SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1309

INTELIGENTNO PRETRAŽIVANJE INFORMACIJSKOG PROSTORA INFORMACIJSKE INFRASTRUKTURE

Marko Čupić

Zagreb, rujan 2002.

Uz najiskrenije zahvale mentoru Vladi Glaviniću

SADRŽAJ

1 UVOD

Količina informacija dostupna preko Interneta danas je toliko velika, da je gotovo svaki pokušaj pronalaženja željene informacije istog trena ugušen ogromnom količinom dodatnih informacija koje vrlo često nisu čak niti tematski vezane sa traženim podacima. Pretraživači po ključnim riječima koji se danas koriste vrlo često vračaju neželjene informacije upravo zbog činjenice da samo ključne riječi uvijek vade informacije iz konteksta i nakon toga više nije moguće precizno zadati što nas zanima.

U ovom radu naglasak je bačen na mehanizam opisivanja dokumenata posebnim načinom koji će očuvati i informaciju i kontekst, a dodatno će omogućavati definiranje odnosa među pojedinim konceptima i na taj način dozvoliti traženje dokumenata o sličnim konceptima, proširenjima koncepta, specijalizacijama koncepta i sl.

Isto tako, osim sinkronih pretraživanja kada korisnik zada upit i odmah dobije povratnu informaciju u radu se razmatra i sustav koji bi omogućavao rad sa asinkronim upitima – korisnik zada upit na kraju radnog vremena, ugasi računalo, vrati se slijedeći dan, upali računalo i dobije rezultate koji su mogli biti skupljani cijelu noć.

Također se predlaže platforma koja bi bila pogodna za obavljanje ovakvih pretraživanja te se govori nešto detaljnije kako bi takav sustav mogao izgledati.

U poglavlju 2 ukratko se daje pregled metoda koje se koriste za pretraživanje dokumenata i navode se njihove karakteristike. Poglavlje 3 objašnjava teorijske postavke pretraživanja pomoću agenata temeljenog na ontologijama. Definiraju se pojmovi "agent" i "ontologija", te se ukratko objašnjavaju tri najčešća formata dokumenata na Internetu: HTML, XML i XHTML. Poglavlje 3 opisuje predloženi sustav za pretraživanje. Definiraju se agentske okoline, iznosi se prijedlog kako bi agent trebao izgledati, pobliže se definira način na koji će se opisivati dokumenti. Dat je prikaz ontologije feronto1 koja je razvijena u sklopu ovog rada. Objašnjen je način na koji korisnik može zadavati upite i prikazan je rad sustava koji vrši obradu upita i zaključivanje. Na kraju je prikazano kako je cijeli sustav implementiran i od kojih komponenti se sastoji. Poglavlje 6 sadrži zaključak, a u prilogu se nalazi konkretan primjer opisa dokumenata feronto1 ontologijom, kao i detalji protokola za komunikaciju agentske okoline s "Agent Server Directory" komponentom.

2 METODE PRETRAŽIVANJA DOKUMENATA

U današnje doba opće je priznata činjenica da količina informacija koja je dostupna korisniku kroz razne mreže nadilazi mogućnosti korisnika da u njima pronađe one informacije koje su mu doista bitne. U pravilu korisnici imaju na raspolaganju tri metode pretrage za informacijama: (1) browsing (prateći hipertekstualne veze koje se čine zanimljivima), (2) slanjem upita pretraživačkim strojevima (engl. search engines) poput Altaviste, ili (3) praćenjem postojećih kategorija u pretraživakim strojevima (poput Yahoo-a i Lycos-a). Učestala je pojava da korisnici provode sate i sate proučavajući nepotrebne i njima nezanimljive podatke, u pokušaju da pronađu ono što ih doista zanima.

Danas se sve više i više informacija stavlja on-line, i dolazi do pojave poznate pod nazivom "paradoks produktivnosti" – pojava da se uslijed preobilne količine informacija dostupnih korisnicima zapravo postiže efekt nemogućnosti pronalaska željenih informacija. Zbog toga je korisnicima potrebno osigurati način da iz brzo i lagano dođu do informacija koje ih doista zanimaju.

Na rješenju ovog problema radi se već niz godina, i postoji nekoliko različitih pristupa. Najjednostavniji pristup bazira se na indeksiranju ključnih riječi svih dokumenata, te uporabe tezarusa za otkrivanje sinonima. Tehnike za dohvat informacija (IR, engl. Information Retrieval) razvijene iz modela frekvencije riječi (engl. word frequency model) [1] omogućavaju dohvat informacija zasnovan na modelu dohvata booleovom logikom (engl. Boolean retrieval model), odnosno proširenje ovog modela neizrazitom logikom [2]. Naprednija metoda koja se danas koristi jest pristup prostora vektora (engl. Vector space approach) [3], kao i neke sofisticiranije metode poput latentnog semantičkog indeksiranja [4], metode bazirane na Bayesovim mrežama poput Inference Net system autora Turtle i Croft[5], te neuronske mreže [6][7].

Sve metode koje rade direktno sa dokumentima obično imaju nekoliko faza u pretprocesiranju dokumenata koje uključuju određivanje tokena (leksičkom analizom), određivanje riječi, rad sa punktuacijom, proširivanje akronima, izolacija rečenica, uklanjanje zaustavnih riječi (engl. stop-words) te uklanjanje prefiksa i sufiksa riječi kako bi se došlo do korijena riječi.

Drugi pristup kako ispravno shvatiti o čemu se u dokumentu radi (bez da se analizira sam dokument) jest analiza referenci na taj dokument. Naime, ideja je slijedeća: kada se autor u dokumentu X referencira na dokument Y, tada se u okolini reference u dokumentu X vrlo vjerojatno nalazi opis bitnoga u dokumentu Y. Ovaj pristup opisan je u Guiding People to Information: Providing an Interface to a Digital Library Using Reference as a Basis for Indexing [12], gdje je implementiran sustav ROSETTA.

Jedno od rješenja koje bi korisnika djelomično oslobodilo prekapanja po informacijama jest uporaba inteligentnih agenata, koji bi taj posao obavljali za korisnika. Jedan takav primjer jest Intelligent Agent for High-Precision Text

Filtering [8], koji radi sa člancima novosti (engl. news articles), ali je primjenjiv i na ostale vrste teksta. Sustav (agent) korisnika prati kroz njegov korisnički profil, koji je predstavljen asocijativnom mrežom čiji su čvorovi izrazi ili riječi interesantne korisniku. Čvorovi su povezani težinskim vezama u fraze. Za prikaz dokumenata također se koriste asocijativne mreže (pri čemu su veze bez težina, a čvorovi imaju aktivacijski nivo početno postavljen na nulu). Konačno, radi se usporedba sličnosti korisničkog profila i članka, i time se određuje relevantnost članka za korisnika.

Zanimljiv primjer predstavlja i agent WebMate opisan u WebMate: A Personal Agent for Browsing and Searching [13]. Agent obradu dokumenata zasniva na modelu prostora vektora, odnosno TF-IDF metodu [3] sa višestrukim vektorskim prikazom. Agent korisnikove interese pamti preko višestrukih korisničkih profila, pri čemu je svaki profil zapravo jedan vektor. Dokument se također pretvara u vektor i zatim se radi usporedba sličnosti sa zapamćenim korisničkim profilima. WebMate profile korisnika uči inkrementalno i kontinuirano. Agent je ostvaren kao proxy (dakle, agent se nalazi između korisnikovog preglednika i Interneta). Stranice koje su korisniku zanimljive, agent analizira klasičnim metodama, uz primjenu heuristike pri čemu se posebno pazi na riječi unutar <TITLE>, <H1>, <H2> i <H3> oznaka, i pridaje im se veće značenje, jer se očekuje da su te riječi bitne za sadržaj dokumenta. Nakon što agent nauči profile korisnika, korisnik može zadati zadatak praćenja određenih sajtova (npr. LAN time news). Agent tada kontinuirano, npr. svaku noć, dohvati sve stranice koje prati i pogleda ima li stranica koje bi mogle biti zanimljive korisniku, te takve stranice slaže u kolekciju i prezentira korisniku kada se on pojavi, npr. slijedeće jutro.

Međutim, ovakav pristup i dalje je baziran na obradi cijelih dokumenata od strane sustava. Kako je obrada prirodnog jezika još uvijek veliki problem, ovakva rješenja daju ograničene rezultate. Zbog toga je potreban drugačiji pristup. Jedno od rješenja je uvođenje meta-informacija, odnosno informacija o onome o čemu je određeni dokument. Ideja ovakvog pristupa je otkloniti potrebu za čitanjem i razumijevanjem dokumenta od strane sustava koji radi za korisnika (jer ionako sustavi u tome nisu dobri). Umjesto toga, na određeni način potrebno je napisati ukratko o čemu se radi u pojedinom dokumentu, i to jezikom (ontologijom) koji je biti razumljiv pretraživačkim sustavima.

Prvi korak u ovom smjeru jest definiranje ontologije. Naime, formalno predstavljanje znanja bazirano je na konceptualizaciji – objekata, koncepata i ostalih entiteta za koje se podrazumijeva da postoje u području od interesa, te veza koje vrijede među njima [22]. Konceptualizacija jest apstraktni, pojednostavljeni pogled na svijet koji želimo predstaviti iz nekog razloga. Svaka baza znanja, sustav baziran na znanju ili agent koji radi na razini znanja zasnovan je na određenoj konceptualizaciji, eksplicitno ili implicitno. Ontologija je jednostavno eksplicitna specifikacija konceptualizacije [21]. Prilikom definiranja ontologija potrebno je voditi računa o nekoliko stvari:

1. *jasnoća* – potrebno je jasno definirati značenje u kojem se koriste određeni termini.
2. *koherentnost* – aksiomi trebaju biti logički konzistentni.

3. *proširivost* – ontologija treba biti dizajnirana uporabom dijeljenog rječnika, i pružiti konceptualni temelj za široko područje primjene. Isto tako, ontologija mora dopuštati specijalizacije i proširenja.
4. *minimalni "encoding bias"* – ontologija treba biti specificirana na razini znanja, neovisno o određenom kodiranju na simboličkoj razini.
5. *minimalni zahtjevi ontologije* – ontologija treba postavljati što manji broj pretpostavki o svijetu koji modelira.

Za potrebe opisivanja meta-informacija razvijeno je nekoliko inicijativa, međutim, uporaba RDF-a [9] sve je učestalija. RDF (engl. Resource Description Framework) je baziran na XMLu (engl. Extensible Markup Language) [10], čime se osigurava standardiziran način pohrane RDF dokumenta, kao i njihova obrada. RDF omogućava jednostavan način za definiranje izjava (npr. "tvorac dokumenta xxx je Pero"). Osnova svega (prema RDF-u) su resursi (a resurs može biti sve; dokument, osoba ili čak i sama izjava) – sve izjave definiraju se o resursima. Svaki resurs ima svoj URI (engl. Unique Resource Identifier) koji može, a ne mora pokazivati na neki fizički dokument.

Međutim, iako RDF nudi mehanizam za pohranu znanja, problem koji se javlja jest nestandardiziranost tumačenja u RDF-u napisanih izjava. Npr. jedna osoba može zapisati izjavu "autor" dokumenta X je Pero (koristeći ontologiju A), druga osoba može napisati izjavu "tvorac" dokumenta X je Pero (koristeći ontologiju B), dok treća osoba može napisati izjavu "author" dokumenta X je Pero (koristeći ontologiju C).

Tablica 2-1 Različiti načini zapisivanja iste izjave

`<rdf:Description about="X">`
`  <A:autor>Pero</A:autor>`
`</rdf:Description>`

`<rdf:Description about="X">`
`  <B:tvorac>Pero</B:autor>`
`</rdf:Description>`

`<rdf:Description about="X">`
`  <C:author>Pero</C:author>`
`</rdf:Description>`

Očito je da svi misle na isto – problem je u tome što je svaka osoba tvrdnju izrekla na njoj jasan način. Ovo dovodi do problema u tumačenju zapisanih meta-informacija. Naime, da bi agent mogao ispravno pretraživati dokumente za odgovarajućim informacijama, on mora biti u stanju razumjeti (sebi protumačiti) meta-informacije o tom dokumentu. Dolazi se do toga da smo problem razumijevanja prirodnog jezika (dokumenata) prebacili u problem razumijevanja meta-informacija. Međutim, ovaj drugi je nešto lakše riješiti. Postoje tri pravca u rješavanju problema:

- Uvođenje standardiziranog jezika (ontologije)
- Uvođenje prevodilačkih servisa
- Učenje nepoznatih koncepata

Primjer uvođenja standardiziranih ontologija je Dublin Core Metadata Initiative (DCMI) [23], koja definira ontologiju za opis dokumenata, članaka i slično. Pri tome se jasno definira "jezik" kojim se treba koristiti i način kako zapisivati izjave. DCMI definira kao način zapisivanja uporabu XML-a, pri čemu je svaki DCMI dokument ujedno i ispravan RDF dokument. Isto tako, DCMI definira 15 standardnih termina koji se mogu koristiti za opisivanje dokumenata (*Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage, Rights*). Tako se definira da izjavu da je Pero tvorac dokumenta X treba reći na slijedeći način:

Tablica 2-2 Uporaba Doublin Core ontologije

```
<rdf:Description about="X">
  <dc:Creator>Pero</dc:Creator>
</rdf:Description>
```

Ovo je jedan od načina na koji se može osigurati da svaki agent razumije što je zapravo rečeno u meta-podacima. No da bi ovakvo rješenje zaživjelo, svi se moraju pridržavati ovih pravila i koristiti upravo DCMI, što vrlo često nije slučaj. Termini koji se koriste u pojedinim izjavama mogu biti proizvoljni, ili pak iz kontroliranog ograničenog rječnika. Tako se za popunjavanje DCMI Subject elementa preporuča uporaba riječi iz kontroliranog rječnika. Veze između pojedinih koncepata mogu se izraziti semantičkim vezama tezarusu. Tri glavne veze su ekvivalencija (ekvivalentni izrazi), hijerarhija (uži, širi izrazi) te asocijacija (slabije povezani izrazi). Primjeri poznatijih tezarusa su Medical Subject Headings [15] te Art and Architecture Thesaurus [16].

Međutim, za semantički opis dokumenata tezarus sadrži samo dio potrebnog znanja, te je potrebno izgraditi odgovarajuću ontologiju. Primjer izrade ontologije za opis antiknog namještaja bazirajući se na Art and Architecture Thesaurus opisan je u From Thesaurus to Ontology [18].

Razvoj vlastitih ontologija oduvijek je bio puno fleksibilnije rješenje od standardiziranih ontologija. Naime, ono je omogućavalo da se sve izjave kažu na način kako to samom korisniku odgovara, a ujedno je omogućavalo i da se kaže sve što se je željelo reći. Naime, DCMI definira samo 15 pojmova koji se mogu vezati uz dokument; bilo što drugo nije moguće reći uporabom DCMI-a. Problem ovog pristupa je taj što svi agenti tada nisu u stanju razumjeti što pojedina izjava znači jer ne poznaje korištenu ontologiju. Problem se može riješiti uporabom prevodilačkih servisa koji bi meta-podatke napisane uporabom ontologije A preveli u meta-podatke napisane ontologijom B koju razumije agent, bilo direktno ($A \rightarrow B$), bilo postupnim prevođenjem ($A \rightarrow Q1 \rightarrow Q2 \rightarrow ... \rightarrow Qn \rightarrow B$). Neki od projekata koji se bave problematikom prevođenja ontologija, odnosno preslikavanjem između ontologija su Stanford Scalable Knowledge Composition (SKC) [19] i Bremer Semantic Translation [20]. Projekt Stanford Scalable Knowledge Composition pokušava razviti algebru nad ontologijama koje sadrže terminologiju iz različitih, tipično autonomnih domena. Presjek će u tom slučaju biti najbitnija operacija, budući da identificira zajedničke točke iz ontologija, odnosno termine u kojima se događa povezivanje među domenama. Bremer Semantic Translation zagovara drugi pristup u

rješavanju istog problema. Polazeći od razmatranja kako uvođenje filtera za preslikavanje svake ontologije u svaku drugu ontologiju raste izuzetno nelinearno, zagovara se uvođenje inteligentnih brokera. Polazi se od pretpostavke da postoji više izvora podataka, i da je svakom izvoru podataka pridružena njegova ontologija. Svi izvori registrirani su kod brokera. Kada korisnik postavi zahtjev za dohvat određenih informacija, broker koristeći meta-podatke pronalazi onaj izvor koji može odgovoriti na postavljeni upit. Više izvora može se povezivati tako da se automatski pronalaze podudaranja koncepata i termina iz jedne ontologije sa konceptima i terminima iz druge ontologije, uporabom automatskog sustava zaključivanja.

Treći pristup rješavanju problema nerazumijevanja ontologije je postupak učenja. U Agents Teaching Agents to Share Meaning [11] prikazan je postupak kojim se dva agenta (od kojih svaki razumije različite ontologije) mogu postupkom dogovaranja usuglasiti o značenju izjava napisanih u različitim ontologijama. U članku je prikazano:

1. Kako agenti mogu odrediti da li znaju iste semantičke koncepte?
2. Kako agenti mogu odrediti da li njihovi različiti semantički koncepti zapravo imaju isto značenje?
3. Kako agenti mogu poboljšati svoju interpretaciju semantičkih objekata rekurzivnim učenjem nedostajućih pravila?

Npr. za lociranje sličnih semantičkih koncepata predlaže se slijedeći pristup.

- Agent A šalje agentu B naziv semantičkog koncepta (u ontologiji agenta A) i niz pokazivača na primjere tog koncepta.
- Agent B dobivene primjere interpretira i pokušava pronaći semantički koncept kojim on opisuje te primjere.
- Agent B može odgovoriti agentu A na nekoliko načina: a) "da, znam taj koncept", b) "možda znam taj koncept" i c) "ne znam taj koncept". Ako B zna ili možda zna taj koncept, on vraća naziv pod kojim on zna koncept, i niz pokazivača na primjere tog koncepta.
- Agent A prima odgovor i provjerava da li agent B doista zna odgovarajući koncept provjerom nad vraćenim primjerima.
- Ako agent A uspije provjeriti da agent B zna koncept, tada u svoju bazu znanja dodaje naziv pod kojim agent B zna odgovarajući koncept, odnosno izjavu tipa "Agent B zna koncept X pod nazivom Y". Uz ovo znanje agent A sada može komunicirati sa agentom B o traženom konceptu.

Vrlo često znanje koje jedan agent uspije pribaviti nije kompletno, pa se prilikom zaključivanja agent mora služiti pretpostavkama. Jedan interesantan način kako se niz agenata koji istražuju određeni prostor hipoteza mogu koordinirati prikazan je u Abductive Coordination for Logic Agents [14]. Prikazana je slijedeća ideja. Kada agent pokušava abducirati pretpostavku h, od svih preostalih agenata zahtjeva provjeru konzistentnosti hipoteze h. Ukoliko od svih agenata dobije potvrdu konzistentnosti hipoteze, h se ugrađuje u bazu znanja agenta.

Jedno od mogućih rješenja u postizanju semantičkog weba prikazuje SEAL — A Framework for Developing SEmantic PortALs [17]. Opisan je Web portal AIFB zasnovan na semantičkim tehnologijama. Pri tome se definiraju tri tipa korisnika portala – agenti, korisnici iz zajednice (engl. community users) te općenito korisnici. Korisnici iz zajednice mogu dodavati novo znanje, dok općenito korisnici i agenti to ne mogu. Sustav ima RDF generator koji za potrebe agenata određeno znanje prezentira u obliku RDF-a. Isto tako, sustav omogućava da se stranice, osim praćenjem hipertekstualnih veza obilaze i preko semantičkih veza (dakle, semantički slični dokumenti).

3 PRETRAŽIVANJE UPORABOM AGENATA TEMELJENO NA ONTOLOGIJAMA

Uporaba agentskih sustava općenito danas je područje koje se intenzivno istražuje. Ideja agenata u bitnome se razlikuje od svih prethodnih pokušaja distribuiranja aplikacija. Naime, koncept agenata omogućava da se na udaljeno računo pošalju ne samo parametri funkcije koju treba obaviti (poput RMI-ja, odnosno RPC-a), već kompletno tijelo funkcije zajedno sa svim parametrima. Ovo znači da se na udaljenom računalu može obavljati širok spektar zadataka, a ne samo oni zadaci koji su unaprijed instalirani. Zbog toga se agentski sustavi mogu koristiti za širok niz problema, a naročito su pogodni za područja koja se intenzivno mijenjaju, poput umjetne inteligencije, gdje nije unaprijed moguće instalirati sve što će biti potrebno (upravo zbog čestih izmjena).

Agentski sustavi mogu vrlo dobro poslužiti kao platforma za razvoj pretraživačkih usluga. Kako je osnova svakog agentskog sustava upravo agent, u nastavku ćemo se pobliže upoznati sa konceptom agenta kao i nekim osnovnim podjelama agenata ().

Kako bismo agentski sustav mogli iskoristiti za pretraživanje dokumenata, a naročito za inteligentno pretraživanje u kojem se obavlja i proces zaključivanja, postavljaju se dva zahtjeva:

1. agent mora razumjeti što se od njega očekuje da pronađe
2. agent mora razumjeti o čemu govore pojedini dokumenti i zaključiti da li se to poklapa sa zahtjevima tražitelja ili ne

Razvoj umjetne inteligencije danas napreduje velikom brzinom, međutim konačni cilj u kojem će računalo "razumjeti" čovjeka u punom smislu riječi "razumjeti" još je vrlo daleko. Zbog toga se može napraviti određeni kompromis: dokumenti se mogu prilagođavati tako da budu razumljivi i čovjeku i računalu. Da bi dokument bio razumljiv čovjeku, ne treba se raditi ništa posebno – svaki pisani dokument čovjeku će biti razumljiv. No da bismo isti dokument učinili razumljivim i agentima, može se sadržaj dokumenta opisati na poseban način koji će biti razumljiv i agentima. Metodologija za opisivanje dokumenata na takav način jest uporaba ontologija, o čemu će također biti riječi u nastavku.

Posljednji dio ovog poglavlja biti će posvećen pregledu formata dokumenata koji se danas najčešće mogu pronaći na Internetu. U takvim formatima ujedno se nalazi i najveća količina informacija pa bi razumjevanje sadržaja takvih dokumenata moglo uvelike poboljšati pronalaženje traženih dokumenata i informacija. Međutim, pokazati će se da neki od formata predstavljaju vrlo velike prepreke pokušajima da se izrade mali i jednostavni agenti koji bi mogli na ispravan način protumačiti njihov sadržaj. Upravo zbog toga ideja da se dokumenti dodatno opisuju određenim ontlogijama dobiva na svom značaju i može bitno popraviti trenutno stanje u izradi pretraživačkih usluga.

3.1. AGENTI

Ideja o agentima došla je od John McCarthy-a sredinom pedesetih. Izraz "agent" je stvorio Oliver G. Selfridge nekoliko godina kasnije, a izveden je iz Latinskog glagola *agere*: voziti, voditi, djelovati, činiti. Danas još ne postoji jednoznačna definicija pojma agent, međutim, mogli bismo se složiti da:

> *"Pojam "agent" opisuje sustav koji, kada mu se zada cilj, može obaviti sve detalje vezane uz prikladne računalne operacije, te može pitati za, i prihvatiti savjet, ponuđen u čovjeku razumljivom jeziku, ukoliko zaglavi. Agent bi bio softverski robot koji bi živio i obavljao svoj posao unutar svijeta računala."*

U današnje doba dolazi i do učestale zlouporaba termina agent, što je i razumljivo zbog činjenice da je ovo još dosta neistraženo područje, a kako se uz agente obično veže pojam inteligencije, svi žele svojim proizvodima nadjenuti ovaj naziv (pa tako imamo i Microsoftove agente – dosadne spajalice koje tu i tamo uspiju pogoditi u čemu je problem).

Slika 3-1 Pomoćnik iz Microsoft Office paketa

Postoji mnoštvo pokušaja da se definira agent, a kada bismo proveli jednostavnu anketu na ovu temu, odgovori bi vjerojatno bili:

- Programi koje se može isprogramirati unaprijed da obave neku operaciju u zadano vrijeme na udaljenom računalu (scheduling)
- Programi koji obavljaju low-level zadatke a može ih se programirati u high-level jezicima ili skriptama
- Programi koji enkapsuliraju ili skrivaju razlike između različitih izvora podataka ili računalnih servisa
- Programi koji preuzimaju ulogu "inteligentnih pomagača"
- Programi koji mogu "svojevoljno" prelaziti iz jednog računala u drugo
- Programi koji se korisnicima predstavljaju kao likovi koji ulijevaju povjerenje
- Programi koji "govore" komunikacijski jezik agenata (agent communication language)
- Programi koji se korisniku predstavljaju osjećajima i "mentalnim stanjima"

Međutim, sa ovakvim definicijama treba biti dosta oprezan jer to mogu doista biti svojstva agenata, ali i ne moraju; naime:

- Primjer programa koji u zadano vrijeme nešto može obaviti jest cron na unixu, Windowsi imaju Scheduled Tasks u koji se mogu dodavati zadaci i sl.; pa ipak ih ne zovemo agentima

- C programski jezik je programski jezik visoke razine koji zatim izvodi low-level zadatke – ne zovemo ga zato agentom
- Java sučelja (interface) i abstraktne klase enkapsuliraju / skrivaju razlike između različitih izvora podataka ili računalnih servisa – ne zovemo ih zato agentom
- Programe koji mogu "svojevoljno" prelaziti iz jednog računala u drugo obično nazivamo virusima
- I sl.

3.1.1 Definiranje pojma "agent"

Postoje dva uobičajena načina na koja se obično definira pojam agent. To su:

1. Svojstvima koje im osobe pripisuju.
2. Opisom atributa koje agenti posjeduju jer su tako dizajnirani.

3.1.1.1 Definicija agenata svojstvima koja im se pripisuju

Definiciju agenta koja bi vrijedila "jednom-za-svagda" vrlo je teško dati, jer "inteligentni agent" jednoj osobi je "pametni objekt" drugoj osobi a već sutra je "glupi program" svima. No ipak, uočene neke "obiteljske sličnosti". American Heritage Dictionary definira agenta kao:

"onaj koji može djelovati ili ima snagu ili autoritet da djeluje ... ili predstavlja nekoga" ili pak *"sredstva kojima se nešto može obaviti ili prouzrokovati; instrument."*

Međutim, gotovo sve se može opisati kao agent. U nastavku je naveden jednostavan primjer.

