UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Pintar

PREGLED IN ANALIZA ORODIJ ZA ODKRIVANJE SEMANTIČNIH SPLETNIH STORITEV

DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

mentor: prof. dr. Marjan Krisper

Železniki, 2008

Zahvala

Najprej bi se rad zahvalil prof. dr. Marjanu Krisperju za mentorstvo, vodenje in usmerjanje pri izdelavi diplomske naloge.

Zahvalil bi se tudi Dejanu Lavbiču, ki mi je s svojimi konstruktivnimi komentarji in znanjem izredno pomagal pri delu.

Hvala staršem in Nini za vzpodbude v času študija.

Hvala Bojanu, Janezu in Ariani, ki so poskrbeli za jezikovno neoporečnost pričujočega dela.

Povzetek

Storitveno usmerjene arhitekture postajajo vedno bolj pomemben pristop k povezovanju raznolikih računalniških sistemov v široko porazdeljenih omrežjih. Velik problem trenutno predstavlja iskanje ustrezne spletne storitve, ki jo hočemo vključiti v poslovni proces, saj opisi spletnih storitev trenutno niso formalno določeni. Medtem, ko so sintaktične lastnosti spletnih storitev dobro definirane, je semantika na zelo nizki ravni. Zaradi težnje po uvajanju večje stopnje avtomatizacije in neodvisnosti od ljudi so se na področju odkrivanja in vnosa semantike v spletne storitve stvari začele odvijati z veliko naglico. Pojavili so se številni predlogi za obogatitev opisa spletnih storitev, kot tudi algoritmi in orodja.

V pričujočem delu bom predstavil različne ontologije, s katerimi lahko dosežemo, da opis spletne storitve ne bo berljiv samo človeškim uporabnikom, ampak ga bo razumel tudi računalnik. Ogledali si bomo nabor algoritmov, ki se razlikujejo po kompleksnosti in načinu primerjanja elementov spletnih storitev. Podal bom tudi analizo orodij, ki te algoritme uporabljajo in tako omogočajo odkrivanje semantičnih spletnih storitev. Na koncu so predstavljeni primeri iskanja spletne storitve, ki je opisana v treh različnih ontologijah.

KLJUČNE BESEDE: semantični splet, semantične spletne storitve, ontologija, algoritmi za odkrivanje semantičnih spletnih storitev, orodja za odkrivanje semantičnih spletnih storitev

Abstract

Service oriented architecture is becoming a more and more important approach for connecting heterogeneous computer systems in wide area networks. One of the mayor problems is the discovery of suitable web service which we would like to include in business process, because the descriptions of web services are not formally defined. Meanwhile the syntactic properties are well defined, the semantics present in description of web service are at a very low level. Because of the tendency to create very high level of automatization and independence of the users, the research area of semantic web services has advanced very rapidly. Large numbers of proposals for semantically annotated web services, algorithms and tools have arisen in the recent years.

The following work will illustrate different ontologies with which we can achieve that the description of web service is not understandable only to the human users but also to computers. We will examine a selection of algorithms, which are distinguished by complexity and ways of comparing elements of semantic web services. I will analyse the tools which use the above mentioned algorithms and therefore enable the semantic web service discovery. In the end I will present examples of web service discovery described in three different ontologies.

KEYWORDS: semantic web, semantic web services, ontology, algorithms for semantic web service discovery, tools for semantic web service discovery

