebu kako bi se došlo do što relevantnijih informacija. Primarnu ulogu u razvoju standarda i tehnologija, kao i u kreiranju preporuka i specifikacija koje bi obezbedile dugoročni razvoj veba u pravcu semantičkog veba ima World Wide Web Consortium[2](W3C), međunarodno telo za standardizaciju veba. Pokretač inicijative semantičkog veba je direktor W3C-a, britanski inženjer i profesor MIT univerziteta,Tim Berners-Li (*Sir Timothy John "Tim" Berners-Lee*), jedan od izumitelja veba i čovek koji je uz pomoć saradnika u CERN-u implementirao prvu uspešnu internet komunikaciju između HTTP klijenta i servera. On od ove inicijative očekuje realizaciju svoje izvorne vizije veb-a u kojoj značenje informacije ima daleko važniju ulogu nego u postojećem vebu.

Definicija semantičkog veba koju je dao Berners-Li[3]*: Semantički veb je nastavak, ekstenzija postojećeg veba gde je informaciji dato precizno definisano značenje i koji bolje omogućava saradnju između računara i korisnika.*

Semantički veb je vizija veba sledeće generacije, koja omogućava aplikacijama da automatski prikupljaju veb dokumenta iz različitih izvora, integrišu i procesiraju informacije i sarađuju sa drugim aplikacijama da bi izvršile sofisticirane zadatke u korist ljudi [2]. Semantički veb je proširenje postojećeg veba u kom su podaci definisani i povezani na način koji omogućava da ih mašine koriste ne samo za potrebe prikazivanja, već i za automatizaciju, integraciju i ponovno korišćenje u raznovrsnim aplikacijama. Vizija semantičkog veba je da podaci koji se

[1] http://www.wikipedia.org/
[2] W3C, World Wide Web Consortium, http://www.w3.org/

nalaze bilo gde na vebu budu dostupni i razumljivi, kako ljudima, tako i mašinama. Veb sadržaj je trenutno oblikovan tako da bude čitljiv za ljude a ne programe. HTML je dominantno jezik na kome su zapisane veb stranice (kreirane ili direktno ili pomoću alata). Ovakav način prikaza informacija je zadovoljavajući kada su u pitanju ljudi ali nije razumljiv i dostupan mašinama. Semantički veb pristup rešavanju ovog problema je ideja da se HTML zameni jezicima koji bi omogućili da veb stranice pored oblikovanih informacija čitljivih za ljude sadrže i semantičke metapodatke – mašinski čitljive podatke o izvorima informacija, njihovim relacijama sa drugim resursima, i razlozima za međusobno povezivanje tih izvora[1]. Razmotrimo naredni primer. Neka postoji recimo virtualna veb prodavnica. Današnje veb stranice te prodavnice sadržale bi informacije koju može da razume osoba ali ne i računar. Naime, osobi je jasno da stranica prikazuje sliku, naziv i cenu proizvoda, međutim računar svakako nema način da razume da je u pitanju više od jedne slike i dva kratka teksta pored te slike. Ideja semantičkog veba je da se svakom bitnom delu informacije koja je prikazana na veb stranici dodaju određeni metapodaci koji će jednoznačno opisati da se ovde radi o tačno tom proizvodu sa tačno tom cenom i da je taj proizvod na prodaju. Tako će i svaki drugi računar koji "pročita" ovu stranicu imati dovoljno podataka da "razume" o čemu se radi.

1.3 Cilj

Cilj semantičkog veba je da pomogne korisnicima u njihovim svakodnevnim aktivnostima na vebu u smislu kreiranja naprednijih sistema za upravljanje informacijama, tj. sistema koji bi trebalo da zadovolje sledeće kriterijume:

- Informacije bi trebalo da budu organizovane u konceptualne celine u skladu sa svojim značenjem.
- Automatizovani alati bi trebalo da podrže održavanje tačnosti infomacija proverom nedoslednosti i dolaženjem do novog znanja.
- Pretraga po ključnim rečima bi trebalo da bude zamenjena odgovorima na postavljena pitanja.
- Tražene informacije bi trebalo da budu prikupljene, izdvojene i predstavljene na način pogodan za čoveka.
- Trebalo bi da bude obezbeđena mogućnost da se odgovori na pitanje sintetizuju iz nekoliko dokumenata.
- Trebalo bi da bude obezbeđena mogućnost definisanja prava za uvid u informacije.
- Trebalo bi da se smanji ili potpuno nestane potreba za programiranjem omotača klasa (*engl.wrapper class*) korišćenjem programskih jezika niskog nivoa.

1.4 Razvoj

1.4.1 Principi na kojima je zasnovan semantički veb

Razvoj semantičkog veba zasnovan je na sledećim tehnološkim i organizacionim principima[5]:

- jednostavnost,
- modularni dizajn,
- decentralizacija
- tolerancija.

Druga dva principa objašnjavaju pravac razvoja semantičkog veba. Decentralizacija se ne odnosi samo na činjenicu da se informacije nalaze na različitim lokacijama na internetu, već i na to da je svako odgovoran za informacije koje postavi na veb, kako za njihov sadržaj, tako i za oblik u kome se one nalaze. Šta više, decentralizacija se ne odnosi samo na sadržaj, već i na tehnološki razvoj veba – proizvođači softvera i pružaoci različitih usluga na vebu mogu u njega da uključuju sopstvena tehnološka rešenja. Sledeći važan princip razvoja veba jeste tolerancija pod kojom se podrazumeva da nove tehnologije ne smeju onemogućiti korišćenje starih tehnologija. Time se obezbeđuje postepen razvoj veba.

1.4.2 Semantičke tehnologije

Semantički veb zahteva da u mašinski čitljivom obliku bude predstavljena znatno potpunija slika oblasti koja se predstavlja, a takav semantički bogatiji opis neke oblasti kojom se bavimo naziva se *ontologija*. Reč ontologija prvi put je korišcena u filozofiji i ona predstavlja spoj dve reči grčkog porekla "*ontos*" što znači biće, stvarnost, i "*logia*" što znači nauka. Dakle u svom izvornom filozofskom značenju ontologija predstavlja nauku o biću, o onome što postoji. Pojam ontologije u računarstvu i informatici se odnosi na predstavljanje znanja u obliku formalno definisanog sistema pojmova i relacija između tih pojmova. Ako se ima u vidu da se inteligentni sistemi bave i predstavljanjem i obradom znanja, javlja se potreba za ponovnim korišćenjem znanja nekog domena. Na ovaj način se postiže da se znanje koje je prikupljeno u toku rešavanja jednog problema može ponovo koristi u novim verzijama inteligentnog sistema. Međutim, pored mogućnosti za ponovnu upotrebu znanja, korisno je omogućiti da se znanje jednog domena može deliti između više različitih korisnika (ljudi ili softverskih agenata). U suštini, istraživanja u oblasti veštačke inteligencije i predstavljanja znanja, pojam ontologije vezuju za mogućnost ponovne upotrebe (*engl. reusability*) i deljenja (*engl. sharing*) znanja nekog domena. Ovo znači da je glavna svrha ontologije da omogući prenošenje i razmenu znanja.

Na primer, može se desiti da određeni program treba da poredi ili kombinuje informacije iz dve različite baze podataka. Kod dve različite baze podataka mogu da se koriste različiti identifikatori za ono što je u stvari isti pojam, kao što je npr. poštanski broj. U ovom slučaju javlja se problem da program mora da zna da su ova dva termina upotrebljena tako da znače istu stvar. U idealnom slučaju, potrebno je da program poseduje način da otkrije takva zajednička značenja kakve god baze podataka da koristi. Rešenje ovog problema nudi jedna od osnovnih komponenata semantičkog veba - ontologija. Ontologija je dokument ili datoteka koja formalno definiše relacije izmedu termina. Tipična vrsta ontologije za veb sastoji se od klasifikacije i skupa pravila zaključivanja. Klasifikacija definiše klase objekata i relacije između njih. Na primer, adresa može biti definisana kao potklasa klase lokacija, a poštanski kod grada može biti definisan tako da se odnosi samo na lokacije. Klase, potklase i relacije između njihovih instanci su veoma važne i korisne na vebu. Može se izraziti veliki broj relacija izmedu instanci klasa tako što se dodeljuju osobine klasama i dozvoljava se potklasama da nasleđuju takve osobine. U navedenom primeru ako je poštanski kod grada potklasa klase grad, a gradovi u opštem slucaju imaju veb stranice, može se zaključiti da je veb stranici pridružen poštanski kod grada čak i ako nijedna baza podataka ne povezuje poštanski kod grada direktno sa veb stranicom. Ontologije mogu poboljšati funkcionisanje veba na razne načine. Na jednostavan način mogu se upotrebiti da poboljšaju tačnost veb pretraživanja. Program za pretraživanje može tražiti samo one stranice koje referišu precizan pojam, umesto svih onih koje koriste dvosmislene ključne reči.

Pored ontologije, za neke primene, mogu se definisati i složenija pravila zaključivanja koja omogućavaju da se iz neposredno datih podataka na semantičkom vebu izvedu nove informacije. Za ovo se koriste različite metode koje su razvijene u oblasti veštačke inteligencije, kao što su ekspertski sistemi i logičko programiranje. Logika je disciplina koja proučava principe rezonovanja još od Aristotela. Logika predstavlja prvi, formalni jezik za izražavanje znanja. Logika obezbeđuje dobro razumevanje formalne semantike jer je u logici formalno definisano značenje rečenica. Automatizovani rasuđivači (*engl. automatic reasoners*) mogu da izvedu zaključke iz postojećeg znanja, čime ono postaje upotrebljeno. Takvi rasuđivači su intenzivno proučavani u oblasti veštačke inteligencije. Važna osobina logike je da može da pruži objašnjenja za zaključke. Pored toga istraživači veštačke inteligencije su razvili načine predstavljanja objašnjenja na lako čitljiv način, organizovanjem dokaza kroz prirodnu dedukciju i grupisanjem više elementarnih koraka dokaza u pojedinačne krupnije korake koji bi čovek obično i uzeo u obzir kao jedan korak dokaza.

Agenti su delovi softvera – programi koji rade samostalno i aktivno. Konceptualno su evoluirali iz koncepta objektno-orijentisanog programiranja i razvoja softvera zasnovanog na komponentama (*engl.component-based*). Lični agenti semantičkog veba dobijaju neke zadatke i prioritete od korisnika, traže informacije na veb izvorima, komuniciraju sa drugim agentima, upoređuju informacije o zahtevima i podešavanjima korisnika, vrše izbor informacija, i daju odgovore korisniku. Agenti ne zamenjuju korisnika na semantičkom vebu, niti obavezno donose

odluke, već je u većini slučajeva njihova uloga da prikupe, organizuju i predstave korisniku skup mogućih informacija. Semantički veb agenti koriste: metapodatke, za identifikaciju i izdvajanje informacija sa veb izvora; ontologije, za pomoć u veb pretragama, za interpretiranje preuzete informacije i za komunikaciju sa drugim agentima; logiku, za obradu preuzetih informacija i donošenje zaključaka. Dok su neki agenti u stanju da donose logičke zaključke, drugi imaju mogućnost da potvrde dokaze. Većina tehnologija potrebnih za realizaciju semantičkog veb-a izgrađena je u oblasti veštačke inteligencije.

1.5 Primene

Trenutno najveća potreba semantičkog veba je integracija, standardizacija, razvoj alata, kao i usvajanje od strane korisnika. Njegov napredak je usko povezan sa tehnološkim napretkom. Sve češće, tehnologije semantičkog veba nalaze primenu u izgradnji sistema baziranih na znanju, posebno u domenu prikupljanja i integracije resursa znanja, kao i za pretraživanje i prezentovanje znanja na vebu. Semantičke veb tehnologije podstiču i nove trendove u razvoju softverskih aplikacija, kao što su npr. razvoj novih alata otvorenog koda (*engl.open-source*), ponovna upotreba već gotovih ontologija, korišćenje besplatnih veb servisa i aplikacija, itd.

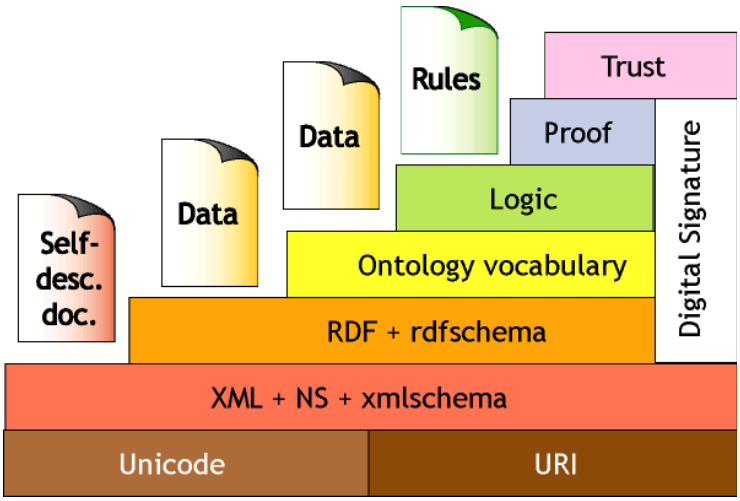
2 Semantički veb – tehnologije

Arhitekturu semantičkog veba čine sledeći standardi:

- XML(*Extensible Markup Language*) - definiše opštu sintaksu za označavanje podataka pomoću odgovarajućih oznaka (*engl. tag*) koje imaju poznato ili lako razumljivo značenje;
- XML Schema - mehanizam za definisanje strukture XML dokumenta, na osnovu XML sheme program može da odredi da li je neki dokument formalno ispravan.
- RDF(*Resource Description Framework*) - model podataka za opis objekata i veza između objekata, može biti izražen XML sintaksom
- RDF shema (RDFS) - rečnik za definisanje svojstava, klasifikaciju i generalizaciju objekata opisnih RDF modelom
- OWL (*Ontology Definition Language*) - bogat jezik za opisivanje osobina i klasa, odnosa između klasa (npr. disjoint), kardinalnosti, jednakosti, sadrži bogatiji prikaz svojstava, karakteristika i numerisanih klasa. OWL je jezik za opisivanje ontologija na semantičkom veb-u. OWL proširuje RDFS u smislu opisivanja sadržaja, relacija i kardinalnosti.

Razvoj semantičkog veba teče u slojevima, svaki sloj se koristi za "izgradnju" sloja iznad. U izgradnji jednog sloja semantičkog veba na vrhu drugog moraju biti ispoštovana dva principa:

- Kompatibilnost naniže. Svaki sloj treba da bude u stanju da interpretira i koristi informacije napisane na nižim nivoima. Na primer, agenti koji koriste OWL semantiku treba da budu u stanju da u potpunosti iskoriste informacije napisane u RDF-u i RDFS-u.
- Razumevanje naviše. Dizajn novog sloja treba da bude takav da je sloj u stanju da bar delimično koristi informacije na višim slojevima. Na primer, agenti koji koriste RDF i RDFS semantike mogu interpretirati i znanje napisano u OWL-u, bez obzira na one elemente koji nisu poznati RDF-u i RDFS-u.

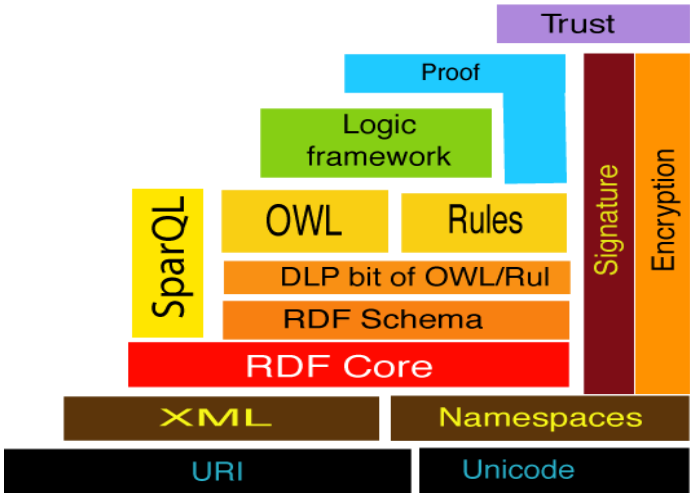


Slika 2. 1 Arhitektura semantičkog veba [11]

Slika 2.1 prikazuje "Slojevitu tortu semantičkog veba" (*engl. Semantic Web layer-cake*) , kako je ovu slojevitu arhitekturu nazvao Tim Berners-Li, koja opisuje glavne slojeve dizajna semantičkog veba.

- Na dnu strukture semantičkog veba je XML jezik koji dozvoljava struktuirano pisanje veb dokumenata korišćenjem korisnički-definisane terminologije. XML je posebno pogodan za objavljivanje dokumenata preko veba.
- Na slici 2.1, RDF se nalazi na iznad sloja XML sheme. RDF je osnovni model podataka, model entiteta i odnosa za definisanje jednostavnih objekata i veza između njih. RDF model podataka ne oslanja se direktno na XML, ali je RDF sintaksa zasnovana na XML-sintaksi. RDF shema obezbeđuje modelovanje primitiva za organizovanje veb objekata u hijerarhiju. Ključne primitive su klase i svojstva, odnosi potklasa i podsvojstava, domeni i ograničenja.
- RDF shema(*engl.RDF schema*) se zasniva na RDF-u. RDF shema se može posmatrati kao primitivni jezik za opis ontologije objekata ali postoji potreba za složenijim jezicima.
- Ontologije proširuju RDF sheme i obezbeđuju reprezentaciju složenijih odnosa između objekata u vebu.
- Logički sloj (*engl.Logic*) se koristi za poboljšanje jezika ontologija i za omogućavanje zapisa znanja za određene aplikacije u formi subjekat-predikat-objekat.
- Sloj dokaza (*engl.Proof*) se odnosi na deduktivni proces, kao i na predstavljanje dokaza u veb jezicima i provere validnosti veb stranice.
- Na kraju, sloj poverenja (*engl.Trust*) se pojavljuje kroz korišćenje digitalnih potpisa i znanja proizlašlog iz rada pouzdanih agenata. Na vrhu piramide se nalazi poverenje jer je ključni koncept semantičkog veba činjenica da će veb postići svoj puni potencijal kada korisnici budu imali poverenje u bezbednost i kvalitet dobijenih informacija.

Na slici 2.2 prikazan je alternativni raspored slojeva semantičkog veba, koji je izgrađen uzimajući u obzir tekući razvoj.



Slika 2. 2 Alternativni raspored slojeva semantičkog veba

U ovakvom rasporedu slojeva, sloj ontologija je predstavljen sa dve alternative: trenutni standardni veb jezik ontologije - OWL i jezik zasnovan na pravilima. DLP[3](*Description Logic Programs*) predstavlja presek OWL-a i logike, i služi kao zajednički osnov za ove dve alternative.