"*Posve je u redu tretirati prekidač za svjetlo kao (čak i vrlo kooperativnog) agenta koji ima mogućnost provođenja električne energije kada god to želi, te koji uvijek provodi struju kada vjeruje da mi to želimo; "prebacivanje" prekidačem jednostavno je način na koji mi njemu prenosimo naše želje…* "

3.1.1.2 Definicija agenata atributima koje posjeduju zbog dizajna

Prema atributima koje agent posjeduje zbog svog dizajna, možemo reći da je agent:

"*Softverski entitet koji funkcionira kontinuirano i autonomno u specifičnom okruženju, često nastanjeno drugim agentima i procesima.*"

Želimo da agenti:

- obavljaju zadatke na fleksibilan i inteligentan način koji odgovara na promjene u njihovoj okolini bez potrebe za stalnim ljudskim nadzorom i intervencijom.
- funkcioniraju kontinuirano kroz dugačak period vremena i da tijekom tog vremena uče iz vlastitog iskustva

- koji "prebivaju" u okruženju u kojem koegzistiraju i drugi agenti budu sposobni komunicirati i surađivati sa njima, te da se mogu seliti sa jednog mjesta na drugo

Termin "softverski agent" zapravo je ovojnica koja obuhvaća mnoge specifičnije tipove agenata. Svaki agent u manjoj ili većoj mjeri posjeduje svojstva nabrojana od Etzioni i Weld (1995) i Franklin i Graesser (1996):

- **reaktivnost** – svojstvo da selektivno osjeća i djeluje
- **autonomnost** – usmjerenost na cilj, proaktivno i samopokretano ponašanje
- **kolaborativno ponašanje** – može raditi/surađivati sa ostalim agentima u svrhu postizanja zajedničkog cilja
- **sposobnost "Knowledge-level" komunikacije** – sposobnost komunikacije sa osobama i ostalim agentima jezikom koji je sličniji ljudskom, a ne program-program protokolima na simboličkoj razini (symbolic-level communication)
- **sposobnost zaključivanja** – može djelovati pri apstraktnim specifikacijama zadataka koristeći prethodno znanje o općenitim ciljevima i preferiranim metodama da bi se postigla fleksibilnost; iz danih informacija izvodi nove, može imati eksplicitan model sebe, korisnika, situacije i/ili drugih agenata
- **vremenski kontinuitet** – otpornost identiteta i stanja kroz duži vremenski period
- **osobnost** – sposobnost manifestiranja karakteristika poput emocija i sl.
- **adaptivnost** – sposobnost da uči i da se poboljšava kroz iskustvo
- **mobilnost** – sposobnost svojevoljnog "seljenja" od jednog domaćina na drugog

3.1.2 Koncept direktne manipulacije u odnosu na koncept "agenta"

Shneiderman (zastupnik direktne manipulacije) zabrinut je nad konceptima inteligentnih sučelja općenito:

"*Prvo, ovakve klasifikacije ograničavaju maštu. Trebali bismo imati daleko veće ambicije od izrade računala koje se ponaša kao inteligentni "butler" ili neki drugi ljudski agent…*

Drugo, kvalitete predvidivosti i kontrole su poželjne. Ako su strojevi inteligentni i adaptivni, mogli bi imati manje ovih kvaliteta…

Treće, zabrinut sam da ako dizajneri budu uspješni u uvjeravanju korisnika da su računala inteligentna, korisnici će imati smanjen osjećaj odgovornosti za pogreške…

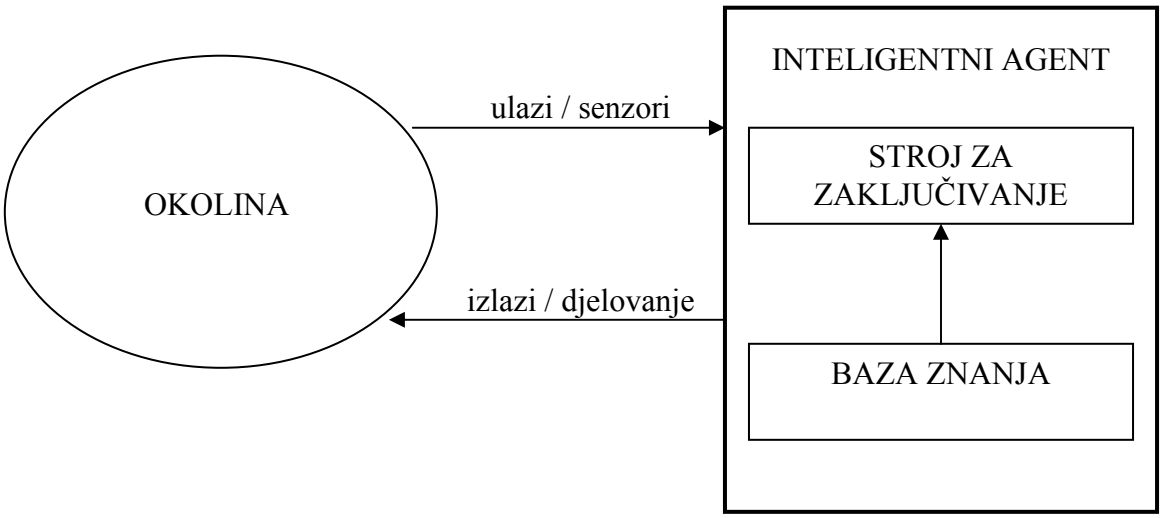
I konačno, … strojevi nisu ljudi… i ako pobrkate način na koji tretirate strojeve sa načinom na koji tretirate ljude… mogli biste završiti tretirajući ljude kao strojeve. "

U prilog činjenici da još ne znamo kako definirati agenta idu i slijedeća pitanja:

- Da li je jezični prevodioc (engl. kompajler) agent?
- A optimirajući jezični prevodioc (engl. kompajler)?
- Da li je upit u bazu podataka agent?
- Da li je print monitor agent?
- Da li je e-mail dostavljen agentom?
- Da li je VCR raspoređivač zadataka (engl. scheduler) agent?

3.1.3 Inteligentni agent (Intelligent agent)

Inteligentni agent je sustav baziran na znanju koji osjeća svoju okolinu (koja može biti fizički svijet, korisnik preko grafičkog korisničkog sučelja, kolekcija drugih agenata, Internet ili neko drugo kompleksno okruženje); razmišlja da bi interpretiralo osjete, izvodi zaključke, rješava probleme i određuje akcije; djeluje u tom okruženju i shvaća ciljeve i zadatke za koje je stvoren. Agent međudjeluje sa čovjekom ili nekim drugim agentima kroz neku vrstu agentskog komunikacijskog jezika i ne mora slijepo izvršavati naredbe, već ima sposobnost da modificira zahtjeve, traži pojašnjenja ili čak odbije zadovoljiti neki zahtjev. Može prihvatiti zahtjeve visoke razine koji govore ŠTO korisnik želi, i može odlučiti kako da zadovolji svaki od zahtjeva sa nekim stupnjem slobode ili autonomije, pokazuje ponašanje usmjereno ka cilju i dinamički odabire koje akcije treba poduzeti, i u kojem redoslijedu. Može surađivati sa korisnicima kako bi poboljšao izvršavanje zadataka i može izvoditi zadatke umjesto korisnika koristeći neki oblik reprezentacije znanja o korisnikovim ciljevima i željama. Može motriti događaje i procedure za korisnika, može savjetovati korisnika kako da obavi zadatak, može trenirati i učiti korisnika, ili može pomoći da različiti korisnici surađuju.



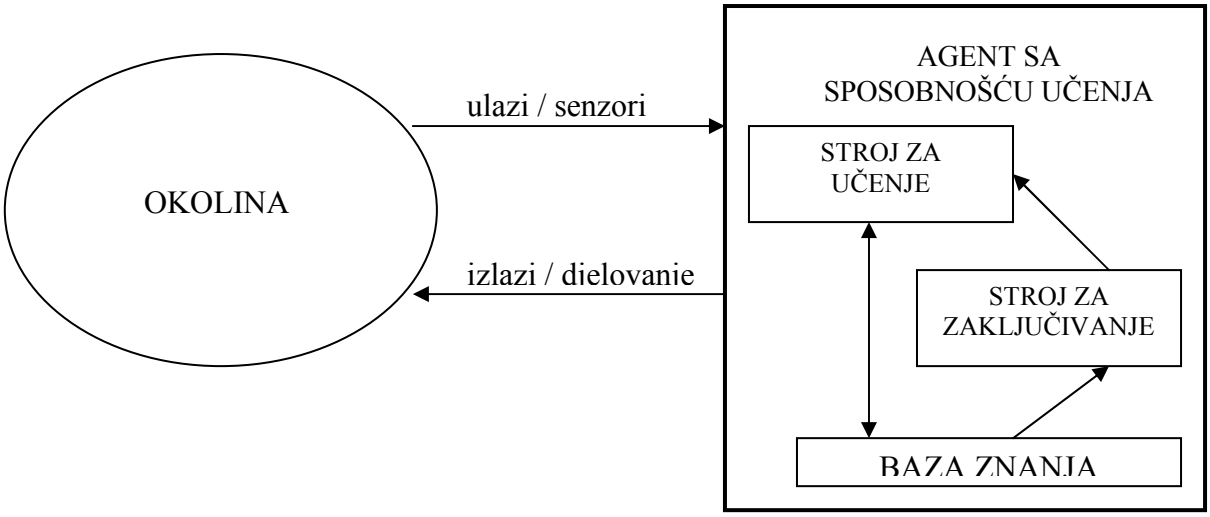
Slika 3-2 Građa inteligentnog agenta

3.1.4 Agent sa sposobnošću učenja (Learning agent)

Najveći problem pri razvoju i korištenju inteligentnih agenata je kodiranje znanja u bazi znanja (*knowledge acquisition bottleneck*!) i modifikacija ovog znanja kao odgovor na promjene u aplikacijskoj domeni ili zahtjevima na agenta (*the knowledge maintenance bottleneck*!).

"*Agent koji je sam sposoban prihvaćati i održavati svoje znanje naziva se agent sa sposobnošću učenja (learning agent).*"

- Agent koji može učiti vrlo pojednostavljuje proces izgradnje i održavanja baze znanja.
- Izgradnja ovakvog agenta vrlo teška.
- Ne razumijemo još kognitivne procese učenja.
- Učenje bez znanja ovdje nije moguće.
- Osoba koja razvija agenta mora ugraditi inicijalnu (malu) količinu znanja u bazu znanja.
- Inicijalna baza znanja može biti djelomično nekompletna i netočna jer će je agent tijekom učenja dovesti u konzistentno stanje.



Slika 3-3 Građa agenta sa sposobnošću učenja

3.1.5 Faze u razvoju baze znanja

Tri osnovne faze:

- Sistematično "izvlačenje" znanja iz eksperta
- Poboljšavanje baze znanja (refinement)
- Reformuliranje baze znanja

Tijekom sistematičnog "izvlačenje" znanja iz eksperta kreira se temeljno znanje agenta odabirući prikladnu shemu za reprezentaciju znanja i razvojem osnovne terminologije i konceptualne strukture baze znanja. Rezultat je inicijalno nekompletna i djelomično netočna baza koja se poboljšava tijekom slijedećih faza.

U fazi poboljšavanje baze znanja baza se proširuje novim znanjem i ispravlja se netočno znanje. Rezultat ove faze bi trebala biti baza koja je kompletna i točna u dovoljnoj mjeri da na većinu problema koje će agent rješavati daje ispravna rješenja.

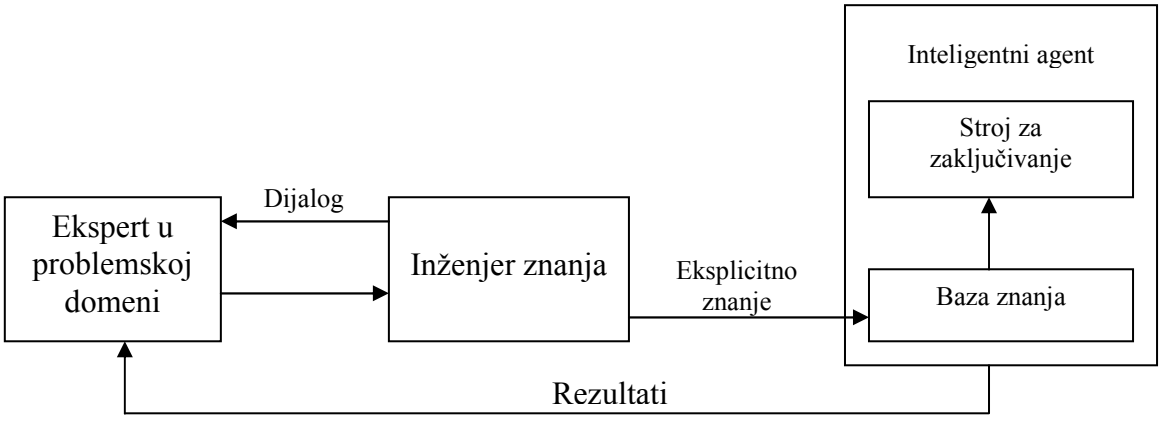
Tijekom reformuliranja baze znanja baza se reorganizira u svrhu povećanja djelotvornosti prilikom rješavanja zadataka.

Postoje dva osnovna pristupa razvoju baze znanja:

- Prikupljanje znanja (Knowledge acquisition approaches)
- Strojno učenje (Machine learning approaches)

3.1.5.1 Prikupljanje znanja (Knowledge acquisition)

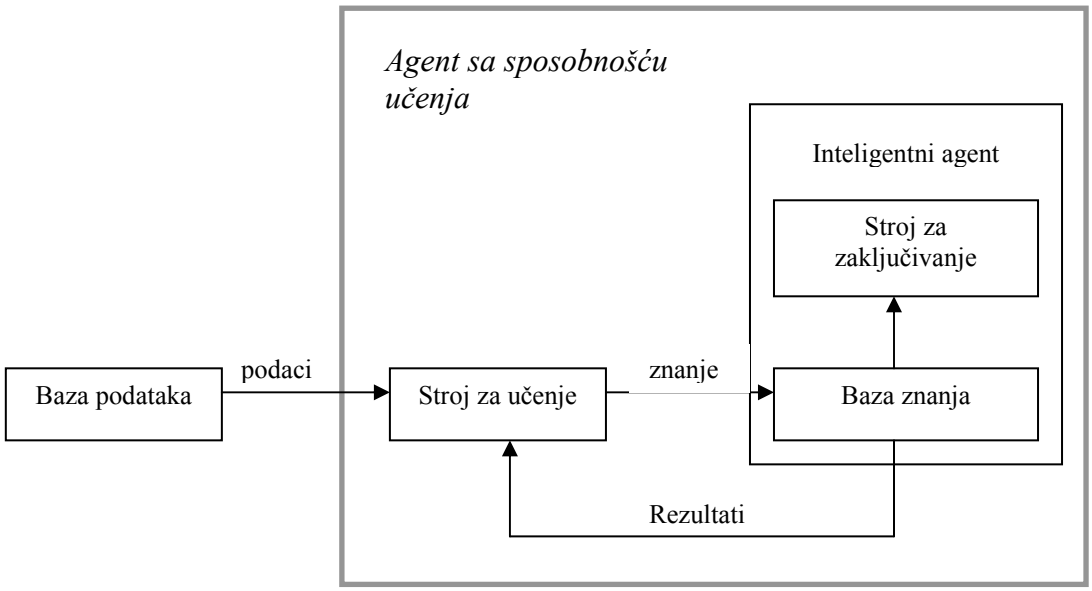
Fokusirano je na poboljšavanje i djelomično automatiziranje prikupljanja znanja od ljudskog eksperta ili inžinjera znanja. Cilj je konstruiranje točne i djelotvorne formalne reprezentacije znanja eksperta. Inženjer znanja surađuje sa ekspertom u problemskoj domeni kako bi shvatio na koji način ekspert rješava probleme. Tada odabire reprezentaciju znanja, gradi stroj za zaključivanje, "izvlači" znanje iz eksperta, konceptualizira ga i predstavlja u bazu znanja.



Slika 3-4 Građa i rad sa inteligentnim agentom

3.1.5.2 Strojno učenje

Fokusirano je na razvoj autonomnih algoritama za prikupljanje znanja iz podataka te za prevođenje i organiziranje znanja. Cilj je poboljšavanje kompetencije i djelotvornosti rješavanja problema. Znanje se testira eksperimentom nad podacima (ali ne nad onim iz kojih je učeno).



Slika 3-5 Građa i rad sa agentom sa sposobnošću učenja

U nastavku su nabrojane najčešće metode strojnog učenja.

- Empiričko induktivno učenje iz primjera – učenje definicije koncepta usporedbom pozitivnih i negativnih primjera, njihovih sličnosti i razlika i induktivnim kreiranjem poopćenog opisa pozitivnih primjera
- Učenje temeljeno na objašnjenjima – učenje definicije koncepta dokazivanjem da je primjer instanca tog koncepta i deduktivnim poopćavanjem dokaza
- Učenje analogijom – učenje novog znanja o entitetu transferiranjem u neki drugi poznati entitet
- Abduktivno učenje – hipotetiziranje o uzrocima bazirano na posljedicama
- Conceptual clustering – grupiranje objekata u razrede
- Kvantitativna otkrića – otkrivanje zvantitativnih zakona među varijablama koje opisuju objekt
- Reinforcement learning – ažuriranje znanja, bazirano na povratnoj vezi iz okoline
- Učenje pomoću genetskih algoritama – model nasljeđivanja i evolucije
- Učenje neuronskim mrežama – uporaba mreže jednostavnih jedinica da bi se postigle složene funkcije; bazirano na pojednostavljenju modela dendrita i aksona iz mozga

3.2. ONTOLOGIJE

Već je u prethodnom poglavlju spomenuto da je formalno predstavljanje znanja bazirano na konceptualizaciji – objekata, koncepata i ostalih entiteta za koje se podrazumijeva da postoje u području od interesa, te veza koje vrijede među njima [22]. Konceptualizacija jest apstraktni, pojednostavljeni pogled na svijet koji želimo predstaviti iz nekog razloga. Svaka baza znanja, sustav baziran na znanju ili agent koji radi na razini znanja zasnovan je na određenoj konceptualizaciji, eksplicitno ili implicitno. Ontologija je jednostavno eksplicitna specifikacija konceptualizacije [21]. Budući da su ontologije u ovom radu odabrane kao metoda za predstavljanje znanja o dokumentima, u nastavku ćemo pogledati kako se ontologije mogu definirati.

Trenutno postoje dva vrlo raširena načina za definiranje ontologija. Prva metoda, i ujedno najjednostavnija, temelji se na uporabi samog RDFa, odnosno definiranju RDF sheme. RDF ima ugrađeno nekoliko osnovnih konstrukcija kojima se mogu definirati veze među pojedinim objektima (definiranje razreda, nasljeđivanja i sl.). Međutim, kako se je uočilo da je ovo vrlo često nedovoljno, definirano je proširenje RDFa, DAML+OIL, koje uvodi niz novih konstrukcija kojima je moguće definirati puno složenije odnose između objekata. U nastavku su ukratko opisane obje metode.

3.2.1 Definiranje ontologija RDFom

RDF inherentno definira dva osnovna koncepta: resurs (rdfs:Resource) i razred (rdfs:Class). Notacija *namespace:konceptname* znači da se radi o konceptu *konceptname* koji je definiran ontologijom *namespace*, odnosno u našem slučaju kratica *rdfs* odnosi se na " http://www.w3.org/2000/01/rdf-schema# ", dok se kratica *rdf* odnosi na " http://www.w3.org/1999/02/22-rdf-syntax-ns# ". Osim ova dva koncepta, definiraju se i dvije temeljne veze: tip_resursa (rdf:type), te potklasa (rdfs:subClassOf). RDF polazi od pretpostavke da je najosnovnija klasa upravo rdfs:Resource, i svi resursi su primjerci (instance) ove klase. Veza rdf:type govori da je R resurs član određene klase C, odnosno da ima sva svojstva koja se očekuju od te klase; drugim riječima, resurs R je primjerak klase C. Veza rdfs:subClassOf govori da je određeni resurs R_1 (koji je određena klasa) podklasa od neke šire klase, resursa R_2.

Npr. možemo definirati klasu motorno vozilo (MotorVehicles), odnosno jezikom RDFa, definirati ćemo resurs MotorVehicles i reći da je taj resurs po tipu klasa, i da je podklasa od najopćenitije klase rdfs:Resource.

```
<rdf:Description rdf:ID="MotorVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>
```

Zatim možemo definirati klasu putničko vozilo (PassengerVehicles), i reći da je ta klasa specijalizacija klase MotorVehicles (odnosno njezina podklasa).

```
<rdf:Description rdf:ID="PassengerVehicle">
```

```
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

Svaka klasa osim što ima stablo nasljeđivanja, ima i određena svojstva (elemente, članove, engl. Properties). U RDFu sva svojstva su primjerci klase rdf:Property. Naime, umjesto klasičnog načina na koji većina programskih jezika definira elemente klase (klasa A ima svojstvo S koje je tipa T), RDF na ove definicije gleda s drugog aspekta – u RDFu se opisuje svojstvo S koje pripada klasi A i koje je tipa T. Shodno tome, sva su svojstva primjerci klase rdf:Property, i definiraju se dvije veze: veza kojom se svojstvo povezuje sa klasom (rdfs:domain), i veza sa kojom se svojstvo povezuje sa svojim tipom koji može poprimiti (rdfs:range). Npr. da bismo rekli da klasa motorno vozilo ima svojstvo "registrirano na" i to svojstvo poprima podatak tipa "Osoba", definirati ćemo novi resurs pod nazivom "registeredTo" i reći da je to primjerak klase rdf:Property, te da kao domenu ima klasu MotorVehicle, a kao tip podatka koji može poprimiti "Person".

```
<rdf:Description rdf:ID="registeredTo">
  <rdf:type
   rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2000/03/example/classes#Person"/>
</rdf:Description>
```

RDF nam dodatno omogućava i definiranje veza među svojstvima (baš kao što se definiraju i veze među klasama) – moguće je reći da je jedno svojstvo (npr. B) podsvojstvo nekog drugog svojstva (npr. A) vezom rdfs:subPropertyOf. Primjer kada ovo može biti korisno jest definiranje svojstva biološkiRoditelj i biološkiOtac – očito je da ako resurs R ima za svojstvo biološkiOtac vrijednost "Pero", tada i svojstvo biološkiRoditelj ima vrijednost "Pero". Vezom rdfs:subPropertyOf upravo je omogućeno da se definira kako je svojstvo biološkiOtac specijalizacija svojstva biološkiRoditelj.

```
<rdf:Description rdf:ID="biologicalParent">
  <rdf:type
   rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Description>

<rdf:Description rdf:ID="biologicalFather">
  <rdf:type
   rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:subPropertyOf rdf:resource="#biologicalParent"/>
</rdf:Description>
```

RDF nam omogućava još i definiranje ograničenja (iz klase rdfs:ConstraintResource za resurse, odnosno iz klase rdfs:ConstraintProperty za svojstva), međutim ne propisuje nikakvo značenje ovim ograničenjima. Drugim riječima, osim nekoliko ograničavajućih svojstava kojima značenje definira sam RDF (npr. svojstvo rdfs:range je primjerak ograničenja, pa je izveden kao primjerak klase rdfs:ConstraintProperty), ako korisnik sam definira nova ograničenja, agent koji pročita ontologiju shvatiti će da je to novo svojstvo ili resurs nekakvo ograničenje,

ali neće znati kakvo, odnosno kakvo je značenje tog ograničenja. Naime, na ovom je mjestu očito koji su nedostaci vezani uz RDF:

> *RDFom se mogu definirati složeni odnosi među objektima kao i složeni razredi sa pripadnim svojstvima. Međutim, RDF ne može definirati značenje (semantiku) onoga što je njime opisano. To je najbolje vidljivo na primjeru same definicije RDF sheme i RDF sintakse. Postoje dvije ontologije koje definiraju niz objekata i veza među njima (ono što je dostupno i agentima), i postoji niz tekstova napisanih prirodnim jezikom koji objašnjavaju semantiku onoga što je napisano u tim ontologijama, jer ontologija sama za sebe ne govori apsolutno ništa.*

3.2.2 Definiranje ontologija pomoću DAML+OIL

Nakon što je uočeno da sam RDF nudi vrlo slabu podršku za specifikaciju složenih odnosa među objektima, a gotovo nikakvu podršku za specifikaciju složenih tipova ograničenja na te objekte i njihova svojstva, napisana je još jedna ontologija i još jedan dokument koji objašnjava semantiku napisane ontologije, s ciljem da se uvedu nova standardizirana ograničenja i svojstva, koja će omogućavati definiranje kompleksnijih odnosa. DAML+OIL proširenje je RDFa i kao svoju osnovu koristi isključivo RDF.

DAML+OIL definira niz novih razreda i svojstava. Svaka ontologija trebala bi sadržavati izjavu da je ona sama ontologija:

```
<daml:Ontology rdf:about="">
  <daml:versionInfo>some version</daml:versionInfo>
  <rdfs:comment>
    An example ontology
  </rdfs:comment>
  <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
</daml:Ontology>
```

Ovime je rečeno da je resurs "trenutni dokument" primjerak klase daml:Ontology, tj. semantičko značenje ove izjave jest da je trenutni dokument ontologija. Definira se još i svojstvo daml:versionInfo, a koristi svojstvo rdfs:comment. Svojstvo daml:imports govori na kojima se sve ontologijama zasniva trenutna ontologija.

DAML+OIL zatim reslikava niz objekata iz rdfs ontologije u daml ontologiju, pa tako postaje nebitno da li je neki objekt primjerak rdfs:Class klase, ili daml:Class klase, jer se te dvije klase vežu relacijom daml:sameClassAs.

Npr. možemo definirati klase Animal, Male, Female, Man, Woman i odnose među njima. Isto tako, DAML+OIL dozvoljava da postavimo ograničenje da skup objekata koji su tipa Female nema zajedničkih objekata sa skupom objekata koji su tipa Male (svojstvo daml:disjointWith). Primjer je prikazan u nastavku:

```
<daml:Class rdf:ID="Animal">
  <rdfs:label>Animal</rdfs:label>
</daml:Class>
```

```
<daml:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</daml:Class>

<daml:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <daml:disjointWith rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Woman">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Female"/>
</daml:Class>
```

Za razliku od RDFa, DAML+OIL svojstva dijeli u dvije kategorije: svojstva koja kao vrijednost poprimaju resurse (daml:ObjectProperty) i svojstva koja kao vrijednost poprimaju jednostavnije tipove podataka (daml:DatatypeProperty). Npr. možemo definirati svojstvo ima_roditelja (hasParent) te kao domenu i opseg zadati resurs Animal. Dakle, samo životinja može imati roditelja koji je također životinja.