Kazalo

Slike

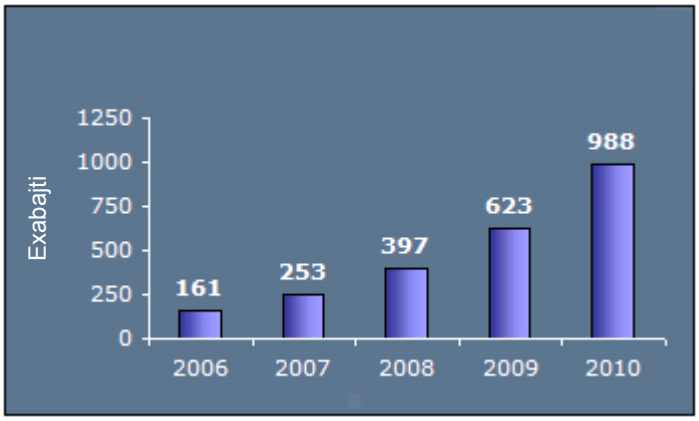
Tabele

Kratice, okrajšave, simboli

Semantic Web	**Semantični splet** je nova tehnologija na katero lahko gledamo kot na nadgradnjo svetovnega spleta. Cilj semantičnega spleta je predvsem čimbolj avtomatsko graditi modele znanja ali ontologije iz poljubnih dokumentov, oz. te modele in ontologije uporabljati za označevanje pomembnih informacij z metapodatki.
Semantic Web Services	**Semantične spletne storitve** so storitve, katerih cilj je učinkovito povezovanje tehnologije semantičnega spletna in spletnih storitev.
Ontology	**Ontologije** so ključni element pri semantičnem spletu, kjer se prepleta človeško razumevanje simbolov z zmožnostjo računalniškega procesiranja. Ontologija je formalna, eksplicitna specifikacija skupne konceptualizacije.
SPARQL SPARQL Protocol and RDF Query Language	Eden izmed najpomembnejših poizvedovalnih jezikov s katerim lahko poizvedujemo po podatkih zapisanih v RDF.
OWL-S Semantic Markup for Web Services	Ontologija za opis spletnih storitev v OWL.
OWL Web Ontology Language	Semantični označevalni jezik za objavljanje in izmenjavo ontologije.
WSMO Web Service Modeling Ontology	Ontologija za opis spletnih storitev.
WSML Web Service Modeling Language	Semantični označevalni jezik za objavljanje in izmenjavo WSMO.
SWSO Semantic Web Services Ontology	Ontologija za opis spletnih storitev.
SWSL Semantic Web Services Language	Semantični označevalni jezik za objavljanje in izmenjavo SWSO.
XML eXtensible Markup Language	Razširljiv označevalni jezik.
WSDL Web Service Description Language	Jezik za opisovanje spletnih storitev in načina dostopa do njih s pomočjo XML.
UDDI Universal Description, Discovery and Integration	Industrijsko standardiziran imenik, ki omogoča objavo in iskanje spletnih storitev.
IR Information Retrieval	**Pridobivanje informacij** se ukvarja s predstavitvijo, shranjevanjem, iskanjem in pridobivanjem informacij iz velikih zbirk tekstovnih dokumentov.

1 Uvod

Svetovni splet je postal skladišče človeškega znanja in kulture, ki omogoča izmenjavo idej in dostop do informacij na globalni ravni. Leta 2006 je bilo na spletu ustvarjenih 161 Eksabajtov podatkov, kar je približno tri milijonkrat več od vseh podatkov, zbranih v knjigah. Do leta 2010 pa se pričakuje letno povečanje za 57 odstotkov[1]. Zanimiva je tudi trditev, da spletni uporabniki izvedejo približno 7 milijard poizvedb na mesec [1].



Slika 1: Podatki prisotni na svetovnem spletu[2].

Informacije na spletu so dostopne v različnih oblikah in na različne načine, ki pa med sabo niso združljivi. Tudi HTML je namenjen predstavljanju informacij in ni primeren za računalniško obdelavo. Prav tako je zaradi velike količine podatkov, ki so slabo organizirani na svetovnem spletu izredno težko in zamudno najti dobre informacije. Zaradi tega so se pojavile številne rešitve na področju reševanja tega problema. Eden izmed prvih pristopov so bili spletni iskalniki, kjer je uporabnik na podlagi ključne besede dobil množico povezav, ki so se najbolje ujemale z iskanim nizom besed. Ta pristop je danes zelo razširjen, vendar pa ima to slabost, da je lahko vrnjena množica izredno velika in je ročno pregledovanje zelo zamudno opravilo. Poleg tega pa je niz besed, na podlagi katerega poteka iskanje, potrebno pametno izbrati, saj napačna izbira besed vrne slabe rezultate. Npr. če v iskalnik Google vnesemo samo niz »jaguar« in s tem mislimo na sesalca (kar dela veliko uporabnikov), bo ustrezna povezava predlagana šele na četrtem mestu.