2.1 Struktuirani veb dokument – XML

Danas je HTML standardni jezik za pisanje veb stranica. HTML je izveden iz SGML[4]-a (*Standard Generalized Markup Language*), sistema za definisanje jezika za označavanje, i međunarodnog standarda ISO 8879 za definisanje metode za predstavljanje podataka nezavisnih od uređaja i sistema, razumljvih i za ljude i za mašine. Iako je HTML doprineo ogromnoj popularnosti Interneta, sa razvojem Interneta primećeni su i njegovi nedostaci. Jedan od krupnijih nedostataka je složena obrada elementa opisanih pomoću HTML-a. W3C je 1998.godine predložio uvođenje novog jezika za obeležavanje XML (*Extensible Markup Language*). XML je deskriptivan jezik za obeležavanje i omogućava čuvanje, obradu i lak prenos podataka koje opisuje. Cilj nije bio da XML zameni HTML, već da se napravi jedan meta-jezik za obeležavanje koji bi poslužio kao osnova za kreiranje raznih drugih jezika. XML je jezik kojim se opisuje označavanje: nema fiksnog skupa oznaka već omogućava korisnicima da definišu sopstvene oznake. Na primer, veb stranica koja sadrži podatke o određenoj knjizi može da sadrži sledeći HTML segment:

[3] http://logic.aifb.uni-karlsruhe.de/
[4] http://www.w3.org/MarkUp/SGML/

<h2>Nonmonotonic Reasoning: Context-Dependent Reasoning</h2>
by V. Marek M. Truszczynski
*
Springer 1993*
*
ISBN 0387976892*

XML reprezentacija iste informacije može izgledati ovako:

<knjiga>
<naslov>Nonmonotonic Reasoning: Context-Dependent Reasoning</naslov>
<autor>V. Marek</ autor>
<autor>M. Truszczynski</autor>
<izdavac>Springer</ izdavac>
<godina>1993</godina>
<ISBN>0387976892</ISBN>
</knjiga>

Obe reprezentacije koriste oznake kao što su *<h2>* i *<godina>*. HTML i XML su jezici za označavanje: oni omogućavaju upis nekog sadržaja i informaciju o tome kakvu ulogu ima taj sadržaj. XML se zasniva na oznakama (*engl.tag*) kao i HTML. Sve oznake u XML-u moraju biti zatvorene. Oba jezika su izgrađena tako da budu lako razumljivi i upotrebljivi za ljude, ali ostaje pitanje koliko su "razumljivi" mašinama. Pretpostavimo da se prethodnoj HTML stranici pristupa veb pretraživačem. Ne postoji eksplicitna informacija o tome ko su autori knjige. Može se zaključiti da se imena autora pojavljuju odmah posle naslova ili neposredno iza reči *by*, ali nema garancije da se ova pravila uvek prate a čak i da je tako nije jasno da li postoje dva autora, "V. Marek "i" M. Truszczinski ", ili samo jedan, pod nazivom"V Marek i M. Truszczinski". Za odgovor na ovo pitanje potrebna je dodatna obrada teksta u kojoj može doći do greške. Problemi nastaju iz činjenice da HTML dokument ne sadrži struktuirane informacije, t.j. informacije o delovima dokumenta i njihovim relacijama. U navedenom primeru, druga varijanta je dostupnija mašini jer se koriste oznake koje opisuju pojmove i relacije od interesa (autor, izdavač, ...) umesto fiksiranog skupa HTML oznaka dizajniranog pre svega za opis hipertekstualnih dokumenata. XML dokument je daleko lakše dostupan mašini jer je svaki podatak opisan. Neke forme njihovih odnosa su definisane kroz struktuirano stablo. Na primer, *<autor>* oznake se pojavljuju unutar *<knjiga>* oznaka tako da opisuju svojstva određene knjige. Aplikacija za obradu XML dokumenta može da zaključi da se element *autor* odnosi na element *knjiga*. HTML ima za cilj da opiše prikaz hipertekstualnih dokumenata, tako da je skup oznaka ograničen na naslove, pasuse, liste, tabele i sl. Informacije opisane korišćenjem XML-a se mogu koristiti na različite načine, pa i za to da se korisniku definiše skup elemenata i oznaka pogodan za aplikaciju.

Od svog nastanka pa do danas XML je stekao veliku popularnost. Razvijen je niz novih jezika za obeležavanje u skladu sa pravilima XML-a. Jezici za obeležavanje nastali iz XML-a često se

nazivaju XML-aplikacije. XML aplikacije su definisane u različitim oblastima: XHTML[5](*Extensible Hypertext Markup Language*), SVG[6] (*Scalable Vector Graphics*), SMIL[7](*Synchronized Multimedia Integration Language*), MathML[8] (*Mathematical Markup Language*), BSML[9](*Bioinformatic Sequence Markup Language)*, HRML[10](*Human Resources Markup Language*), AML[11](*Astronomical Markup Language*), NewsML[12](*News Markup Language*) i IRML[13](*Investment Research Markup Language*). Svaka specifična XML aplikacija definiše pojedinačan format podataka, dok XML meta jezik za opis formata podataka za razmenu između aplikacija daleko nadmašuje njegovu prvobitnu namenu kao jezika za označavanje dokumenata.

XML dokument je dobro formiran ako poštuje određena sintaksička pravila. XML dokument je validan ako je dobro formiran, koristi struktuirane informacije i poštuje strukturiranje informacija. Ako dve aplikacije pokušavaju da komuniciraju i žele da koriste isti rečnik, potrebno je definisati sve elemente, atribute i imena koja se mogu koristiti, a osim toga, trebalo bi definisati i moguće vrednosti atributa, koji elementi se mogu ili moraju pojaviti u okviru drugih elemenata i sl. Uz takvo strukturiranje podataka poboljšana je mogućnost provere validnosti dokumenta. XML parser je program koji čita XML datoteku i čini dostupnim podatke koji se u njoj nalaze.

Svaki XML dokument mora da ima zaglavlje. Zaglavlje XML dokumenta se sastoji od XML deklaracije i opcionog poziva spoljašnjeg strukturiranja dokumenata. XML deklaracija : *<?xml version="1.0" encoding="UTF-16"?>* ima značenje da je dokument XML dokument, i definiše verziju i kodiranje karaktera koji se koriste u određenom dokumentu (kao što su na primer UTF-8, UTF-16 i ISO 8859-1). Kodni raspored nije obavezan, ali se njegova specifikacija smatra dobrom praksom. Prema XML 1.0[14] specifikaciji, svi procesori su obavezni da automatski podržavaju i prepoznaju UTF-8 i UTF-16 kodiranje karaktera. Ako se koristi kodni raspored različit od UTF-8 ili UTF-16, onda je neophodno koristiti odgovarajuću XML deklaraciju.

XML elementi predstavljaju stvari o kojima govori XML dokument. Oni sačinjavaju glavni koncept XML dokumenta. Element se sastoji od oznake za otvaranje, sadržaja i oznake za zatvaranje. Na primer:

<autor> Marija Bogdanovic </autor>

[5] http://www.w3.org/TR/xhtml-modularization/
[6] http://www.w3.org/TR/SVG/
[7] http://www.w3.org/TR/SMIL/
[8] http://www.w3.org/Math/
[9] http://xml.coverpages.org/bsml.html
[10] http://xml.coverpages.org/hrmlAnn19981006.html
[11] http://xml.coverpages.org/aml.html
[12] http://www.newsml.org/
[13] http://xml.coverpages.org/irml.html
[14] http://www.w3.org/TR/REC-xml/

Ime elementa može biti izabrano gotovo slobodno uz vrlo malo ograničenja. Najvažnije je da prvi znak mora da bude slovo, podcrta (_) ili dve tačke (:), i da ime ne može početi sa "xml". Sadržaj može da bude tekst, da sadrži druge elemente ili da bude prazan. Na primer

<autor><ime> Marija Bogdanovic </ime><tel> +1234567</tel></autor>

Ako nema sadržaj, element se naziva *prazan element*. Prazan element kao što je *<autor></autor>* može biti skraćen kao *<autor/>.* Za razliku od elemenata, atributi ne mogu biti ugnježdeni. Atribut predstavlja par ime-vrednost unutar para oznaka elementa:

<autor ime=" Marija Bogdanovic" tel="+1234567"/>

Primer atributa za neprazan element je:

<narudžbenica brojNarudžbenice="23456" kupac="Pera Peric" datum="15 Oktobar 2002">
<stavka brojStavke="a528" kolicina="1"/><stavka brojStavke ="c817" kolicina="3"/>
</narudžbenica>

Iste informacije mogu biti napisane na sledeći način, zamenjujući atribute ugnježdenim elementima:

<narudžbenica>
<brojNarudžbenice>23456 </ brojNarudžbenice>
<kupac>Pera Peric </ kupac>
<datum>15 Oktobar 2002 </ datum>
<stavka>
<itemNo>a528 </itemNo>
<kolicina>1 </ kolicina>
</ stavka>
<stavka>
<brojStavke>c817 </ brojStavke>
<kolicina>3 </ kolicina>
</stavka>
< narudžbenica>

Komentar je deo teksta koji treba da bude ignorisan od strane parsera. Komentar ima sledeći oblik:
<!—ovo je komentar -->

Uputstva za obradu obezbeđuju mehanizam za prenos informacije o tome kako aplikacija obrađuje elemente. Opšti oblik je *<?target instruction?>*, na primer:

<?stylesheet type="text/css" href="mystyle.css"?>.

Da bi XML dokument bilo moguće obraditi, moraju biti poštovana sledeća pravila:

- Postoji samo jedan koren (*engl.root*) element u dokumentu.
- Svaki element sadrži oznake za otvaranje i odgovarajuće zatvaranje.
- Elementi mogu biti ugnežđeni, ali se ne smeju preklapati, a svaki element, osim korenskog, mora se u potpunosti sadržati u drugom elementu.
- Prazan element može biti označen oznakom praznog elementa.
- Jedan element na može imati dva atributa sa istim imenom.
- Vrednosti atributa moraju biti unutar navodnika.
- Komentari i instrukcije za obradu se ne mogu navoditi unutar oznake.
- Karakteri < i *&* ne smeju da se pojavljuju unutar vrednosti atributa, ni unutar karakterskog sadržaja elemenata.
- Nazivi elemenata mogu sadržati slova, brojeve i posebne znakove, moraju počinjati slovom, ali ne smeju počinjati tekstom *xml* ni *XML*, a ne smeju sadržati ni praznine.
- Imena elemenata su osetljiva na razliku između malih i velikih slova (*eng*. *case-sensitive*). Tako elementi *STAVKA*, osoba i *Stavka* za XML nisu isti.

Postoje dva načina za definisanje strukture XML dokumenata: definicija tipa dokumenta – DTD (*Document Type Definition*), stariji i ograničeniji način, i XML shema, koja nudi proširene mogućnosti, pre svega za definisanje tipova podataka. Ponekad se i navodi da li dokument sadrži sve delove u sebi, to jest, da li ima poziva spoljašnjeg strukturiranja dokumenata: *<?xml version="1.0" encoding="UTF-16" standalone="no"?>*. Referenca na spoljašnje strukturiranje dokumenta izgleda ovako: *<!DOCTYPE book SYSTEM "book.dtd">* i znači da se informacije o struktuiranju dokumenta nalaza u lokalnoj datoteci sa imenom *book.dtd*. Referenca može biti i URL. Ako se koristi samo lokalno ime ili samo URL, onda se koriti oznaka SYSTEM, a ako se koristi i lokalni naziv i URL adresa onda bi trebalo koristiti oznaku PUBLIC, na primer:

<! DOCTYPE book PUBLIC "-//MITPress//DTD DocBook XML//EN" "../dtds/book.dtd">

Komponente DTD mogu da se definišu u posebnoj datoteci (spoljni DTD) ili u samom XML dokumentu (interni DTD). Obično se koristite spoljni DTD-ovi, jer se njihova definicija može koristiti za nekoliko dokumenata, u suprotnom dolazi do dupliranja i održavanje konzistentnosti tokom vremena postaje teško. Na primer, DTD za tip elementa

<predavač><ime> Dejvid Billington </ime><telefon> +61-7-3875 507 </telefon></predavač>

izgleda ovako:

<!ELEMENT predavač (ime,telefon)>
<!ELEMENT ime (#PCDATA)><!ELEMENT telefon (#PCDATA)>.

Smisao ovog DTD je sledeći:
Element tipa *predavač* može da koristi elemente tipa *ime* i *telefon* unutar svog sadržaja i element tipa *predavač* sadrži elemente tipa *ime* i *telefon* u tom redosledu. U navedenom DTD-u, jedini atomični tip elemenata je #PCDATA. Na primer, za element :

<narudžbenica brojNarudžbenice="23456" kupac="John Smith" datum="15 Oktobar 2002">
<stavka brojStavke="a528" kolicina="1"/>
<stavka brojStavke ="c817" kolicina ="3"/>
</narudžbenica>

DTD može izgledati ovako:

<!ELEMENT narudžbenica (stavka+)>
<!ATTLIST narudžbenica
brojNarudzbenice ID #REQUIRED
kupac CDATA #REQUIRED
datum CDATA #REQUIRED>
<!ELEMENT stavka EMPTY>
<!ATTLIST stavka
brojStavke ID #REQUIRED
kolicina CDATA #REQUIRED
komentar CDATA #IMPLIED>

Novina je da se tip elementa *stavka* definiše kao prazan. Još jedan novi aspekt je pojava znaka *+* nakon stavke u definiciji tipa elementa *narudžbenica*. To je jedan od operatora kardinalnosti:

- ?: Opcioni.Elemet se pojavljuje nula puta ili jednom.
- *: Opcioni i ponovljiv. Element se pojavljuje nula ili više puta.
- +: Obavezan i ponovljiv. Element se pojavljuje jednom ili više puta. Na primer: *<!ELEMENT TABLE (CAPTION?,TR+)>*

Ukoliko nema operatora kardinalnosti, podrazumeva se tačno jedna pojava elementa specificiranog tipa. Pored definisanja elemenata, moraju se definisati i atributi. Atributi se definišu u *atribut listi*. Prva komponenta je ime tipa elementa koji je dopušten u listi, a zatim sledi spisak trojki ime atributa, tip atributa i vrsta vrednosti. Ime atributa je ime koje se može koristiti u XML dokumentu pomoću DTD.

Tipovi atributa su slični predefinisanim tipovima podataka, ali izbor je vrlo ograničen. Najvažniji tipovi atributa su:

- CDATA, niz karaktera
- ID, ime koje je jedinstveno na celom XML dokumentu
- IDREF, referenca na drugi element sa ID atributom koji ima istu vrednost kao i IDREF atribut
- IDREFS, niz IDREF-a
- (V_1 |... | V_r), nabrajanje svih mogućih vrednosti

Postoje četiri vrste vrednosti:

- #REQUIRED - Atribut mora da se pojavljuje u svakom pojavljivanju tipa elementa u XML dokumentu. U prethodnom primeru, *stavkaNo* i *quantity* moraju uvek da se pojave u okviru svakog *stavka* elementa.
- #IMPLIED – Pojava atributa nije obavezna. U primeru, komentari su opcioni.
- #FIXED "value" - Svaki element mora da ima ovaj atribut čija vrednost uvek mora da bude jednaka vrednosti navedenoj posle #FIXED u DTD.
- "value" - Određuje podrazumevane vrednosti za atribut. Koristi se u slučaju da u dokumentu nije eksplicitno navedena vrednost.

XML shema nudi znatno bogatiji jezik za definisanje strukture XML dokumenata. Jedna od njenih karakteristika je sintaksa zasnovana na samom XML-u. Ovaj dizajn obezbeđuje značajno poboljšanje čitljivosti, ali takođe omogućava značajno iskorišćenje tehnologije. Nije više potrebno pisati posebne parsere i uređivače (*engl. editor)* koji imaju posebnu sintaksu, kao što je potrebno za DTD. Još važnije poboljšanje je mogućnost ponovnog korišćenja i usavršavanja sheme. XML shema omogućava da se definišu novi tipovi proširivanjem ili ograničavanjem već postojećih što omogućava da se izgrade sheme od drugih shema, čime se smanjuje obim posla. XML shema obezbeđuje veći skup tipova podataka koji se mogu koristiti u XML dokumentima (DTD su ograničeni samo na niske). XML schema je objavljena kao W3C preporuka 2001. godine. XML shema je element sa oznakom za otvaranje:

<xs:schema xmlns:xs=http://www.w3.org/2000/10/XMLSchema version="1.0"> .

Najvažniji sadržaj XML sheme su definicije elemenata i tipova atributa, koji se definišu pomoću tipova podataka. Sintaksa tipa elementa je *<element ime=". . ."/>* i može da ima veliki broj opcionih atributa, kao što je *type = ""*... ili ograničenja kardinalnosti *minOccurs = "x",* gde x može biti bilo koji prirodan broj (uključujući i nulu), *maxOccurs = "x",* gde je x može biti bilo koji prirodan broj (uključujući i nulu) . *minOccurs* i *maxOccurs* su generalizacije operatera

kardinalnosti ?,* i + koji su definisani u DTD-u. Kada ograničenja kardinalnosti nisu eksplicitno data, *minOccurs* i *maxOccurs* imaju podrazumevanu vrednost 1.

```
<element name ="email"/>
<element name ="zaglavlje" minOccurs="1" maxOccurs="1"/>
<element name ="posiljalac" minOccurs="1"/>
```

Sintaksa tipa atributa je *<attribute ime=". . ."/>* i može da ima veliki broj opcionih atributa, kao što je *type* = ""... ili postojanje (odgovara #IMPLIED u DTD), use = "x", gde x može biti *optional* ili *required* ili *prohibited* ili podrazumevana vrednost (odgovara #FIXED i podrazumevanoj vrednosti u DTD).

```
<attribute name="id" type="ID" use="required"/>
<attribute name ="jezik" type="Language" use="optional" default="en"/>
```

Ključna slabost DTD je veoma ograničen izbor tipova podataka dok XML shema obezbeđuje velike mogućnosti za definisanje tipova podataka. Postoji niz već izgrađenih tipova podataka, a neki od njih su :

- Numerički tipovi podataka, uključujući i *integer, short, Byte, long, float,decimal*
- String tipovi podataka, uključujući *string, ID IDREF, CDATA*
- Vremenski tipovi podataka, uključujući *time, date, gMonth, gYear*

Takođe je omogućeno kreiranje korisnički definisanih tipova podataka, koje čine jednostavni tipovi podataka, koji ne mogu da koriste elemente i atribute, i složeni tipovi podataka, koji mogu da koriste elemente i atribute. Složeni tipovi podataka su definisani iz već postojećih tipova podataka kroz definisanje nekih atributa (ako ih ima) i uz korišćenje konstruktora:

- *sequence*, sekvenca postojećih tipova podataka, u kom je važan predefinisani poredak
- *all*, skup elementa koji moraju da se pojavljuju, ali poredak nije važan
- *choice*, skup elemenata, od kojih će jedan biti izabran.

```
<complexType name="predavacType">
  <sequence>
      <element name="ime" type="string" minOccurs="0" maxOccurs="unbounded"/>
      <element name ="prezime" type="string"/>
  </sequence>
  <attribute name ="titula" type="string" use="optional"/>
</complexType>
```

Postojeći tipovi podataka mogu se proširiti novim elementima i atributima.

```
<complexType name ="extendedPredavacType">
  <sequence>
        <element name ="ime" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name ="prezime" type="string"/>
        <element name ="email" type="string" minOccurs="0" maxOccurs="1"/>
  </sequence>
  <attribute name ="titula" type="string" use="optional"/>
  <attribute name ="zvanje" type="string" use="required"/>
</complexType>
```

Postojeći tipovi podataka mogu se ograničiti dodavanjem ograničenja na određenim vrednostima. Novi *type* i *use* atributi ili numerička ograničenja mogu biti dodati. Važno je razumeti da ograničenja nisu suprotan proces od proširenja. Ograničenje se ne postiže brisanjem elemenata i atributa.

XML *prostor imena* je kolekcija imena, zadata URI[15](*Uniform Resource Identifier*) referencom, koji se koriste kao elementi ili atributi XML dokumenta. XML specifikacija prostora imena omogućava da imena elemenata i atributa u XML dokumentu budu jedinstvena. XML prostor imena omogućava da isti XML dokument sadrži elemenate i atributa uzete iz različitih rečnika.