```
<daml:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</daml:ObjectProperty>
```

Koduporabe DAML+OILa možemo slobodno koristiti i RDF konstrukcije, pa tako možemo definirati da je svojstvo hasFather podsvojstvo od hasParent, te da svojstvo hasFather može primati samo poprimati vrijednosti koje su isključivo tipa Male (iako je svojstvo koje je specijalizirano, dakle hasParent, dozvoljavalo da se na tom mjestu pojavi objekt tipa Animal).

```
<daml:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>
```

Svojstva koja poprimaju podatke koji nisu resursi izvode se iz DatatypeProperty razreda. Npr. svojstvo veličina cipela poprima podatak tipa xmls:decimal, i osoba može imate samo jedan broj cipela. To možemo izreći na slijedeći način:

```
<daml:DatatypeProperty rdf:ID="shoesize">
  <rdfs:comment>
    shoesize is a DatatypeProperty whose range is xsd:decimal.
    shoesize is also a UniqueProperty (can only have one shoesize)
  </rdfs:comment>
  <rdf:type
   rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
</daml:DatatypeProperty>
```

DAML+OIL uvodi niz načina za opisivanje raznih ograničenja. Npr. možemo definirati klasu Osoba (Person), i reći da je to specijalizacija klase Animal. Međutim,

zatim želimo reći da osoba za roditelja može imati isključivo drugu osobu, te da može imati samo jednog oca, i mora imati barem jednu veličinu cipela. Ove uvjete ćemo u DAML+OILu ostvariti definiranjem ograničavajućih klasa (daml:Restriction) koji će ispunjavati jedno od ograničenja, i zatim ćemo klasu Parent proglasiti podklasom takvih klasa. Primjer je prikazan u nastavku:

```
<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasFather"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#shoesize"/>
      <daml:minCardinality>1</daml:minCardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Drugo rdfs:subClassOf svojstvo govori da je klasa Person podklasa anonimne klase koja je primjerak daml:Restriction klase, i koja postavlja ograničenje na svojstvo hasParent, i zahtjeva da kao vrijednost to svojstvo poprimi isključivo objekt koje je tipa Person. Slično zahtijevaju i preostala rdfs:subClassOf svojstva.

DAML+OIL omogućava i definiranje kardinalnosti za svojstva (minimalnu, maksimalnu ili obje), tranzitivnosti i sl. Npr.

```
<daml:TransitiveProperty rdf:ID="hasAncestor">
  <rdfs:label>hasAncestor</rdfs:label>
</daml:TransitiveProperty>
```

govori da je hasAncestor tranzitivno svojstvo. Međutim, interesantno je spomenuti da DAML+OIL ne može reći da je to tranzitivna verzija svojstva hasParent!

DAML+OIL omogućava definiranje ograničenja nad klasama i preko skupovnih operacija. Npr. možemo reći da je određena klasa:

```
<rdfs:subClassOf>
  <daml:Class>
    <daml:complementOf rdf:resource="#Person"/>
  </daml:Class>
</rdfs:subClassOf>
```

Ovime smo definirali klasu koja predstavlja sve objekte koji nisu tipa Person. Postoji još niz sličnih konstrukcija.

3.2.3 Problem ontologije – definiranje značenja?

Već je prije spomenuto da je svako definiranje nove ontologije popraćeno generiranjem dodatne dokumentacije koja ljudima govori koje je značenje definiranih koncepata u ontologiji. Naime, sama ontologija o značenju ne govori ništa. U ovo ćemo se najbolje uvjeriti kroz jednostavan primjer. Definirajmo u ontologiji A (="http://ex.com/a#") klasu C, i definirajmo da klasa C ima svojstvo S:

```
<rdf:Description rdf:ID="C">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="S">
  <rdf:type
   rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#C"/>
</rdf:Description>
```

Zatim napišimo ontologiju B (="http://ex.com/b#") u kojoj definiramo svojstvo B:qwerty koje pripada bilo kojoj klasi i poprima kao vrijednost opet neku klasu, i definirajmo razred D, koji ima svojstvo B:qwerty sa vrijednošću resursa C.

```
<rdf:Description rdf:ID="D">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <B:qwerty rdf:resource="http://ex.com/a#C">
</rdf:Description>

<rdf:Description rdf:ID="qwerty">
  <rdf:type
   rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>
```

Dozvolimo sada našem agentu da pročita jednu i drugu ontologiju, i zatim mu postavimo pitanje "Da li objekt O koji je primjerak klase B:D može imati svojstvo A:S?". Može li agent odgovoriti na to pitanje? Logičkim razmatranjem agent može zaključiti (odnosno vidjeti) da svojstvo A:S mogu imati samo primjerci klase A:C, zatim da je objekt O primjerak klase B:D, i da je klasa B:D preko svojstva B:qwerty povezana sa klasom A:C. Ništa drugo nije moguće zaključiti iz ove dvije ontologije, i odgovor agenta bi trebao biti "ne znam". Naime, odgovor ne može biti niti potvrdan, niti negativan jer ništa od toga ne možemo dokazati.
Postavlja se pitanje da li bi se išta promijenilo kada bi naziv definiranog svojstva umjesto B:qwerty glasio B:sameClassAs? Kako bismo mi tada odgovorili na postavljeno pitanje? Objekt O je tipa B:D, a tip B:D je isto što i A:C, a budući da A:C ima svojstvo A:S, tada i O može imati to svojstvo. Međutim, ključan trenutak je spoznaja da je "tip B:D isto što i A:C" što zaključujemo iz samog naziva svojstva (sameClassAs). Dakle, mi smo ovdje na temelju naziva otkrili (pretpostavili?) semantiku tog svojstva. Da smo rješavali problem uz početno definirani zadatak, završili bismo isto kao i agent. Naime, iz naziva B:querty ne bismo mogli izvući semantiku i zaključiti da je "tip B:D isto što i A:C". U tom slučaju trebali bismo prolistati dokumentaciju uz samu ontologiju koja bi trebala definirati semantiku svojstva B:qwerty prirodnim jezikom.

Iz ovog razmatranja očito je da sama ontologija nije dovoljna kako bi agent na temelju nje mogao izvoditi zaključivanje kakvo od njega očekujemo.

3.3. FORMATI DOKUMENATA NA INTERNETU

U današnje doba najveći dio dokumenata koji sadrže tekstualne informacije pohranjen je u jednom od slijedeća tri formata: HTML, XML te XHTML, od kojih je većina još uvijek u klasičnom HTML formatu. Zbog toga će u nastavku biti dat kratak prikaz ovih formata, kao i problemi vezani uz njih.

3.3.1 HTML

HTML je kratica od Hypertext Mark-up Language. Ovaj format baziran je na SGML-u (Standard Generalized Markup Language). Ovo je jedan od najuspješnijih formata dokumenata u povijesti. HTML dokumente moguće je kreirati kroz širok spektar alata (od jednostavnih tekst-uređivača pa do vrlo složenih WYSIWYG alata), budući da se koriste tekstualne datoteke.

Kako bi strukturirao tekst, HTML format koristi oznake (npr. *<H1>...</H1>*). U HTML dokumentima moguće je prikazivati tekst i slike, a postoji i oznaka za umetanje datoteka sa animacijama (*.AVI, .MPEG, .MOV* i sl.), zvuka (*.MID, .WAV* i sl.) i sl. HTML format ujedno omogućava i direktno umetanje raznih skriptnih jezika u sam dokument pomoću posebne oznake (*SCRIPT*).

Ovakav jednostavan format u početku je bio zamišljen kao pomagalo pri automatizaciji weba. Međutim, pokazuje se ipak da format i nije baš previše prijateljski sa pokušajima automatizacije. Naime, jedan od problema je vrlo slaba podrška metapodacima (standardom je definirano tek nekoliko osnovnih). Isto tako, bilo kakav pokušaj izrade sustava koji bi ispravno tumačio ove dokumente svodi se na izradu sustava koji mora razumjeti mnoštvo različitih verzija ovog formata, budući da trenutno postoje verzije HTML 2.0, HTML 3.2, HTML 4.0 te HTML 4.01.

U prilog tezi da ovaj format nije velika pomoć pri automatizaciji govori i činjenica da se specifikacija formata prostire na oko četristotinjak stranica. No unatoč tome što sam standard definira mnoštvo oznaka, može se pokazati da se opet mogu postaviti pitanja: "ima li previše oznaka" te "ima li premalo oznaka" i na oba pitanja odgovori mogu biti i potvrdni i negativni. Naime, želimo li napraviti sustav automatizacije ili pak jednostavno sustav za prikaz HTML dokumenata, tada ćemo doći do zaključka da oznaka ima puno previše. Razvoj jednog ovakvog sustava i timu ljudi trajao bi mjesecima, ako ne i godinama. S druge pak strane, pitamo li bilo kojeg korisnika (osobu koja koristi HTML za kreiranje svojih dokumenata) ista pitanja, odgovor će bez sumnje biti "ima ih premalo – ja sam baš htio napraviti *xxx* a HTML nema oznake kojim bih to mogao".

Da stvar bude još gora, specifikacija postavlja zahtjeve i na parsere HTML dokumenata: parseri moraju ignorirati sve oznake koje ne razumiju, i moraju

ignorirati sve atribute koje ne razumiju u oznakama koje razumiju. Ovakvi zahtjevi autorima dokumenata otežavaju ispravno pisanje HTML dokumenata jer ignoriraju sve greške koje autor može napraviti pa nema povratne informacije da li je dokument ispravno napisan ili nije. Nadalje, u slučaju neispravno napisanih HTML dokumenata parser će prolaziti kroz stanja grešaka i morati će koristiti sofisticirane metode za oporavak, što nikako ne ide u prilog izradi malih i brzih sustava automatizacije weba.

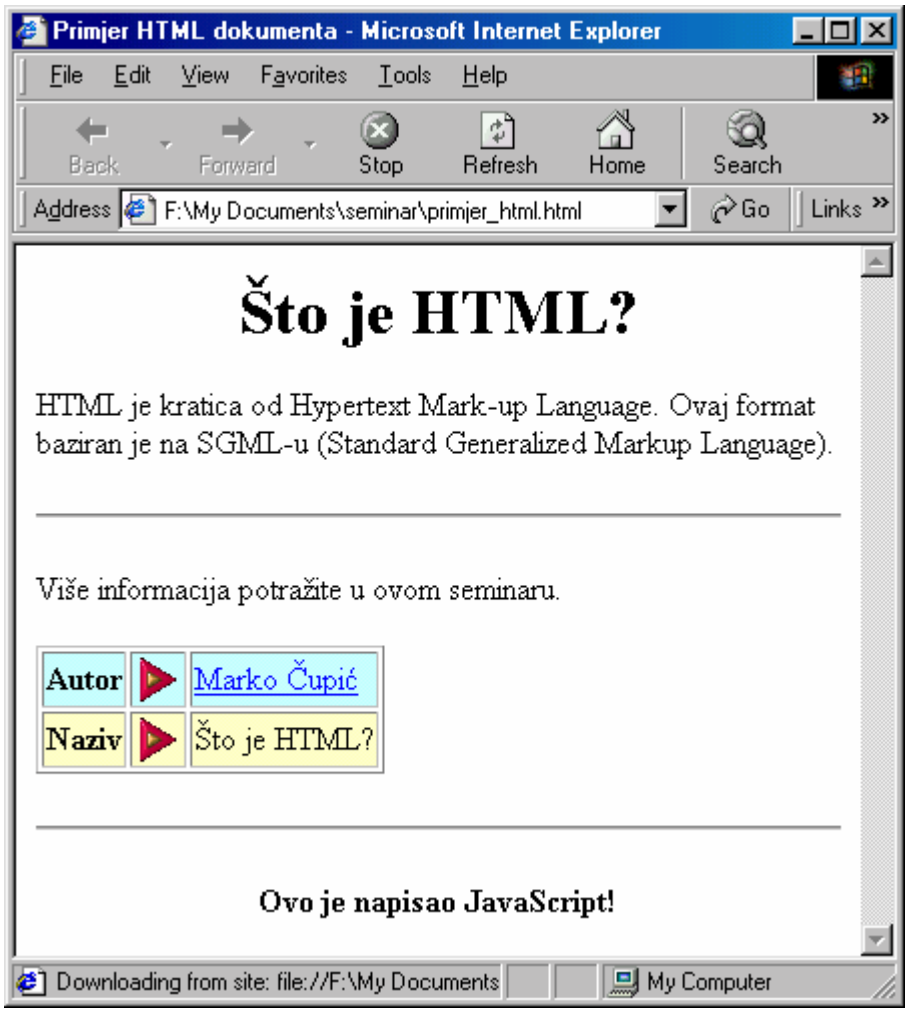
3.3.1.1 Ukratko o specifikaciji kroz primjer HTML dokumenta

HTML specifikacija pobliže je opisana kroz jednostavan primjer HTML dokumenta (Primjer 3.3-1).

Primjer 3.3-1 Jednostavan HTML dokument

```
<HTML>
  <HEAD>
   <META http-equiv="Content-Type"
         CONTENT="text/html; charset=windows-1250">
   <META NAME="Author"
         CONTENT="Marko Cupic">
   <META NAME="GENERATOR"
         CONTENT="Mozilla/4.72 [en] (Win98; I) [Netscape]">
   <META NAME="Description"
         CONTENT="Ovaj dokument govori o HTML specifikaciji.">
   <META NAME="KeyWords"
         CONTENT="HTML specifikacija, HTML parseri">
   <TITLE>Primjer HTML dokumenta</TITLE>
  </HEAD>

  <BODY>

    <H1><CENTER>Što je HTML?</CENTER></H1>

    <P>HTML je kratica od Hypertext Mark-up Language. Ovaj format
    baziran je na SGML-u (Standard Generalized Markup Language).

    <P><HR>

    <P>Više informacija potražite u ovom seminaru.

    <P>
    <TABLE BORDER="1">
      <TR BGCOLOR="#CCFFFF">
        <TD><B>Autor</B></TD>
        <TD><IMG SRC="Strelica.gif"></TD>
        <TD><A HREF="http://marko.cupic.com/">Marko Čupić</A></TD>
      </TR>
      <TR BGCOLOR="#FFFFCC">
        <TD><B>Naziv</B></TD>
        <TD><IMG SRC="Strelica.gif"></TD>
        <TD>Što je HTML?</TD>
      </TR>
    </TABLE>

    <P><HR>

    <P><SCRIPT Language="JavaScript"><!--
```

```
      document.write(
         "<CENTER><B>Ovo je napisao JavaScript!</B></CENTER>"
      );
      //--></SCRIPT>

  </BODY>
</HTML>
```



Slika 3-6 Primjer HTML dokumenta

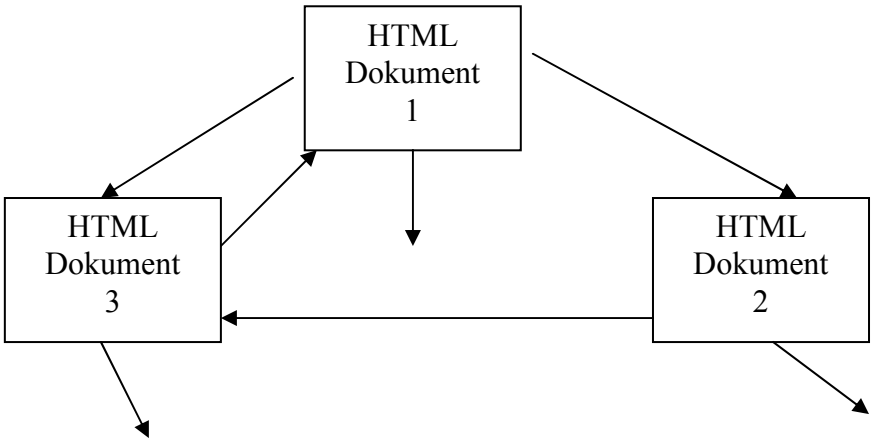
Primjer 3.3-1 prikazuje jednostavan HTML dokument, dok Slika 3-6 prikazuje kako taj dokument tumači Microsoft Internet Explorer. Iz primjera je vidljivo na koji se način zapisuju HTML oznake. Oznaka započinje šiljastom otvorenom zagrade iza koje slijedi naziv. Zatim mogu slijediti atributi koji su oblika *NazivAtributa* ili pak *NazivAtributa* = *VrijednostAtributa* pri čemu *VrijednostAtributa* može ali i ne mora biti napisana kao tekst pod navodnicima.

Specifikacija razlikuje dvije vrste oznaka: prazne oznake (*
, <IMG ...>* i sl.) te neprazne oznake (*<P>...</P>, <A ...>...</A>* i sl.). Prazne oznake koriste se za izdavanje naredbi tipa "ovdje prijeđi u novi red" (*
), "ovdje ubaci sliku" (<IMG>*) i sl. Neprazne oznake su oznake koje se sastoje od tri dijela: početka

oznake (<*P*>), tijela oznake (*"Ovo je paragraf"*) te završetka oznake (</*P*>). Neprazne oznake imaju tijelo i pregledniku govore što da radi, odnosno kako da prikaže to tijelo.

Primjer 3.3-1 demonstrira neke od mogućnosti HTML-a. Prikazan je unos metapodataka u sam dokument (META oznaka), davanje naslova dokumentu (koji je ispisan na vrhu prozora Microsoft Internet Explorera), strukturiran unos teksta u dokumentu (naslov je upisan unutar H1 oznaka, običan tekst unutar P oznake), unos tablica (TABLE oznaka), unos slika (IMG oznaka), unos hipertektualnih veza (A oznaka) te unos skriptnog jezika u sam dokument (SCRIPT oznaka). Primjer 3.3-1 također jasno demonstrira osnovni kostur svakog HTML dokumenta: cijeli dokument sastoji se od jedne HTML oznake koja pak ima dvije osnovne cjeline – HEAD oznaku koja donosi informacije o samom dokumentu, te BODY oznaku koja je de facto sadržaj dokumenta.

Pomoću hipertekstualnih veza moguće je povezivati dokumente. Pri tome mogu nastati stablaste tvorevine, ali najčešće nastaju ciklički grafovi budući da autori HTML dokumenata često na stranice stavljaju vezu na stranicu sa koje je korisnik upravo došao.



Slika 3-7 Mrežasta struktura dokumenata

U nastavku su navedene mogućnosti HTML-a.

3.3.1.2 Kratak pregled mogućnosti HTML

HTML specifikacija definira mnoštvo oznaka čija svrha varira od unosa metapodataka, pa do opisa teksta (vrsta slova, veličina i sl.), unosa tablica, slika i sl. Slijedeća lista daje kratak popis najčešće korištenih HTML oznaka.

Tablica 3-1 Oznake opće namjene

naslov dokumenta	**<title>** ...**</title>**
naslovi u tekstu	** ** ...** **, ..., ** ** ...** **
odlomci	** ** ... ** **
naglašavanje	**** ...****

unaprijed formatirani tekst	**<pre>** ... **</pre>**
slike	**<img src=... width=... height=... alt=...** **longdesc=... border=... usemap=...>**
forsirani prijelom retka	** **
mape	**<map name=...>** **<area shape="circle" coords="50,100,20"** **href=... alt=...>** **</map>** oblici: *circle*, *rect*, *poly*
okviri	**<frameset> <frame …> … </frameset>** **<noframes> … </noframes>**

Potrebno je spomenuti da se neke oznake ne moraju uparivati sa oznakama za zatvaranje. Tako npr. početak paragrafa započinjemo s oznakom , ali ukoliko iza njega slijedi novi paragraf, nije potrebno prvo zatvoriti stari paragraf oznakom već je dovoljno započeti novi paragraf oznakom .

Tablica 3-2 Oznake za definiranje hipertekstualnih veza

općenito	**** ... ****	
označavanje ciljnih točaka	staro	**** ... ****
	novo	**id="**ime**"**
vrste ciljnih točaka	lokalne	**** ... ****
	vanjske	**...**

Hipertekstualne veze su jedan od najbitnijih elemenata web stranica. One omogućavaju da se s jedne stranice uspostavi "veza" s nekim drugim dijelom te iste stranice, općenito s nekom drugom stranicom (pri ćemu se stranica prikazuje od početka) ili pak s određenim dijelom neke druge stranice (koji je označen ciljnom točkom) pri ćemu se prikazuje druga stranica od točno zadane pozicije.

Tablica 3-3 Oznake za definiranje lista

neporedane	** ... **
poredane	** ... **
definicijske	**<dl> <dt> </dt> <dd> </dd> </dl>**

Liste su također vrlo često korištene u izradi web stranica. Sam standard podržava više tipova lista, među koje se ubrajaju neporedane liste (u kojima se pojedini stavci ne numeriraju nego se označavaju točkicama, kružićima i sl.), poredane liste (u kojima se pojedini stavci numeriraju), te definicijske liste (kod kojih se svaka stavka

sastoji od dvije cjeline: dijela koji je želi definirati i dijela koji sadrži definiciju koncepta iz prvog dijela stavke).

Tablica 3-4 Oznake za često korištene specijalne znakove

Razmaci	** **
Copyright	**©** ©
Registered trademark	**®** ®
Trademark	**™** ™
Ostalo	**<**, **>**, **&**, **"**

Budući da HTML dokumenti u tekst umeću oznake, očito je da se neki simboli ne smiju pojavljivati u dokumentu jer će inaće biti krivo interpretirani kao početak oznake. Zbog taga postoji mehanizam kako se ti simboli umeću u dokument a da ne prouzrokuju pogreške prilikom interpretiranja dokumenta. Primjer je simbol manje ('<') koji označava početak oznake. Ukoliko je taj simbol potreban u dokumentu, umetnuti će se kao slijed znakova '<'. Slično vrijedi i za druge specijalne znakove.

Tablica 3-5 Oznake za rad s tablicama

Tablica	**<table> ... </table>**
Opis tablice	**<caption> ... </caption>**
Redak tablice	**<tr> ... </tr>**
Ćelija tablice	**<td> ... </td>**
Podnaslov u tablici	**<th> ... </th>**

Slijedeći vrlo bitan element dizajna web stranice su tablice. Naime, osim što se tablice koriste za tablični prikaz podataka, daleko češće se koriste za precizno pozicioniranje grafičkih elemenata na stranici, i za izradu finih prijeloma teksta. Ovdje opet postoje oznake koje nije potrebno zatvarati, a pravila su još kompliciranija nego u prethodno opisanom primjeru.

Tablica 3-6 Oznake za rad sa skriptama

Umetanje skripte	**<script language="***JavaScript***"> ... </script>**
Umetanje skripte	**<script type="***text/javascript***"> ... </script>**
Onemogućene skripte	**<noscript> ... </noscript>**
Događaji	** ... **

Kako bi se stranice učinile što dinamičnijima, često se koriste različiti skriptni jezici koji se umeću direktno u tijelo dokumenta. Za to služi posebna oznaka <script>. Međutim, veliki je problem pri uporabi ove oznake to što stariji preglednici ne

prepoznaju tu oznaku (pa je zanemare) a tijelo skripte koje slijedi tumače kao dio dokumenta što rezultira katastrofom u pokušaju interpretiranja dokumenta, a vrlo često i rušenjem samog preglednika.

Tablica 3-7 Oznake za definiranje Cascading style sheets (CSS)

```
<style type="text/css">
  body { color: black; background: white; }
  pre { color: green; font-family: monospace; }
  table {
    margin-left: -4%
    font-family: sans-serif;
    background: white;
    border-width: 2;
    border-color: white;
  }
  th { font-family: sans-serif; background: rgb(204, 204, 153) }
  td { font-family: sans-serif; background: rgb(255, 255, 153) }
</style>
```

I konačno, jedan od novijih dijelova u HTML specifikaciji su i stilovi za čije je opisivanje razvijen poseban jezik. Stilovi se najčešće umeću <style> oznakom, no postoje još neki načini. Problem starih preglednika ovdje je posebno izražen jer stranice koje koriste stilove čak niti u novim preglednicima različitih proizvođača ne izgledaju jednako.