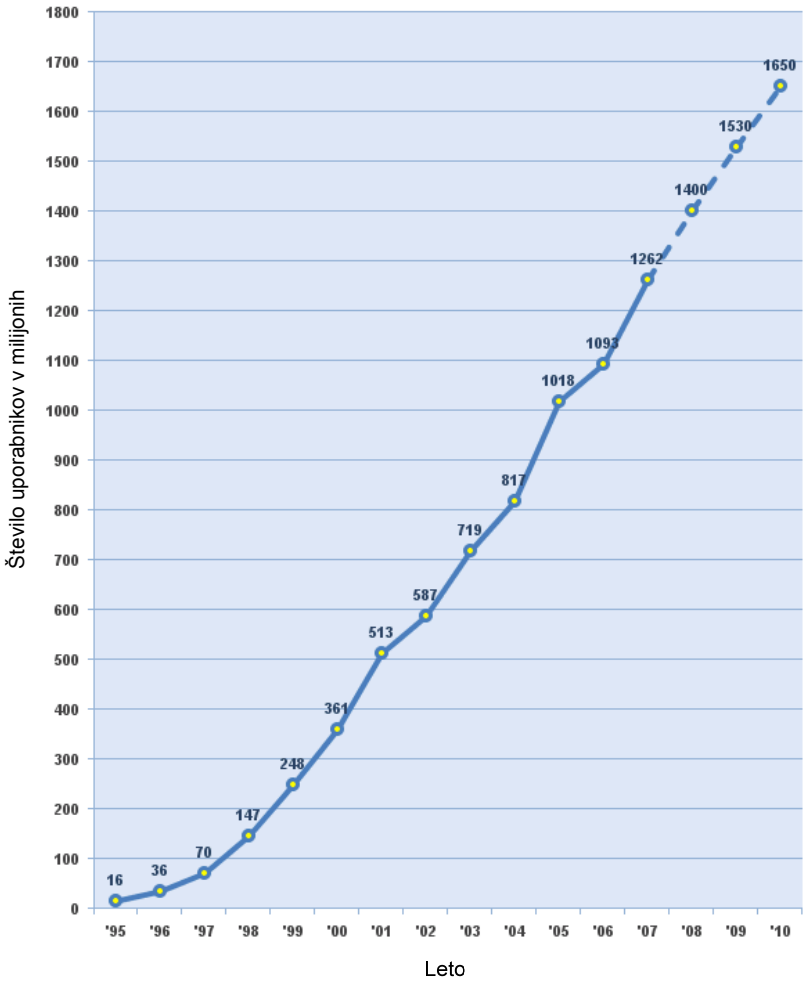
Spletne storitve igrajo pomembno vlogo pri integraciji aplikacij in tako postajajo sestavni gradniki sodobnih informacijskih sistemov. Ključni cilj storitvenih arhitektur je zagotoviti integracijo aplikacij z namenom doseči čim višjo stopnjo avtomatizacije poslovanja. Spletne storitve opišemo z WSDL, te opise pa objavimo v UDDI imeniku. Trenutno je iskanje v UDDI imeniku dokaj okorno, tako da brez posredovanja uporabnika spletnih storitev ni mogoče vključiti v poslovni proces. V primeru, da je neka storitev nedosegljiva, trenutno ni mogoče, da bi sistem avtomatično našel ustrezno nadomestno storitev.

Zaradi želje, da se poleg ljudi tudi računalniki začnejo zavedati vsebine, se je pojavila tehnologija, ki nadgrajuje obstoječi splet in spletne storitve s tem, da vnaša semantiko, ki je za to nujno potrebna.