Jedna od glavnih prednosti korišćenja XML-a je mogućnost da se pristupi informacijama iz različitih izvora, jer XML dokumenti mogu da koriste više od jednog DTD ili sheme. Problem višeznačnih odrednica za validaciju XML dokumenata se prevazilazi pomoću različitih prefiksa za svaki DTD ili shemu. Prefiks je odvojen od lokalnog imena dvotačkom: *prefix:ime.* Prostori imena su definisani u okviru elementa i mogu se primeniti na taj element i svu njegovu decu (elemente i atribute). Deklaracija prostora imena ima oblik: *xmlns: prefix = "lokacija",* gde lokacija može biti adresa DTD-a ili sheme. Ako prefiks nije naveden, kao u xmlns = "lokacija", onda se koristi podrazumevana vrednost.

```
<?xml version="1.0" encoding="UTF-16"?>
  <vu:instructors xmlns:vu="http://www.vu.com/empDTD"
                  xmlns=http://www.gu.au/empDTD
                  xmlns:uky="http://www.uky.edu/empDTD">
  <uky:faculty uky:title="assistant professor" uky:name="John Smith"
                                  uky:department ="Computer Science"/>
  <akademskoOsoblje titula="predavač" ime="Mate Jones"
                  skola="Information Technology"/>
 </vu:instructors>
```

U elementu koji definiše XML shemu:

```
<xs:schema xmlns:xs=http://www.w3.org/2000/10/XMLSchema version="1.0">…</xs:schema>
```

[15] http://tools.ietf.org/html/rfc2396

prefiks *xs* označava prostor imena te sheme, a ako je izostavljen u *xmlns* atributu, onda se koriste elementi iz podrazumevanog prostora imena:

<schema xmlns=http://www.w3.org/2000/10/XMLSchema version="1.0"> .

U relacionim bazama podataka, delovi baze podataka mogu biti izabrani i preuzeti preko upitnih jezika kao što je SQL (*Structured Query Language*). Isto važi i za XML dokumente, za koje postoji veliki broj upitnih jezika, kao što su XQL[16], XML-QL[17], i XQuery[18].

2.2 Opis veb objekata (resursa) – RDF

XML pruža jedinstven okvir i skup alata za razmenu podataka i metapodataka između aplikacija. Međutim, XML ni na kakav način ne govori o značenju podataka. Pretpostavimo da želimo da izrazimo sledeću činjenicu:

Milutin Milanković je predavač racionalne mehanike.

Postoje razni načini za predstavljanje ove rečenicu u XML-u:

```
<kurs ime="Racionalna mehanika">
        <predavač> Milutin Milanković </ predavač>
</kurs>

<predavač ime=" Milutin Milanković ">
        <predaje> Racionalna mehanika </predaje>
</predavač>

<predavanje>
        <predavač> Milutin Milanković </predavač>
        <kurs> Racionalna mehanika </kurs>
</predavanje>
```

Prva dva načina su međusobno inverzno ugnježđena iako predstavljaju istu informaciju. Ne postoji standardni način da se oznaci elementa definiše smisao. Iako je često definisan kao jezik, RDF (*Resource Description Framework*) je u suštini model podataka. RDF ima XML sintaksu i kao rezultat toga, nasleđuje sve prednosti XML-a. Međutim, moguća je i druga sintaksička reprezentacija RDF-a koja nije zasnovana na XML-u, pa XML-sintaksa nije neophodna komponenta RDF modela.

[16] XML Query Language http://www.ibiblio.org/xql/xql-proposal.html
[17] A Query Language for XML http://www.w3.org/TR/NOTE-xml-ql/
[18] An XML Query Language http://www.w3.org/TR/xquery/

RDF je model podataka nezavistan od domena. Korisnici definišu svoje pojmove pomoću jezika sheme koji se zove RDF shema (RDFS). XML shema ograničava strukturu XML dokumenata, dok RDF model podataka koristi rečnik koji definiše RDF shema. Taj rečnik sadrži klasifikaciju objekata, opis hijerarhije između objekata, opis svojstva koja imaju određene vrste objekata. Rečenica

Predavač je potklasa akademskog osoblja.

znači da su svi predavači istovremeno i akademsko osoblje. Važno je razumeti značenje "je potklasa" koje mora da se poštuje od strane svih programa za obradu RDF-a. RDFS čini semantičke informacije dostupne mašini, što je u skladu sa vizijom semantičkog veba.

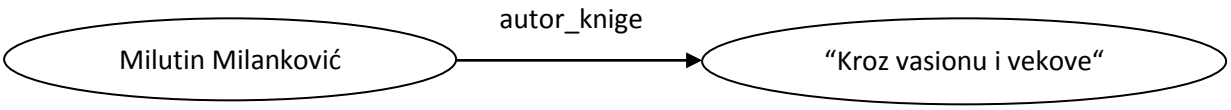
Resurs (*engl. resource*), predstavlja objekat, "stvar" koju želimo da predstavimo. Resursi mogu biti autori, knjige, mesta, ljudi, hoteli, sobe, pretrage, i tako dalje. Svaki resurs ima URI. URI može biti URL (*Uniform Resource Locator*, ili veb adresa) ili neka druga vrsta jedinstvenog identifikatora. Identifikator ne mora omogućiti pristup resursima. URI sheme su definisane, ne samo za veb lokacije, već i za različite uloge objekata kao što su brojevi telefona, ISBN brojevi, kao i geografske lokacije. Generalno, pretpostavka od koje se polazi je da je URI identifikator veb resursa [7].

Svojstva (*engl. property*) su posebna vrsta resursa; opisuju odnose između resursa, na primer "napisao", "naslov", i tako dalje. Svojstva su u RDF-u takođe identifikovana sa URI (i u praksi URL). Ideja korišćenja URI-a za identifikaciju "stvari" i odnosa između njih je veoma važna jer daje izbor za lako jedinstveno imenovanje na globalnom nivou. Korišćenje takve sheme u velikoj meri smanjuje problem homonima.

Iskazi (*engl. statement*) povezuju svojstva i resurse. Iskaz je osnovni gradivni element RDF-a i predstavlja trojku subjekat-predikat-objekat. Subjekat može biti bilo koji resurs. Predikat može biti resurs ili svojstvo. Objekat može biti resurs ili literal. Literali su atomične vrednosti (niske).

Iskaz *Milutin Milanković je autor knige "Kroz vasionu i vekove"* može se interpretirati kao trojka (*Milutin Milanković*, autor knige, "Kroz vasionu i vekove"). Trojka (x,P,y) se može interpretirati kao logička formula P (x, y), gde se binarni predikat P odnosi na relaciju subjekta x i objekta y. RDF nudi samo binarne predikate (svojstva).

Drugi način prikaza je graf:



To je usmeren graf sa oznakom čvora grafa i lukovima koji su usmereni od resursa (subjekta iskaza) na vrednost (predmet iskaza). Ova vrsta grafa je poznata u veštačkoj inteligenciji kao semantička mreža resursa. Vrednost iskaza može da bude i resurs, tako da resurs može biti povezan sa drugim resursima.

Vizija semantičkog veba zahteva reprezentaciju informacija u obliku XML-a koji mašina može da obradi. U skladu sa tim postoji i treća mogućnost interpretacije iskaza. RDF dokument se predstavlja kao XML element sa oznakom *rdf:RDF*. Sadržaj ovog elementa je spisak opisa koji koriste *RDF:Description* oznake. Svaki opis daje iskaz o resursima.

RDF reprezentacija prethodnog iskaza je sledeća:

```
<XML  Version = "1.0" encoding = "UTF-16">?
<rdf: RDF xmlns: rdf = " http://www.w3.org/1999/02/22-rdf-sintak-ns#"
            xmlns: mydomain = "http://www.mydomain.org/my-rdf-ns">
        <rdf:Description rdf:about="Kroz vasionu i vekove">
                <mydomain:autor_knige rdf:resource="Milutin Milanković"/>
        </ rdf: Description>
</ rdf: RDF>
```

Prva linija određuje da se koristi XML. *RDF:Description* elemenat predstavlja iskaz o resursu “Kroz vasionu i vekove“, i u okviru njega svojstvo se koristi kao oznaka, a sadržaj je vrednost svojstva. Opisi se daju u određenim redosledom, međutim redosled opisa (ili resursa) nije značajan u apstraktnom RDF modelu što pokazuje da graf predstavlja pravi RDF model podatka i da je XML samo mogući prikaz grafa.

Prilikom čitanja podatka parser ne može da zna da li podatak treba da se tumači kao broj ili kao string, osim ako aplikacija eksplicitno daje informacije koje opisuju resurs. Najčešća praksa u programskim jezicima ili bazama podataka kojom se pruža ova vrsta informacije, jeste dodela tipa podataka literalu. U RDF se koriste tipizirani literali da bi se obezbedila ova vrsta informacija. Korišćenjem tipiziranih literala može se opisati iskaz

Ana ima 5 godina

kao

(Ana,http://www.mydomain.org/godina,"5"^^http://www.w3.org/2001/XMLSchema#integer).

Ovaj primer pokazuje dve stvari: korišćenje ^^ notacija da se ukaže na tip literala, i korišćenje tipova podataka koji su unapred definisani u XML shemi. Korišćenje korisnički-definisanih tipova literala je dozvoljeno u RDF dokumentima, ali se u praksi najčešće koriste tipovi podataka koji su definisani u XML shemi. XML shema sadrži širok spektar tipova podataka, uključujući i logičke, cele brojeve, brojeve sa pokretnim zarezom, datume i vreme.

RDF koristi samo binarna svojstva. Ovo ograničenje izgleda vrlo ozbiljno, jer se često koriste predikati sa više od dva argumenta. Takvi predikati mogu biti predstavljeni sa više binarnih predikata. Ako se iskaz *X je sudija u partiji šaha između Y i Z* intuitivno posmatra kao predikat *sudija(X, Y, Z)*, uvođenjem pomoćnog resursa *partija šaha* i binarnih predikata *sudija, igrač1 i igrač2*, predikat *sudija(X, Y, Z)* se može predstavitii kao :

sudija(partija šaha, X),igrač1 (partija šaha, Y),igrač2(partija šaha, Z).

RDF dokument se sastoji od *rdf: RDF* elemenata čiji sadržaj čine brojni opisi.

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
                xmlns:uni="http://www.mydomain.org/uni-ns#">
        <rdf:Description rdf:about="949352">
                <uni:ime>Grigoris Antoniou</uni:ime>
                <uni:titula>Profesor</uni:titula>
        </rdf:Description>
        <rdf:Description rdf:about="949318">
                <uni:ime>David Billington</uni:ime>
                <uni:titula> Profesor</uni:titula>
                <uni:starost rdf:datatype="&xsd;integer">27</uni:starost>
        </rdf:Description>
        <rdf:Description rdf:about="949111">
                <uni:ime>Michael Maher</uni:ime>
                <uni:titula>Profesor</uni:titula>
        </rdf:Description>
        <rdf:Description rdf:about="CIT1111">
                <uni:kurs>Diskretna matematika </uni: kurs>
                <uni:Predavac>David Billington</uni: Predavac>
        </rdf:Description>
        <rdf:Description rdf:about="CIT1112">
                <uni:kurs>Analitička geometrija</uni:kurs>
                <uni:Predavac>Grigoris Antoniou</uni:Predavac>
        </rdf:Description>
        <rdf:Description rdf:about="CIT2112">
                <uni:kurs>ProgramiranjeI</uni:kurs>
                <uni:Predavac>Michael Maher</uni:Predavac>
        </rdf:Description>
        <rdf:Description rdf:about="CIT3112">
                <uni:kurs>Teorija algoritama</uni:kurs>
                <uni:Predavac>David Billington</uni:Predavac>
        </rdf:Description>
        <rdf:Description rdf:about="CIT3116">
                <uni:kurs>Veštačka inteligencija</uni:kurs>
```

```
        <uni:Predavac>Grigoris Antoniou</uni:Predavac>
    </rdf:Description>
</rdf:RDF>
```

U navedenom primeru se koristi proširen mehanizam prostora imena XML-a. Spoljašnji prostori imena se koriste da uvezu spoljašnji RDF dokument u tekući RDF dokument. Ovaj mehanizam dozvoljava ponovnu upotrebu resursa od strane drugih korisnika koji mogu da odluče da ubace dodatne mogućnosti u ove resurse. Rezultat je pojava velikih, distribuiranih baza znanja. *Rdf:about* atribut elementa *rdf:Description* ima ekvivalentno značenje kao *ID atribut*, ali se često koristi da sugeriše da je objekat o kojem je dat iskaz već definisan na drugom mestu. Sadržaj *rdf: Description* elemenata se naziva *svojstvo elementa*. Elementi *uni:kurs* i *uni:Predavac* iz prethodnog primera definišu parove svojstvo-vrednost za resurs CIT3116. Atribut *rdf:datatype="&xsd;integer"* se koristi za označavanje tipa podataka za vrednost svojstva starosti. Iako svojstvo starosti ima definisan tip "&xsd;integer", ipak je potrebno da se specificira tip podatka za vrednost ovog svojstva svaki put kada se koristi da bi se obezbedilo da RDF procesor dodeli vrednosti svojstva tačan tip čak i ako nije ranije video odgovarajuću definiciju RDF sheme.

U prethodnom primeru odnosi između kursa i predavača nisu definisani formalno već se uvode korišćenjem istog imena predavača. Formalna definicija tog odnosa može biti realizovana definisanjem resursa u RDF dokumentu a zatim definisanjem vrednosti atributa *rdf:resource* elementa *uni:Predavac*:

```
<rdf:Description rdf:about="CIT1111">
        <uni:kurs> Diskretna matematika </uni: kurs>
        <uni:Predavac rdf:resource="949318"/>
</ RDF: Description>
<rdf:Description rdf:about="949318">
        <uni:ime> Dejvid Billington </uni: ime>
        <uni:titula>Profesor</uni: titula>
</ RDF: Description>
```

U navedenom primeru opisi spadaju u dve kategorije: kursevi i predavači. Ova činjenica je jasna za čitaoce, ali nije formalno definisana, tako da nije dostupna mašini. U RDF-u je moguće to definisati korišćenjem *rdf: type* elementa. *RDF: type* element omogućava da se uvedu neke strukture u RDF shema dokumentu. Atribut *rdf:resource="&uni;kurs"* elementa *rdf: type* označava da je resurs tipa *kurs,* a tip *kurs* je definisan kao resurs u uvezenom RDF dokumentu.

```
<rdf:Description rdf:about="CIT1111">
        <rdf:type rdf:resource="&uni;kurs"/>
        <uni:kurs> Diskretna matematika </uni:kurs>
        <uni:Predavac rdf:resource="949318"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="949318">
        <rdf:type rdf:resource="&uni;profesor"/>
        <uni:ime>David Billington</uni:ime>
        <uni:titula> Profesor</uni:titula>
</rdf:Description>
```

Kontejner elementi se koriste za prikupljanje resursa ili atributa o kojima je potrebno dati iskaze u celini.
U RDF-u postoje tri vrste kontejnera :

- *RDF:Bag*, neuređeni kontejner, koji može imati više pojavljivanja istog elementa (ne važi za skup). Tipični primeri su članovi odbora fakulteta i fascikle sa dokumenatima.
- *RDF:Seq*, uređeni kontejner, koji može imati više pojavljivanja istog elementa. Tipični primeri su moduli kursa, tačke dnevnog reda, spisak radnika
- *RDF:Alt*, skup alternativa. Tipičan primer su prevodi dokumenata u različitim jezicima.

Mehanizam materijalizacije (*engl. reification*) obezbeđuje mogućnost davanja iskaza o drugom iskazu korišćenjem RDF identifikatora. Mehanizam opredmećivanja pretvara iskaz u resurs. Na primer,

```
<rdf:Description rdf:about="949352">
        <uni:ime> Grigoris Antoniou </uni:ime>
</rdf:Description>
```

može se napisati kao:

```
<rdf:Statement rdf:about="Informacija o 949352">
        <rdf:subject rdf:resource="949352"/>
        <rdf:predicate rdf:resource="&uni;ime"/>
        <rdf:object> Grigoris Antoniou </rdf:object>
</rdf:Statement>
```

RDF: subject, RDF: predicate, i RDF: object omogućavaju pristup delovima iskaza. Ako se više od jednog elementa resursa nalazi u opisu, element je sadržan u više od jednog iskaza.

RDF je univerzalni jezik koji omogućava korisnicima da opisuju resurse koristeći svoj rečnik. RDF ne daje pretpostavke o nekom domenu, niti definiše semantiku bilo kog domena, već je prepušteno korisniku da to učini u RDF shemi (RDFS) koristeći klase i svojstva [1]. Klasa se može posmatrati kao skup elemenata. Pojedinačni objekti koji pripadaju klasi predstavljaju instance te klase. Odnos između instance i klase u RDF u definisan je korišćenjem *rdf:type*. Važno korišćenje klasa je da se uvedu ograničenja na ono što može biti navedeno u dokumentu korišćenjem RDF-a.

Kada su klase definisane potrebno je uspostaviti vezu između njih. Na primer, definisane su klase: nastavno osoblje, asistenti, akademsko osoblje,administrativno osoblje, profesori, tehnička podrška, vanredni profesori. Ove klase nisu nepovezane međusobno, svaki profesor je deo akademskog osoblja, "profesor" je potklasa klase "akademsko osoblje", ili "akademsko osoblje" je natklasa klase "profesor". Odnos potklasa definiše hijerarhiju klasa. Ako je klasa potklasa i B_1 i B_2, to znači da je svaka njena instanca i instanca B_1 i B_2. Hijerarhijska organizacija klasa ima veoma važan praktičan značaj. Na primer, ako se uzme u obzir ograničenje da učenicima na kursu moraju biti predavači samo pripadnici akademskog osoblja i ako je David Billington definisan kao profesor onda njemu nije dozvoljeno da bude predavač na kursevima. Razlog za to je da nema iskaza koji definiše da je David Billington deo akademskog osoblja. Ovaj problem se može prevazići nasleđivanjem klase profesora od klase akademskog osoblja. Upravo to se radi u RDF shemi. RDF shema određuje značenje "je potklasa" što znači da to značenje mora da se koristi od strane svih RDF alata za obradu. RDF shema je primitivni jezik ontologije.

Hijerarhijski odnos osim između klasa može biti definisan i između svojstava. Na primer, "je predavač" je podsvojsvo svojstva "učestvuje u kursu". Ako kurs k predaje neko ko je deo akademskog osoblja, a, onda k uključuje a. Suprotno nije obavezno tačno. U principu, P je podstvojsvo od Q ako je Q (x, y) uvek kada je P (x, y).

RDF shema omogućava modelovanje osnove za izražavanje informacija. RDF omogućava da se bilo koji iskaz predstavi kao resurs, i da bilo koji URI može biti resurs.
Izvorne klase su:

- *rdfs:Resource*, klasa svih resursa
- *rdfs:Class*, klasa svih klasa
- *rdfs:Literal*, klasa svih literala(stringova)
- *rdf:Property*, klasa svih svojstava
- *rdf:Statement*, klasa svih prihvaćenih iskaza

Osnovna svojstva za definisanje odnosa su:

- *rdf:type* povezuje resurs sa pripadajućom klasom. Resurs je definisan kao instanca te klase.
- *rdfs:subClassOf*, povezuje klasu sa nekom od njehih natklasa. Sve instance klase su instance svojih natklasa.
- *rdfs:subPropertyOf*, povezuje svojstvo sa nekim od njegovih nadsvojstava.

Osnovna svojstva za definisanje ograničenja su:

- *rdfs:domain*, precizira domen svojstva P i navodi da je svaki resurs koji predstavlja vrednost svojstva instanca klase domena.
- *rdfs:range*, određuje opseg svojstva P i navodi da su vrednosti svojstva instance klase opsega.

Još neka korisna svojsva su:

- *rdf: subject*, povezuje iskaz sa njenim subjektom
- *rdf:predicate*, povezuje iskaz sa njenim predikatom
- *rdf:object*, povezuje iskaz sa njenim objektom
- *rdf:Bag***,** klasa neuređenih kontejnera
- *rdf:Seq***,** klasa sekvenci - uređenih kontejnera
- *rdf:Alt***,** klasa alternativa
- *rdfs:Container***,** natklasa svih kontejener klasa

Resurs može biti definisan i opisan u mnogim mestima na veb-u. Sledeća svojstva omogućavaju da se definišu veze ka odgovarajućim adresama:

- *rdfs: seeAlso* referencita resurs na drugi resurs koji ga objašnjava
- *rdfs: isDefinedBy* je podsvojstvo od *rdfs:seeAlso* i povezuje resurs sa mestom gde se nalazi njegova definicija, obično RDF shema.

Obezbeđivanje više informacija namenjenim korisnicima može biti realizovano korišćenjem karakteristika:

- *rdfs:comment*, obično duži tekst, može biti povezan sa resursima
- *rdfs:label*, kratko ime, povezano sa resursom. Između ostalog, ona može poslužiti kao ime elementa grafičkom prikazu RDF dokumenta.

2.3 SPARQL – upitni jezik

Odgovarajući upitni jezik mora da razume RDF, to jest, mora da razume ne samo sintaksu već i model podataka RDF-a i značenje RDF rečnika.

SPARQL[19] upitni jezik je W3C preporuka za upitivanje RDF-a, i kao takav je brzo postao standardni upitni jezik za ovu namenu.

SPARQL upitni jezik se zasniva na podudaranju grafovskih obrazaca. Najjednostavniji grafovski obrazac je trostruki obrazac, koji je sličan RDF iskazu, ali sa mogućnošću promenljive umesto RDF subjekta, predikata ili objekata [10]. Kombinovanje trostrukih obrazaca daje osnovni grafovski obrazac, gde je za potrebno potpuno podudaranje da grafovski ispuni obrazac.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c
WHERE
{
?c rdf:type rdfs:Class .
}
```