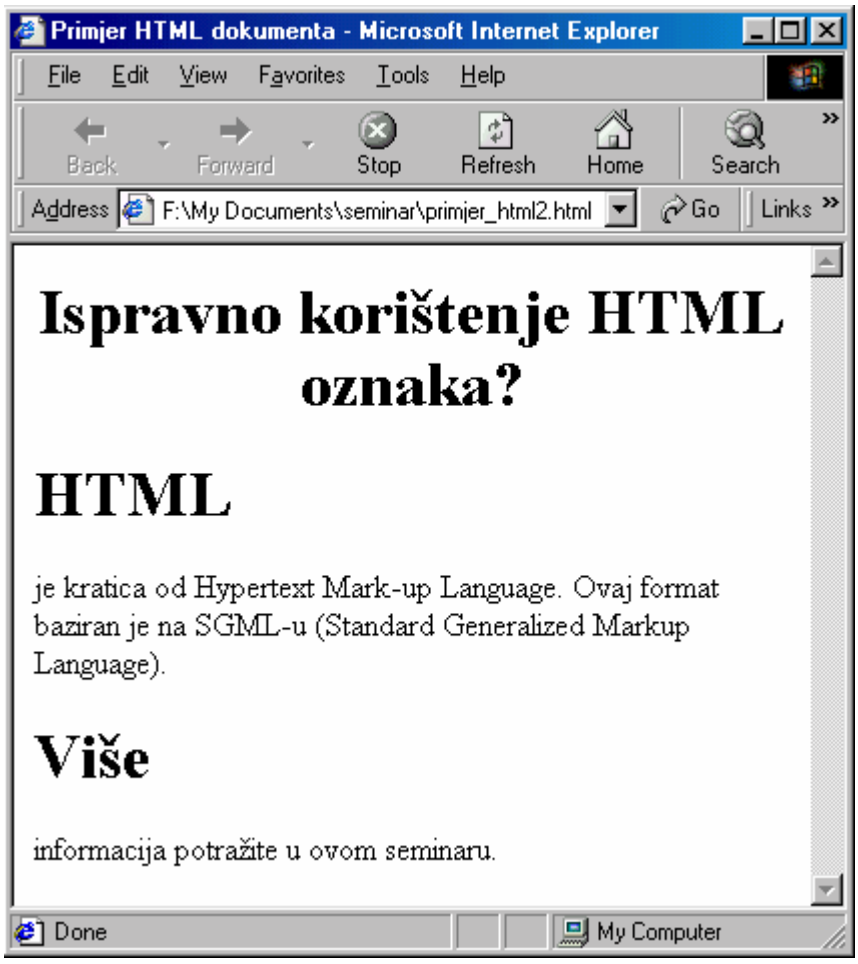
Dosta velik problem u izradi malih i jednostavnih parsera ovog formata predstavlja i neobaveznost zatvaranja pojedinih oznaka koji imaju sadržaj. Primjer je već i jednostavna P oznaka koja se ne mora zatvarati, ali ako se nađu dvije otvorene P oznake za redom, druga P oznaka automatski zatvara prvu, i započinje novu. Ovo je jednostavno pravilo koje i nije toliko teško implementirati, međutim, ovakvih detalja ima mnoštvo, a s druge strane postavlja se pitanje što ako iza P oznake ne dođe nova P oznaka već neka druga; da li tada zatvoriti oznaku ili ne?

3.3.1.3 Dobre i loše strane specifikacije

Iz ovog kratkog pregleda može se jasno zaključiti da je HTML jedan sveobuhvatan format. Međutim, mana mu je što je istovremeno i deskriptivni i komandni jezik koji svojim oznakama određuje i značenje pojedinih dijelova teksta i način prikaza istih. Dodatni problem je što je pojedinim oznakama koje određuju značenje dijelova teksta istovremeno propisan i način prikaza, pa mnoštvo autora HTML dokumenata te oznake ne koristi zbog deskriptivnog karaktera već upravo zbog komandnog karaktera kako bi u dokumentu postigli određeni grafički učinak (npr. uporaba H1 oznake koja ima značenje naslova najviše razine koristi se kao sredstvo za prikaz teksta najvećim slovima).

Nadalje, HTML specifikacija je vrlo složena kako bi se omogućilo formatiranje teksta, uključivanje slika, animacija, glazbenih datoteka te skripti. Zbog toga HTML

parser mora biti vrlo složen kako bi ispravno protumačio dokument. Jedan vrlo čest primjer je uključivanje skripti u tijelo HTML dokumenta – skripta se uključuje SCRIPT oznakom, a sve što se nalazi unutar SCRIPT oznake nije formatirano prema HTML specifikaciji, već prema pravilima skriptnog jezika. Vrlo često se dogodi da parseri koji na ovo ne paze počnu i u tom dijelu tražiti HTML oznake što rezultira katastrofom; naime, ima li smisla izvorni C kod parsirati HTML parserom? Upravo ovaj primjer jedan je od razloga što su autori HTML stranica koji koriste skripte tijela skripti zatvarali u znakove komentara kako bi ih stariji preglednici koji ne razumiju SCRIPT oznaku ispravno protumačili. Međutim, praksa pokazuje da ovo ne reda svi – i tako na internetu postoji šuma razno raznih formata i doskočica koje može koliko toliko ispravo prikazati jedino preglednik koji nije stariji od pola sata… I upravo je ovo glavni razlog moje tvrdnje da je ovaj format gotovo nemoguće automatizirati na jednostavan način.



Slika 3-8 Ispravna uporaba oznaka?

3.3.2 XML

XML je format koji je stvoren pod pokroviteljstvom W3C-a (World Wide Web Consortium). Kratica XML (engl. *eXtensible Markup Language*) označava proširivi jezik. Začetak razvoja ove specifikacije smješten je u srpanj 1996. godine što znači

da je ova specifikacija još vrlo mlada. U javnost je pušten u studenom 1996. godine a prvi parser napravljen je u siječnju 1997. godine. Prva aplikacija koja je koristila XML napravljena je u ožujku 1997. godine. XML specifikacija je vrlo jednostavna (svega četrdesetak stranica). To je pojednostavljeni oblik SGML-a. XML je lagan i proširiv, te garantirano neće srušiti aplikaciju koja ga koristi.

Poučeni problemima sa HTML-om, tvorci XML-a odlučili su da XML bude isključivo deskriptivni jezik, pa oznake ne opisuju KAKO formatirati, već ŠTO to predstavlja. Ideja je da se odvoji dokument od njegove prezentacije. Loša strana ove odluke je otežano dizajniranje dokumenta i prezentacije; međutim, dobra strana je da se svaki od ovih problema sada može napasti zasebno, i postići kvalitetnija rješenja. Evo i jednostavnog primjera zašto je dobro odvojiti dokument od njegove prezentacije. Uzmimo za prijer da smo procesiranjem XML dokumenta dobili e-mail adresu koju sada želimo prikazati na 21-inčnom monitoru te na zaslonu minijaturnog mobitela; isto tako, adresu želimo kvalitetno otisnuti na papiru, i pripremiti za ispis na faks. Očito je da će način prezentacije te e-mail adrese biti vrlo različit na 21-inčnom monitoru (npr. font veličine 50 za slabovidne osobe) i na zaslonu minijaturnog mobitela (gdje je cijeli manjih dimenzija od fonta 50). Isti komentar vrijedi i za pripremu ispisa.

XML donosi još jednu novost – jedan logički dokument može se protezati kroz više fizičkih datoteka.

XML specifikacija također postavlja uvjete na parsere XML dokumenata, ali bitno drugačije od HTML specifikacije. Naime, od parsera XML dokumenata se zahtjeva da ispravno protumače sve ispravne XML dokumente (dakle, nema ignoriranja i nerazumijevanja oznaka; ako je dokument ispravan, parser ga mora ispravno protumačiti). Nadalje se zahtjeva da u slučaju pojave greške parsr mora aplikaciji javiti grešku, i najstrože mu je zabranjeno pokušati bilo kakav oporavak! Isključivo u slučaju da korisnik tako želi, parser smije pokušati nastaviti daljnji rad, ali sva se komunikacija sa aplikacijom dalje mora odvijati na nestandardan način, te se ovakav način rada dozvoljava isključivo u svrhu pronalaska svih preostalih pogrešaka u dokumentu.

XML dalje uvodi kategorizaciju ispravnosti XML dokumenta. Dokument može biti dobro formiran (well-formed), i može biti ispravan (valid). Ispravni XML dokument sastoji se od dva dijela: DTD-a (Document Type Definition) te tijela dokumenta.

3.3.3 Bitne razlike između HTML I XML specifikacija

Koje su bitne razlike između ove dvije specifikacije, najjednostavnije je pokazati kroz primjer dokumenta koji se prema HTML specifikaciji tumači kao ispravan, dok se prema XML specifikaciji tumači kao neispravan.

Primjer 3.3-2 Ispravni HTML a istovremeno neispravni XML dokument

```
<title>Ispravni HTML</title>
Neki tekst, i ja <i>doista</i> ne želim
ovdje razmak između "ovdje" i "razmak".
<p>Evo i slike: <IMG src=madonna.jpg>
```

Pogledajmo zašto ovo nije dobar XML dokument (čak niti well-formed).

- Nema "root" elementa koji obuhvaća sve (trebao bi biti `<HTML> ... </HTML>`).
- Entitet `nbsp` je korišten a nije deklariran.
- Postoji `<p>` oznaka bez odgovarajuće `</p>`.
- `<IMG>` oznaci `nedostaje zatvaranje />`, pa XML parser ne može znati da bi ona trebala biti prazna.
- Vrijednost `src` atributa, `madonna.jpg`, morala bi biti pod navodnicima.

Primjer 3.3-2 pokazuje da je XML specifikacija vrlo striktna i uvodi jasna i nedvosmislena pravila. Npr. oznaka može biti prazna, ali ako jest, nužno mora imati oznaku da je prazna. Dakle, prazne oznake koje se u HTML-u pišu npr.
, u XML-u moraju biti napisane s kosom crtom na kraju:
. Ukoliko oznaka ima atribute, i ti atributi imaju vrijednosti, vrijednosti nužno moraju biti navedene pod navodnicima.

3.3.3.1 Proširivost XML-a

XML specifikacija dopušta izradu novih XML oznaka i time jezik čini proširivim. Kako se ovo može izvesti najbolje će se vidjeti iz jednostavnog primjera u kojem je e-mail poruka opisana XML-om.

Primjer 3.3-3 Primjer e-mail poruke u XML-u

```
<email>
  <head>
    <from>
      <name>Marko Čupić</name>
      <address>marcupic@pinus.cc.fer.hr</address>
    </from>
    <to>
      <name>Pero Perić</name>
      <address>perperic@pinus.cc.fer.hr</address>
    </to>
    <subject> XML specifikacija </subject>
  </head>
  <body>
    <p>Budući da si rekao da ti pošaljem XML
    specifikaciju ako je pronađem, šaljem ti XML
    1.0 specifikaciju, što je trenutno aktualna
    inačica, iako je već drugo izdanje.</p>
    <attach encoding="mime" name="xml-spec.html"/>
  </body>
</email>
```

Ovaj primjer pokazuje jedan XML dokument koji poštuje sve zahtjeve na oznake. Prazne oznake završavaju kosom crtom; neprazne oznake imaju završni dio i sl. Međutim, što oznake iz ovog primjera znače? Kako definirati da li su prazne, što smiju sadržavati ako nisu prazne i sl.? Odgovor je DTD. DTD trebamo jer:

- Ne želimo stvarati novi skup oznaka svaki puta.
- Želimo provjerili ispravnost dokumenta.
- Želimo omogućili stvaranje dokumenta sa aplikacijom koja nam jednostavno neće dopustiti da generiramo loši dokument.
- Želimo omogućiti automatizaciju.

3.3.3.2 DTD

Da bi se XML dokument mogao ispravno protumačiti, potrebno je definirati i DTD. DTD za Primjer 3.3-3 prikazan je kroz .

Primjer 3.3-4 DTD za e-mail poruku

```
<!element email   (head, body)>
<!element head    (from, to+, cc*, subject)>
<!element from    (name?, address)>
<!element to      (name?, address)>
<!element name    (#PCDATA)>
<!element address (#PCDATA)>
<!element subject (#PCDATA)>
<!element body    (p | attach)*>
<!element p       (#PCDATA)>
<!element attach  EMPTY>
<!attlist attach  encoding (mime|binhex) "mime"
                  name     CDATA         #REQUIRED>
```

U nastavku će biti opisana struktura DTD-a.

3.3.3.3 Struktura DTD-a

DTD se piše prema određenim pravilima. Kratak pregled dan je u nastavku. DTD može biti lokalni ili vanjski:

- lokalni:
  <!DOCTYPE ime [….]>
- referenca na vanjski:
  <!DOCTYPE ime SYSTEM literal_sustava>

Oznake <!element>, <!attlist> i <!entity> pišu se unutar tijela DTD. Opis oznaka slijedi u nastavku:

- <!element> oznaka definira oznake (naziv, što mora sadržavati)
- <!attlist> oznaka definira atribute pojedinih oznaka
- <!entity> oznaka definira entitete

Pojedini entitet definira se na slijedeći način:

- <!ENTITY ime literal>
- <!ENTITY ime SYSTEM literal_sustava>
- <!ENTITY % ime literal>

Primjer 3.3-5 Jedan logički XML dokument koji se proteže kroz nekoliko datoteka

```
<!doctype book SYSTEM "book.dtd"
[
 <!entity naslov "Vrlo nezanimljiva knjiga">
 <!entity toc SYSTEM "toc.xml">
 <!entity chap1 SYSTEM "chapters/c1.xml">
 <!entity chap2 SYSTEM "chapters/c2.xml">
]>
<book>
<title>&naslov;</title>
<head>&toc;</head>
<body>
&chap1;
&chap2;
</body></book>
```

Definicije oznaka u DTD-u mogu biti sastavni dio XML dokumenta, ali i ne moraju, jer se one mogu spremiti u zasebnu datoteku, i zatim pozvati iz svakog XML dokumenta koji ih koristi. Slijedeća dva primjera ovo ilustriraju.

Primjer 3.3-6 XML sa referencom na DTD

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

Primjer 3.3-7 XML sa potpunim DTD-om

```
<?xml version="1.0"?>
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

Sada se može pokazati kako bi izgledao ispravno napisan Primjer 3.3-3.

Primjer 3.3-8 Potpuni XML dokument