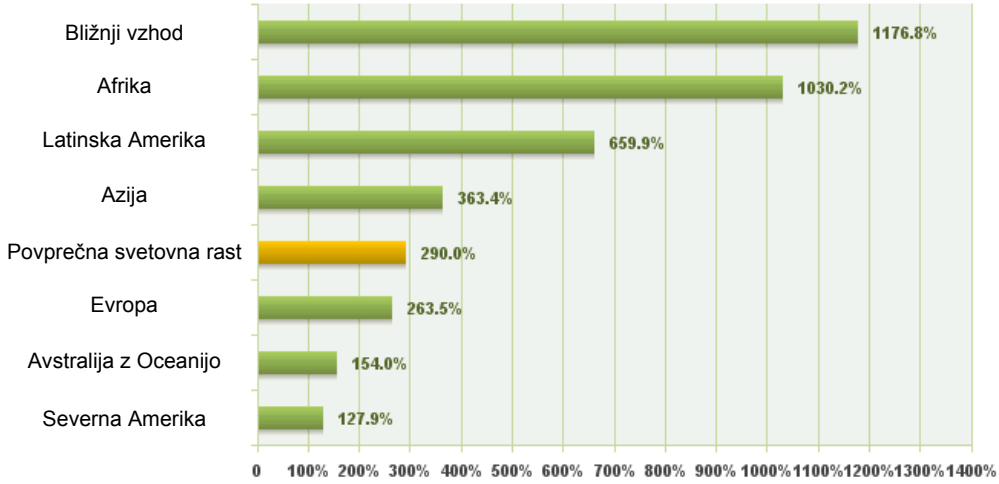
---

[1] http://www.internetnews.com/stats/article.php/3663641

[2] http://www.lk-design.net/the_market.html#internet_size



Slika 2: Rast števila uporabnikov v letih od 1995 do 2008[3].



Slika 3: Rast števila uporabnikov v letih od 2000 do 2008 v [%][4].

[3] http://www.internetworldstats.com/emarketing.htm

1.1 Semantični splet in semantične spletne storitve

Semantični splet je nova tehnologija na katero lahko gledamo kot na nadgradnjo svetovnega spleta. Cilj semantičnega spleta je predvsem čimbolj avtomatsko graditi modele znanja ali ontologije iz poljubnih dokumentov, oz. te modele in ontologije uporabljati za označevanje pomembnih informacij z metapodatki. Ontologija je formalna, eksplicitna specifikacija skupne konceptualizacije [8]. Izbira jezika pri gradnji ontologije je predvsem odvisna od stopnje formalnosti in granularnosti, ki jo želimo vnesti v model.

Tako kot na področju semantičnega spleta, se je tudi na področju spletnih storitev v zadnjih šestih letih veliko ukvarjalo z koristno uporabo semantike v spletnih storitvah. Cilj semantičnih spletnih storitev je učinkovito povezovanje tehnologije semantičnega spleta in spletnih storitev. Semantično obogateno procesiranje podatkov, s podporo logičnemu sklepanju, bo postopoma omogočalo razvoj tehnologij za samodejno odkrivanje, izgradnjo in izvajanje storitev na svetovnem spletu [3]. Pri semantičnih spletnih storitvah naletimo na več ontologij. Prva izmed njih je OWL-S, ki izhaja iz DAML-S in predstavlja ontologijo za opis spletnih storitev z OWL. Ostali najbolj vidni pristopi so WSMO, SWSF in WSDL-S. Vsak pristop ima različen način vpeljave semantike v spletno storitev, ki pa se loči tudi po kompleksnosti.

V odkrivanje semantičnih spletnih storitev je bilo v zadnjih letih vloženo veliko truda. Obstaja vrsta algoritmov, ki uporabljajo različne pristope pri določitvi ujemanja storitev. Prav tako obstaja vrsta orodij, kjer so ti algoritmi realizirani in omogočajo izbiro ustrezne storitve. Zanesljivo in natančno odkrivanje igra pomembno vlogo, saj je le tako mogoče realizirati avtomatsko izvajanje ustreznih spletnih storitev.