[19] SPARQL Protocol and RDF Query Language http://www.w3.org/TR/rdf-sparql-query/

Ovaj upit preuzima sve trostruke obrasce kojima je svojstvo *rdf:type* i objekat je *rdfs:class*. Drugim rečima, ovaj upit, kada se izvrši, preuzeće sve klase. Da bismo dobili sve instance određene klase, potreban je sledeći upit:

```
PREFIX uni: <http://www.mydomain.org/uni-ns#>
SELECT ?i
WHERE
{
?i rdf:type uni:kurs .
}
```

Kao i u SQL i u SPARQL upitima postoji *SELECT-FROM-WHERE* struktura:

- SELECT određuje broj i redosled preuzetih podataka.
- FROM se koristi da se navede izvor koji se upituje. Ova klauzula je opciona, ako nije navedena, može jednostavno da se pretpostavi da se upituje baza znanja određenog sistema.
- WHERE nameće ograničenja mogućim rešenjima

Upit kojim se vraćaju zaposleni koji su profesori i njihovi brojevi telefona izleda ovako:

```
SELECT ?x ?y
WHERE
{
?x rdf:type uni:Profesor ;
uni:telefon ?y .
}
```

Klauzula *?x rdf:type uni:Profesor ;* prikuplja sve instance klase Profesor, i vezuje rezultat za promenljivu ?x. Drugi deo prikuplja sve trojke čiji je predikat telefon. Ali postoji implicitno pridruživanje koje ograničava drugi šablon samo na one trojke čiji je predmet trojke sadržan u promenljivoj ?x. U ovom slučaju korišćena je sintaksa prečica, tačka-zarez ukazuje na to da sledeći obrazac deli predmet prethodnog. Ekvivalentni upit je:

```
SELECT ?x ?y
WHERE
{
?x rdf:type uni:Predavac .
?x uni:telefon ?y .
}
```

U SPARQL se koristi FILTER uslov da bi se označila i logička ograničenja. Na primer, ako je potrebno prikazati imena kurseva čiji je predavač profesor sa ID=949352:

```
SELECT ?n
WHERE
{
?x rdf:type uni:Kurs ;
uni:Predavač :949352 .
?c uni:ime ?n .
FILTER (?c = ?x).
}
```

Za potrebe prikazivanja svih traženih resursa koji nemaju vrednosti za sve atribute koristi se uslov OPTIONAL. Na primer, ako je poterbno prikazati sve profesore i njihove e-mail adrese, a e-mail adresa nije navedena za svakog profesora, upit izgleda ovako:

```
SELECT ?ime  ?email
WHERE
{ ?x rdf:type uni:Profesor ;
uni:ime ?ime .
OPTIONAL { ?x uni:email ?email }
}
```

SPARQL omogućava korisnicima da pišu globalno nedvosmislene upite čime ispunjava viziju semantičkog veba da tretira veb kao jednu veliku bazu podataka.

2.4 Jezici ontologija – OWL

OWL (*Web Ontology Language*) je jezik za opisivanje ontologija na vebu. Osnovna namena OWL-a je da omogući korisnicima prikazivanje značenja termina i odnose među tim terminima u rečenicima, a to znači da je informacije koje se nalaze u dokumentima moguće obraditi aplikacijama i da je njihov sadržaj razumljiv čoveku. OWL ima više mogućnosti za iskazivanje značenja od XML-a i RDF-a. OWL je izveden iz DAML[20](*DARPA Agent Markup Language*) + OIL *(Ontology Inference Layer*) ontološkog jezika razvijenog kao proširenje XML-a i RDF-a [8]. S obzirom da je OWL zasnovan na RDF-u, to znači da OWL ontologija predstavlja RDF graf, koji se dalje može predstaviti u obliku RDF iskaza. Zbog toga je moguće za OWL koristiti slične sintaksne forme kao i za RDF. Informacije opisane OWL-om postaju znanje, a ne više skup podataka. Kada se OWL-u pridoda i mogućnost izvođenja činjenica koje nisu eksplicitno navedene, informacije stvarno postaju znanje, a OWL postaje jezik za predstavljanje znanja. Iako OWL proizilazi iz ideje semantickog veba, počeo je da se koristi u područjima koja nisu povezana sa vebom.

[20] http://www.daml.org/

Sa ontološkog stanovišta moguće je donositi zaključke o sledećem:

- *Pripadnost klasi*. Ako je x instanca klase C, a C je potklasa D, onda može se zaključiti da je x instanca D.
- *Ekvivalencija klasa*. Ako je klasa A ekvivalent klase B i klase B je ekvivalent klase C, onda je A ekvivalent klase C
- *Doslednost*. Ako je x instanca klase A i A je potklasa klase B, pri tom je A je potklasa klase C i B i C su disjunktne klase, onda postoji nedoslednost, jer bi A trebalo da bude prazna, ali ona ima instancu x. Ovo je indikacija greške u ontologiji.
- *Klasifikacija*. Ako ako jedinka x ispunjava uslove i ako su određeni parovi svojstvo-vrednost dovoljan uslov za članstvo u klasi A, može se zaključiti da je x instanca klase A.

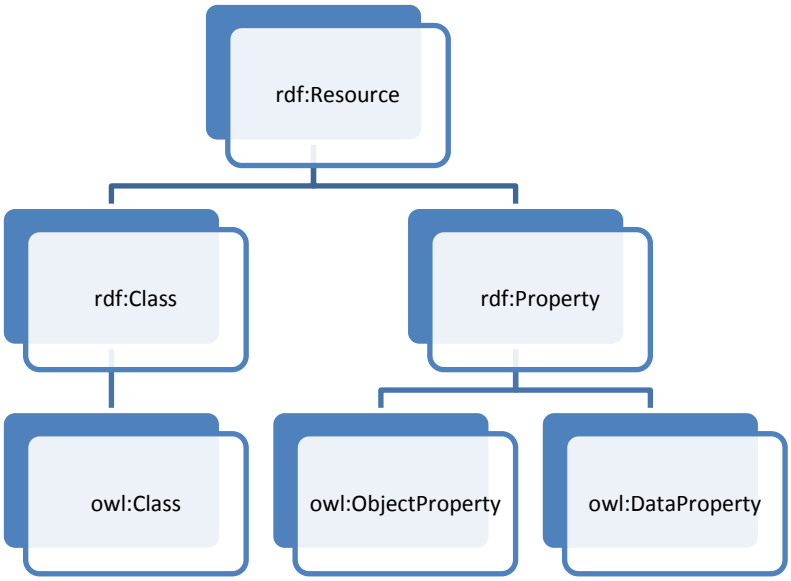
Formalna semantika je preduslov za podršku rezonovanju. Prethodna izvođenja mogu biti automatska umesto da se prave ručno. Podrška rezonovanju je važna jer omogućava:

- proveru doslednosti ontologije i znanja,
- proveru odnosa između klasa
- automatsku klasifikaciju instanci klasa

Automatska podrška rezonovanju omogućava proveru mnogo više slučajeva nego što se može ručno proveriti. Formalna semantika i podrška rezonovanju obezbeđuje prevođenje jezika ontologije na logičke formule i korišćenje već postojećih automatizovanih rasuđivača rasuđivača koji barataju sa takvim formulama. OWL je preveden na deskriptivne logike, i koristi postojeće rasuđivače [12]. Deskriptivna logika je podskup predikatske logike prvog reda za koju je moguća podrška efikasnom rezonovanju o ontologijama.

OWL koristi RDF i RDF sheme :

- Sve vrste OWL-a koriste RDF sintaksu.
- Instance klasa se deklarišu u RDF-u, korišćenjem RDF opisa i informacija o tipu podatka.
- OWL konstruktori kao što su *owl:Class*, *owl:DatatypeProperty*, i *owl:ObjectProperty* su specijalizacije njihovih odgovarajućih RDF parnjaka.



OWL je izgrađen na RDF-u i RDF shemi i koristi sintaksu RDF-a koja nije lako čitljiva i obzirom na to u OWL-u su definisane sledeće sintaksne forme:

- XML-bazirana sintaksa koja ne prati RDF konvencije i tako je lakše čitljiva
- Apstraktna sintaksa, koja se koristi u dokumentu specifikacije jezika, koja je kompaktnija i čitljivija od XML sintakse i RDF/XML sintakse
- Grafička sintaksa bazirana na UML-u (*Unified Modeling Language*), koji je u širokoj upotrebi i na taj način jednostavan i razumljiv za ljude.

OWL dokumenti u RDF sintaksi se obično nazivaju OWL ontologije i to su RDF dokumenti ontologije. Koren element OWL-a je *rdf: RDF* element, koji definiše prostore imena:

```
<rdf:RDF 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 ontologija može da počne kolekcijom iskaza za interne svrhe. Ovi iskazi su grupisani *owl:Ontology* elementom, koji sadrži komentare, podatke o verzijama, i uključivanje drugih ontologija.

```
<owl:Ontology rdf:about="">
        <rdfs:comment>Primer OWL ontologije</rdfs:comment>
        <owl:priorVersion rdf:resource="http://www.mydomain.org/uni-ns-old"/>
        <owl:imports rdf:resource="http://www.mydomain.org/persons"/>
        <rdfs:label>UNI  Ontology</rdfs:label>
</owl:Ontology>
```

Samo jedan od ovih iskaza ima uticaja na logično značenje ontologije: *owl:imports*, koji navodi ime druge ontologije čiji sadržaj postaje deo nove ontologije. Obično se koristi uvoz elemenata za svaki prostor imena, ali je moguć uvoz dodatnih ontologija, na primer, ontologije koje obezbeđuju definicije, bez uvođenja novih naziva. *Owl:imports* je tranzitivno svojstvo: ako ontologija A uvozi ontologiju B i ontologija B uvozi ontologiju C, onda ontologija A takođe uvozi ontologiju C.

owl:Class elementom se definiše klasa. Postoje dve unapred definisane klase, *owl:Thing* i *owl:Nothing*. *Owl:Thing* je najopštija klasa, koja sadrži sve (sve je resurs), a *owl:Nothing* je prazna klasa. Tako je svaka klasa potklasa *owl:Thing* i natklasa *owl:Nothing*.

rdfs:subClassOf označava da je klasa C potklasa klase C', tako da je svaka instanca klase C i instanca klase C'. Iskaz C je potklasa klase C', u OWL-u znači da C' prikuplja sve objekte klase C koji zadovoljavaju određene uslove. *rdfs:subClassOf* je tranzitivno svojstvo. Navođenjem jednog ili vise iskaza koji kao predikat imaju ovo svojstvo definiše se hijerarhija klasa.

```
<owl:Class rdf:ID="RedovniProfesor">
        <rdfs:subClassOf rdf:resource="#AkademskoOsoblje"/>
</owl:Class>
```

Korišćenjem *owl:disjointWith* elementa može se navesti da su dve klase disjunktne. Ovaj element može biti uključen u prethodnoj definiciji, ili dodat korišćenjem *rdf:about* elementa. Ovaj mehanizam je nasleđen od RDF-a.

```
<owl:Class rdf:about="#RedovniProfesor ">
        <owl:disjointWith rdf:resource="#Asistent"/>
</owl:Class>
```

Korišćenjem elementa *owl:equivalentClass* može biti dodata ekvivalentna klasa.

```
<owl:Class rdf:ID="Fakultet">
        <owl:equivalentClass rdf:resource="#AkademskoOsoblje"/>
</owl:Class>
```

U OWL postoje dve vrste svojstava:

- Objektna svojstva - *ObjectProperty*, koja se odnose na objekte drugih objekata

- Svojstva koja se odnose na tip podatka vrednosti - *DatatypeProperty*. OWL ne sadrži predefinisane tipove podataka, niti obezbeđuje definiciju posebnih objekata. Umesto toga, OWL omogućava korišćenje XML sheme tipova podataka, koristeći slojevitu arhitekturu semantičkog veba.

Primeri:

```
<owl:DatatypeProperty rdf:ID="starost">
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="jePredavač">
        <rdfs:domain rdf:resource="#kurs"/>
        <rdfs:range rdf:resource="#akademskoOsoblje"/>
        <rdfs:subPropertyOf rdf:resource="#jeUljučen"/>
</owl:ObjectProperty>
```

OWL omogućava povezivanje inverznih svojstava, kao što su *jePredavač* i *uči:*

```
<owl:ObjectProperty rdf:ID="uči">
        <rdfs:range rdf:resource="#kurs"/>
        <rdfs:domain rdf:resource="#akademskoOsoblje"/>
        <owl:inverseOf rdf:resource="#jePredavač"/>
</owl:ObjectProperty>
```

Ekvivalentna svojstva mogu biti definisana korišćenjem elementa *owl:equivalentProperty*.

```
<owl:ObjectProperty rdf:ID="podučava">
        <owl:equivalentProperty rdf:resource="#jePredavač"/>
</owl:ObjectProperty>
```

U OWL-u postoji mogućnost definisanja ograničenja mogućih vrednosti svojstava. Ograničenja se uvode korišćenjem *owl:Restriction* elementa. *owl:Restriction* definiše *anonimnu klasu*, klasu koja nije definisana *owl:Class* elementom i može biti samo upotrijebljena na jednom mjestu gdje se ograničenje pojavljuje. Svaki *owl:Restriction* element sadrži *owl:OnProperty* element i jednu ili više restriktivnih deklaracija.

Ako je potrebno uvesti ograničenje da kurseve prve godine studija predaju samo profesori uvode se ograničenja korišćenjem elementa *owl:allValuesFrom* kojim se navode klase mogućih vrednosti svojstva navedenog u *owl:onProperty* elementu.

```
<owl:Class rdf:about="#kursPrveGodine">
        <rdfs:subClassOf>
                <owl:Restriction>
                        owl:onProperty rdf:resource="#jePredavač"/>
                        <owl:allValuesFrom rdf:resource="#Profesor"/>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Ukoliko je potrebno uvesti ograničenje da neko svojstvo ima specifičnu vrednost koristi se element *owl:hasValue.*

```
<owl:Class rdf:about="#matematika">
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#jePredavač"/>
                        <owl:hasValue rdf:resource="#949318"/>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Ukoliko je potrebno uvesti ograničenje da neko svojstvo mora sadržati neku od vrednosti, npr. da svaki član akademskog osoblja mora biti predavač bar jednog obaveznog kursa, koristi se element *owl:someValuesFrom*.

```
<owl:Class rdf:about="#akademskoOsoblje">
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#jePredavač"/>
                        <owl:someValuesFrom rdf:resource="#obavezanKurs"/>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Osim *owl:hasValue*, *owl:allValuesFrom* i *owl:someValuesFrom,* restriktivne deklaracije mogu biti i elementi koji ograničavaju kardinalnost:

- *owl:cardinality* – specificira tačan broj elemenata koji predstavljaju vrednost svojstva.
- *owl:minCardinality* – specificira minimalni broj elemenata koji predstavljaju vrednost svojstva.
- *owl:maxCardinality* – specificira maksimalni broj elemenata koji predstavljaju vrednost svojstva.

Osobine svojstava se mogu definisati direktno korišćenjem elemenata:

- *owl:TransitiveProperty* - definiše tranzitivna svojstva kao što su *“je stariji”*,*”je viši” .*
- *owl:SymmetricProperty* - definiše simetrična svojstva kao što su *“isto godište”,”isti razred”.*
- *owl:FunctionalProperty* - definiše funkcionalna svojstva, svojstva koja imaju najmanje jednu vrednost za svaki objekat, kao što su *“starost”,”visina”.*
- *owl:InverseFunctionalProperty* - definiše svojstva koja za dva različita objekta ne mogu imati istu vrednost, kao što je *“jmbg”.*

OWL jezik podržava i uvođenje ograničenja korišćenjem logičkih kombinacija – unije, komplementa i preseka, upotrebom elemenata *owl:union, owl:complement, owl:intersection.*

```
<owl:Class rdf:about="#kurs">
        <rdfs:subClassOf>
                <owl:Class>
                        <owl:complementOf rdf:resource="#osoblje"/>
                </owl:Class>
        </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ljudiNaUniverzitetu">
        <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#osoblje"/>
                <owl:Class rdf:about="#studenti"/>
        </owl:unionOf>
</owl:Class>
<owl:Class rdf:ID="katedraPrimenjeneMatamatike">
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#katedra"/>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#pripadajućiKursevi"/>
                        <owl:hasValue rdf:resource="#PrimenjenaMatemtika"/>
                </owl:Restriction>
        </owl:intersectionOf>
</owl:Class>
```

Enumeracije u OWL su *owl:oneOf* elementi koji se koriste za definisanje klase navođenjem svih njenih instanci.

```
<owl:Class rdf:ID="daniUSedmici">
        <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#Ponedeljak"/>
                <owl:Thing rdf:about="#Utorak"/>
```

```
            <owl:Thing rdf:about="#Sreda"/>
            <owl:Thing rdf:about="#Četvrtak"/>
            <owl:Thing rdf:about="#Petak"/>
            <owl:Thing rdf:about="#Subota"/>
            <owl:Thing rdf:about="#Nedelja"/>
        </owl:oneOf>
</owl:Class>
```

Instanca klase se definiše isto kao u RDF-u

```
<rdf:Description rdf:ID="949352">
        <rdf:type rdf:resource="#akademskoOsoblje"/>
</rdf:Description>
```

Ili

```
<akademskoOsoblje rdf:ID="949352"/>
```

Za razliku od tipičnih sistema baza podataka, OWL ne usvaja pretpostavku da su dve instance različite ako imaju različito ime ili ID, već se to mora eksplicitno navesti korišćenjem elementa *owl:differentFrom*

```
<autor rdf:ID="949318"><owl:differentFrom rdf:resource="#949352"/></autor>
```

Iako XML shema pruža mehanizam za kreiranje korisnički definisanih tipova podataka izvedeni tipovi podataka ne mogu da se koriste u OWL-u . OWL lista referenci sadrži sve XML shema tipove podataka koji se mogu koristiti, ali to uključuje najčešće korišćene tipove kao što su string, ceo broj, logički tipovi, vreme i datum.