```
<?xml version="1.0"?>
<!DOCTYPE email [
<!element email   (head, body)>
```

```
<!element head    (from, to+, cc*, subject)>
<!element from    (name?, address)>
<!element to      (name?, address)>
<!element name    (#PCDATA)>
<!element address (#PCDATA)>
<!element subject (#PCDATA)>
<!element body    (p | attach)*>
<!element p       (#PCDATA)>
<!element attach  EMPTY>
<!attlist attach  encoding (mime|binhex) "mime"
                  name     CDATA         #REQUIRED>
]>
<email>
  <head>
    <from>
      <name>Marko Čupić</name>
      <address>marcupic@pinus.cc.fer.hr</address>
    </from>
    <to>
      <name>Pero Perić</name>
      <address>perperic@pinus.cc.fer.hr</address>
    </to>
    <subject> XML specifikacija </subject>
  </head>
  <body>
    <p>Budući da si rekao da ti pošaljem XML
    specifikaciju ako je pronađem, šaljem ti XML
    1.0 specifikaciju, što je trenutno aktualna
    inačica, iako je već drugo izdanje.</p>
    <attach encoding="mime" name="xml-spec.html"/>
  </body>
</email>
```

Danas se XML počinje primjenjivati u mnoštvu različitih aplikacija. Neke od primjena navedene su:

- RDF (Resource Description Framework)
- OFX (Open Financial Exchange)
- CML (Chemical Markup Language)
- MML (Mathematical Markup Language)
- OSD (Open Source Distribution)
- XHTML

3.3.4 XHTML

XHTML nastao je kao proširenje HTML 4.0 specifikacije, i to je jezik baziran na XML specifikaciji. Trenutno aktualna specifikacija je XHTML 1.0. Naime, kako se je pokazalo da je u svom razvoju HTML specifikacija postala preglomazna i vrlo

teška, a u međuvremenu je razvijen XML koji donosi striktna pravila i jednostavnost, napravljena je fuzija ovih dvaju jezika te je HTML specifikacija prilagođena tako da u potpunosti zadovoljava XML zahtjeve. Stoga je svaki XHTML dokument ispravni XML dokument što povlači mogućnost pregleda, uređivanja i provjere standardnim XML alatima. XHTML dokumenti rade jednako dobro (a i bolje) u starim HTML 4.0 kompatibilnim preglednicima, a rade i u novim XHTML 1.0 kompatibilnim preglednicima.

Nadalje, XHTML dokumenti mogu koristiti aplikacije koje se oslanjaju na HTML DOM, kao i na XML DOM (Document Object Model).

Ljudi neprestano pronalaze nove načine kako svoje znanje i ideje strukturirati i opisati pomoću oznaka – XML dodavanje oznaka čini vrlo jednostavnim. Uz to, alternativni načini pristupa Internetu se neprekidno razvijaju; prema nekim procjenama, do 2002. godine pregled dokumenata sa Interneta u 75% slučajeva biti će preko alternativnih platformi, i upravo je XHTML razvijen imajući u vidu interoperabilnost sa velikim brojem platformi (ova prognoza možda je malo preuranjena, ali za još koju godinu sigurno neće biti).

4 SUSTAV ZA PRETRAŽIVANJE S DETALJIMA IMPLEMENTACIJE

Temeljeći se na zapažanjima iz prethodnog poglavlja, u nastavku je predložen sustav za pretraživanje koji se temelji na uporabi ontologija i agentskih sustava. Kako bi se agenti mogli izvoditi i seliti s računala na računalo, potrebno je definirati nužnu potpornu platformu – agentsku okolinu. Nakon toga treba definirati i osnovni oblik agenta – na koji način ga možemo napisati i slično. Ove dvije teme obrađene su u 4.1 i 4.2. Konkretan način na koji će se vršiti opisivanje dokumenata prikazan je u 4.3. Nakon toga objašnjen je način na koji se definiraju i obrađuju upiti (4.4), i zatim slijedi opis konkretne realizacije sustava sa svim komponentama (4.5).

4.1. OKOLINA ZA PODRŠKU AGENATA

U poglavlju 3.1 pokušali smo definirati agenta, tj. značenje pojma "agent". Sa programerskog stajališta, agenta je vrlo jednostavno definirati – to je dio programa. Ovakva jednostavna definicija ipak otkriva mnoštvo problema. Agent je program – znači da ga netko mora znati izvršavati; znači da umjesto korisnog posla taj dio programa može postati opasan po sustav koji ga izvodi te se treba zaštititi; znači da tom djelu programa mogu zatrebati neki resursi pa treba odlučiti što i u kolikoj mjeri staviti na raspolaganje za uporabu; i još mnoštvo pitanja i problema.

S druge strane, ako govorimo o mobilnom agentu, tada zahtjevi postaju još teži jer agent mora biti izveden tako da se u zadanom trenutku svi relevantni podaci o stanju agenta mogu prikupiti, zatim mora postojati mogućnost da se agent odvoji od svoje okoline koja ga je do tada izvršavala, i na kraju, nakon što se agent preseli u novu okolinu, i ta okolina mora znati kako izvršavati agenta. Ovo su vrlo veliki zahtjevi čak i u današnje doba, te pisanje agenta u programskim jezicima kod kojih je rezultat prevođenja niz instrukcija mikroprocesoru ne dolaze u obzir jer nema garancije da će sve okoline sklopovski biti izvedene jednako, odnosno bazirane na istom mikroprocesoru.

Jasno je, dakle, da mobilni agent mora biti napisan preko nekog od programskih jezika koji se ne prevode direktno u instrukcije mikroprocesora, već u nešto viši apstraktni kod koji nije ovisan o mikroprocesoru. Stoga se kao moguće rješenje nameće upravo Java programski jezik, jer se Java programi prevode u Java byte kod, i zahvaljujući tome Java programi mogu se izvoditi na svim računalima neovisno o mikroprocesoru na kojem je računalo bazirano. Naime, za različite mikroprocesore dovoljno je napisati Java Virtual Machine koji će tada biti u stanju izvoditi byte kod, neovisno o tome tko ga je proizveo i na kojem računalu.

Java programski jezik nameće se zbog još nekih specifičnosti. Naime, to je čisti objektni programski jezik, vrlo dobro je prihvaćen na Internetu te gotovo sva računala imaju instaliran Java Virtual Machine što im daje mogućnost izvođenja Java programa. Isto tako, Java ima ugrađeno rukovanje sa dretvama na vrlo jednostavan način a sinkronizacija je u svojoj osnovnoj izvedbi prisutna u samoj bazi jezika. Slijedeći veliki plus Javi su i ugrađeni mehanizmi serijalizacije / deserijalizacije objekata što automatski odvajanje agenta od njegove okoline čini daleko

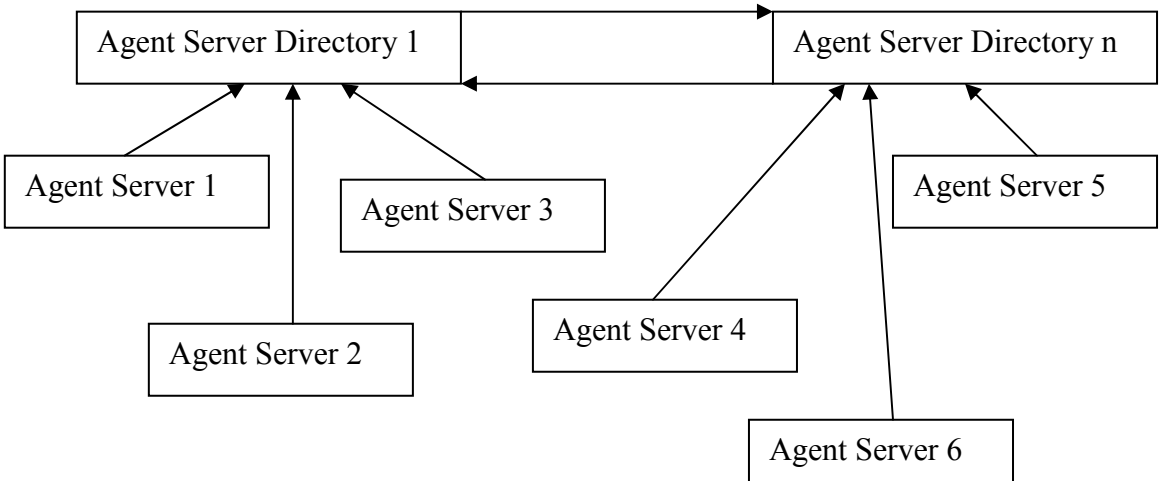
jednostavnijim poslom nego što je to moguće u klasičnim programskim jezicima (Basic, C, Pascal, …).

Isto tako, Java programski jezik dizajniran je imajući u vidu okolinu u kojoj će se izvršavati (Internet) pa je puno pažnje posvećeno upravo sigurnosti. Razvijeni su mehanizmi nadzora nad kodom koji se izvršava, i mehanizmi dodjeljivanja dozvola tako da se dosta jednostavno može odrediti što pojedini program smije, a što ne smije raditi. I konačno, Java programski jezik omogućuje i dosta jednostavan rad sa utičnicama (sockets), čime je i dohvat podataka sa mreže sveden na prihvatljivo laganu razinu.

Ovo su osnovni razlozi zbog kojih je kao jezik za razvoj agenata u ovom seminaru prihvaćen Java programski jezik.

4.1.1 Elementi okoline za podršku agenata

Minimalna okolina koja se preporuča ovim radom sastoji se od dva dijela: komponente "Agent Server" te komponente "Agent Server Directory". Ideja je slijedeća: svako računalo koje želi omogućiti izvršavanje agenata pokrenuti će komponentu "Agent Server". "Agent Server" je osnovni element okoline koji zna kako prihvatiti agenta, odrediti koje resurse agent smije koristiti, kako agenta poslati do nekog drugog "Agent Server"-a (ukoliko to agent zatraži), te u konačnici i kako izvoditi agenta. Ujedno "Agent Server" može ponuditi i neke dodatne mogućnosti koje agent može koristiti ukoliko ih smatra korisnima. Jedna od preporučenih dodatnih mogućnosti je hibernacija agenta, o kojoj će biti govora kasnije, te mogućnost garantiranja perzistencije agenta u određenoj mjeri (kako se ne bi dogodilo da netko zabunom izvuče utikač od struje pa da svi agenti budu izgubljeni).



Slika 4-1 Međudjelovanje komponenti sustava

Drugi bitan element okoline je komponenta "Agent Server Directory". "Agent Server Directory" je servis koji čuva katalog "Agent Servera" komponenti. U trenutku kada agent odluči da bi zbog nekog razloga bilo poželjno premjestiti se u neku drugu okolinu (sa računala 1 na računalu 2), agent će se raspitati upravo preko komponente "Agent Server Directory"-a da li na računalu 2 postoji komponenta "Agent Server", i ako postoji, kako do nje doći (odnosno, na kojim vratima [portu] taj "Agent Server"

prima zahtjeve). Stoga je vidljivo da će "Agent Server Directory" biti prisutan tek na nekoliko računala, a zadatak svih "Agent Servera" je da se prilikom svog podizanja prijave na jedan od "Agent Server Directory"-a, te da se prilikom prestanka rada odjave s tog istog "Agent Server Directory"-a. Isto tako, "Agent Server Directory"-i bi se trebali povezivati u mrežu kako bi izgradili jedan globalni katalog (npr na način kako se danas povezuju DNS servisi).

4.1.2 Komponenta "Agent Server"

Komponenta Agent Server je osnovni element okoline zadužen za direktni rad sa agentima, i komunikaciju sa drugim Agent Serverima. Osnovni zadaci "Agent Servera" su:

- Prijavljivanje na jedan od "Agent Server Directory"-a prilikom startanja komponente.
- Prihvat agenta.
- Izvršavanje agenta.
- Transport agenta.
- Hibernacija i buđenje agenta.
- Proslijeđivanje poruka koje šalje agent nekom drugom agentu u toj istoj okolini, ili u nekoj drugoj okolini.
- Odjavljivanje s "Agent Server Directory"-a prilikom gašenja komponente.

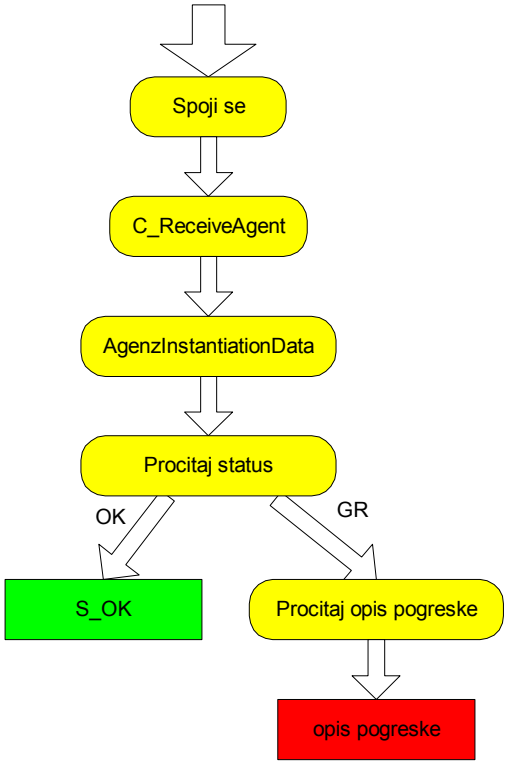
Prihvat agenta / transport agenta dva su posla koja su međusobno povezana. Naime, s jedne strane nalazi se Agent Server koji agenta šalje dok je sa druge strane Agent Server koji prihvaća agenta. Za ovaj zadatak razvijen je poseban protokol. Isti protokol koristi se i u slučaju slanja agenta u prvu okolinu (tada ga ne šalje sama okolina).

U nastavku su grafički prikazane faze definirane protokolom za različite vrste zadataka. Agentska okolina (Agent Server) agentima nudi tri funkcije vezane uz komunikaciju sa drugim okolinama:funkciju pronalaženja agenta, funkciju slanja poruke agentu te funkciju slanja samog agenta (bilo originala, bilo klona).

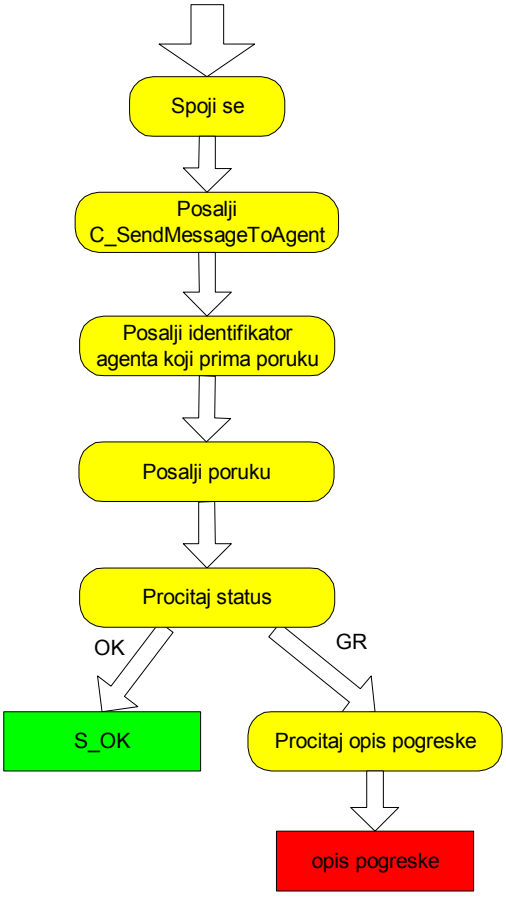
Za komuniciranje sa drugim agentima agent od okoline zahtjeva stvaranje AgentProxy objekta preko kojega teče sva komunikacija. Na ovaj način agent pošiljatelj oslobođen je od razmišljanja da li komunicira sa lokalnim agentom ili ne, te na koji način da to obavi.

Na grafičkim prikazima u žutim zaobljenim pravokutnicima nalaze se akcije koje se obavljaju ili podaci koji se šalju. Zeleni pravokutnici predstavljaju uspješan završetak započete radnje i podatak koji se u tom slučaju šalje. Crveni pravokutnici predstavljaju neuspjeh i podatke koji se u tom slučaju šalju. Sivi trokut predstavlja grananje, a sivi romb ponavljanje (konačno ili beskonačno).

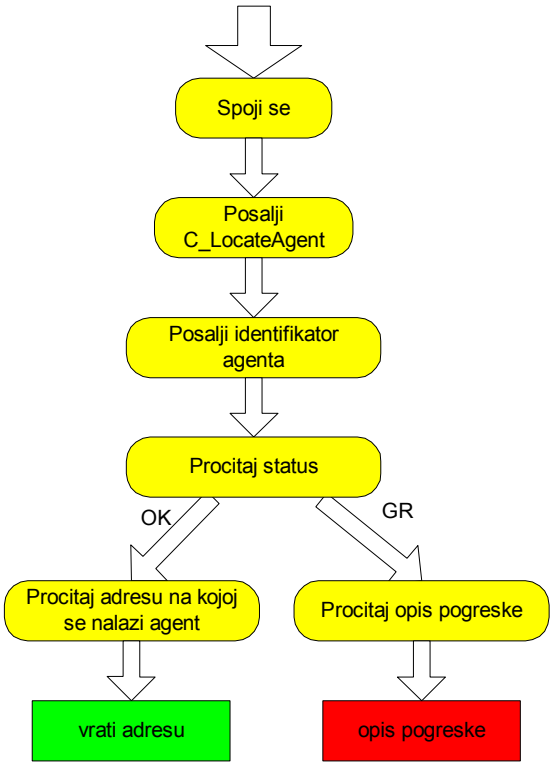
4.1.2.1 Protokoli na klijentskoj strani



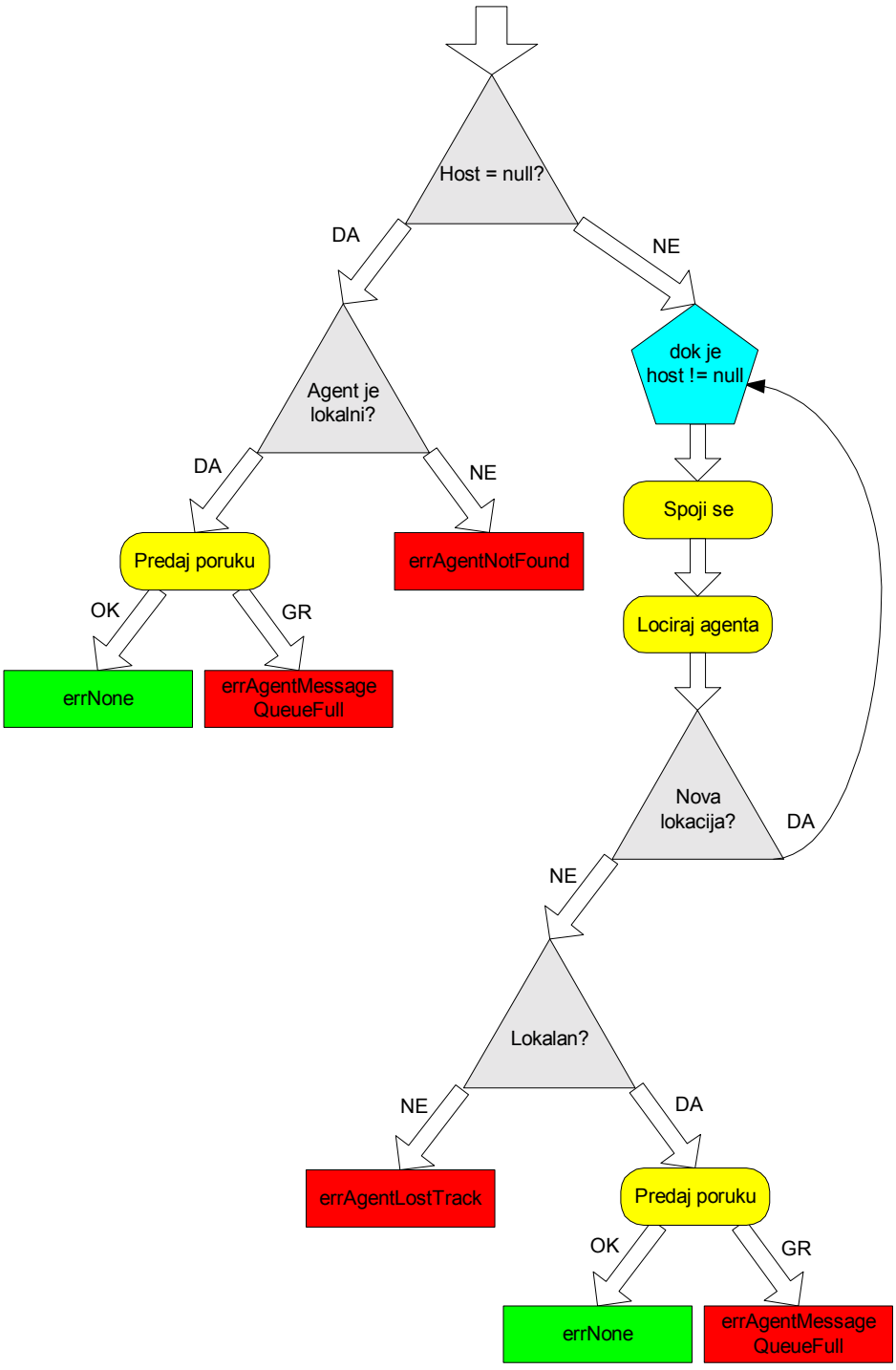
Slika 4-2 Protokol slanja agenta



Slika 4-3 Protokol slanja poruke agentu

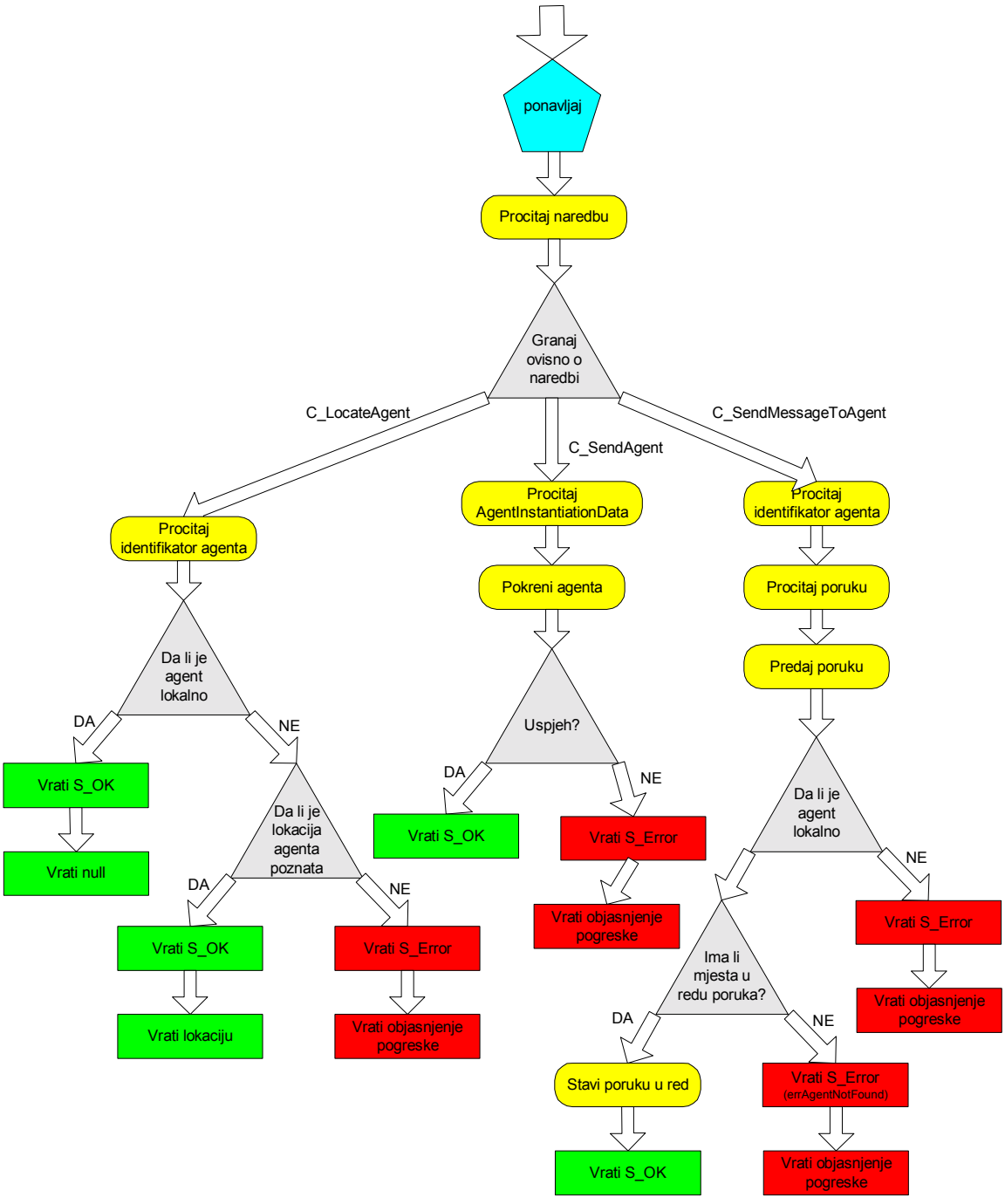


Slika 4-4 Protokol traženja agenta



Slika 4-5 Predaja poruke preko AgentProxy objekta

4.1.2.2 Protokoli na poslužiteljskoj strani



Slika 4-6 Protokol poslužitelja

4.1.3 Komponenta "Agent Server Directory"

Servis pod nazivom Agent Server Directory predlaže se u svrhu izgradnje jednog globalnog kataloga Agent Servera, kako bi agenti mogli doznati da li na određenom računalu postoji Agent Server.

Za ove potrebe razvijen je protokol koji koriste agenti ali i Agent Serveri za komunikaciju sa Agent Server Directory-em. Riječ je o protokolu kojim se Agent Serveri prijavljuju / odjavljuju na globalni katalog, te kojim agenti ispituju da li na zadanom računalu postoji Agent Server. Ovo je čisti tekstualni protokol, i opisan je u nastavku.

Protokol definira četiri osnovne naredbe: BYE, EXISTS, REGISTER te UNREGISTER. Svaka naredba mora biti terminirana nizom ascii znakova 13, 10. U prilogu su prikazane definirane naredbe protokola i njihov opis.

Pod pojmom klijent smatra se računalo koje je iniciralo vezu (agent ili Agent Server), a pod pojmom server smatra se Agent Server Directory.

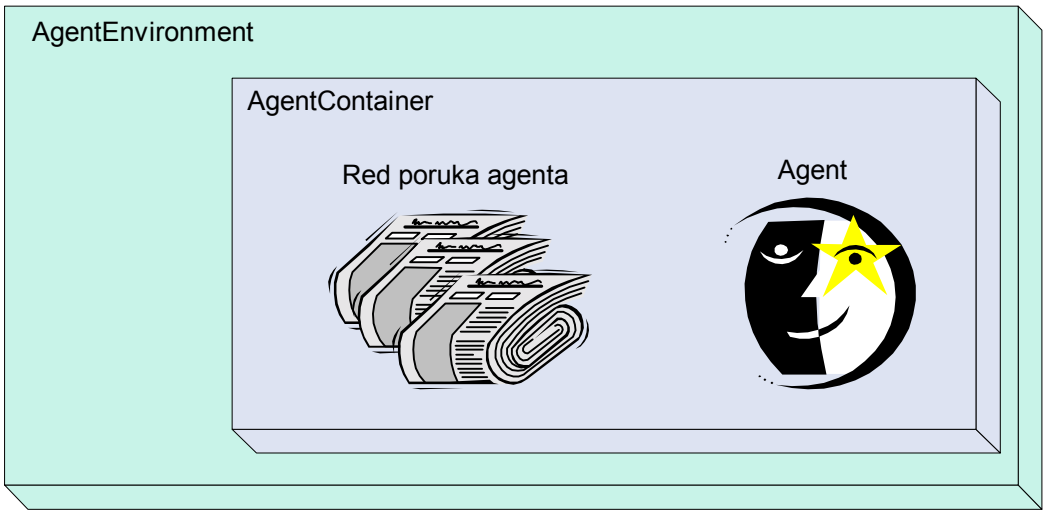
U slučaju pogreške, server odgovara jednom linijom koja započinje sa tekstom "ERROR:" iza kojeg slijedi jedna praznina (ascii 32) te zatim kratko objašnjenje greške.

Na neka pitanja još treba odgovoriti, a to su pitanja poput kako efikasno izgraditi mrežu Agent Server Directory-a kako bi pretraživanje kataloga koji tako nastaje bilo što brže. U svojoj osnovnoj izvedbi, implementiran je Agent Server Directory koji se ne povezuje u mrežu sa drugim Agent Server Directory-ima.

4.2. PRIJEDLOG OSNOVNOG OBLIKA AGENTA

Kako bi se omogućila mobilnost agenta, agent mora biti izveden na način koji omogućava da se cijela komunikacija agenta i okoline odvija preko dobro definiranog sučelja, budući da okolina ne može znati (a ponekad je i poželjno da ne zna) internu strukturu i građu agenta. Ista opaska vrijedi i za agenta, jer što manje agent zna o internoj strukturi okoline u kojoj se izvršava, to je manja opasnost od namjernog narušavanja funkcionalnosti okoline. Prije same građe agenta upoznati ćemo se sa položajem agenta u agentskoj okolini, kao i definiranim sučeljima te načinom na koji se odvijaju određeni zadaci koje agent može zahtjevati.

Slika 4-7 prikazuje položaj agenta u jednoj agentskoj okolini.



Slika 4-7 Agentska okolina, spremnik agenta, agent

Agent koristi dva osnovna sučelja: AgentContainerI te AgentEnvironmentI. Preko sučelja AgentContainerI agent ostvaruje najveći dio zadataka vezanih za komunikaciju sa ostalim agentima, kao i transport i kloniranje. Detaljan opis ovih sučelja dat je u nastavku.

4.2.1 Sučelje AgentContainerI

Popis svih metoda koje definira ovo sučelje prikazan je u nastavku.

```
public interface AgentContainerI {
      public AgentMessage getMessage( long timetowait );
      public AgentEnvironmentI getAgentEnvironment();
      public AgentIdentifier cloneMe(AgentProtocolResult apr);
      public AgentIdentifier cloneAndTransportMe(String host,
                  short port, AgentProtocolResult apr);
      public AgentIdentifier transportMe(String host, short port,
                  AgentProtocolResult apr);
      public AgentIdentifier whoAmI();
      public AgentProxy getAgentProxy(AgentIdentifier a);
      public AgentProxy getAgentProxy(AgentIdentifier a, String
host,
                  short port);
      public void startMessageThread();
      public void stopMessageThread();
      public void blockMessageThread();
      public void unblockMessageThread();
      public Object getService(String serviceName,Object[]
parameters);
}
```

Metoda *getMessage* iz reda poruka agenta dohvaća prvu poruku. Ukoliko je red prazan, čeka je dok ne pristigne poruka, ili dok ne prođe vrijeme *timetowait* koje se predaje kao argument. Metoda *getAgentEnvironment* pribavlja agentu sučelje prema samoj agentskoj okolini. Metoda *cloneMe* pokreće proces kloniranja agenta, i klonirani agent se također pokreće u trenutnoj agentskoj okolini. Metoda *cloneAndTransportMe* obavlja proces kloniranja agenta i klon šalje u novu agentsku

okolinu. Metoda *transportMe* trenutnog agenta transportira u novu agentsku okolinu. Metoda *WhoAmI* vraća identifikator samog agenta. Ovo je izvedeno na ovaj način iz sigurnosnih razloga, te sam agent nikada ne može doći u priliku izmjeniti svoj identifikator (i time doći u priliku lažno se predstavljati, što bi moglo imati određenih sigurnosnih posljedica). Metoda *getAgentProxy* stvara proxy objekt prema drugom agentu koji se može nalaziti lokalno ili pak u nekoj drugoj okolini. Agenti međusobno komuniciraju preko proxy objekata. Metode startMessageThread, stopMessageThread, blockMessageThread te unblockMessageThread posvećene su radu sa redom poruka agenta, o čemu će više riječi biti u nastavku, kada se objasni građa agenta.

Metoda *getService* naročito je važna metoda jer omogućava različitim okolinama da nude agentima različite servise koji nisu standardni dio agentskih okolina.

4.2.2 Sučelje AgentEnvironmentI

Popis svih metoda koje definira ovo sučelje prikazan je u nastavku.

```
public interface AgentEnvironmentI {
      public String getEnvironmentHost();
      public short getEnvironmentPort();
}
```

Metoda *getEnvironmentHost* vrača naziv računala na kojem se izvršava trenutna agentska okolina. Metoda *getEnvironmentPort* vrača broj porta na kojem agentska okolina očekuje komunikacijske zahtjeve.

4.2.3 AgentProxy objekti

Već je spomenuto da agenti međusobno komuniciraju preko AgentProxy objekata. AgentProxy razred ima slijedeću strukturu:

```
public class AgentProxy {
      public boolean sendMessage(AgentProtocolResult res,
            AgentMessageDataI msg);
      public String getHost();
      public short getPort();
}
```

Agent ove objekte nikada ne stvara direktno, već preko zahtjeva upućenog AgentContainer komponenti. Preko ovog objekta agent može slati poruke drugom agentu (metoda *sendMessage*), kao i doznati gdje se drugi agent trenutno nalazi (metode *getHost* i *getPort*, nakon uspješnog slanja barem jedne poruke).

Postupak traženja agenta izveden je na slijedeći način: svaka agentska okolina pamti na koju je lokaciju transportirala pojedine agente (neko određeno vrijeme). Ukoliko u okolinu stigne zahtjev za lociranjem dotičnog agenta, okolina će vratiti posljednju adresu na koju je agent poslan. Tada će se okolina koja pokušava pronaći agenta otpojiti, spojiti na tu novu okolinu i postupak će se ponavljati sve dok se agent ne

pronađe, ili dok se naiđe na okolinu koja nema podataka o daljnjem putovanju agenta.

4.2.4 Građa agenta

Agent je za potrebe ovog rada definiran osnovnim Java razredom "Agent". Ovaj razred ima slijedeću strukturu.