1.2 Vsebina in namen diplomskega dela

Namen diplomskega dela je podati pregled najvidnejših orodij, ki se ukvarjajo z odkrivanjem semantičnih spletnih storitev. Predstavil bom tudi tehnologije semantičnega spleta, ki so neposredno povezane s spletnimi storitvami. Natančneje si bomo ogledali ontologije, s katerimi lahko različno natančno opišemo spletne storitve. Poleg orodij bom podal pregled algoritmov, ki jih orodja uporabljajo pri odkrivanju. Za lažje razumevanje bom predstavil primere uporabe odkrivanja spletnih storitev, ki so opisne z različnimi ontologijami. Prav tako bom poizkušal odgovoriti na vprašanje ali so tehnologije dovolj zrele za uporabo v resnejše namene.

V 2. poglavju si bomo natančneje ogledali področje semantičnega spleta. Spoznali bomo osnovne pojme. Podal bom tudi pregled jezikov s katerimi opišemo ontologije semantičnih spletnih storitev in okvirno predstavil trenutno stanje na področju semantičnega spleta.

V 3. poglavju bodo predstavljene semantične spletne storitve. Natančneje si bomo ogledali ontologije za opis semantičnih spletnih storitev. Za lažjo primerjavo med njimi pa sem izdelal tabelo, v kateri je povzet strnjen pregled nad njimi.

4. poglavje se nanaša na algoritme, ki jih uporabljamo pri odkrivanju semantičnih spletnih storitev, medtem ko so v 5. poglavju predstavljena nekatera izmed orodij, ki imajo te algoritme realizirane in tako omogočajo odkrivanje semantičnih spletnih storitev.

V 6. poglavju bom predstavil nekaj primerov uporabe odkrivanja spletnih storitev. Vsak primer bo opisan z različno ontologijo, posledično pa bo odkrivanje izvedeno z ustreznim orodjem.

---

[4] http://www.internetworldstats.com/stats.htm

2 Semantični splet

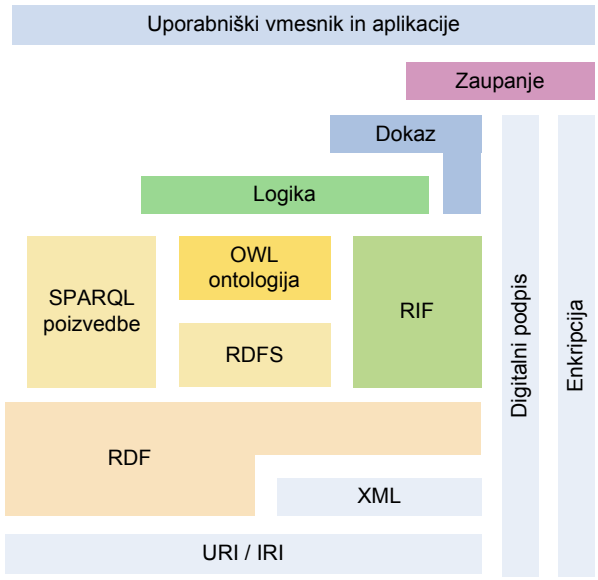
Semantični splet je nadgradnja svetovnega spleta, kjer so spletnim informacijam pripisani nedvoumni računalniško razumljivi metapodatki, ki bistveno olajšajo sodelovanje med ljudmi in računalniki [*Berners-Lee, Hendler, in ostali, 2001*].