Imajući u vidu da je teško odgovoriti svim zahtevima koje treba da ima jezik ontologije, W3C je dao specifikaciju koja definiše tri vrste OWL-a: OWL Full, OWL DL i OWL Lite.

Ceo jezik se zove *OWL Full* i koristi sve osnovne primitive OWL jezika. On omogućava i kombinaciju ovih primitiva na proizvoljni način sa RDF-om i RDF šemom što uključuje mogućnost promene značenja predefinisanih primitiva (RDF ili OWL primitive). OWL je u potpunosti kompatibilan unazad sa RDF-om, i sintaksno i semantički: svaki pravilan RDF dokument je takođe pravilan OWL Full dokument, slično validan RDF/RDF shema zaključak je validan OWL Full zaključak. Mana OWL Full jezika je kompleksnost, što otežava obezbeđivanje efikasne podrške rezonovanju. U OWL Full, sve jezičke kontrukcije se mogu se koristiti u bilo kojoj kombinaciji, sve dok je rezultat validan RDF dokument.

OWL DL (DL-*Description Logic*) je podjezik OWL Full jezika u kome je ograničen način korišćenja konstrukcija iz OWL i RDF. OWL DL ima dobru formalnu osnovu pošto je zasnovan na deskriptivnoj logici, pa je stoga omogućena efikasna podrška rezonovanju. Nedostatak je da se gubi puna kompatibilnost sa RDF-om. RDF dokument mora da se proširi ili ograniči na neki način pre nego što postane validan OWL DL dokument. Svaki validan OWL DL dokument je validan RDF dokument. Da bi bila iskorišćena formalna osnova desktriptivne logike, OWL DL ontologija mora poštovati neka ograničenja:

- *Podela rečnika*. Resurs može biti samo klasa, tip podataka, svojstvo tipa podatka, objektno svojstvo, instanca, vrednost svojstva, ili deo ugrađenog rečnika, ali ne više od jednog od njih. To znači da klasa ne može istovremeno da bude instanca, ili da svojstvo ne može imati neke vrednosti iz tipa podataka i neke vrednosti iz klase (isto svojsvo ne može biti i svojstvo tipa podatka i objektno svojstvo).
- *Eksplicitno tipiziranje*. Svi resursi moraju biti podeljeni (kao što je propisano u prethodnom ograničenju), i ta podela mora biti eksplicitno navedena.
- *Razdvajanje objekata*. Na osnovu prvog ograničenja, skupovi objektnih i svojstava tipova podataka su disjunktni. To znači da svojstvo tipa podatka nikada ne može biti određeno korišćenjem: *owl:inverseOf, owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:SymmetricProperty*.
- *Nema prenosa ograničenja kardinalnosti*. Ograničenje kardinalnosti ne može biti postavljeno za tranzitivna svojstva ili njihova podsvojstva.
- Ograničena anonimnost klasa. Anonimne klase su dozvoljene samo kao domen i opseg *owl:equivalentClass* ili *owl: disjointWith*, i kao opseg (ali ne i domen) *rdfs: subClassOf.*

OWL Lite je najjednostavniji OWL podjezik.OWL Lite je namenjen da podrži klasifikacionu hijerarhiju i jednostavna ograničenja. Stoga je lakši za primenu i lakše razumljiv za korisnike. Mana OWL Lite-a je ograničena izražajnost. Izbor između OWL Lite i OWL DL jezika zavisi od kompleksnosti konstrukcija koje zahteva korisnik. Izbor između OWL DL i OWL Full uglavnom zavisi od potrebe za podrškom meta modelovanju (definisanje potklasa, pridruživanje svojstava klasi) .

Postoji stroga definicija kompatibilnosti naviše između ova tri OWL podjezika [8]:

- Svaka validna OWL Lite ontologija je validna OWL DL ontologija.
- Svaka validna OWL DL ontologija je validna OWL Full ontologija.
- Svaki validan OWL Lite zaključak je validan OWL DL zaključak.
- Svaki validan OWL DL zaključak je validan OWL Full zaključak.

OWL Lite ontologija mora biti OWL DL ontologija i moraju se zadovoljiti sledeća ograničenja:

- Konstruktori *owl:oneOf*, *owl:disjointWith*, *owl:unionOf*, *owl:complementOf* i *owl:hasValue* nisu dozvoljeni
- Ograničenje kardinalnosti (minimalna, maksimalna, i tačna kardinalnost) može biti postavljeno samo na vrednosti 0 ili 1 i više nije proizvoljan nenegativan ceo broj.
- *owl:equivalentClass* konstruktor se više ne može uspostaviti između anonimnih klasa, već samo između klasa

2.5 Logika i zaključivanje

Jezici RDF i OWL (Lite i DL) se mogu posmatrati kao specijalizacija predikatske logike. OWL Lite i OWL DL odgovaraju deskriptivnoj logici, podskupu predikatske logike za koju postoji efikasni sistem za rezonovanje.

Još jedan podskup predikatske logike sa efikasnim sistemom za rezonovanje se sastoji od tzv. sistema pravila (*logika Hornovih klauza*[21]). Programski jezik PROLOG(*PROgramming in LOGic*) je izgrađen na osnovi logike Hornovih klauza.

2.5.1 Logika Hornovih klauza

Svako pravilo ima oblik $B_1...B_r \rightarrow A$ gde su B_i ($i \in N$) i A atomičke formule. Postoje dva intuitivna načina čitanja takvog pravila:

- Ako je poznato da su $B_1...B_r$ tačne formule, onda je A takođe tačna formula. Pravila sa ovakvim tumačenjem su *deduktivna pravila*.
- Ako su uslovi $B_1...B_r$ tačne formule onda treba sprovesti formulu A. Pravila sa ovom interpretacijom se nazivaju *reaktivna pravila*.

U pravilu oblika $B_1...B_r \rightarrow A$, A je glava pravila, a $B_1,...,B_r$ su premise pravila. $B_1,..,B_r$ se naziva telo pravila. Zapete u telu pravila predstavljaju konjukcije: ako su B_1, B_2 i... B_r tačne formule, onda je i A tačna formula. Najopštija interpretacija pravila $B_1,...,B_r \rightarrow A$ je formula koja će biti označavana sa pl(r): $\forall X_1...\forall X_v((B_1 \wedge . . . \wedge B_r) \rightarrow A)$

Ili

njoj ekvivalentna formula: $\forall X_1...\forall X_v(A \vee \neg B_1 \vee ... \vee \neg B_v)$, gde su $X_1,...,X_v$ sve promenljive koje se pojavljuju u $A,B_1,...,B_r$.

[21] http://www.w3.org/2005/rules/wg/wiki/Horn_Logic

Logika programa P je konačan skup činjenica i pravila, pa stoga njegova logika *pl(P)* predstavlja skup svih predikatskih logičkih interpretacija pravila i činjenica u P. [4]

Cilj označava upit logici programa. On ima oblik $B_1,..., B_r \rightarrow$. Ako je n=0 onda je to prazan cilj. Interpretacija cilja predikatske logike ima sledeći oblik:

$\forall X_1 ... \forall X_v(\neg B_1 \vee ... \vee \neg B_v)$, gde su $X_1,...,X_v$ promenljive koje se koriste u $B_1,...,B_r$.

Ova formula je skoro ista kao i pl(r), a jedina razlika je što je glava pravila izostavljena. Ekvivalentna reprezentacija cilja predikatske logike je: $\neg \exists X_1 ... \exists X_v(B_1 \wedge ... \wedge B_r)$.

Ako je p(X) upit, i ako se zna da je p(a) tačna formula, potreban je odgovor na pitanje da li postoji X za koje je p(X) tačno. Očekivani odgovor je pozitivan zbog činjenice da je p(a) tačno i p(X) je egzistencijalno kvantifikovan. Za dokazivanje tačnosti formule koristi se dokaz pobijanjem. Ova tehnika pokazuje da formula A sledi iz formule B pretpostavljajući da je A netačna formula i izvođenjem protivrečnosti, u kombinaciji sa formulom B. Na upit se može pozitivno odgovoriti negirajući ga i dokazujući da je dobijena kontradikcija, koristeći logiku programa. Na primer, s obzirom na logiku programa p(a) i upit $\neg \exists X\, p(X)$ dobijaju se logičke protivrečnosti: druga formula kaže da nijedan element nema svojstvo p, ali prva formula kaže da vrednost a ima svojstvo p. Tako ∃X p(x) sledi iz p (a).

2.5.2 Deskriptivna logika

Deskriptivna logika i logika Hornovih klauza su ortogonalne u smislu da nijedna od njih nije podskup one druge [1]. Na primer, u deskriptivnoj logici je nemoguće izraziti odnos *stric (X, Y).* Ovaj odnos zahteva sposobnost ograničenja da vrednost svojstva *brat* koji se odnosi na term *X* bude vrednost svojstva *roditelj* za drugi term Y. Svojstvo *brat* primenjeno na *X* mora proizvesti rezultat koji je takođe vrednost svojsva *roditelj* kada se primenjuje na *Y*. Ovo pridruživanje predikata je izvan izražajnih mogućnosti OWL DL-a, međutim ovo znanje se lako može predstaviti korišćenjem pravila:

brat(X, Y), roditelj(Z, Y) → stric(X,Z)

Pravila logike Hornovih klauza ne mogu da predstave negaciju ili dopunu klase, disjunktne informacije i egzistencijalne kvantifikacije. OWL je u stanju da izrazi dopunu i uniju klasa i određene oblike egzistencijalne kvantifikacije.

Ako se razmotri jednostavno pravilo koje glasi da svi lojalni kupci stariji od 60 imaju pravo na popust:

LojalniKupac(*X*)*, starost*(*X*) > 60 →*pravoNapopust*(*X*)

može se primetiti nekoliko delova iz kojih se sastoji pravilo:

- promenljive, kojima se menjaju vrednosti X.
- konstante, koja označavaju fiksne vrednosti: 60 .
- predikati, koji predstavljaju odnose objekata: *LojalniKupac*, *pravoNapopust.*
- funkcionalni simboli koji vraćaju vrednost za određene argumente: *starost.*

U pokušaju da logika Hornovih klauza i deskriprivna logika ostvare svoju integraciju u jednom okviru, najjednostavniji pristup je da se razmotri njihov presek, to jest, da se pronađe deo jednog jezika koji se može prevesti na semantički prihvatljiv način u drugi jezik, i obrnuto. Presek logike Hornovih klauza i OWL-a se zove Deskriptivna programska logika (DLP)[22]; to je Horn-definisani OWL deo, ili OWL-definisani deo logike Hornovih klauza.

Prednosti DLP jezika su [1]:

- Iz perspektive onoga ko modeluje, postoji sloboda da se koristi ili OWL ili pravila (i pridruženi alati i metodologije) za modelovanje, u zavisnosti od iskustva i postavki koje se koriste.
- Prilikom rezonovanja mogu se koristiti ili rasuđivači deskriptivne logike ili sistem deduktivnih pravila. Tako je moguće da se za modele koristi jedan okvir, na primer OWL, i da se koristi sistem obrazloženja iz drugih okvira, na primer, pravila. Ova funkcija pruža dodatnu fleksibilnost i omogućava interoperabilnost sa mnoštvom alata.
- Iskustvo u korišćenju OWL-a je pokazalo da postojeće ontologije vrlo retko koriste konstrukcije van DLP jezika.

2.5.3 Izražavanje OWL konstrukcija u logici Hornovih pravila

Trojka (a,P,b) u RDF se može izraziti kao činjenica P(a,b); slično se deklaracija instance klase C tipa (a, C) može izraziti kao činjenica C(a). Činjenica da je C potklasa klase D se lako može izraziti kao C(x) → D(x); slično važi i za podsvojstva. Domeni i ograničenja mogu biti izraženi u logici Hornovih klauza, na primer, sledeće pravilo kaže da je činjenica C domen svojstva P*: P(x, y) → C(x).* U pogledu OWL-a *sameClassAs(C, D)* se može izraziti parom pravila *C(x) →D(x) i D(x) → C(x*), slično važi za *samePropertyAs.* Tranzitivnost svojstva P može biti izraženo kao *P*(*X, Y*)*, P*(*Y,Z*) *→ P*(*X,Z*). Presek klasa C_1 i C_2 je potklasa klase D može se izraziti na sledeći način: C_1 *(x),* C_2 *(x) →* D *(x)* i u drugom smeru, C je potklasa preseka D_1 i D_2: *C (x) →* D_1 *(x) i C (x) →* D_2 *(x*). Može se izraziti i da je unija klasa C_1 i C_2 potklasa klase D koristeći sledeći par pravila: C_1 *(x) →* D *(x) i* C_2 *(x) →* D *(x*). Da bi se izrazila činjenica da je C potklasa unije klasa D_1 i D_2, potrebna je disjunkcija odgovarajućih pravila na desnoj strani implikacije, što nije dostupno u logici Hornovih klauza. Za svojstvo P i

[22] http://logic.aifb.uni-karlsruhe.de/

klasu D, *allValuesFrom(P, D)* označava anonimnu klasu svih x takvih da y mora biti instanca klase D svaki put kada je ispunjeno P (x,y). OWL iskaz *C subClassOf allValuesFrom(P, D)* se može izraziti u logici Hornovih klauza kao *C(x), P (x,y) → D (y). someValuesFrom(P, D)* označava anonimnu klasu svih x za koje postoji najmanje jedno y koje je instanca klase D, tako da važi P (x, y). OWL iskaz *someValuesFrom*(*P,D*) *subClassOf C* se može izraziti u logici Hornovih klauza kao $P(x,y),D(y) \rightarrow C(x)$.

2.5.4 SWRL

SWRL[23] (*Semantic Web Rule Language*) je jezik semantičkog veba koji kombinuje OWL DL (a time i OWL Lite) sa logikom Hornovovih klauza tako da omogućava da se pravila logike Hornovih klauza kombinuju sa OWL DL ontologijom [6]. Pravilo u SWRL ima oblik $B_1,...,B_r \rightarrow A_1,...,A_v$ gde zapete označavaju konjunkciju i $A_1,...,A_v,B_1,...,B_r$ mogu biti u obliku *C (x), P (x, y), sameAs (x, y)* ili *differentFrom (x, y)*, gde je C OWL opis, P je OWL svojstvo, i x i y promenljive, OWL instance ili OWL vrednosti promenljive. Ako glava pravila ima više od jednog atoma (ako je konjunkcija atoma bez deljenja promenljivih), pravilo se može transformisati u ekvivalentan skup pravila sa jednim atomom u glavi pravila. Glavna složenost SWRL jezika proizilazi iz činjenice da OWL izrazi (kao što su ograničenja) mogu da se pojavljuju u glavi ili telu pravila.

U skladu sa vizijom semantičkog veba, neophodno je da znanje u obliku pravila bude dostupno računaru. RuleML[24] je važan standard za označavanje pravila na vebu. To nije jedan jezik, već porodica jezika za označavanje pravila koji odgovaraju različitim vrstama jezika za predstavljanje pravila: derivaciona pravila, pravila za ograničenja integriteta, reakciona pravila, itd. Izražavanje pravila korišćenjem RuleML je jasno. Na primer, pravilo "popust za kupca za kupovinu proizvoda je 7,5 odsto ukoliko klijent ima status premium, a proizvod predstavlja luksuz" predstavlja se kao:

```
<Implies>
    <head>
        <Atom>
            <Rel>popust</Rel>
            <Var>kupac</Var>
            <Var>proizvod</Var>
            <Ind>7.5 odsto</Ind>
        </Atom>
    </head>
    <body>
        <And>
            <Atom>
```

[23] http://www.w3.org/Submission/SWRL/
[24] http://ruleml.org/

```
                    <Rel>premium</Rel>
                    <Var>kupac</Var>
                </Atom>
                <Atom>
                    <Rel>luksuz</Rel>
                    <Var>proizvod</Var>
                </Atom>
            </And>
        </body>
</Implies>
```

SWRL je proširenje RuleML-a[7]. Pravilo *brat (x, y), roditelj (z, y) → stric (x, y)* u XML sintaksi SWRL-a se predstavlja kao:

```
<ruleml:Implies>
    <ruleml:head>
        <swrlx:individualPropertyAtom swrlx:property="stric">
            <ruleml:Var>x</ruleml:Var>
            <ruleml:Var>z</ruleml:Var>
        </swrlx:individualPropertyAtom>
    </ruleml:head>
    <ruleml:body>
        <ruleml:And>
            <swrlx:individualPropertyAtom swrlx:property="brat">
                <ruleml:Var>x</ruleml:Var>
                <ruleml:Var>y</ruleml:Var>
            </swrlx:individualPropertyAtom>
            <swrlx:individualPropertyAtom swrlx:property="childOf">
                <ruleml:Var>Z</ruleml:Var>
                <ruleml:Var>Y</ruleml:Var>
            </swrlx:individualPropertyAtom>
        </ruleml:And>
    </ruleml:body>
</ruleml:Implies>
```

Semantički rasuđivač (*engl. semantic reasoner*) je program koji ima mogućnost da zaključi logičke posledice skupa činjenica ili zadatih aksioma (tvrđenja koji se ne dokazuju). Pravila zaključivanja su obično navedena korišćenjem jezika ontologije. Većina rasuđivača za rezonovanje koristi predikatsku logiku prvog reda. Često korišćenji semantički rasuđivači su FACT[25], Pellet[26], Jena[27], RaserPro[28], Jess Rule Engine[29].

[25] http://www.cs.man.ac.uk/~horrocks/FaCT/
[26] http://clarkparsia.com/pellet
[27] http://jena.sourceforge.net/
[28] http://www.racer-systems.com/

2.6 Jess Rule Engine

Jess Rule Engine (Jess) je semantički rasuđivač koji koristi poboljšanu verziju Rete algoritma za obradu pravila koji je efikasan mehanizam za rešavanje komplikovanih problema koji sadrže podudaranje više-prema-više. Jess ima mnogo značajnih osobina, uključujući efikasnost u radu sa velikim brojem instanci i mogućnost izvršavanja upita u radnoj memoriji. Jess podržava rad SWRL u sprezi sa SQWRL, što je bilo od najvećeg značaja za izbor ovog rasuđivača.

2.7 Protégé – alat za uređivanje ontologija

Protégé[30] je platforma otvorenog koda koja pruža paket alata za pravljenje domenskih modela i aplikacija korišćenjem ontologija. *Protégé* pruža bogat skup struktura za modelovanje i aktivnosti koje podržavaju stvaranje, vizuelizaciju i manipulaciju ontologijama koje su zastupljene u različitim formatima. *Protégé* se može proširiti putem plug-in arhitekture i Java Application Programming Interface (API) za izgradnju alata i aplikacija. *Protégé* ima snažnu podršku od strane korisnika, koji ga koriste za rešenja u raznolikom skupu područja kao što su biomedicina, prikupljanje obaveštajnih informacija, korporativna modelovanja, itd. *Protégé* je razvijen u Stenford istraživačkom centru za biomedicinsku informatiku na Medicinskom fakultetu Univerziteta Stenford u Kaliforniji. [9]

Protégé platforma podržava dva osnovna načina modelovanja ontologije:

- *Protégé* - *Frame* uređivač omogućava korisnicima izgradnju i popunjavanje ontologija koje su zasnovane na okvirima, u skladu sa *Open Knowledge Base Connectivity* protokolom (OKBC). U ovom modelu, ontologija se sastoji od skupa hijerarhijski organizovanih klasa koje predstavljaju kostur koncepta domena, skup isečaka vezanih za klase koji opisuju svojstva i odnose, kao i niz instanci klasa - primeraka klasa koji imaju određene vrednosti za svoja svojstva. Karakteristike *Protégé* - *Frame* uređivača su:
    - širok skup elemenata korisničkog interfejsa koji se može prilagoditi da bi se omogućilo korisnicima da kreiraju modele znanja i unose podatke u prihvatljivoj formi.
    - plug-in arhitektura koja se može proširiti prilagođenim elemenata, kao što su grafički elementi (na primer, grafikoni i tabele), mediji (na primer, zvuk, slike i video), razni formati za skladištenje (npr., RDF, XML, HTML i baze podataka), kao i alati za dodatnu podršku (npr. za upravljanje ontologijama, vizuelizaciju ontologija, rezonovanje, itd.)