```
public class Agent implements java.io.Serializable {
     public AgentIdentifier parentAgent;
     public transient AgentContainerI aci;
     public void init();
     public void destroy();
     public void transported();
     public void awakened();
     public void cloned(boolean sameEnvironment);
     public void execute();
     public void messageArrived(AgentMessage am);
}
```

Polje *parentAgent* sadrži identifikator agenta koji je stvorio ovog agenta (prilikom kloniranja). Sam identifikator građen je od dva dijela: prvog dijela koji je zajednički roditelju i svim njegovim klonovima, i drugog dijela koji je jedinstven za svakog agenta. Polje *aci* je referenca na AgentContainerI sučelje preko kojeg agent obavlja sve akcije u trenutnoj agentskoj okolini.

Metoda *init* poziva se samo jednom, prilikom stvaranja agenta, i u njoj se smješta kod koji inicijalizira sve potrebno za rad samog agenta. Ova metoda se ne poziva prilikom kloniranja ili transporta. Metoda *destroy* poziva se kada agent završava sa radom. To je prilika agentu da oslobodi sve zauzete resurse. Metoda *transported* poziva se nakon što je agent preseljen u novu okolinu i uspješno rekonstruiran. Metoda *awakened* poziva se nakon što je agent probuđen iz hibernacije. Metoda *cloned* poziva se nakon što je agent kloniran. Metoda *execute* sadrži kod koji obavlja glavni zadatak agenta. Ona se poziva u nekoliko različitih scenarija, a više o tome biti će u nastavku. Konačno, metoda *messageArrived* poziva se asinkrono sa izvođenjem ostalih metoda u trenutku kada stigne nova poruka u red poruka agenta. Redoslijed poziva pojedinih metoda prikazuje Tablica 4-1.

Tablica 4-1 Redoslijed pozivanja osnovnih metoda agenta

Redoslijed...	Kada se pojavljuje...

init();
execute();
....
destroy();
	Prvo pokretanje agenta.

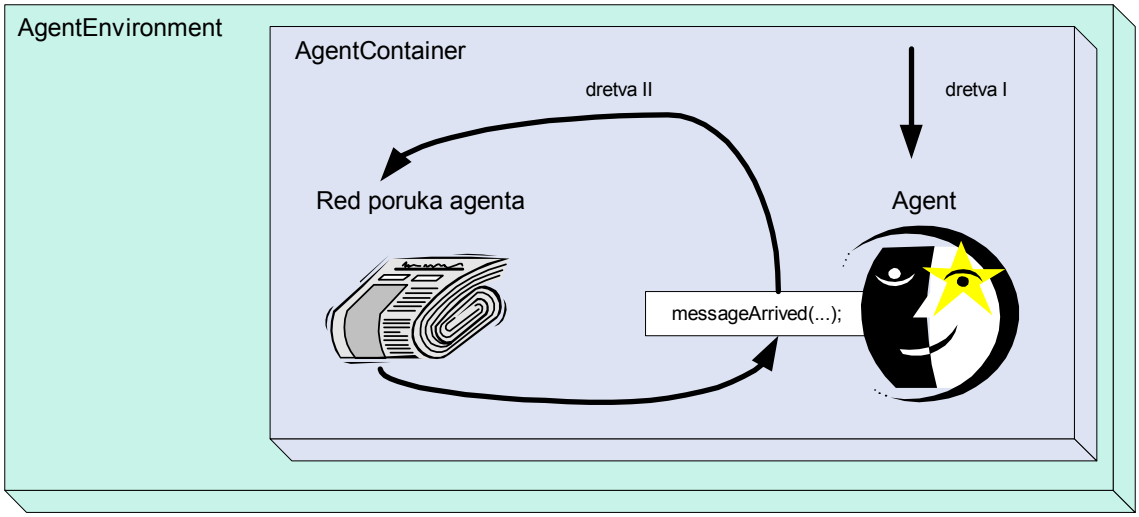
cloned();
execute();
...
	Nakon što je stvoren novi klon.

destroy();	

transported();
execute();
...
destroy();
	Nakon što je agent transportiran u novu okolinu.

awakened();
execute();
...
destroy();
	Nakon što je agent probuđen iz hibernacije.

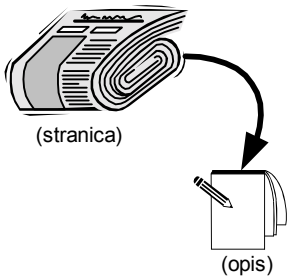
Agentska okolina inherentno je višedretveni program. Svaki agent minimalno koristi jednu dretvu koja izvodi kod agenta, a najčešće dvije: jednu koja izvodi kod agenta i jednu koja asinkrono poziva *messageArrived()* metodu kada pristigne nova poruka. Situaciju prikazuje Slika 4-8.



Slika 4-8 Dretve vezane uz agenta

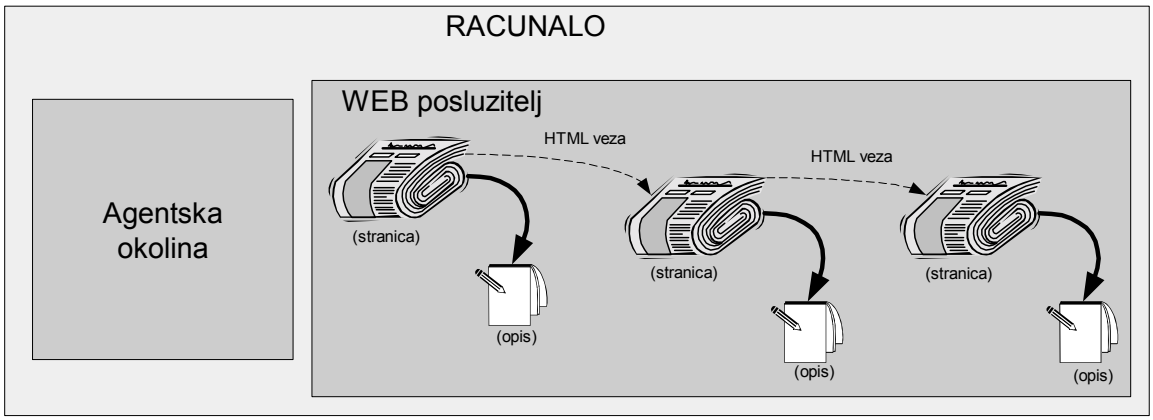
4.3. PRIJEDLOG NAČINA OPISIVANJA DOKUMENATA

Kroz prethodna poglavlja već je objašnjeno da računala danas još ne mogu razumjeti sadržaj pisanih dokumenata. Zbog toga ćemo koristiti slijedeći pristup. Svakom dokumentu (vidi Slika 4-9) pridjeliti će se dodatni dokument u kojem će pomoću određenih ontologija biti opisan sadržaj dokumenta. Koristiti ćemo ontologije zbog toga što će agenti u određenoj mjeri "razumjeti" takve dokumente.



Slika 4-9 Individualni opis dokumenta

U trenutku kada agent dođe u odredišnu agentsku okolinu, ovisno o servisima koje okolina nudi moguća su dva pravca vršenja pretrage. Ukoliko okolina ne nudi servis preko kojega se može dobiti cjelokupni sadržaj svih opisnih dokumenata na jednom mjestu (Slika 4-10), agent će morati krenuti u fazu prikupljanja opisnih datoteka – krenuti će od naslovne web stranice na dotičnom računalu, i zatim će analizirati sve HTML dokumente, prikupljati opisne datoteke, izvlačiti sve linkove i slijediti ih dok tako ne obiđe sve dokumente na dotičnom računalu. Ukoliko u agentskoj okolini postoji servis koji ovo obavlja automatski (vidi Slika 4-11), prethodni korak se može preskočiti.

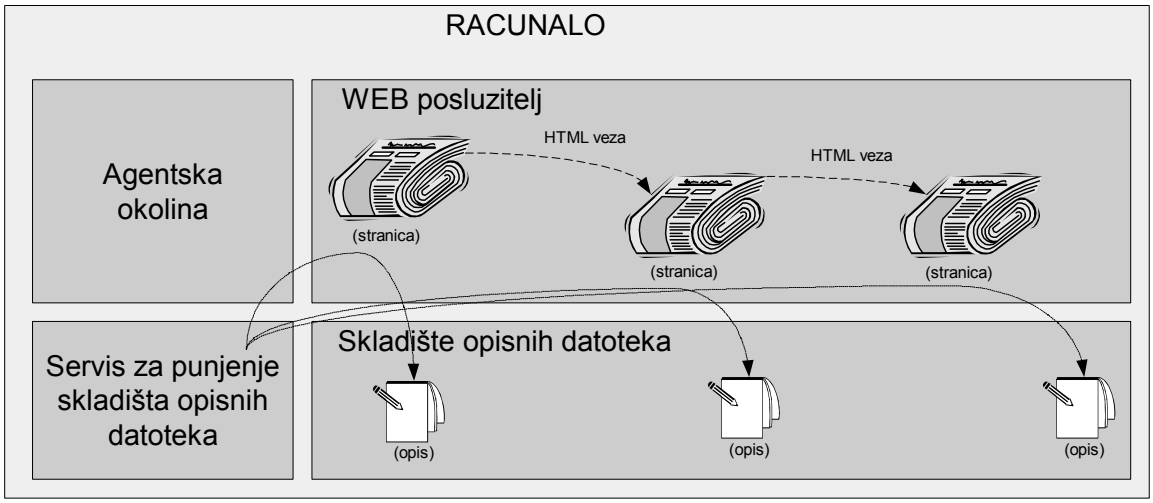


Slika 4-10 Struktura računala s pojedinačnih opisom dokumenata

Automatsko prikupljanje opisnih datoteka može se izvesti na nekoliko načina, i to ne mora nužno biti stvarno fizičko prikupljanje. Mogučnosti su slijedeće.

- Postoji jedan centralizirani opisnik svih datoteka na određenom računalu koji pune i održavaju sami autori dokumenata.
- Dokumenti se opisuju pojedinačno, a sustav ima "pauka" koji u određenim vremenskim intervalima analizira sve dokumente na računalu i prikuplja opisne datoteke.
- Dokumenti se opisuju pojedinačno, a postoji i skladište opisnih datoteka. Autori stranica opisne datoteke ostavljaju u skladištu, umjesto da povezuju dokumente i opisne datoteke.

Analizira li se ukratko postupak koji svaki agent obavlja kada ne postoji jedno mjesto na kojemu bi se moglo dohvatiti sve opisne datoteke, vidimo da svaki agent zapravo simulira jednu varijantu gore opisanog sustava. Već samo to opravdava implementaciju ovakvog sustava, jer bi inače u realnom svijetu gdje bi se pretraživanje vršilo puno puta u kratkom vremenskom intervalu došlo do zagušenja sustava jer bi svi agenti pokušavali dohvatiti sve web stranice.



Slika 4-11 Računalo sa servisom za prikupljanje opisnih datoteka

Kada agent pribavi sve opisne datoteke, započinje sa procesom "čitanja" informacija i traženja odgovarajućih dokumenata. Način na koji se vrši analiza biti će opisan u sljedećem poglavlju, a u nastavku će biti prikazana ontologija *feronto1* razvijena za potrebe ovog rada.

4.3.1 Ontologija feronto1

Budući da je cilj ovog rada opisivanje dokumenata, razvijena je ontologija *feronto1*, koja opisuje neke osnovne objekte i relacije među njima koji će se koristiti za opis dokumenata. Ontologija je prikazana u nastavku.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:fer1="http://www.fer.hr/#">

        <!--=====================================================-->
        <!--              ONTOLOGY DEFINITION                    -->
        <!--=====================================================-->

        <rdfs:Class rdf:about="http://www.fer.hr/#Document">
        </rdfs:Class>

        <rdfs:Class rdf:about="http://www.fer.hr/#HTMLDocument">
```

```
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#Document"/>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.fer.hr/#Picture">
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#Document"/>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.fer.hr/#AudioFile">
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#Document"/>
</rdfs:Class>

<rdf:Property rdf:about="http://www.fer.hr/#DocumentTitle">
        <rdfs:label>Title of general document.</rdfs:label>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Document"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#Title">
        <rdfs:label>Title of HTML document.</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#DocumentTitle"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#HTMLDocument"/>
</rdf:Property>

<rdfs:Class rdf:about="http://www.fer.hr/#Concept">
</rdfs:Class>

<rdf:Property rdf:about="http://www.fer.hr/#conceptLabel">
        <rdfs:label>This is a name of concept</rdfs:label>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Concept"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#relatedTo">
        <rdfs:label>Document Z is related to concept C on some
                unspecified way.</rdfs:label>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Document"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#Concept"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#isAbout">
        <rdfs:label>Document Z is about concept C</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#relatedTo"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Document"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Course"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#moreAbout">
        <rdfs:label>See more about this concept on given host</rdfs:label>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Concept"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#relatedConcept">
        <rdfs:label>Concept A is related in some way with concept B</rdfs:label>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Concept"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#Concept"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#dependsOn">
        <rdfs:label>Understanding of concept A depends on concept B</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#relatedConcept"/>
```

```
        <rdfs:domain rdf:resource="http://www.fer.hr/#Concept"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#Concept"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#extendsConcept">
        <rdfs:label>Concept A is extension of concept B</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#dependsOn"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Concept"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#Concept"/>
</rdf:Property>


<rdfs:Class rdf:about="http://www.fer.hr/#Faculty">
</rdfs:Class>

<rdfs:Class rdf:about="http://www.fer.hr/#Department">
</rdfs:Class>

<rdfs:Class rdf:about="http://www.fer.hr/#Course">
</rdfs:Class>


<!-- WebPage URI is actual URL of web page. -->
<!-- In other words, URI must exists!       -->

<rdfs:Class rdf:about="http://www.fer.hr/#WebPage">
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#HTMLDocument"/>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.fer.hr/#WebSite">
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#WebPage"/>
</rdfs:Class>


<rdf:Property rdf:about="http://www.fer.hr/#Name">
        <rdfs:label>Name in general sense</rdfs:label>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#FacultyName">
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#Name"/>
        <rdfs:label>Name of faculty</rdfs:label>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#DepartmentName">
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#Name"/>
        <rdfs:label>Name of department</rdfs:label>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#CourseName">
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#Name"/>
        <rdfs:label>Name of course</rdfs:label>
</rdf:Property>

<rdf:Property rdf:about="http://www.fer.hr/#HomePage">
        <rdfs:label>Points to homepage of Faculty, Department
                or Course</rdfs:label>
        <rdfs:range rdf:resource="http://www.fer.hr/#WebPage"/>
</rdf:Property>
```

```
    <rdf:Property rdf:about="http://www.fer.hr/#isPartOf">
        <rdfs:label>Some resource is partOf another resource</rdfs:label>
    </rdf:Property>

    <rdf:Property rdf:about="http://www.fer.hr/#belongsTo">
        <rdfs:label>Some resource belongs to another resource</rdfs:label>
    </rdf:Property>

    <rdf:Property rdf:about="http://www.fer.hr/#belongsToWebSite">
        <rdfs:label>WebPage X belongs to web site Y</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#belongsTo"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#WebPage"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#WebSite"/>
    </rdf:Property>

    <rdf:Property rdf:about="http://www.fer.hr/#belongsToFaculty">
        <rdfs:label>Department X belongs to faculty Y</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#belongsTo"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Department"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#Faculty"/>
    </rdf:Property>

    <rdf:Property rdf:about="http://www.fer.hr/#belongsToDepartment">
        <rdfs:label>Course X belongs to department Y</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://www.fer.hr/#belongsTo"/>
        <rdfs:domain rdf:resource="http://www.fer.hr/#Course"/>
        <rdfs:range rdf:resource="http://www.fer.hr/#Department"/>
    </rdf:Property>

</rdf:RDF>
```

Ontologija definira slijedeće razrede:

- `FER1:Document` – opći tip dokumenta.
- `FER1:HTMLDocument` – dokument html tipa, podrazred od FER1:Document.
- `FER1:Picture` – slikovni dokument, podrazred od FER1:Document.
- `FER1:AudioFile` – audio dokument, podrazred od FER1:Document.
- `FER1:Concept` – koncept.
- `FER1:Faculty` – fakultet.
- `FER1:Department` – odjel.
- `FER1:Course` – tečaj, seminar, kolegij.
- `FER1:WebPage` – Web stranica, podrazred od FER1:HTMLDocument.
- `FER1:WebSite` – Tematski vezana skupina stranica, sam objekt ujedno predstavlja i početnu stranicu; podrazred od FER1:WebPage.

Ontologija definira slijedeća svojstva razreda:

- `FER1:DocumentTitle` – naslov dokumenta; svaki objekt tipa FER1:Document može imati ovo svojstvo.

- `FER1:Title` – naslov HTML dokumenta; svaki objekt tipa FER1:HTMLDocument može imati ovo svojstvo. Podsvojstvo od FER1:DocumentTitle.
- `FER1:ConceptLabel` – naziv koncepta; svaki objekt tipa FER1:Concept može imati ovo svojstvo.
- `FER1:relatedTo` – određeni dokument odnosi se na / povezan je sa određenim konceptom. Svaki objekt tipa FER1:Document može imati ovo svojstvo, a vrijednost svojstva mora biti objekt tipa FER1:Concept.
- `FER1:isAbout` – određeni dokument govori o određenom konceptu, ili određeni kolegij obrađuje određeni koncept. Svaki objekt tipa FER1:Document ili FER1:Course može imati ovo svojstvo, a vrijednost svojstva mora biti objekt tipa FER1:Concept.
- `FER1:moreAbout` – više o određenom konceptu može se pronači na navedenom računalu. Svaki objekt tipa FER1:Concept može imati ovo svojstvo. Ovo je primjer izvedbe semantičkih veza – veza za koje agent zna kamo ga vode i da li ih se isplati slijediti ili ne.
- `FER1:relatedConcept` – koncept je povezan s drugim konceptom na neki neodređeni način. Svaki objekt tipa FER1:Concept može imati ovo svojstvo, a vrijednost svojstva mora biti objekt tipa FER1:Concept.
- `FER1:dependsOn` – koncept ovisi o drugom konceptu, odnosno da bismo razumjeli ovaj koncept, moramo najprije razumjeti drugi koncept. Svaki objekt tipa FER1:Concept može imati ovo svojstvo, a vrijednost svojstva mora biti objekt tipa FER1:Concept.
- `FER1:extendsConcept` – koncept je proširenje drugog koncepta, odnosno da bismo razumjeli ovaj koncept, moramo najprije razumjeti drugi koncept. Primjer su koncepti "Skup" i "Neizraziti skup", gdje je koncept "Neizraziti skup" očito proširenje koncepta "Skup". Svaki objekt tipa FER1:Concept može imati ovo svojstvo, a vrijednost svojstva mora biti objekt tipa FER1:Concept.
- `FER1:Name` – ime resursa u općenitom smislu.
- `FER1:FacultyName` – ime fakulteta. Podsvojstvo od FER1:Name.
- `FER1:DepartmentName` – ime odjela. Podsvojstvo od FER1:Name.
- `FER1:CourseName` – ime kolegija. Podsvojstvo od FER1:Name.
- `FER1:HomePage` – adresa "domaće stranice". Kao vrijednost mora poprimiti objekt tipa FER1:WebPage.
- `FER1:isPartOf` – resurs je dio nekog drugog resursa.
- `FER1:belongsTo` – resurs pripada drugom resursu.
- `FER1:belongsToWebSite` – web stranica pripada određenom skupu stranica (engl. site-u). Objekti tipa FER1:WebPage mogu imati ovo svojstvo, a kao vrijednost svojstvo mora poprimiti objekt tipa FER1:WebSite. Podsvojstvo od FER1:belongsTo.
- `FER1:belongsToFaculty` – odjel pripada određenom fakultetu. Objekti tipa FER1:Department mogu imati ovo svojstvo, a kao vrijednost svojstvo mora poprimiti objekt tipa FER1:Faculty. Podsvojstvo od FER1:belongsTo.
- `FER1:belongsToDepartment` – kolegij pripada određenom odjelu. Objekti tipa FER1:Course mogu imati ovo svojstvo, a kao vrijednost svojstvo

mora poprimiti objekt tipa FER1:Department. Podsvojstvo od FER1:belongsTo.

Nešto detaljniji primjer koji pokazuje način opisa dokumenata pomoću ove ontologije, naveden je u .

Primjer definira:

- *Fakultet* – FER
- *Odjel* – ZEMRIS, pripada FER-u
- *Kolegije* – IS, ML, EFNC, pripadaju ZEMRIS-u; navodi se i popis koncepata koji se obrađuju u pojedinom kolegiju.
- *Koncepti* – "Set", "FuzzySet", "Relation", ..., zajedno sa međuovisnostima; npr. "FuzzySet" je proširenje koncepta "Set".
- *Skupovi stranica* (engl. web sites) – "FER home page", "ZEMRIS home page", "IS Home page" i sl. s međuodnosima (npr. "ZEMRIS home page" pripada "FER home page"-u, barem gledajući logičku strukturu).
- *Pojedine stranice* – stranice na kojima se obrađuju pojedine teme, zajedno sa konceptima na koje se te teme odnose.

Sada kada je definirana ontologija i kada imamo određen broj dokumenata opisan tom ontologijom, možemo krenuti u opis metoda pretraživanja i zaključivanja. Za potrebe ovog rada pretpostaviti ćemo agentsku okolinu koja ima implementiranu uslugu automatskog prikupljanja opisnih datoteka, te ćemo smatrati da je prethodni primjer rezultat spajanja prikupljenih opisnih datoteka u jednu veliku bazu.

Prirodno se postavlja pitanje – što sada možemo tražiti, i kako to zadati ono što tražimo? Npr. Zanima nas "*koje koncepte obrađuju dokumenti vezani za kolegij EFNC*"?

4.4. OBRADA UPITA I SUSTAV ZA ZAKLJUČIVANJE

Kako bismo mogli vršiti određene upite, potrebno je razjasniti kako se točno pohranjuju RDF dokumenti. Naime, kada to shvatimo, način za postavljanje upita doći će prirodno. RDF dokument prevodi se u niz tvrdnji, takozvanih tripleta: (subjekt, predikat, objekt). Npr. poslužimo se djelom ontologije feronto1:

```
<rdfs:Class rdf:about="http://www.fer.hr/#Document">
</rdfs:Class>
```

Ovaj isječak prevodi se točno u jedan triplet (pretpostavimo da je prethodno definirano fer1=" http://www.fer.hr/#", kao i značenja rdf odnosno rdfs prefiksa):

(fer1:[Document], rdf:[type], rdfs:[Class])

Prema RDF terminologiji, sve što se definira RDFom su resursi. Tako je npr. definirani objekt "http://www.fer.hr/#Document" jedan resurs, rdfs:Class drugi resurs

itd. Jedino što nisu resursi su literali – obični nizovi znakova (tekst) koji nikada ne mogu biti na mjestu subjekta ili predikata; literal se može pojaviti isključivo na mjestu objekta. Literali se obično pišu kao tekst pod navodnicima, pa da bi se izbjeglo mješanje sa resursima, u ovom radu resursi će se označavati u uglatim zagradama. Specijalan oblik sintakse će se koristiti kada se resurs definira pomoću prethodno definiranog prefiksa; tada prefiks i dvotočka idu izvan uglatih zagrada, a preostali dio naziva resursa (odnosno URI-ja) slijedi u uglatim zagradama. Npr. [http://www.fer.hr/#Document], odnosno fer1:[Document] su resursi (i to isti!), a "http://www.fer.hr/#Document" je literal.

Pogledajmo još jedan isječak iz feronto1:

```
<rdfs:Class rdf:about="http://www.fer.hr/#HTMLDocument">
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#Document"/>
</rdfs:Class>
```

Ovo se prevodi u dva tripleta:

(fer1:[HTMLDocument], rdf:[type], rdfs:[Class])
(fer1:[HTMLDocument], rdfs:[subClassOf], fer1:[Document])

Te isječak iz feronto1:

```
<rdfs:Class rdf:about="http://www.fer.hr/#HTMLDocument">
        <rdfs:subClassOf rdf:resource="http://www.fer.hr/#Document"/>
</rdfs:Class>
```

Ovo se prevodi u dva tripleta:

(fer1:[HTMLDocument], rdf:[type], rdfs:[Class])
(fer1:[HTMLDocument], rdfs:[subClassOf], fer1:[Document])

Sada je jasno da se kompletna ontologija zajedno sa našom testnom bazom opisa prevodi u veliki broj tripleta.

Podsjetimo se sada još jedne bitne stvari vezane uz RDF: mogu se definirati razredi objekata te relacija podrazreda, kao i svojstava te relacija podsvojstva. Odnos razred / podrazred je sam po sebi jasan, međutim svojstvo / podsvojstvo zahtjeva kratko objašnjenje. Pretpostavimo da "resurs" Pero ima biološkog oca "resurs" Juru. Isto tako, "resurs" Ante ima očuha "resurs" Ivu. Govoreći terminologijom RDF-a, "biološki otac" i "očuh" su svojstva koja povezuju resurse Pero-Jura, Ante-Ivo. Može se uvesti novo svojstvo "otac": osoba X ima oca Y ako X ima biološkog oca Y ili ako X ima očuha Y. Svojstvo "otac" je tada generalno svojstvo, a biološki otac odnosno očuh su njegove specijalizacije, tj. podsvojstva. Iskazivanje ovakvih odnosa RDF nam omogućava upravo svojstvom rdfs:subPropertyOf.

Najjednostavnija vrsta upita koju možemo postaviti je oblika:

- Da li je resurs fer1:[HTMLDocument] preko svojstva rdf:[type] povezan sa resursom rdfs:[Class], odnosno da li je fer1:[HTMLDocument] po tipu rdfs:[Class]?
- Da li je resurs fer1:[HTMLDocument] preko svojstva rdfs:[subClassOf] povezan sa resursom fer1:[Document], odnosno da li je fer1:[HTMLDocument] podrazred od fer1:[Document]?

Rješenja ovih upita svode se na provjeravanje da li postoje tripleti:

(fer1:[HTMLDocument], rdf:[type], rdfs:[Class])
(fer1:[HTMLDocument], rdfs:[subClassOf], fer1:[Document])

Ako postoje, odgovori na gornja pitanja su potvrdni!

Slijedeća vrsta pitanja može glasiti ovako:

- Da li je resurs "R" koji je tipa fer1:[HTMLDocument] ujedno i tipa fer1:[Document]?

Imajući u vidu semantiku koja je vezana uz definiranje osnovnih RDF ontologija, možemo reči ovako: "R" je tipa fer1:[Document] ako postoji triplet:

([R], rdf:[type], fer1:[Document])

što očito iz prethodne tvrdnje ne postoji, ili pak ako postoji triplet

([R], rdf:[type], [nestoDrugo])
a resurs [nestoDrugo] je po tipu razred, i to podrazred od fer1:[Document]! Sada već imamo malih problema kako da ovakvo tumačenje zapišemo kao triplet. A zapravo ono što želimo pitati jest: da li je resurs [R] u *relaciji* rdf:[type] sa resursom fer1:[Document].

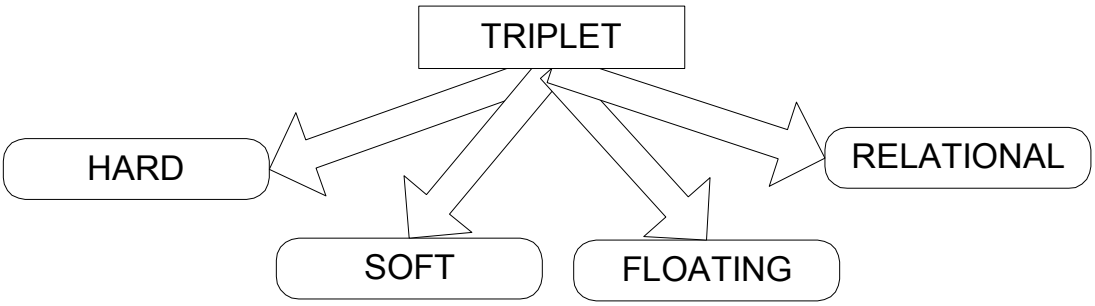
Zanimljiv je i problem sa svojstvima i podsvojstvima – kako pitati ima li Pero oca Juru i na to dobiti potvrdan odgovor, iako u bazi ne postoji takav triplet?

Konačno, umjesto da ispitujemo da li u bazi postoje određeni tripleti, zašto ne bismo pitali "tko ima oca Juru"? Za ovo će nam dodatno trebati koncept varijable! Nazivi varijable se tvore od znaka '?' iza kojeg slijedi niz slova, brojeva i podvlaka (prvi znak ne smije biti broj). Primjeri varijabli: ?x, ?course, ?var123, ?var__1.

Može se pronaći još niz sličnih varijacija upita, pa nakon određene analize, u radu ćemo koristiti četiri tipa tripleta:

1. HARD triplet, kratica HT
2. SOFT triplet, kratica ST
3. FLOATING triplet, kratica FT
4. RELATIONAL triplet, kratica RT

Pojedine vrste tripleta objašnjene su u nastavku.



Slika 4-12 Vrste tripleta

4.4.1 HARD triplet

HARD tripleti su tripleti koji se razriješavaju direktnim uvidom u bazu tripleta. HARD tripleti na svim mjestima mogu imati i varijable. HARD triplete označavati ćemo na slijedeći način:

HT(subjekt, predikat, objekt)

Primjeri:

HT(?x, rdf:[type], fer1:[Concept])

Način rješavanja:

Pronalaženje svih tripleta koji kao predikat imaju rdf:[type] a kao objekt fer1:[Concept].

4.4.2 SOFT triplet

SOFT tripleti su ekvivalent HARD tripletima. Razlika je što se mogu rješavati na nekoliko načina, a jedan od njih je prevođenje u SOFT triplete. Razlog zbog kojeg su uvedeni biti će objašnjen u nastavku. SOFT triplete označavati ćemo na slijedeći način:

ST(subjekt, predikat, objekt)

4.4.3 FLOATING triplet

FLOATING tripleti su tripleti koji prilikom razriješavanja vode računa od odnosu svojstvo / podsvojstvo. Ovaj tip tripleta koristiti će se za upite poput: "Tko je otac od Pere"? Razriješavanje se svodi na pokušaj praćenja postojećih tripleta i uporabe semantičke rdfs:subPropertyOf svojstva. FLOATING triplete označavati ćemo na slijedeći način:

FT(subjekt, predikat, objekt)

Varijable se mogu pojavljivati na svim mjestima, ali se ne preporuča uporaba na mjestu predikata.