2.1 Svetovni splet v povezavi s semantičnim spletom

Trenutno je večina dokumentov na svetovnem spletu napisanih v HTML. Ta jezik je zelo uporaben pri predstavitvi informacij, ki so namenjene človeški uporabi, saj temelji na vizualni predstavitvi podatkov v spletnem brskalniku. Celo dokumenti, izdelani na podlagi šablon z dobro definirano strukturo in pomenom, ne olajšajo dela programskim agentom. Npr. pri izmenjavi podatkov v formatu XML morata obe strani nujno določiti XML-shemo, da lahko razbereta ustrezen pomen.

Tim Berners-Lee, ustanovitelj svetovnega spleta, je v začetku tega tisočletja predstavil idejo semantičnega spleta kot nadgradnjo obstoječega spleta, kjer informacije dobijo pomen, kar omogoča boljše medsebojno sodelovanje ljudi in računalnikov. Proces gradnje semantičnega spleta je v velikem razmahu. Pod okriljem W3C poteka množica aktivnosti in pobud pri določitvi standardov, jezikov in tehnologij, ki bodo omogočale predstavitev in povezovanje informacij na spletu ter njihovo učinkovitejše odkrivanje, strnjevanje in ponovno uporabo.

Semantični splet je zgrajen v obliki sklada, ki je trenutno predlagan, in ga prikazuje Slika 4.



Slika 4: Večplastna struktura semantičnega spleta[5].

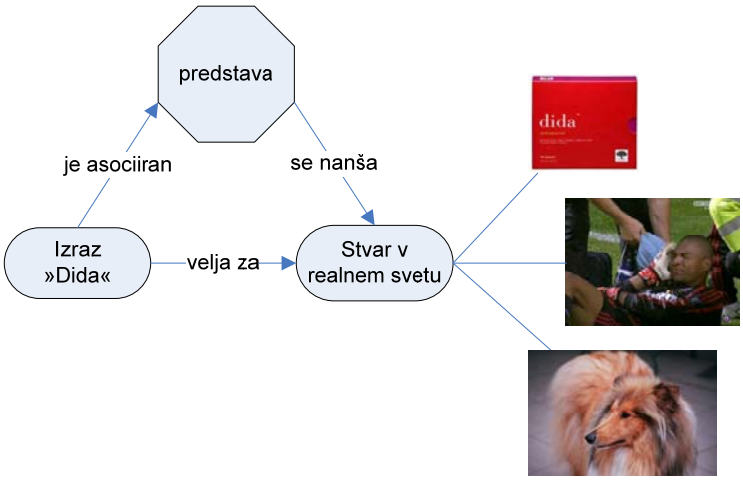
V nadaljevanju si bomo ogledali nekatere izmed ravni semantičnega spleta.

URI in **IRI**[6] poznamo že iz svetovnega spleta in jih v povezavi s semantičnim spletom uporabimo kot nize, s katerimi predstavimo objekte (resurse). Ti objekti predstavljajo konkretne stvari, kot npr. pes Dida, nogometaš Dida,

[5] http://www.w3.org/2007/03/layerCake.svg
[6] razlika med URI in IRI je, da slednji temelji na Unicode standardu, medtem ko prvi na uporablja standard ASCII.

spletni naslov strani www.pintar.tusek.info, itd. Z njimi rešujemo probleme dvoumnosti, ki se lahko pojavi z uporabo naravnega jezika. Kot primer si oglejmo izraz »Dida«, ki lahko v realnem svetu predstavlja številne stvari (Slika 5):

- Tablete s cimetovim oljem
- Brazilski nogometaš Nelson Dida, ki brani za AC Milan
- Ime samice, ki je pasme škotski ovčar