[29] http://www.jessrules.com
[30] http://protege.stanford.edu

    - Java Application Programming Interface (API) koji omogućava plug-in i pristup i korišćenje od strane drugih aplikacija.
- *Protégé* - OWL uređivač omogućava korisnicima da izgrade ontologije za semantički veb, posebno u OWL jeziku veb ontologija. OWL ontologije mogu da sadrže klase, svojstva i instance. OWL formalna semantika precizira kako da se izvuku logične posledice, odnosno činjenice koje nisu bukvalno opisane u ontologiji, ali su podrazumevane semantikom, što se zasniva na jednom ili više distribuiranih dokumenata koji su kombinovani korišćenjem OWL mehanizama. *Protégé* - OWL uređivač omogućava:
    - Učitavanje i čuvanje OWL i RDF ontologije.
    - Izmenu i vizualizaciju klasa, svojstava i SWRL pravila.
    - Definisanje logičkih karakteristika klasa kao što je OWL izraz.
    - Rezonovanje.
    - Izmenu OWL instanci.

Jedan od značajnijih dodataka koji su definisani za *Protégé* je SQWRL(*Semantic Query-Enhanced Web Rule Language*)[31] jezik za izvršavanje upita OWL ontologije zasnovan na SWRL-u. On pruža operacije za preuzimanje znanja iz OWL slične SQL – u. SQWRL ima sledeće karakteristike:

- Osnovni operatori koriste SWRL pravila kao obrazac i zamenjuju pravila u zavisnosti od izbora i oblikovanja operatora.
- Operatori nad kolekcijama podržavaju više naprednih mogućnosti upita, SQWRL ima izbor operatora nad kolekcijama koji pružaju napredne funkcionalnosti grupisanja i agregacija i ograničene forme negacije i disjunkcije.

SQWRL se definiše korišćenjem biblioteke ugrađenih komponenti (*engl. built-ins*) SWRL jezika. Ove ugrađene komponente su definisane u SQWRL ontologiji. Kopija ove ontologije se može naći kao standardni deo *Protégé* - OWL baze znanja. SQWRL upiti mogu da rade u sprezi sa SWRL-om i može da se koristi za preuzimanje znanja dobijenog iz SWRL pravila. Sposobnost da se slobodno definišu i koriste ugrađene komponente u upitu obezbeđuje sredstvo za stalno proširenje mogućnosti ovog upitnog jezik.

SQWRL podržava dve vrste kolekcija : *set* i *bag*. *Set*, za razliku od *bag*, ne dozvoljava duplikate elemenata. Funkcija *sqwrl:makeSet* obezbeđuje konstrukciju *set* kolekcije i ima formu *sqwrl:makeSet(<set>,<element>),* slično *sqwrl:makeBag* obezbeđuje konstrukciju *bag* kolekcije i ima formu *sqwrl:makeBag(<bag>,<element>).* Operatori nad kolekcijama kao što je *sqwrl:size* mogu biti primenjeni na obe vrste kolekcija. Operator grupisanja *sqwrl:groupBy* omogućava složenije upite za koje je potrebna podela elemenata kolekcije.

[31] http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL

SQWRL podržava standardne operacije agregacija nad kolekcijama čiji elementi imaju prirodno definisano uređenje *(sqwrl:avg, sqwrl:max, sqwrl:min, sqwrl:sum* ...).

3 Primena semantičkog veba u automatskom upravljanju uređajima

3.1 Pametne kuće

Kontrola okruženja i upravljanje energijom u standardnim kućnim sredinama je retko podržano naprednim sistemima. Ukoliko postoje sistemi za automatizaciju upravljanja uređajima u kućama ili zgradama, to su najčešće jednostavni sistemi kao na primer sistem za regulaciju temperature. U zavisnosti od zemlje, tipične kuće mogu već da budu opremljene pametnim brojilima sa ekranom i servisom koji omogućuje pregled dijagrama potrošnje energije u vremenskim intervalima. Dostupnost ovih dijagrama povećava mogućnost za uštedu energije, što inače presudno zavisi od svesti kupaca i dobre prakse, na primer, isključivanje svetla ili promena temperature u toku noći ili prilikom odsustva iz kuće. Da bi se smanjila potreba da sami korisnici vode računa o tome, može se uvesti još nekoliko naprednih sistema za automatizaciju uređaja u kući kao što su inteligentna kontrola svetla, grejanja i ventilacije u skladu sa poznatim situacijama. Ovakvi sistemi obično zahtevaju programiranje, a korisnik ima ulogu administratora sistema. Performanse takvih sistema izuzetno zavise od pravilne administracije, koja je najčešće suviše težak zadatak za tipičnog korisnika. Značajan napredak može biti uveden samo pomoću tehnologije koja sa jedne strane može da poveže one delove sistema koji bi trebalo da razmene informacije direktno, a sa druge strane može da ostvari interakciju sa korisnikom na intuitivan i lako prihvatljiv način. Semantička tehnologija je jedan od ključnih faktora takvog poboljšanja.

U poslednjih nekoliko godina, napredak u nekoliko oblasti istraživanja imao je značajan uticaj na modelovanje i implementaciju sistema “pametne kuće”. Pametna kuća je kompleksan i dinamičan sistem koji mora biti otvoren za jednostavnu integraciju i interakciju sa mnogo različitih uređaja (centralno kontrolisano svetlo, sistemi za grejanje i ventilaciju, automatske roletne i sl.) i sa mnogo različitih korisnika. Da bi rad u ovako dinamičnom okruženju bio moguć, uređaji, servisi i agenti moraju biti svesni konteksta tj. informacija koje karakterišu određenu situaciju. Za izgradnju ovakvih sistema koji prepoznaju kontekst potrebno je da budu zadovoljena sledeća dva uslova: nivo posredne podrške i odgovarajući kontekstni model. Nivo posredne podrške je mesto na kom se informacije prikupljaju sa fizičkih senzora, iz baze podataka i od agenata, kontekstno tumače i blagovremeno distribuiraju. Kada je u pitanju pristup dizajnu kontekstnih modela, postoje:

- aplikativno-orijentisani,
- model-orijentisani i
- ontološki-orijentisani pristupi.

Aplikativno-orijentisanim pristupom se predstavlja kontekst relevantan samo za specifičnu aplikaciju i zato takvim modelima nedostaje formalnost i određenost i oni ne podržavaju razmenu

znanja između različitih sistema. Model-orijentisani pristup se obično zasniva na formalnom kontekstu modela i sadrži i model veza među objektima i model uloga objekata i njime se lako upravlja korišćenjem relacionih baza podataka. Nedostatak model-orijentisanog pristupa je nemogućnost razmene znanja i kontekstnog rezonovanja. Ontološki-orijentisani pristup se zasniva na kontekstu ontologije i koristi pogodnosti automatskog kontekstnog rezonovanja baziranog na tehnologijama semantičkog veba.

3.2 SESAME projekat

SESAME[32] *(SEmantic SmArt MEtering)* je međunarodni projekat realizovan u okviru mreže kojom koordinira Austrijki istraživački centar - Forschungszentrum Telekommunikation Wien Betriebs-GmbH[33] (FTW), i kojoj pripadaju četiri inovativna preduzeća, od kojih su dva iz Austrije - eSYS[34] Informationssysteme GmbH (eSYS - Attnang-Puchheim) i Semantic Web Company [35] (SWC - Vienna), jedno iz Srbije - E-Smart Systems d.o.o. Beograd[36](E-SMART, u kojoj sam ja zaposlena), i jedno iz Rusije - The Experimen-tal Factory of Scientific Engineering[37](EZAN - Chernogolovka). Uloga E-SMART-a u ovom projektu je ekspertiza u implementaciji naprednih mernih sistema i sigurnosnih rešenja, implementacija sopstvenog proizvoda za automatizovano daljinsko očitavanje i konfiguraciju elektronskih brojila, kao i sticanje iskustva u oblasti semantičkih tehnologija i automatizaciji zgrada.

SESAME sistem koristi ontologije i servisno orjentisani dizajn za integraciju naprednih brojila za daljinsko očitavanje i automatizaciju upravljanja uređajima u zgradi u fleksibilan sistem koji je kontrolisan korisnički definisanim pravilima. SESAME sistem je projektovan da radi na zajedničkom, semantički opisanom modelu, uključujući i pravila koja definišu individualne potrebe korisnika, operativne osobine perifernih uređaja, senzora i aktuatora (mehanizma koji uzrokuje da se uređaj uključi ili isključi, prilagodi ili pomeri), kao i eksternih informacija o dostupnosti i ceni električne energije. [11]

Semantički model SESAME sistema se sastoji od ontologije uređaja koja opisuje pojmove vezane za uređaje, ambijant i ponašanje uređaja radi dostizanja željenog stanja ambijanta, i ontologije energetskih preferecija koja opisuje osnovni sistem ograničenja i pravila o prednosti korišćenja uređaja u zavisnosti od tipa energije, cene i energetskih tarifa. Održavanje ontologija je ključni zadatak u takvom dinamičnom sistemu. Održavanje ontologija i automatsko rezonovanje

[32] http://sesame.ftw.at/
[33] http://www.ftw.at/
[34]http://www.esys.at
[35] http://www.semantic-web.at
[36] http://www.e-smartsys.com
[37] http://www.ezan.ac.ru

vrši se korišćenjem alata kao što su Pelet ili Jess rasuđivač, Jena Framework i Protege uređivač za kreiranje i obradu ontologija. Sistem uzima u obzir kontekst trenutnog i željenog ambijenta svojih korisnika i pruža kontekstno prepoznate usluge. Definisana je četvoroslojna arhitektura za upravljanje prikupljanjem podataka i kontekstnim informacijma, a ona obuhvata [11]:

- sloj percepcije koji dobija informacije putem senzora,
- sloj konteksta koji daje semantičku reprezentaciju podataka,
- sloj zaključivanja koji obrađuje podatke i određuje sprovođenje određenih akcija koje se odnose na rad aktuatora, i
- sloj akcije koji pokreće akcije koje su određene u sloju zaključivanja.

U okviru ove arhitekture kontekstni model baziran je na ontologiji dizajniranoj je u OWL-u, a sloj zaključivanja je zasnovan na SWRL i SQWRL jezicima za definisanje pravila. Ontologija se popunjava kontekstualnim podacima korišćenjem servisa koji su pripadaju sloju konteksta.

Zbog zahteva za podršku nezavisnom razvoju novih i integraciji postojećih servisa, SESAME koristi ontologiju na koju je prevedena funkcionalnost servisa, kao standard koji je nezavisan od implementacije servisa. Svrha korišćenja ontologije uređaja u okviru SESAME sistema je podrška lakoj konfiguraciji sistema "pametne kuće" od strane kranjeg korisnika tako što su tehnički detalji aktuatora "sakriveni" od krajnjeg korisnika sistema. Krajnjem korisniku je vidljiv samo najviši kontekstni sloj.

3.3 Ontologija uređaja

Rad aktuatora je modelovan u ontologiji uređaja pomoću pojmova aktuator, uređaj, tip uređaja, dok su politike ponašanja sistema modelovane pomoću pojmova željeno stanje, aktivnost stanara, trenutno stanje, lokacija i SWRL i SQWRL pravila. *(Obzirom na to da je ova ontologija sastavni deo većeg sistema koji je urađen u okviru međunarodnog projekta SESAME, nazivi klasa, svojstava i instanci klasa su na engleskom jeziku).*

3.3.1 Klase

Tip aktivnosti uređaja je definisan klasom *DeviceActivityType* tj. njenim instancama koje označavaju mogući način korišćenja uređaja: *Cooling, Heating, Lighting, MusicPlayer, Ventilating, WaterHeating, OnOff.* Obzirom na to da jedan uređaj npr.klima uređaj može obavljati više uloga (može da greje, hladi, provetrava) uvedena je i klasa *DeviceType* kojom je definisana svrha korišćenja uređaja preko svojstva *hasActivityType* kardinalnosti n (n>=1) čije vrednosti su instance klase *DeviceActivityType.* Ove klase su implementirane u ontologiji na sledeći način:

```
<owl:Class rdf:ID="DeviceActivityType">
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#IsInteruptableActivity"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="DeviceType">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDatabaseIdentification"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#dependsOnOutOfDoorCondition"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

Aktuator predstavlja mehanički deo uređaja za kontrolu nekog mehanizma ili sistema. Aktuatori su predstavljeni klasom *Actuator* koja osim opštih svojstava *hasName, hasIdentification* sadrži i svojstva koja označavaju trenutno stanje aktuatora *hasState* (može biti *on* ili *off*) i stanje koje bi trebalo postaviti da bi se dostiglo željeno stanje *StateToBeSet* (može biti *on* ili *off*). OWL repezentacija klase *Actuator*:

```
<owl:Class rdf:ID="Actuator">
    <rdfs:subClassOf>
```

```
    <owl:Restriction>
      <owl:onProperty rdf:resource="#influenceLoudness"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasDatabaseIdentification"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#influenceAirTemperature"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#influenceBoilerWater"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#StateToBeSet"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#influenceHotWater"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasState"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#influenceLight"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
```

```
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#influenceHumidity"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

Klasa *Condition* je opšta klasa koja opisuje vrednosti senzora korišćenjem obaveznog svojstava *hasValue* čija vrednost predstavlja očitanu ili željenu vredost senzora i svojstva *hasTolerance* koje nije obavezno i čija vrednost predstavlja toleranciju za dostizanje željene vrednosti senzora. Potklase ove klase su njene specijalizacije prema tipu senzora : *AirHumidityCondition* – vlažnost vazduha*, AirTemperatureCondition* – temperatura vazduha*, LightCondition* – svetlost, *LoudnessCondition* – buka*, BoilerWaterCondition* – toplota vode u bojleru*, HeatingWaterCondition* – toplota vode za grejanje. Relacije između klase *Actuator* i potklasa klase Condition definisane su preko odgovarajućih svojstava *influenceOnAirTemperature, influenceOnHumidity, influenceOnLight, influenceOnLoudness, influenceOnBoilerWater, influenceOnHeatingWater*. Implementacija klase *Condition* i njenih potklasa :

```
  <owl:Class rdf:ID="Condition">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasValue"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasTolerance"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

```
<owl:Class rdf:ID="AirHumidityCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
</owl:Class>
<owl:Class rdf:ID="AirTemperatureCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
</owl:Class>
<owl:Class rdf:ID="LightCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
  </owl:Class>
<owl:Class rdf:ID="LoudnessCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
  </owl:Class>
<owl:Class rdf:ID="BoilerWaterCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
</owl:Class>
<owl:Class rdf:ID="HeatingWaterCondition">
    <rdfs:subClassOf rdf:resource="#Condition"/>
</owl:Class>
```

Klase *DayProfile* i *ResidentActivityindex* koje imaju samo svojstvo *hasName* se koriste samo kao pomoć pri opisu drugih klasa, tj. njihove instance se koriste kao moguće vrednosti nekih svojstava drugih klasa. *DayProfile* definiše o kom tipu dana se radi tj.da li je to radni dan, vikend ili odmor i njene instance su WorkingDay, Weekend,Holiday. *ResidentActivityindex* definiše aktivnost koja se povezuje za ambijent u kući i njene obavezne, podrazumevane instance su NoActivity i OutOfHome, prva označava da postoji prisustvo u nekoj prostoriji u kući ali ne u zadatoj prostoriji, a druga označava da nema prisustva u kući. Instance klase *ResidentActivityindex* mogu biti korisnički definisane aktivnosti npr. odmaranje, spavanje, vežbanje, itd.

```
 <owl:Class rdf:ID="DayProfile">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
  <owl:Class rdf:ID="ResidentActivityIndex">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
```

```
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

ResidentActivity je jedna od centralnih klasa ontologije i njene instance predstavljaju raspored aktivnosti koji zadaje korisnik. Svojstva koja opisuju ovu klasu su :

- *isDefault* (DataProperty) - označava da li je raspored vezan za neku od obaveznih, podrazumevanih aktivnosti (NoActivity ili OutOfHome); vrednost ovog svojstva je tipa *boolean.*
- *isScheduleActive* (DataProperty) - označava da li je raspored trenutno aktivan tj.da li se uzima u obzir prilikom odlučivanja o postavljanju željenog ambijenta; vrednost ovog svojstva zavisi od tipa tekućeg dana i vrednosti svojstva hasDayProfile; vrednost ovog svojstva je tipa *boolean.*
- *hasTimeFrom* (DataProperty) – označava vremenski početak željene aktivnosti za koju bi trebalo postaviti ambijent; vrednost ovog svojstva je tipa *time.*
- *hasTimeTo* (DataProperty) – označava vremenski kraj željene aktivnosti za koju bi trebalo postaviti ambijent; vrednost ovog svojstva je tipa *time.*
- *hasDayProfile* (ObjectProperty) – označava tip dana za koji je definisan raspored; vrednost ovog svojstva je neka od instanci klase DayProfile.
- *hasResidentActivityIndex* (ObjectProperty) – označava konkretnu aktivnost za koju je definisan raspored, vrednost ovog svojstva je neka od instanci klase ResidentActivityIndex.

```
  <owl:Class rdf:ID="ResidentActivity">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasResidentActivityIndex"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#isScheduleActive"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDayProfile"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasTimeTo"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasTimeFrom"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasIdentification"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#isDefault"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

Klasom *Location* opisane su prostorije u kući. Ova klasa osim osnovih svojstava *hasName* i *hasDatabaseIdentification* prostorije ima i dodatna svojstva kojim opisuje prostorije: *hasStateExpiredPeriod*, čija vrednost je ceo broj koji označava vreme nakon kog je dozvoljena promena aktivnosti u prostoriji ukoliko u njoj nije zabeleženo prisustvo, *hasPossibleActivities* – čije su vrednosti instance klase ResidentActivity i koje predstavlja skup mogućih aktivnosti za jednu prostoriju, pri čemu se mora voditi računa da se vremenski okviri ne preklapaju za isti tip dana.

```
<owl:Class rdf:ID="Location">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasStateExpiredPeriod"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDatabaseIdentification"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasActivity"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

Klasa *ResidentOnLocation* predstavlja samo vezu između klasa Location i ResidentActivity i vrednost njenog svojstva *hasActivity* je neka od instanci klase ResidentActivity koja predstavlja trenutno postavljenu aktivnost u prostoriji.

```
  <owl:Class rdf:ID="ResidentOnLocation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasLocation"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasActivity"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>
```

Konkretni uređaji koji utiču na ambijent u kući opisani su klasom *Device.* Značajna svojstva ove klase su *hasPower*, kojim je opisana snaga uređaja u vatima (W) i čija vrednost je ceo broj, i *hasActuator,* koje predstavlja vezu uređaja sa klasom Actuator tj. vrednost ovog svojstva pridružuje konkretan uređaj aktuatoru koji upravlja njegovim ponašanjem. *isAffected* je svojstvo

ove klase čije vrednosti tipa boolean su krajnji rezultat procesa rezonovanja i njime je označen uticaj konkretnog uređaja na dostizanje željenog ambijenta u prostoriji.

```
<owl:Class rdf:ID="Device">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasPower"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasActionPurpose"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasAmbientPurpose"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasLocation"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasActuator"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDeviceType"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
```

```
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#isAffected"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDatabaseIdentification"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

CurrentOutOfDoorState je klasa koja opisuje vremenske uslove preko vrednosti meteoroloških parametara kao što su vlažnost, temperatura i sl. Vrednosti svojstva *Humidity, Light, Temperature, Wind* ove klase su numeričke vrednosti značajne prilikom odlučivanja koji uređaj bi trebalo aktivirati/deaktivirati pre nekog drugog da bi se postigao željeni ambijent u prostoriji, a da se pri tom vodi računa o potrošnji električne energije.