4.4.4 RELATIONAL triplet

RELATIONAL tripleti su tripleti koji prilikom razriješavanja vode računa od semantici pojedinih svojstava. Ovdje je važno napomenuti da se ova vrsta semantike ne može opisati u ontologiji samim RDF-om, pa je potrebno poduzeti odgovarajuće korake kako bi sustav znao raditi sa ovim tipom tripleta. Budući da je sama relacija određena predikatom, predikat ne smije biti varijabla – subjekt i objekt smiju biti varijable. RELATIONAL triplete označavati ćemo na slijedeći način:

RT(subjekt, predikat, objekt)

4.4.5 Složeni upiti

Napišimo sada upit koji će rješavati pitanje sa kojim smo završili poglavlje 4.3: "*koje koncepte obrađuju dokumenti vezani za kolegij EFNC*"?

Razmišljamo u okvirima ontologije feronto1: treba pronaći resurs koji predstavlja kolegij EFNC (dakle, po tipu je fer1:[Course], i ima odgovarajući fer1:[CourseName]), zatim treba pronaći koji je fer1:[HomePage] tog kolegija. Zatim pronađemo sve dokumente koji pripadaju tom skupu dokumenata (engl. web site-u), i vratimo koncepte na koje se ti dokumenti odnose! Zapravo, razmišljajući na ovaj način vjerojatno se ne bismo sjetili da se ime kolegija navodi specijalnim svojstvom fer1:[CourseName], već bi nam na pamet palo samo da je to nekakvo ime – stavimo zato u upit svojstvo fer1:[Name]!

Upit možemo složiti na slijedeći način:

FT(?course, rdf:[type], fer1:[Course])
FT(?course, fer1:[Name], "Neizrazito, evolucijsko i neuro-racunarstvo")
FT(?course, fer1:[HomePage], ?hp)
FT(?doc, fer1:[belongsTo], ?hp)
FT(?doc, fer1:[isAbout], ?concept)

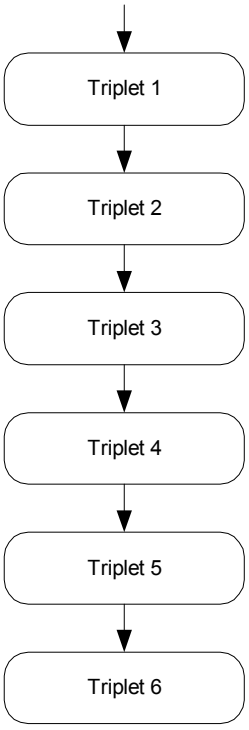
Vidimo da je upit dosta jednostavan – koristimo samo Floating triplete (zahvaljući kojima će problem fer1:[Name] – fer1:[CourseName] biti ispravno riješen; isto tako, obratiti pažnju da isti problem imamo i sa fer1:[belongsTo] – uvidom u ontologiju vidimo da bi za dokumente trebalo pisati fer1:[belongsToWebSite]).

4.4.6 Razriješavanje upita – građa sustava za zaključivanje

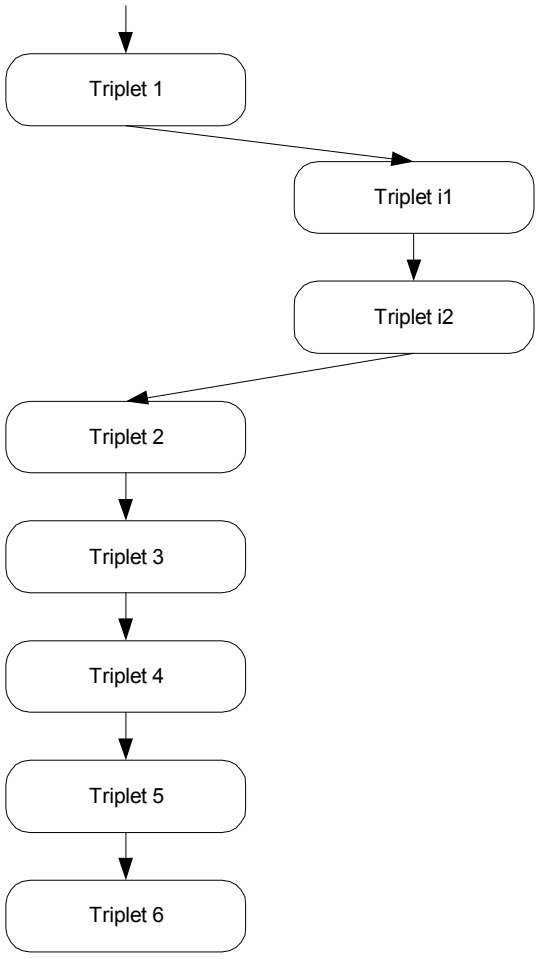
Sustav za razriješavanje upita kao ulaz dobije upit koji se sastoji od jednog ili više tripleta određenih tipova. Sustav na temelju tih tripleta treba dati odgovor – postaviti korištene varijable na vrijednosti koje zadovoljavaju upit.

Postupak je slijedeći:

1. Vrši se pretprocesiranje upita i pronalaze se korištene varijable.
2. Za svaku varijabu stvara se opisnik koji sadrži tri podatka:
    a. Varijabla je privremena?
    b. Varijabla je vezana (tj. postavljenja na neku vrijednost)?
    c. Vrijednost varijable
3. Kompletan upit se stavlja u posebnu strukturu LUSS (lista sa umetnutom stogovnom strukturom).
4. Riješi triplete u LUSS-u.



Slika 4-13 Početak rješavanja upita



Slika 4-14 Rješavanje prvog tripleta

Koraci 3 i 4 definiraju i koriste strukturu LUSS. Slika 4-13 i Slika 4-14 prikazuju dva stanja u rješavanju postavljenog upita. Na početku su u LUSS-u svih 6 tripleta.

Nakon što započne rješavanje tripleta 1, rezultat je potreba za rješavanjem dva nova tripleta (triplet i1 i triplet i2). Zbog toga se nakon tripleta 1 stvara stog na koji se ubacuju tripleti i1 i i2. Nastavlja se sa razriješavanjem tripleta i1, a u slučaju uspjeha i tripleta i2. Ukoliko se triplet i2 ne bi mogao razriješiti, skinuo bi se sa stoga, i pokušalo bi se razriješiti triplet i1 na neki drugi način. Ukoliko niti to ne bi bilo moguće, i triplet i1 bi se skinuo sa stoga, te bi se rješavanje vratilo na triplet 1. Ukoliko bi se pronašla neka druga metoda za razriješavanje ovog tripleta, cijela priča bi se ponovila, a inače bi pretraživanje završilo neuspjehom.

U slučaju da se je u prethodnom koraku triplet i2 uspio razriješiti, nastavilo bi se sa tripletom 2 gdje bi se cijela priča opet ponovila. I tako sve dok se ne dođe do kraja, kada smo pronašli jedno moguće rješenje. Nakon što se to rješenje dostavi pozivatelju, proces se nastavlja kao da posljednji triplet nije uspješno razriješen – traže se novi načini, ili u slučaju neuspjeha skida se sa stoga.

Na ovaj način sustav će pronaći sva moguća vezanja za varijable korištene u postavljenom upitu.

4.4.7 Razriješavanje upita – prijedlozi?

Do sada smo objasnili kako sustav vodi proces razriješavanja upita. Objašnjeno je sve, osim slijedećega – kako sustav zna na koji način razriješiti pojedini triplet? Naime, prethodno je spomenuto da se razriješavanjem jednog tripleta mogu pojaviti novi tripleti! Odgovor na pitanje "od kuda" vrlo je jednostavan. Sustav je utemeljen na konceptu *prijedloga*.

U okviru ovog rada predložen je slijedeći pristup. Svaka ontologija (osim same sebe) ima OntologyLayer – sloj koji zna nešto o semantici dotične ontologije. Ako se je prilikom izgradnje baze znanja koristila samo jedna ontologija, tada će biti dovoljno nabaviti OntologyLayer za tu ontologiju. Ukoliko se koristilo više ontologija, potrebno ja nabaviti / napisati OntologyLayer za svaku od korištenih ontologija.

Prije postupka razriješavanja upita sustav pribavlja potrebne OntologyLayer-e. Nakon toga započinje postupak razriješavanja, koji ide na slijedeći način. Za svaki triplet:

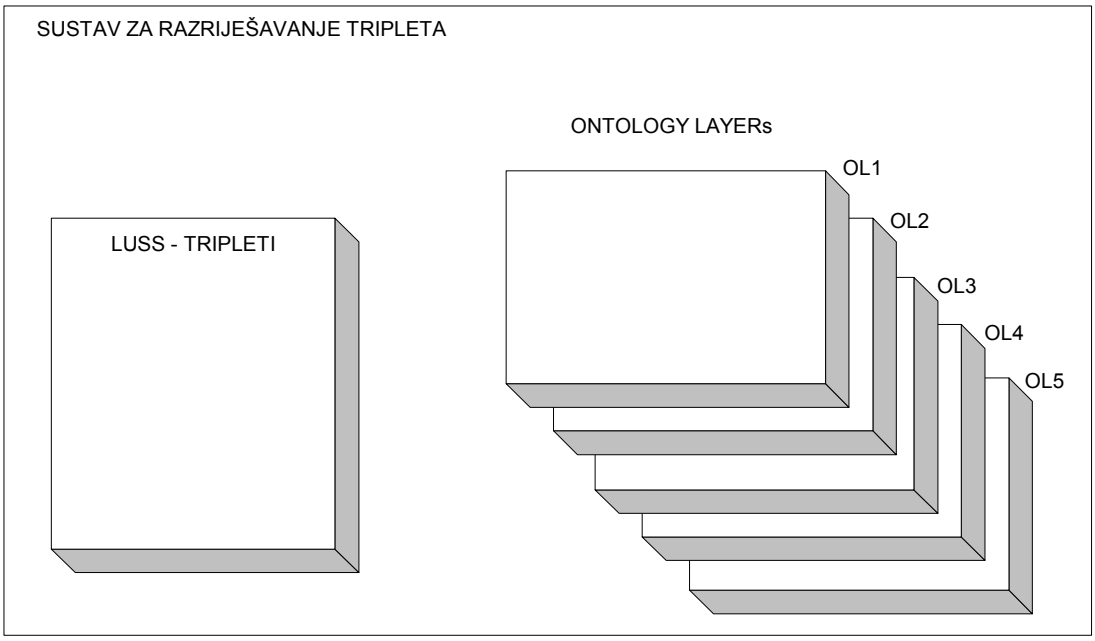
- Ako je triplet tipa HARD triplet, obavi uvid u bazu podataka i pokušaj vezati varijable.
- Inače, inicijaliziraj skup prijedloga na null pa:
    - za svaki OntologyLayer:
        - Pitaj OntologyLayer za prijedloge kako razriješiti dotični triplet.
        - Ponuđene prijedloge dodaj u skup prijedloga.
    - Za svaki prijedlog:
        - Stavi prijedlog na stog
        - Nastavi razriješavanje
        - Ako nije uspjelo, skini prijedlog sa stoga

Ovakvo razriješavanje temeljeno na prijedlozima ima jednu bitnu prednost koju je potrebno spomenuti: ovakav pristup može u određenim uvjetima omogućiti suradnju privatnih ontologija i ispravno zaključivanje! Npr. Neka korisnik koji šalje upit koristi ontologiju O1 koja je bazirana na standardnoj ontologiji B. U okolini gdje se nalazi sustav korisnici su dokumente opisivali svojom privatnom ontologijom O2. Čak i u ovoj situaciji, sustav će pronaći tražene odgovore ukoliko:

- Korisnik uz upit pošalje OntologyLayer ontologije O1.
- Sustav ima OntologyLayer ontologije O2.
- Sustav ima OntologyLayer ontologije B.

Prilikom razriješavanja upita u sustavu se zapravo tada događaju dvije stvari: upit se postepeno prevodi iz ontologije O1 u ontologiju B, a s druge strane podaci se također prevode iz ontologije O2 u ontologiju B.

Dakako, ukoliko ne postoji zajednička ontologija B, ovaj proces neće funkcionirati.



Slika 4-15 Konceptualni prikaz sustava za razriješavanje tripleta

4.5. IMPLEMENTACIJA SUSTAVA ZA PRETRAŽIVANJE

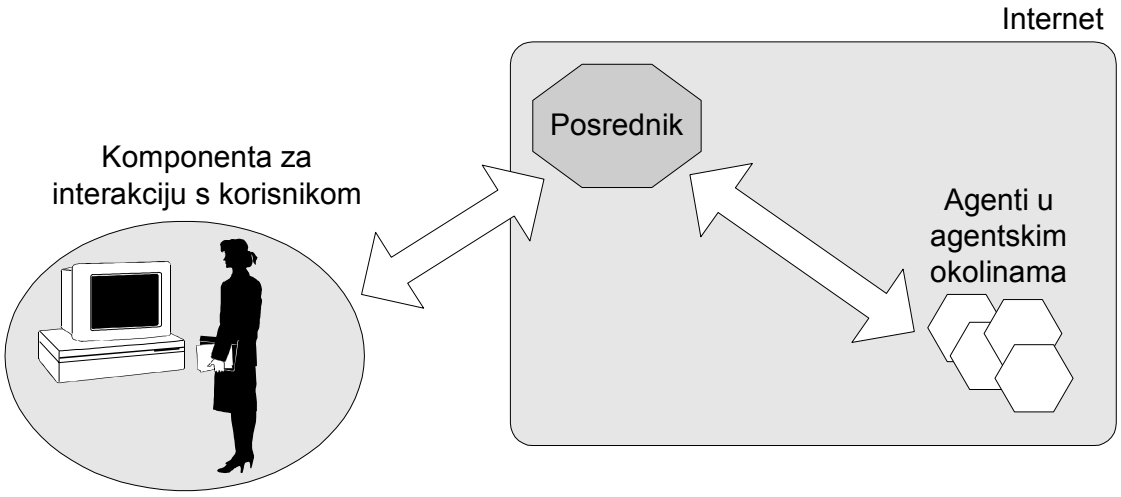
Sustav za pretraživanje obuhvaća paket programa koji zajedno omogućavaju pronalaženje traženih dokumenata ili podataka općenito metodom opisanom u ovom diplomskom radu. Sustav se sastoji od tri komponente koje djeluju kao jedna cjelina:

- agentske okoline (AE, engl. Agent Environment)
- posrednika (M, engl. Mediator)

- komponente za interakciju s korisnikom (UIC, engl. User Interaction Component)

Agentske okoline čine bazu cijelog sustava i omogućavaju rad i egzistenciju mobilnih agenata koji obavljaju posao pretraživanja. Agentske okoline trebale bi se nalaziti na svim računalima koja imaju poslužitelje web stranica.

Posrednik je poseban dio sustava koji odašilje agente i brine se za prikupljanje rezultata i koordinaciju agenata. Isto tako, postojanje posrednika omogućava korisniku zadavanje zadatka i zatim raskidanje spoja s posrednikom (npr. kada se očekuje duže izvođenje procesa pretraživanja, tj. upit se može zadati na kraju radnog vremena, i zatim se raskine spoj s posrednikom; posrednik prikuplja podatke od agenta i skladišti ih; slijedeći dan kada korisnik dođe na posao, upali računalo, spoji se na posrednika i zatim dobije podatke). Posrednik se nalazi na manjem broju računala (ili samo jednom računalu). Naime, posrednik omogućava proces pretraživanja (baš kao danas poznati servis za pretraživanje po ključnim riječima AltaVista).



Slika 4-16 Građa sustava za pretraživanje

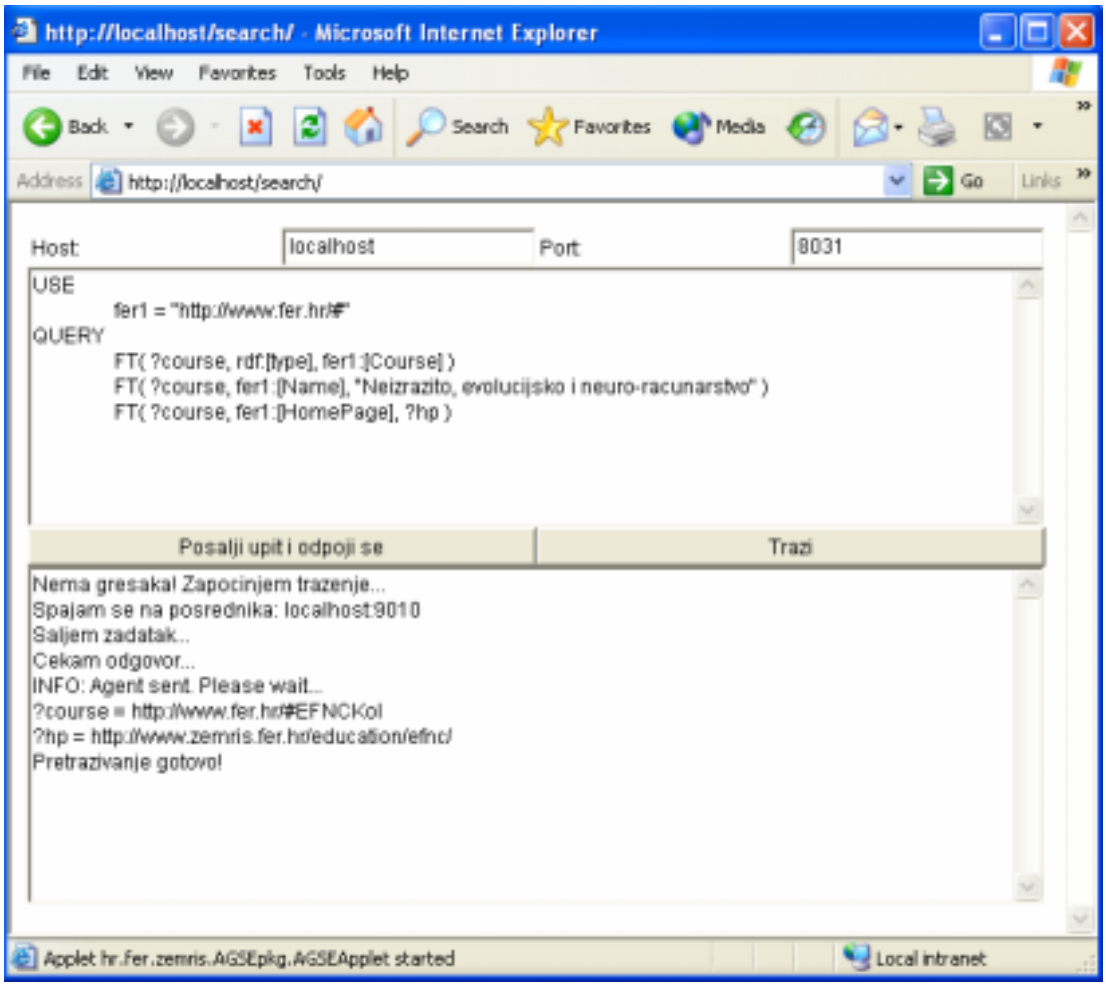
Komponenta za interakciju s korisnikom UIC u neposrednoj je vezi sa posredničkom komponentom. UIC omogućava korisniku zadavanje upita, brine se za slanje upita posredničkoj komponenti te za prikupljanje i prikaz rezultata od posredničke komponente. Bitno je napomenuti da ova komponenta nikada ne "razgovara" direktno sa agentima koji obavljaju pretraživanje. Građu sustava prikazuje Slika 4-16.

Za potrebe ovog rada komponenta za interakciju s korisnikom implementirana je na dva načina: kao standardni Java applet, tako da korisnik uslugama pretraživanja može pristupati direktno iz web preglednika bez potrebe za instalacijama posebnih paketa, te uporabom Java Servlet tehnologije za koju je korišten IBM WebSphere

Application Server (može se implementirati i pomoću Apache Tomcat servera). Applet prikazuje Slika 4-17.

Proces pretraživanja odvija se na slijedeći način: korisnik preko komponente za interakciju s korisnikom zadaje upit. Nakon toga upit se obrađuje i šalje posredniku. Posrednik provjerava o kakvom je upitu riječ, i stvara prvi primjerak agenta koji će obrađivati upit. Agent se zatim šalje u prvu agentsku okolinu, gdje obavlja pretraživanje. Svi rezultati koji se skupe šalju se posredniku koji ih zatim prosljeđuje komponenti za interakciju s korisnikom, gdje se rezultati prikazuju korisniku.

Agent pretraživanje obavlja na sljedeći način: nakon što stigne u agentski okolinu, provjerava pomoću metode *getService()* da li u toj okolini postoji usluga QuerySolverService. Ukoliko ne postoji, agent u toj okolini ne može vršiti pretraživanje jer ne može doći do opisnih datoteka. Ukoliko je navedena usluga prisutna, agent vrši proces pretraživanja.



Slika 4-17 Applet za unos upita i prikaz rješenja (Sinkrono pretraživanje)

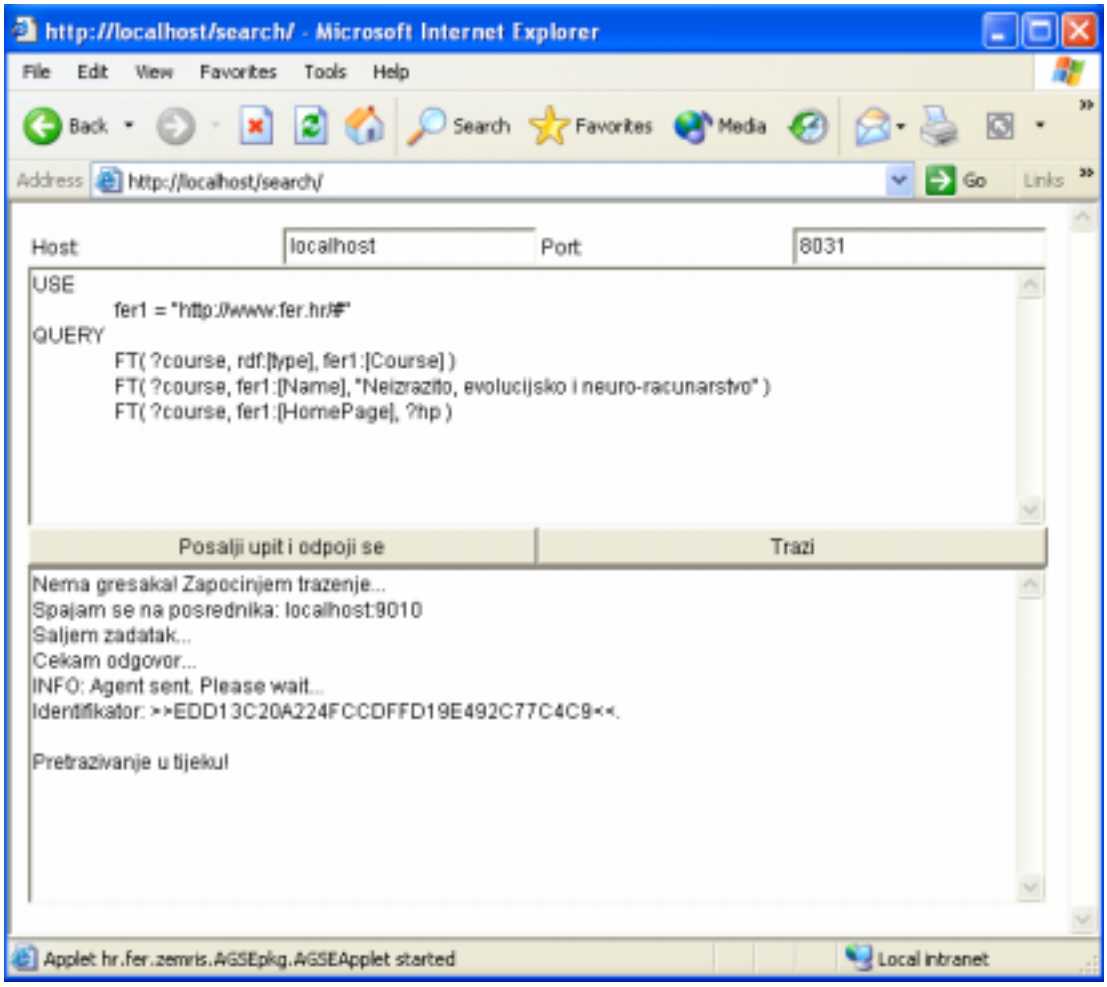
Slika 4-17 prikazuje upit s odgovorima koji su prikazani direktno u appletu. Ovakvo pretraživanje dobije se pritiskom na gumb 'Trazi'. U tom slučaju applet je cijelo

vrijeme spojen na posrednika i dinamički preuzima rezultate brzinom kojom oni postaju dostupni.

Ukoliko se za pretraživanje odabere druga metoda (gumb 'Posalji upit i odpoji se'), applet će poslati upit posredniku, od njega preuzeti identifikator zadatka (vidi Slika 4-18), i zatim prekinuti vezu s posrednikom. U tom trenutku računalo na kojem je korisnik zadao zadatak može se ugasiti (npr. zbog kraja radnog vremena). Status zadatka i pronađena rješenja mogu se doznati naknadnim pozivima servleta koji je zadužen za komunikaciju s posrednikom.