```
<owl:Class rdf:ID="CurrentOutOfDoorState">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Humidity"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Light"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Temperature"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="#Wind"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

Trenutni ambijent u prostoriji opisan je klasom *CurentState*. Svojstvo *hasLocation* predstavlja vezu sa klasom Location, dok svojstvo *hasCondition* predstavlja vezu sa klasom Condition tj. sa potklasama ove klase; ovo svojstvo je veoma značajno za krajnji rezulat rezonovanja jer je njegova vrednost polazna tačka u rezonovanju, tj. vrednost sa kojom se upoređuje željeno stanje ambijenta. Osim ova dva svojstva kojima se definiše ambijent na lokaciji, još jedno značajno svojstvo za rezonovanje u ovoj ontologiji je svojstvo *DetectPresence* čija vrednost tipa *boolean* označava da li je u prostoriji detektovano prisustvo što može bitno da utiče na stanje ambijenta koje će biti postavljeno.

```
<owl:Class rdf:ID="CurrentState">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasLocation"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#DetectPresence"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasCurentDateTime"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasLastDateTimePresence"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

Klasa *ActiveDesiredState* opisuje željena stanja ambijenta i ona kao i klasa CurentState ima svojstvo *hasCondition* koje predstavlja vezu sa klasom Condition tj. sa potklasama ove klase i vrednost ovog svojstva predstavlja željeno stanje ambijenta. Još jedno svojstvo ove klase je svojstvo *dependsOnActivity* koje povezuje ovu klasu sa klasom ResidentActivity i čija vrednost označava za koje sve aktivnosti u prostoriji je potreban takav ambijent. Svojstvo *StandbyDavicesBehavior* je dodatno svojstvo koje može imati neku od vrednosti *NoChange, SwitchOn, SwitchOff* i koje opisuje ambijent, ali ne u smislu vrednosti temperature, vlažnosti, svetlosti već opisuje način ponašanja uređaja koji se mogu isključiti ili uključiti na *stand by* režim rada (npr. televizor).

```
<owl:Class rdf:ID="ActiveDesiredState">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#StandByDevicesBehavior"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasIdentification"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasName"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#isDefault"/>
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

3.3.2 Svojstva

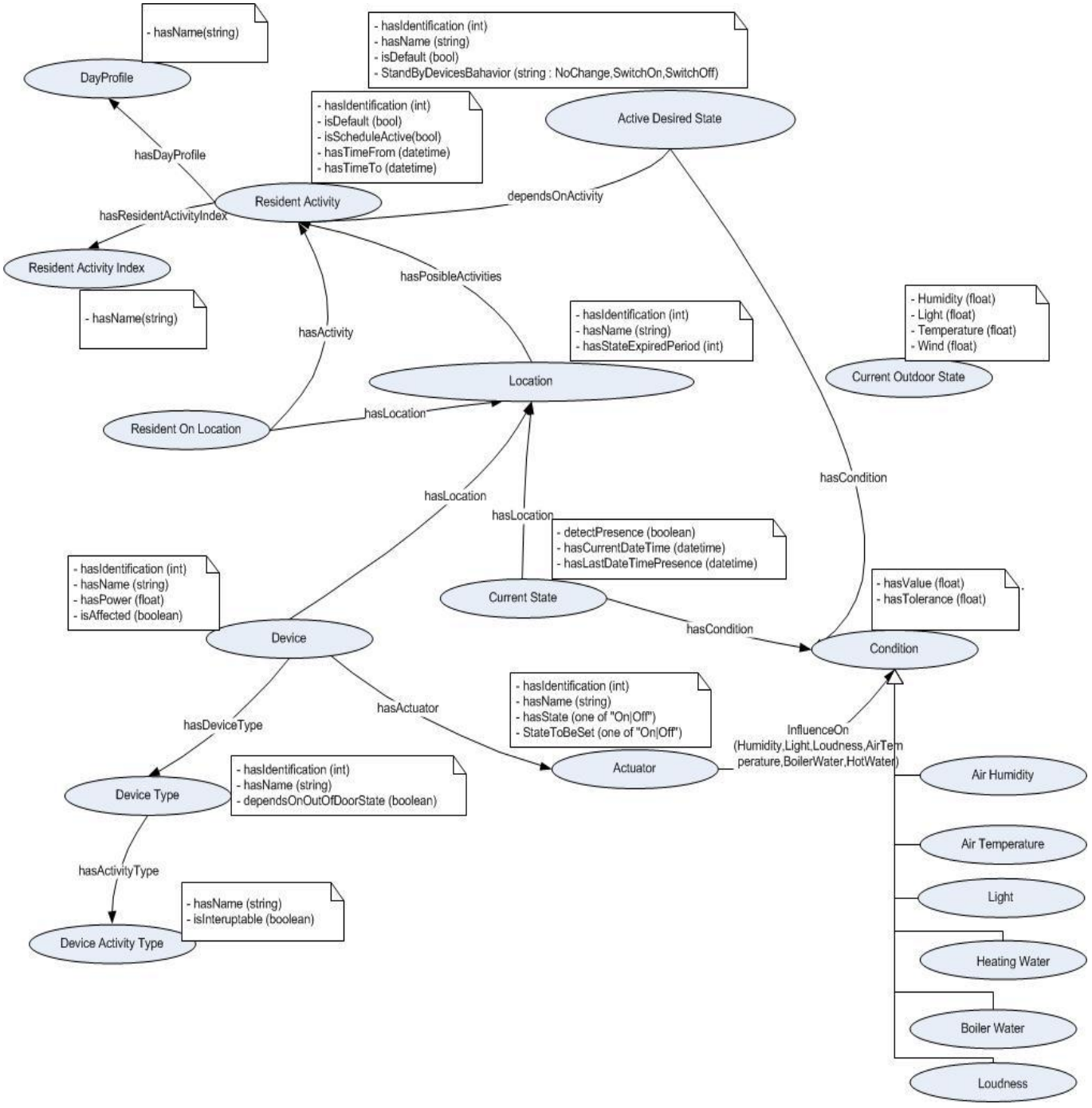
Klase ontologije uređaja opisane su zajedničkim svojstvima, objektnim svojstvima i svojstvima tipova podataka. Zajednička svojstva su opisana navođenjem svih klasa koje ih koriste.

Neka od takvih svojstava (npr. *hasName*, *isDefault, hasActivity, hasCondition, hasLocation*) u ontologiji uređaja imaju sledeću OWL reprezentaciju:

```
<owl:DatatypeProperty rdf:ID="hasName">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#ActiveDesiredState"/>
        <owl:Class rdf:about="#Actuator"/>
        <owl:Class rdf:about="#ConditionInfluenceType"/>
        <owl:Class rdf:about="#DayProfile"/>
        <owl:Class rdf:about="#Device"/>
        <owl:Class rdf:about="#DeviceActivityType"/>
        <owl:Class rdf:about="#DeviceType"/>
        <owl:Class rdf:about="#Location"/>
        <owl:Class rdf:about="#ResidentActivityIndex"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="isDefault">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#ActiveDesiredState"/>
        <owl:Class rdf:about="#ResidentActivity"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasActivity">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Location"/>
        <owl:Class rdf:about="#ResidentOnLocation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#ResidentActivity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasCondition">
  <rdfs:domain>
```

```
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#ActiveDesiredState"/>
            <owl:Class rdf:about="#CurrentState"/>
          </owl:unionOf>
        </owl:Class>
      </rdfs:domain>
      <rdfs:range rdf:resource="#Condition"/>
    </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasLocation">
      <rdfs:domain>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#CurrentState"/>
            <owl:Class rdf:about="#Device"/>
            <owl:Class rdf:about="#ResidentOnLocation"/>
          </owl:unionOf>
        </owl:Class>
      </rdfs:domain>
      <rdfs:range rdf:resource="#Location"/>
    </owl:ObjectProperty>
```

3.3.3 Grafovski prikaz



3.3.4 Prostor imena

Zbog postojanja potrebe za upoređivanjem datumskih i vremenskih vrednosti u ontologiju je uvezena javna *Temporal* ontologija[38] jednostavnim uvođenjem u prostor imena ontologije uređaja:

```
<rdf:RDF xmlns="http://www.e-smartsys.com/2010/4/DevicesOntology.owl#"
    xml:base="http://www.e-smartsys.com/2010/4/DevicesOntology.owl"
    xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
    xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:temporal="http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
```

3.3.5 Pravila rezonovanja

Rezonovanje nad ovom ontologijom se zasniva na *SWRL i SQWRL* pravilima koja se primenjuju korišćenjem *Jess Rule Engine.* Skup pravila na kojima se zasniva rezonovanje u ovoj ontologiji čine pravila o rasporedu aktivnosti, prisustvu/odsustvu u prostorijama i uključivanju/isključivanju uređaja u zavisnosti od željenog stanja ambijenta. U narednim delu teksta biće prikazana neka od tih pravila.

Pravilo koje postavlja aktivnosti stanara u prostorijama (lokacijama) na trenutno aktivne prema rasporedu ukoliko bar u jednoj prostoriji u kući postoji prisustvo uključuje klase *Location, CurrentState, ResidentActivity, ResidentOnLocation* i svojstva *hasLocation, hasPossibleActivities, DetectPresence, isScheduleActive, hasPossibleActivities, isDefault, hasTimeFrom, hasTimeTo, hasCurentDateTime, hasLastDateTimePresence, hasStateExpiredPeriod* i *hasActivity.* Atomi ovog pravila predstavljaju konjunkciju sledećih uslova:

- prisustvo neke osobe na bilo kojoj lokaciji u kući – postojanje prisustva u kući
- aktivnost koja ulazi u izbor nije neka od podrazumevanih aktivnosti, njen raspored je aktivan i definiše trajanje te aktivnosti u periodu kom pripada trenutno vreme

[38] http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl#

- izabrana aktivnost mora biti neka od aktivnosti koju je korisnik definisao kao svoju moguću aktivnost na toj lokaciji
- trenutno stanje na toj lokaciji treba da ispunjava uslov da nije prošlo više od dozvoljenog vremena za odsustvo iz prostorije; vreme proteklo od poslednjeg detektovanog prisustva u prostoriji mora biti manje ili jednako dozvoljenom vremenu.

Ovo pravilo ima sledeći izgled:

Location(?prl) ∧ CurrentState(?cs2) ∧ hasLocation(?cs2, ?prl) ∧ DetectPresence(?cs2, true) ∧ ResidentActivity(?ra) ∧ isScheduleActive(?ra, true) ∧ isDefault(?ra, false) ∧ hasTimeFrom(?ra, ?ratf) ∧ hasTimeTo(?ra, ?ratt) ∧ CurrentState(?cs) ∧ hasLocation(?cs, ?l) ∧ hasCurentDateTime(?cs, ?ct) ∧ temporal:notAfter(?ratf, ?ct) ∧ temporal:before(?ct, ?ratt) ∧ ResidentOnLocation(?rac) ∧ hasLocation(?rac, ?l) ∧ hasPossibleActivities(?l, ?ra) ∧ hasStateExpiredPeriod(?l, ?ep) ∧ hasLastDateTimePresence(?cs, ?ltp) ∧ temporal:duration(?dtp, ?ct, ?ltp, temporal:Seconds) ∧ swrlb:multiply(?eps, ?ep, 60) ∧ swrlb:lessThanOrEqual(?dtp, ?eps) → hasActivity(?rac, ?ra) ∧ hasActivity(?l, ?ac)

Pravilo koje postavlja predefinisanu aktivnost *ResidentActivity_NoActivity,* koja označava da u prostoriji nije detektovano prisustvo neko vreme a u bar jednoj prostoriji u kući postoji prisustvo, na trenutnu aktivnost stanara na toj lokaciji uključuje klase *Location, CurrentState, ResidentOnLocation* i svojstva *hasLocation, DetectPresence, hasCurentDateTime, hasLastDateTimePresence, hasStateExpiredPeriod i hasActivity*. Ovo pravilo sadrži konjunkciju atoma koji predstavljaju ispunjenje sledećih uslova:

- prisustvo neke osobe na bilo kojoj lokaciji u kući – postojanje prisustva u kući
- lokacija na kojoj se postavlja predefinisana aktivnost *ResidentActivity_NoActivity* je lokacija čije trenutno stanje treba da ispunjava uslov da je prošlo više od dozvoljenog vremena za odsustvo iz prostorije, vreme proteklo između poslednja dva detektovana prisustva u prostoriji mora biti veće od dozvoljenog vremena

SWRL reprezentacija ovog pravila je sledeća:

Location(?prl) ∧ CurrentState(?cs2) ∧ hasLocation(?cs2, ?prl) ∧ DetectPresence(?cs2, true) ∧ Location(?l) ∧ CurrentState(?cs) ∧ DetectPresence(?cs, false) ∧ hasLocation(?cs, ?l) ∧ hasStateExpiredPeriod(?l, ?ep) ∧ hasLastDateTimePresence(?cs, ?ltp) ∧ hasCurentDateTime(?cs, ?ct) ∧ temporal:duration(?dtp, ?ct, ?ltp, temporal:Seconds) ∧ swrlb:multiply(?eps, ?ep, 60) ∧ swrlb:greaterThan(?dtp, ?eps) ∧ ResidentOnLocation(?rol) ∧ hasLocation(?rol, ?l) → hasActivity(?l, ResidentActivity_NoActivity) ∧ hasActivity(?rol, ResidentActivity_NoActivity)

Pravilo koje postavlja predefinisanu aktivnost *ResidentActivity_OutOfHome*, koja označava da nema prisustva ni u jednoj od prostorija u kući, na trenutnu aktivnost stanara na toj lokaciji uključuje klase *Location, CurrentState, ResidentOnLocation* i svojstva *hasLocation, DetectPresence i hasActivity.* Ovo pravilo je predstavljeno konjunkcijom atoma koja predstavlja uslov da je broj

prostorija u kom nije detektovano prisustvo jednako ukupnom broju prostorija u kući, njegova SWRL reprezentacija je sledeća :

Location(?l) ∧ CurrentState(?cs) ∧ hasLocation(?cs, ?l) ∧ DetectPresence(?cs, false) ∧ sqwrl:makeSet(?sl, ?l) ∧ Location(?all) ∧ CurrentState(?cs1) ∧ hasLocation(?cs1, ?all) ∧ sqwrl:makeSet(?sall, ?all) ∧ sqwrl:size(?s1, ?sl) ∧ sqwrl:size(?s2, ?sall) ∧ swrlb:equal(?s1, ?s2) ∧ ResidentOnLocation(?rol) ∧ hasLocation(?rol, ?l) → hasActivity(?rol, ResidentActivity_OutOfHome) ∧ hasActivity(?l, ResidentActivity_OutOfHome)

Pravilo za uključivanje uređaja koji bi trebalo da smanji vrednost temperature u prostoriji uključuje klase *Location, CurrentState, ResidentActivity, ResidentOnLocation, ActiveDesiredState, AirTemperatureCondition*, *Actuator, Device, DeviceType* i svojstva *hasLocation, hasActivity, hasCondition, hasValue, hasTolerance, dependsOnActivity, hasState, hasDeviceType, hasActivityType, hasActuator*, i konjunkcija njegovih atoma predstavlja sledeći niz zahteva:

- bira se trenutna aktivnost stanara na lokaciji
- iz tekućeg stanja na lokaciji očitava se temperaturno stanje
- iz željenog stanja za trenutnu aktivnost očitava se temperaturno stanje
- upoređuju se temperaturna stanja tekućeg i željenog stanja i ukoliko je vrednost trenutne temperature veća od sume vrednosti temperature i temperaturne tolerancije željenog stanja znači da je u toj prostoriji potrebno uključiti uređaj za smanjenje temperature
- za uređaje koji ispunjavaju prethodni uslov biraju se njihovi aktuatori koji imaju status da su isključeni

Ovo pravilo ima sledeću SWRL reprezentaciju:

ResidentOnLocation(?r) ∧ ResidentActivity(?ra) ∧ hasActivity(?r, ?ra) ∧ Location(?l) ∧ hasActivity(?l, ?ra) ∧ CurrentState(?cs) ∧ hasLocation(?cs, ?l)∧ AirTemperatureCondition(?cat) ∧ hasCondition(?cs, ?cat) ∧ ActiveDesiredState(?ads) ∧ dependsOnActivity(?ads, ?ra) ∧ AirTemperatureCondition(?dsat) ∧ hasCondition(?ads, ?dsat) ∧ hasValue(?cat, ?catv) ∧ hasValue(?dsat, ?dsatv) ∧ hasTolerance(?dsat, ?dsatt) ∧ swrlb:add(?um, ?dsatv, ?dsatt) ∧ swrlb:greaterThan(?catv, ?um) ∧ Device(?d) ∧ hasLocation(?d, ?l) ∧ DeviceType(?dt) ∧ hasDeviceType(?d, ?dt) ∧ hasActivityType(?dt, Cooling) ∧ Actuator(?a) ∧ hasActuator(?d, ?a) ∧ hasState(?a, "off") → StateToBeSet(?a, "on") ∧ influenceAirTemperature(?a, Decrease)

Slično izgleda i pravilo za isključivanje uređaja koji je uključen kako bi smanjio vrednost temerature u prostoriji, u slučaju da je ta temperature dostignuta ili da se promenila vrednost željene temperature na višu vrednost:

ResidentOnLocation(?r) ∧ ResidentActivity(?ra) ∧ hasActivity(?r, ?ra) ∧ Location(?l) ∧ hasActivity(?l, ?ra) ∧ CurrentState(?cs) ∧ hasLocation(?cs, ?l)∧ AirTemperatureCondition(?cat) ∧ hasCondition(?cs, ?cat) ∧ ActiveDesiredState(?ads) ∧ dependsOnActivity(?ads, ?ra) ∧ AirTemperatureCondition(?dsat) ∧ hasCondition(?ads, ?dsat) ∧ hasValue(?cat, ?catv) ∧ hasValue(?dsat, ?dsatv) ∧ hasTolerance(?dsat, ?dsatt) ∧ swrlb:add(?um, ?dsatv, ?dsatt) ∧ swrlb:lessThanOrEqual(?catv, ?um) ∧ Device(?d)∧ hasLocation(?d, ?l) ∧ DeviceType(?dt) ∧ hasDeviceType(?d, ?dt) ∧ hasActivityType(?dt, Cooling) ∧ dependsOnOutOfDoorCondition(?dt, false) ∧ Actuator(?a) ∧ hasActuator(?d, ?a) ∧ hasState(?a, "on") → StateToBeSet(?a, "off") ∧ influenceAirTemperature(?a, NoInfluence).