Prilikom poziva servleta bez argumenata generirati će se web stranica s formulator u koji će trebati upisati slijedeće podatke: host i port posrednika (može popunjavati i sam servlet, ovisno o konfiguraciji), identifikator zadatka (niz znakova koji je posrednik vratio appletu prilikom pokretanja procesa pretraživanja) te naredba što treba učiniti. Implementirane su slijedeće naredbe:

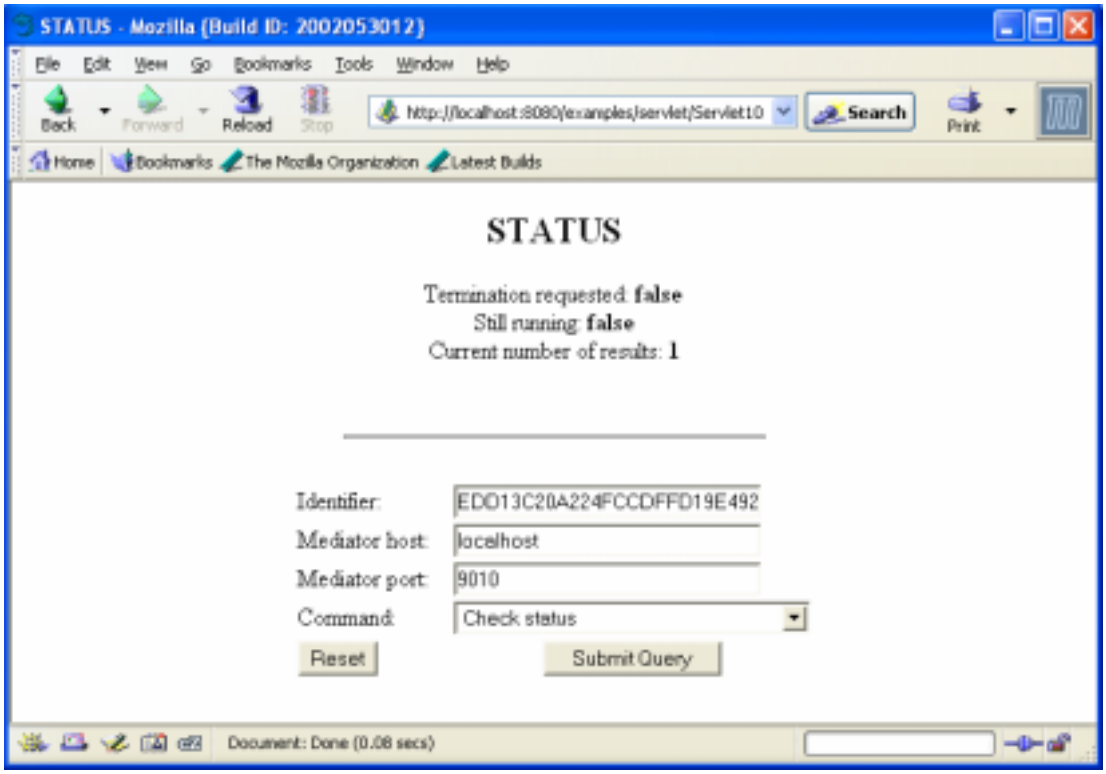
- *status* – trenutno stanje procesa pretraživanja
- *rezultati* – prikaz pronađenih rezultata do trenutka poziva
- *stop* – prekid procesa pretraživanja i brisanje zadatka



Slika 4-18 Asinkrono pretraživanje

Generiranu stranicu za slučaj naredbe 'status' prikazuje Slika 4-19, a generiranu stranicu za slučaj naredbe 'rezultati' prikazuje Slika 4-20.

Uporaba servleta dodatno olakšava zahtjeve na korisnikov preglednik jer Java više nužna na korisničkoj strani. Međutim, Java Appletima se rezultati mogu prikazati na daleko funkcionalniji način (stabla, prozori i sl.).



Slika 4-19 Servlet za komunikaciju s posrednikom - status

4.5.1 Upiti

Zadavanje upita obavlja se preko appleta (UIC), i omogućeno je na sljedeći način: zadaje se računalo na kojem započinje proces pretraživanja (host i port), i zadaje se tijelo upita. Upit je slijedeće strukture:

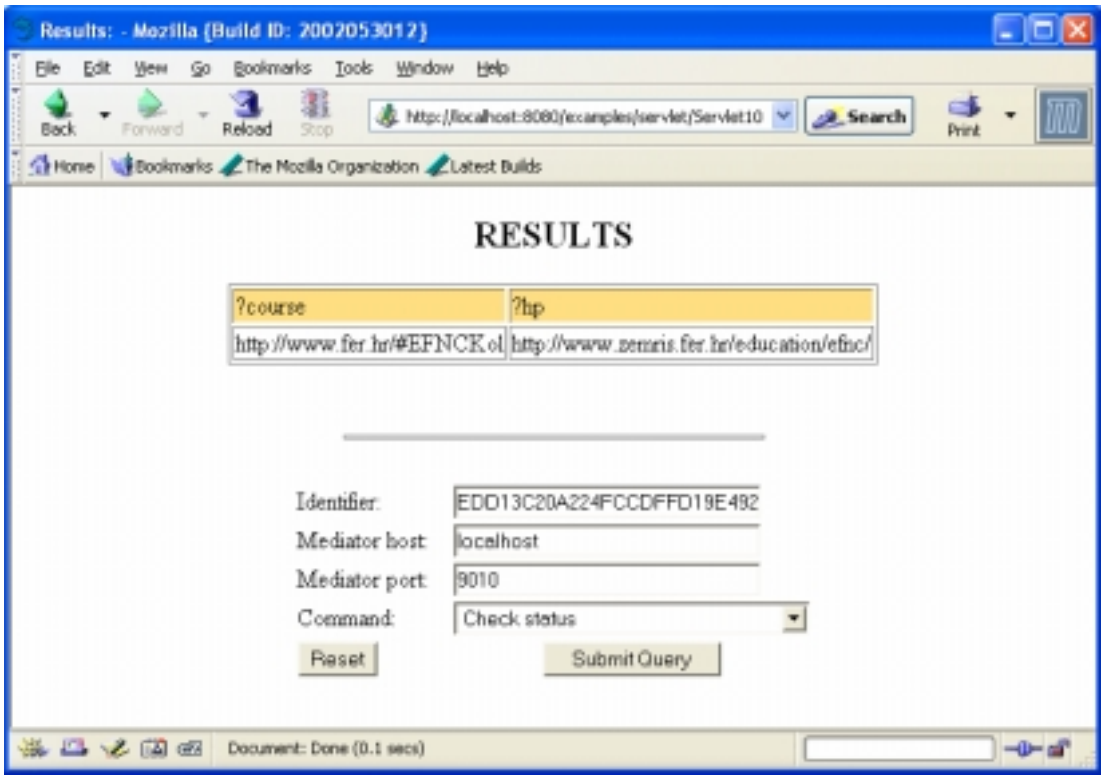
USE-dio (*opcionalan*)
QUERY-dio
ONSOLUTION (*opcionalan*)

USE dio omogućava definiranje kratica koje će se koristiti prilikom zadavanje naziva resursa. Sintaksa je:

USE
nazivkratice_1 = vrijednostkratice_1

nazivkratice_2 = vrijednostkratice_2
…
nazivkratice_n = vrijednostkratice_n

Kratice *rdf* i *rdfs* već su standardno definirane.



Slika 4-20 Servlet za komunikaciju s posrednikom - rezultati

QUERY dio omogućava definiranje samog upita. To je niz tripleta prethodno opisanih tipova koji sadrže nijednu, jednu ili više varijabli. Sintaksa je:

QUERY
triplet_1
triplet_2
…
triplet_n

ONSOLUTION dio definira dodatne akcije koje treba poduzeti kada se pronađe rješenje. Trenutno nije implementirana niti jedna akcija, ali postoje razne ideje kako ovo dodatno iskoristiti.

Slika Y prikazuje applet preko kojega korisnik unosi upit.

U nastavku će biti navedena dva primjera uporabe definiranog jezika za postavljanje upita. Prvi, jednostavniji primjer pokazuje kako možemo doznati koja je "domaća"

stranica (engl. Homepage) kolegija "Neizrazito, evolucijsko i neuro-racunarstvo". Upit ima slijedeću strukturu:

1. Traže se svi kolegiji.
2. Odabire se onaj kolegij koji ima traženo ime.
3. Traži se "domaća" stranica odabranog kolegija.

Upit glasi:

```
USE
      fer1 = "http://www.fer.hr/#"
QUERY
      FT( ?course, rdf:[type], fer1:[Course] )
      FT( ?course, fer1:[Name],
            "Neizrazito, evolucijsko i neuro-racunarstvo" )
      FT( ?course, fer1:[HomePage], ?hp )
```

Slijedeći primjer prikazuje upit koji daje odgovor na pitanje "*koje koncepte obrađuju dokumenti vezani za kolegij EFNC*":

```
USE
      fer1 = "http://www.fer.hr/#"
QUERY
      FT( ?course, rdf:[type], fer1:[Course] )
      FT( ?course, fer1:[Name],
            "Neizrazito, evolucijsko i neuro-racunarstvo" )
      FT( ?course, fer1:[HomePage], ?hp )
      FT( ?document, fer1:[belongsTo], ?hp )
      FT( ?document, fer1:[isAbout], ?concept )
```

Upit je sastavljen od prethodno opisanog upita koji određuje "domaću" stranicu traženog kolegija, i zatim slijedi dio koji traži sve dokumente koji pripadaju tom web mjestu (engl. website), i zatim koncepte o kojima govore ti dokumenti.

4.5.2 POTPORA UČENJU NA DALJINU

Jedan od osnovnih problema kod učenja na daljinu jest prikupljanje dokumenata iz kojih se može naučiti željeno gradivo. Uporaba agentskih sustava temeljenih na ontologijama može na određen način pomoći. Može se definirati agent koji će kao ulaz dobiti skup koncepata koje želimo naučiti. Uporabom semantičkih veza među konceptima agent bi mogao prikupljati dokumente koji govore o traženim konceptima, ali i o konceptima koje treba savladati kako bi se mogli razunjeti traženi koncepti.

Npr. dvije osnovne veze među konceptima koje definira ontologija feronto1 su fer1:dependsOn i fer1:extendsConcept. Ukoliko korisnik želi naučiti koncept "neizraziti skup" agent će vidjeti da je koncept "neizraziti skup" proširenje (veza fer1:extendsConcept) koncepta "skup" pa će pribaviti dokumente koji se bave i klasičnim skupovima, i neizrazitim skupovima.

Uporabom veze fer1:moreAbout agent može doći do novih okolina u kojima postoji više dokumenata o određenom konceptu. Ukoliko agent ima već dovoljno dokumenata o nekom konceptu, jednostavno će ignorirati tu informaciju, a ukoliko nema, može odlučiti sam otići u tu okolinu (ili poslati svog klona kako bi posao obavljali paralelno) i provjeriti što tamo ima o traženim konceptima.

Virtualni kolegiji također se mogu definirati na vrlo jednostavan način: dovoljno je kolegij definirati kao skup koncepata koje taj kolegij obrađuje. Uporabom prethodno opisanog agenta korisnici bi mogli pribaviti sve potrebne dokumente iz kojih bi mogli učiti gradivo kolegija.

5 ZAKLJUČAK

U okviru ovog rada prikazana je implementacija sustava za pretraživanje dokumenata napisanog u programskom jeziku Java. Također je objašnjeno zašto je odabran baš taj jezik. Prikazan je način na koji se uporabom ontologija mogu opisivati dokumenti, i objašnjena je prednost ove metode u odnosu na klasično opisivanje ključnim riječima. Definirana je ontologija feronto1 i prikazan je primjer kako se njezinom uporabom mogu opisivati dokumenti. Također su izneseni primjeri upita koji jasno demonstriraju mogućnosti upita temeljenih na ontologijama (npr. upit poput "koji se sve koncepti obrađuju na kolegiju neizrazito, evolucijsko i neuro-računarstvo"?) na koje klasične metode temeljene na ključnim riječima ne mogu dati odgovor. Objašnjena je uporaba sustava kao pomoć za učenje na daljinu, kao i definiranje virtualnih kolegija. Cijeli sustav zamišljen je kao velika distribuirana aplikacija, te je objašnjen način na koji je proces distribuje proveden uporabom agenata. Definirani su agenti i potporna infrastruktura koju čine agentske okoline i sustav za pronalaženje okolina.

LITERATURA

1 H.P. Luhn. The automatic creation of literature abstracts. IBM Jour. Res. Dev., 2(2), 1958.

2 G. Salton, E. Fox, and H. Wu. Extended boolean information retrieval, Communications of ACM, 26(12):1022-1036, March 1992.

3 E.A. Salton and M.J. McGill. Introduction to Modern Information Retrieval. McGraw Hill International, New York, 1983.

4 S. Deerwester, S. Dumais, T.Landauer, G. Furnas, L. Beck. Improving information retrieval using latent semantic indexing. In Proc. Of the Annual Meeting of the American Society for Information Science, pages 36-40, 1988.

5 H.R. Turtle, W.B. Croft. Evaluation of an inference network-based retrieval model. ACM Trans. Inf. Sys., 3, 1991.

6 K.L.Kwok, A neural network for probabilistic information retrieval. In ACM SIGIR Forum, volume 23, pages 21-30, 1989.

7 R. Belew. Adaptive information retrieval: using a cennectionist representation to retrieve and learn about documents. In Annual Proc. Of the ACM SIGIR, pages 11-20. 1989.

8 A. O'Riordan, H. Sorensen, An Intelligent Agent for High-Precision Text Filtering. CIKM '95, Baltimore MD USA, ACM 0-89791-812-6/95/11, 1995.

9 RDF

10 XML

11 A.B. Williams, Z. Ren, Agents Teaching Agents to Share Meaning, AGENTS'01 May 28 – June 1, 2001, Montréal, Québec, Canada, ACM 1-58113-326-X/01/0005, 2001.

12 S. Bradshaw, A. Scheinkman, K. Hammond. Guiding People to Information: Providing an Interface to a Digital Library Using Reference as a Basis for Indexing. IUI 2000 New Orleans LA USA, ACM 1-58113-134-8/00/1, 2000.

13 L. Chen, K. Sycara. WebMate: A Personal Agent for Browsing and Searching. Autonomous Agents 98 Minneapolis MN USA, ACM 0-89791-983-1/98/5, 1998.

14 A. Ciampolini, E. Lamma, P. Mello, C. Stefanelli. Abductive Coordination for Logic Agents, SAC '99, San Antonio, Texas, ACM 1-58113-086-4/99/0001, 1998.

15 MeSH 1999. Medical Subject Headings homepage. [http://www.nlm.nih.gov/mesh/meshhome.html]

16 Art and Architecture Thesaurus Browser, [online: http://shiva.pub.getty.edu/aat_browser/], 1999.

17 N. Stojanovic, A. Maedche, S. Staab, R. Studer, Y. Sure. SEAL — A Framework for Developing SEmantic PortALs. K-CAP'01, October 22-23, 2001, Victoria, British Columbia, Canada. ACM 1-58113-380-4/01/0010, 2001.

18 B.J. Wielinga, A.Th. Schreiber, J. Wielemaker, J.A.C. Sandberg. From Thesaurus to Ontology. K-CAP'01, October 22-23, 2001, Victoria, British Columbia, Canada. ACM 1-58113-380-4/01/0010, 2001.

19. Stanford Scalable Knowledge Composition (SKC), online: http://www-db.stanford.edu/SKC

20. Bremer Semantic Translation, online: http://www.semantic-translation.com/

21. T.R. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Revision: August 23, 1993, online: http://ksl-web.stanford.edu/KSL_Abstracts/KSL-93-04.html
22. M.R. Genesereth, N.J. Nilsson. Logical Foundations of Artificial Intelligence. San Mateo, CA: Morgan Kaufmann Publishers. 1987.
23. Dublin Core Metadata Iniciative, online: http://purl.org/dc

PRILOZI

5.1. Primjer opisa dokumenata feronto1 ontologijom

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:fer1="http://www.fer.hr/#"
        xmlns:fer2="http://www.fer.hr/extensions/1/#">

        <!--=====================================================-->
        <!--            APPLICATION OF THIS ONTOLOGY             -->
        <!--=====================================================-->

        <!-- Definitions of used concepts. -->

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#Set">
                <fer1:conceptLabel>Set</fer1:conceptLabel>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#Relation">
                <fer1:conceptLabel>Relation</fer1:conceptLabel>
                <fer1:dependsOn rdf:resource="http://www.zemris.fer.hr/concepts/#Set"/>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#FuzzySet">
                <fer1:conceptLabel>Fuzzy set</fer1:conceptLabel>
                <fer1:extendsConcept
                        rdf:resource="http://www.zemris.fer.hr/concepts/#Set"/>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#FuzzyRelation">
                <fer1:conceptLabel>Fuzzy relation</fer1:conceptLabel>
                <fer1:extendsConcept
                        rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzySet"/>
                <fer1:extendsConcept
                        rdf:resource="http://www.zemris.fer.hr/concepts/#Relation"/>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#FuzzyLogic">
                <fer1:conceptLabel>Fuzzy Logic</fer1:conceptLabel>
                <fer1:relatedConcept
                        rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyOperations"/
                >
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#FuzzyOperations">
                <fer1:conceptLabel>Fuzzy operations</fer1:conceptLabel>
                <fer1:relatedConcept
                        rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzySet"/>
                <fer1:relatedConcept
                        rdf:resource="http://www.zemris.fer.hr/concepts/#Set"/>
        </fer1:Concept>
```

```
        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#NeuralNetwork">
                <fer1:conceptLabel>Neural network</fer1:conceptLabel>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#GeneticAlgorithm">
                <fer1:conceptLabel>Genetic algorithm</fer1:conceptLabel>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#ID3">
                <fer1:conceptLabel>ID3</fer1:conceptLabel>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#CA">
                <fer1:conceptLabel>Candidate elimination</fer1:conceptLabel>
        </fer1:Concept>

        <fer1:Concept rdf:about="http://www.zemris.fer.hr/concepts/#FINDS">
                <fer1:conceptLabel>FIND-S</fer1:conceptLabel>
        </fer1:Concept>

        <!-- List of all documents. -->

        <fer1:WebSite rdf:about="http://www.fer.hr/">
                <fer1:Title>FER Home Page</fer1:Title>
        </fer1:WebSite>

        <fer1:WebSite rdf:about="http://www.zemris.fer.hr/">
                <fer1:belongsToWebSite rdf:resource="http://www.fer.hr/"/>
                <fer1:Title>ZEMRIS Home Page</fer1:Title>
        </fer1:WebSite>

        <fer1:WebSite rdf:about="http://www.zemris.fer.hr/education/is/">
                <fer1:belongsToWebSite rdf:resource="http://www.zemris.fer.hr/"/>
                <fer1:Title>Intelligent Systems Home Page</fer1:Title>
        </fer1:WebSite>

        <fer1:WebSite rdf:about="http://www.zemris.fer.hr/education/ml/">
                <fer1:belongsToWebSite rdf:resource="http://www.zemris.fer.hr/"/>
                <fer1:Title>Machine Learning Home Page</fer1:Title>
        </fer1:WebSite>

        <fer1:WebSite rdf:about="http://www.zemris.fer.hr/education/efnc/">
                <fer1:belongsToWebSite rdf:resource="http://www.zemris.fer.hr/"/>
                <fer1:Title>Evolutionary, Fuzzy and Neuro-Computing Home
                        Page</fer1:Title>
        </fer1:WebSite>

        <fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/efnc/documents/FuzzySets.html">
                <fer1:belongsToWebSite
                        rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
                <fer1:Title>Introduction to Fuzzy Set theory</fer1:Title>
                <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzySet"/>
        </fer1:WebPage>

        <fer1:WebPage
rdf:about="http://www.zemris.fer.hr/education/efnc/documents/FuzzyRelations.html">
                <fer1:belongsToWebSite
```

```
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
            <fer1:Title>Introduction to Fuzzy Relations</fer1:Title>
            <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyRelation"/>
        </fer1:WebPage>

        <fer1:WebPage
rdf:about="http://www.zemris.fer.hr/education/efnc/documents/FuzzyOperations.html">
            <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
            <fer1:Title>Introduction to operations with Fuzzy Sets</fer1:Title>
            <fer1:isAbout
            rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyOperations"/>
        </fer1:WebPage>

        <fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/efnc/documents/Sets.html">
            <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
            <fer1:Title>Introduction to Set theory</fer1:Title>
            <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#Set"/>
        </fer1:WebPage>

        <fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/efnc/documents/Relations.html">
            <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
            <fer1:Title>Introduction to relations</fer1:Title>
            <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#Relation"/>
        </fer1:WebPage>

        <fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/efnc/documents/FuzzyLogic.html">
            <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
            <fer1:Title>Introduction to fuzzy logic</fer1:Title>
            <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyLogic"/>
        </fer1:WebPage>

        <fer1:WebPage
            rdf:about="http://www.zemris.fer.hr/education/efnc/documents/NN.html">
                <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
            <fer1:Title>Introduction to neural networks</fer1:Title>
            <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#NeuralNetwork"/>
        </fer1:WebPage>

        <fer1:WebPage
rdf:about="http://www.zemris.fer.hr/education/efnc/documents/GeneticAlgorithm.html">
            <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
        <fer1:Title>Introduction to genetic algorithm</fer1:Title>
            <fer1:isAbout
            rdf:resource="http://www.zemris.fer.hr/concepts/#GeneticAlgorithm"/>
        </fer1:WebPage>
```

```
<fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/ml/documents/FINDS.html">
        <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/ml/"/>
        <fer1:Title>Introduction to FIND-S algorithm</fer1:Title>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#FINDS"/>
</fer1:WebPage>

<fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/ml/documents/CA.html">
        <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/ml/"/>
        <fer1:Title>Introduction to Candidate elimination algorithm</fer1:Title>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#CA"/>
</fer1:WebPage>

<fer1:WebPage
        rdf:about="http://www.zemris.fer.hr/education/ml/documents/ID3.html">
        <fer1:belongsToWebSite
                rdf:resource="http://www.zemris.fer.hr/education/ml/"/>
        <fer1:Title>Introduction to ID3 algorithm</fer1:Title>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#ID3"/>
</fer1:WebPage>

<fer1:Faculty rdf:about="http://www.fer.hr/#FER">
        <fer1:FacultyName>Fakultet elektrotehnike i racunarstva
        </fer1:FacultyName>
        <fer1:HomePage rdf:resource="http://www.fer.hr/"/>
</fer1:Faculty>

<fer1:Department rdf:about="http://www.fer.hr/#ZEMRIS">
        <fer1:DepartmentName>Zavod za elektroniku, mikroelektroniku, racunalne i
                inteligentne sustave</fer1:DepartmentName>
        <fer1:HomePage rdf:resource="http://www.zemris.fer.hr/"/>
        <fer1:belongsToFaculty rdf:resource="http://www.fer.hr/#FER"/>
</fer1:Department>

<fer1:Course rdf:about="http://www.fer.hr/#MLKol">
        <fer1:CourseName>Strojno ucenje</fer1:CourseName>
        <fer1:HomePage rdf:resource="http://www.zemris.fer.hr/education/ml/"/>
        <fer1:belongsToDepartment rdf:resource="http://www.fer.hr/#ZEMRIS"/>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#FINDS"/>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#CA"/>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#ID3"/>
        <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#NeuralNetwork"/>
</fer1:Course>

<fer1:Course rdf:about="http://www.fer.hr/#EFNCKol">
        <fer1:CourseName>Neizrazito, evolucijsko i neuro-racunarstvo
        </fer1:CourseName>
        <fer1:HomePage rdf:resource="http://www.zemris.fer.hr/education/efnc/"/>
        <fer1:belongsToDepartment rdf:resource="http://www.fer.hr/#ZEMRIS"/>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#Set"/>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#Relation"/>
        <fer1:isAbout rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzySet"/>
        <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyRelation"/>
```

```
                <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyOperations"/>
                <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#NeuralNetwork"/>
                <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#GeneticAlgorithm"/>
                <fer1:isAbout
                rdf:resource="http://www.zemris.fer.hr/concepts/#FuzzyLogic"/>
        </fer1:Course>

        <fer1:Course rdf:about="http://www.fer.hr/#ISKol">
                <fer1:CourseName>Inteligentni sustavi</fer1:CourseName>
                <fer1:HomePage rdf:resource="http://www.zemris.fer.hr/education/is/"/>
                <fer1:belongsToDepartment rdf:resource="http://www.fer.hr/#ZEMRIS"/>
        </fer1:Course>

</rdf:RDF>
```

5.2. Protokol za komunikaciju agentske okoline s "Agent Server Directory" komponentom

Za potrebe komunikacije između Agent Servera i Agent Server Directory-a razvijen je protokol čije su naredbe opisane u nastavku. Protokol omogućava prijavu i odjavu Agent Servera, te upit o postojanju Agent Servera koji može koristiti agent kako bi doznao da li se na ciljnom računalu nalazi okolina u koju bi mogao prijeći.

Tablica 0-1. Naredba BYE

Naredba	**BYE**
Sintaksa	**BYE**
Opis	Ovu naredbu izdaje klijent, i naredba završava komunikaciju između klijenta i servera i prekida uspostavljenu vezu.
Primjer	BYE

Tablica 0-2. Naredba EXISTS

Naredba	**EXISTS**
Sintaksa	**EXISTS** host **EXISTS** host:port
Opis	Ovu naredbu izdaje klijent. Naredba zahtjeva od servera da provjeri da li je na zadanom računalu (host) registriran koji Agent Server. Ukoliko se koristi skraćeni oblik upita bez navođenja porta, odgovor će sadržavati sve portove na kojima je prijavljen Agent Server. Ukoliko se koristi potpuni upit sa portom, u odgovor će sadržavati taj port ukoliko je za njega registriran Agent Server. Odgovor je uvijek lista koja sadrži određeni broj portova. Svaki port naveden je u novoj liniji, a cijela lista je terminirana praznim retkom.
Primjer	Neka su registrirani AS-ovi fly.srk.fer.hr:8005, fly.srk.fer.hr:8006. **EXISTS fly.srk.fer.hr** *8006* *8005* *OK.* **EXISTS fly.srk.fer.hr:8005** *8005* *OK.* **EXISTS fly.srk.fer.hr:8001** *OK.*

Tablica 0-3. Naredba REGISTER

Naredba	**REGISTER**
Sintaksa	**REGISTER** host:port
Opis	Ovu naredbu izdaje klijent. Naredba zahtjeva od servera da prihvati registraciju Agent Servera na navedenom računalu i na navedenom portu.
Primjer	**REGISTER fly.srk.fer.hr:8001** *OK. Registration completed.*

Tablica 0-4. Naredba UNREGISTER

Naredba	**UNREGISTER**
Sintaksa	**UNREGISTER** host:port
Opis	Ovu naredbu izdaje klijent. Naredba zahtjeva od servera da poništi registraciju Agent Servera na navedenom računalu i na navedenom portu.
Primjer	**UNREGISTER fly.srk.fer.hr:8001** *OK.*

Svaka naredba mora biti terminirana nizom ascii znakova 13, 10.

U slučaju pogreške, server odgovara jednom linijom koja započinje sa tekstom "ERROR:" iza kojeg slijedi jedna praznina (ascii 32) te zatim kratko objašnjenje greške.