I ostala pravila za upravljanje uređajima koji utiču na ambijent u kući su slična ovim pravilima, a značajniju razliku ima pravilo za isključivanje (uključivanje) *standBy* uređaja koje umesto uslova koji upoređuje vrednosti trenutnog i željenog stanja (temperature,vlažnosti,svetlosti) imaju samo uslov odabira željenog ponašanja *standBy* uređaja. Ovo pravilo predstavlja konjunkciju atoma koji uključuju klase *Location, ResidentActivity, ResidentOnLocation, ActiveDesiredState, Actuator, Device, DeviceType* i svojstva *StandByDevicesBehavior, hasActivity, hasLocation, StateToBeSet, hasTolerance, dependsOnActivity, hasState, hasDeviceType, hasActivityType, hasActuator*:

ResidentOnLocation(?r) ∧ ResidentActivity(?ra) ∧ hasActivity(?r, ?ra) ∧ Location(?l) ∧ hasActivity(?l, ?ra) ∧ ActiveDesiredState(?ads) ∧ dependsOnActivity(?ads, ?ra) ∧ StandByDevicesBehavior(?ads, "SwitchOff") ∧ Device(?d) ∧ hasLocation(?d, ?l) ∧ DeviceType(?dt) ∧ hasDeviceType(?d, OnOffDevice) ∧ Actuator(?a) ∧ hasActuator(?d, ?a) ∧ hasState(?a, "on") → StateToBeSet(?a, "off")

Obzirom na to da može postojati više uređaja koji mogu da utiču na isti parametar stanja ambijenta, uvedena su pravila koja definišu odabir uređaja kome je za rad potrebno najmanje električne energije kao npr. pravilo koje u skupu urađaja koji mogu da se uključe kako bu uticali na porast temperature bira urađaj koji ima najmanju snagu:

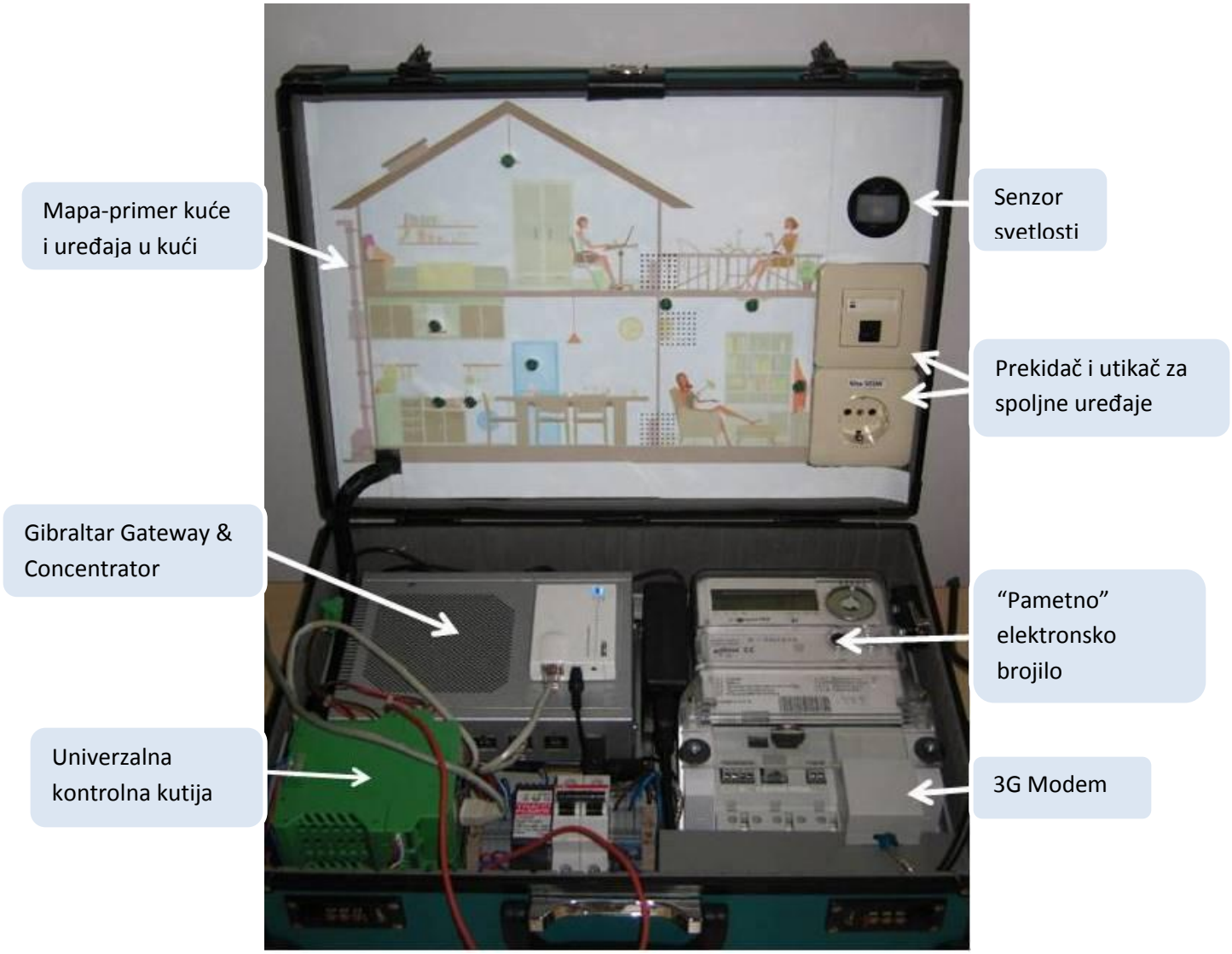
Device(?d) ∧ hasPower(?d, ?dp) ∧ Actuator(?a) ∧ hasActuator(?d, ?a) ∧ Location(?l) ∧ hasLocation(?d, ?l) ∧ influenceAirTemperature(?a, ?inf) ∧ hasName(?inf, "Insrease") ∧ hasState(?a, ?cst) ∧ StateToBeSet(?a, ?nst) ∧ swrlb:notEqual(?cst, ?nst) ∧ sqwrl:makeBag(?pb, ?dp) ∧ sqwrl:groupBy(?pb, ?l) ∧ sqwrl:min(?mp, ?pb) ∧ swrlb:equal(?mp, ?dp) → isAffected(?d, true).

3.4 Korisnički intefejs SESAME sistema

Predloženi koncept SESAME sistema je realizovan proširivom demonstrator-platformom nazvanom *SESAME Demostrator.* SESAME Demonstrator obezbeđuje proveru koncepta ovog inovativnog tehničkog rešenja tako što objedinjuje nekoliko relevantnih fizičkih i virtuelnih uređaja predstavljajući rezultat interakcije korisnika i sistema. Osnovni delovi demonstratora su:

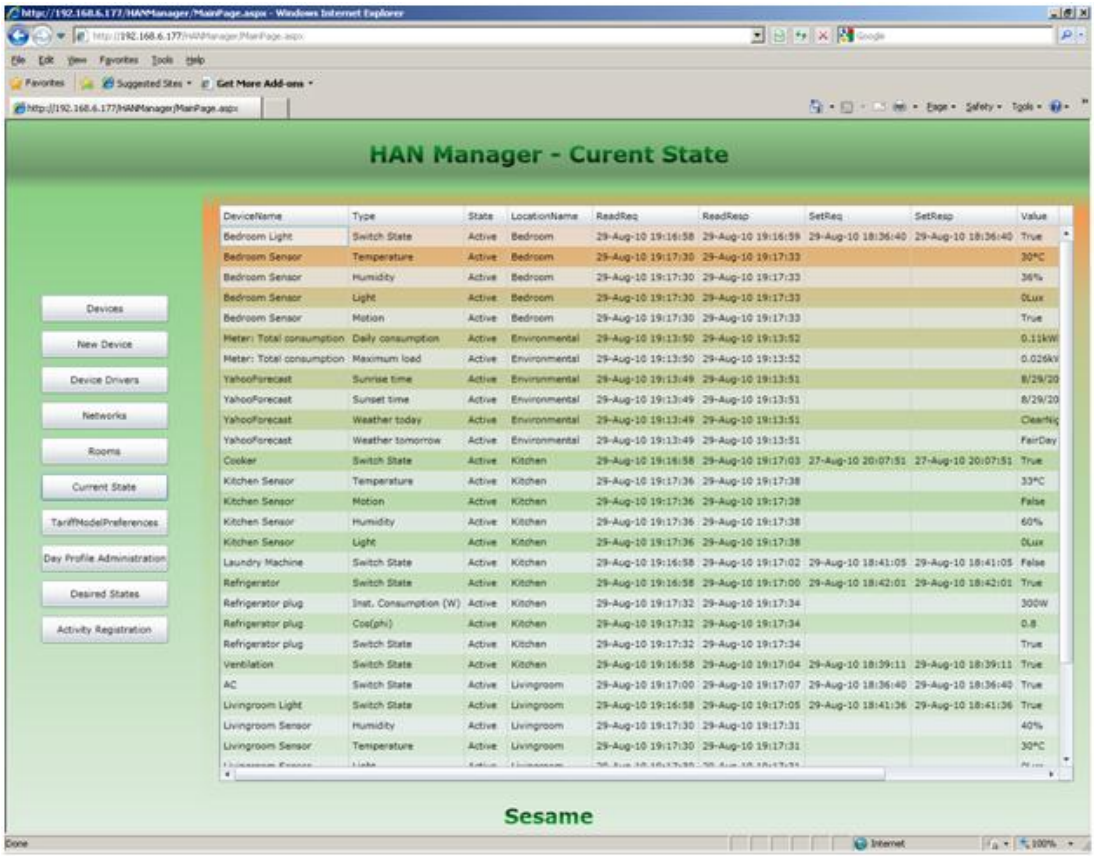
- *“Pametno” elektronsko brojilo*, pruža sve osnovne funkcionalnosti za merenje potrošnje energije.
- *Prekidač i utikač*, prihvataju sve ulazne podatke iz spoljnih uređaja kao što su lampe, veš mašine, itd.
- *Univerzalna kontrolna kutija*, čita i kontroliše sve senzore i utičnice. Trenutno su dostupni senzori za svetlost, vlažnost i temperaturu.
- *Gibraltar Gateway & Concenrator*, sadrži zaštitni zid (*engl.firewall*), skladište za ontologije i semantički rasuđivač.

Naredna slika prikazuje izgled SESAME Demontratora.



U SESAME sistemu je impementirano više funkcija za upravljanje i nadgledanje "pametne kuće". Namera je da se nadgledaju vrednosti senzora, potrošnja energije i rad uređaja. Obezbeđeno je i podešavanje željenog stanja ambijenta za izabrane lokacije i kreiranje planova rada uređaja. Demonstratorom se upravlja pomoću intuitivnog korisničkog interfejsa, za čiji izgled je kao smernica je uzet izgled savremenog ekrana osetljivog na dodir. U nastavku je dat okviran prikaz interfejsa SESAME sistema prema korisniku.

Za upravljanje i konfiguraciju sistema, krajnji korisnik može koristiti dve aplikacije: HAN Menadžer (*Home Appliances Network Manager*) i HAN Monitor (*Home Appliances Network Monitor*). Prvi je veb aplikacija sa ovlašćenim pristupom u kojem korisnik može administrirati i kontrolisati sve senzore i aktuatore u kući. Uz redovno očitavanje svih senzora u kući, sistem dobija sve potrebne informacije o trenutnom stanju u kući.



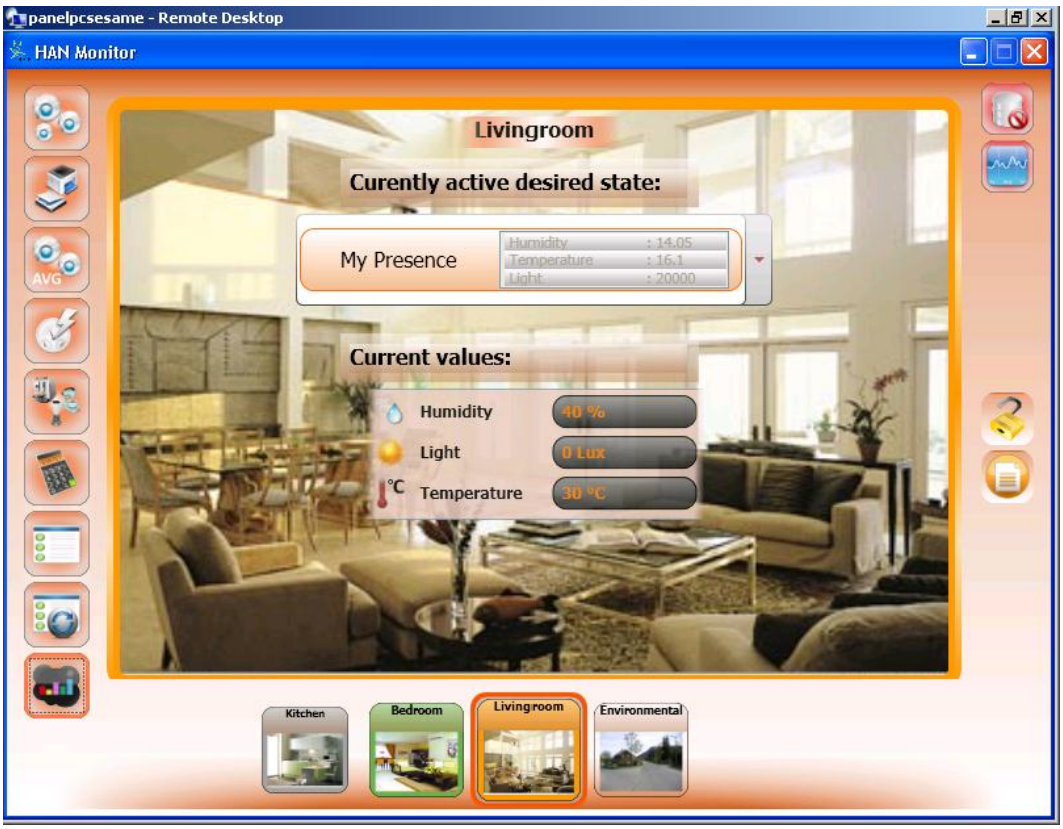
Korisnički interfejs za pregled pročitanih podataka sa svih senzora

HAN Menadžer aplikacija se koristi i za definisanje uslova koja uticati na rezonovanje o planiranju potrošnje i izboru optimalnih tarifnih modela. Korisnik može da definiše koliko su za njega važni cena, izbor distributera električne energije i izvora energije koju koristi.

Korinički interfejs za definisanje uslova koji utiču na planiranje potrošnje i izbor optimalnog tarifnog modela

Što se tiče ambijenta u kući, HAN Menadžer se koristi za definisanje željenih stanja koja mogu biti planirana kao na primer: "Dok se odmaram, želim da temperatura bude između 20° C i 24° C i svetla ugašena", ili odmah primenjena (*engl.on-demand*) izborom odgovarajuće akcije korišćenjem HAN Monitor aplikacije.

Korisnički interfejs za definisanje željenih stanja ambijenta



Korisnički interfejs za pregled trenutnog stanja ambijenta i eventualnu "on-demand" primenu željenog stanja

Korišćenjem HAN Monitor aplikacije može se pregledati trenutno stanje ambijenta u prostorijama kuće, trenutna potrošnja električne energije, kao i evidentirani problemi u čitanja podataka sa uređaja.

Korisnički interfejs za pregled vrednosti pročitanih sa senzora i pregled potrošnje električne energije za izabranu prostoriju

4 Zaključak

Imajući u vidu raznolikosti u kućnom okruženju, aparatima i korisnicima, sistemi za pametne kuće moraju biti konfigurabilni i prilagodljivi. Generičko predstavljanje znanja korišćenjem ontologije koristi se da bi olakšalo prilagodljivost odgovarajuće aplikacije u zavisnosti od profila korisnika i dostupnih uređaja. Iako se tehnonlogije za automatizaciju zgrada ubrzano razvijaju i stiču zrelost, i dalje postoji veliki potencijal za inovacije u oblasti semantičkih tehnologije i pristupu koji je potreban da bi sistem bio zaista fleksibilan i da bi reagovao na promene u konfiguraciji korisničkog okruženja, kao i na potrebe korisnika. To je glavna motivacija za rad i napredak tehnologija u pristupu koji ima projekat SESAME. SESAME sistem poboljšava koncept “pametne kuće“ primenom ontologija za modelovanje i servisno-orijentisane arhitekture, semantički preplićući automatizaciju uređaja u zgradi i infrastrukturu pametnih brojila. Takvo "semantičko lepljenje" nije trivijalan zadatak i zahteva razumevanje brojnih specifičnosti u oblasti energije i automatizacije. Sa takvim sistemom očekuje se veća ušteda energije i bolja pogodnost života nego kod trenutnih sistema za automatizaciju zgrada. SESAME semantički opisuje “pametnu kuću“ kroz modele uređaja, model za automatizaciju zadataka, sistem automatizacije pravila, model energetske politike i korisnička podešavanja za prilagođavanje korišćenja sistema na nivou pravila. U ovom radu je predstavljen formalni i proširiv kontekstni model zasnovan na OWL-u za predstavljanje i manipulaciju informacijama u inteligentnom okruženju. Ontologija uređaja, uz pravila rezonovanja, omogućava definisanje mera koje treba preduzeti u pravcu rada aktuatora i servisa dostupnih u okviru sistema “pametne kuće“. Korišćenjem SWRL pravila, ova ontologija predstavlja inovativni način korišćenja ontologije daleko od njene osnovne svrhe. U budućnosti je planirana dalja optimizacija postojećeg sistema i ontologije i implementacija novih funkcionalnosti.

Projekat SESAME je završen u septembru 2010-te godine sa demonstracijom rada sistema na konferenciji “I-Semantics“ održanoj u Gracu. Nakon završetka, usledila je priprema materijala za nastavak rada na ovom polju, na projektu pod imenom SESAME-S. U novom projektu je predviđeno da ontologija uređaja bude prilagođena, stabilizovana, publikovana i povezana sa drugim ontologijama na veb-u. Takođe, unutar sistema SESAME-S, biće obezbeđeni mehanizmi za učenje o ponašanju korisnika. Učenje će biti sprovedeno kroz intuitivne upitnike o navikama korisnika, odnosno na osnovu senzorima uočenih događaja u kući. Zaključci će biti iskorišćeni za automatizovano donošenje odluka o prilagođavanju uslova u kućnom okruženju.

Literatura

[1] Antoniou, G.; van Harmelen, F., *"A Semantic Web Primer 2nd ed."*, MIT Press, London, 2008.

[2] Anutariya, C.; Wuwongse, V.; Akama, K.; Wattanapailin, V., *"Semantic Web Modeling and Programming with XDD"*, Semantic Web Working Symposium, 2001.

[3] Berners-Lee, T.; Hendler, J.; Lassila, O., *"The Semantic Web"*, Scientific American, May 2001

[4] Burke, E.; Foxley, E., *"Logic and Its Applications"*, Upper Saddle River, N.J.: Prentice Hall, 1996.

[5] Carpenter, B., *"Architectural Principles of the Internet"*, RFC Editor United States, June 1996.

[6] Horrocks, I.; Patel-Schneider, F.P.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M., *"SWRL: A Semantic Web Rule Language, Combining OWL and RuleML"*, W3C Member Submission, 21 May, 2004

[7] Manola, F.; Miller, E., *"RDF Primer"*, W3C Recommendation, 2004. Dostupno na: http://www.w3.org/TR/rdf-primer/

[8] McGuinness, D.; Frank van Harmelen, *"OWL Web Ontology Language – Overview"*, W3C Recommendation, 2004. Dostupno na: http://www.w3.org/TR/owl-features/

[9] PROTÉGÉ. 2009. Dostupno na: http://protege.stanford.edu

[10] Prud'hommeaux, E.; Seaborne, A., *"SPARQL Query Language for RDF"*, W3C Recommendation, 2008. Dostupno na: http://www.w3.org/TR/rdf-sparql-query/

[11] Semantic Smart Metering: Enablers for Energy Efficiency (SESAME), 2009. Dostupno na: http://sesame.ftw.at

[12] Smith M.; Welty, C.; McGuinness, D., *"OWL Web Ontology Language – Guide"*, W3C Recommendation, 2004. Dostupno na: http://www.w3.org/TR/owl-guide/