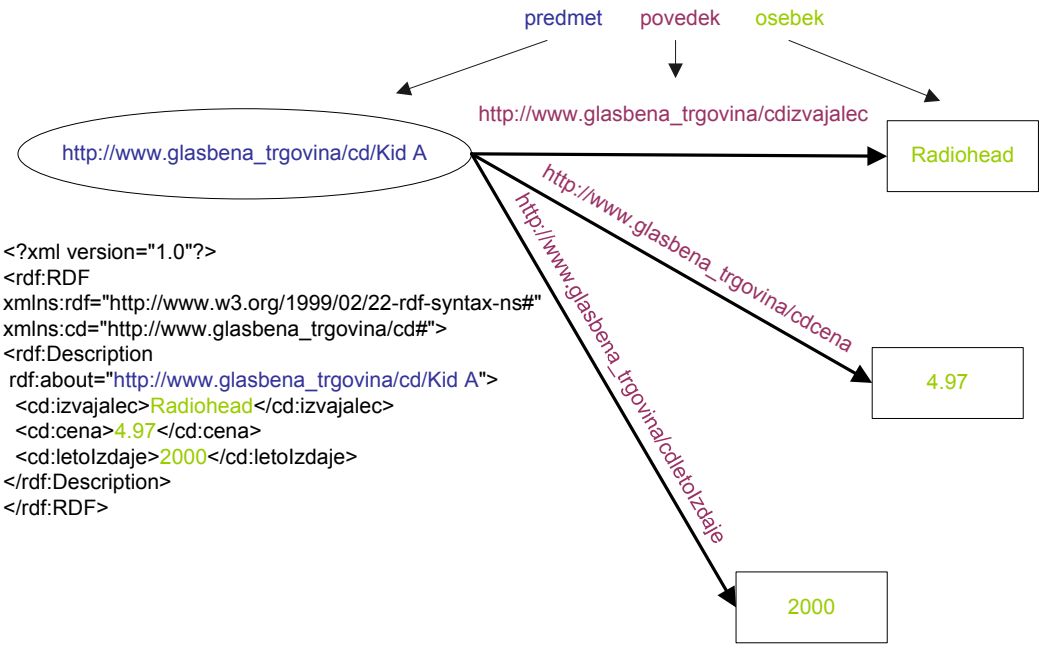


Slika 5: Homonim »Dida« v trikotniku pomena

Ta problem rešimo z ustrezno navedbo URI-jev (npr. http://www.nogomet.si/ACMilan/Dida, http://www.pasme.si/škotskiOvčar/samica/AndromaEdenLas/Dida).

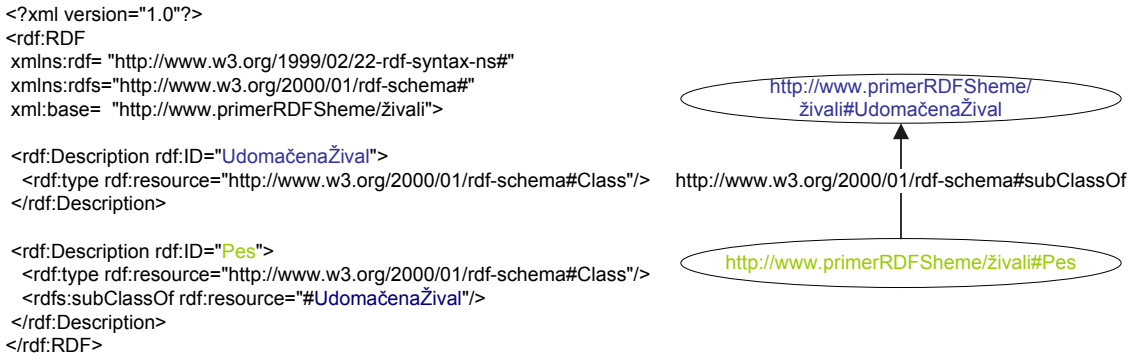
RDF[7] lahko obravnavamo kot prvi nivo semantičnega spleta. Namenjen je opisovanju virov, kjer je vir opisan z identifikatorjem URI. Dokument RDF sestavljajo trojice predmet-povedek-osebek, ki jih lahko zapišemo kot povedek(predmet, osebek). Npr. izvajalec(Kid A, Radiohead). Podatkovni model dokumentov zapisanih v RDF pa je usmerjen, označen graf med objekti.

[7] Resource Description Framework - ogrodje za opisovanje virov.



Slika 6: Primer RDF-grafa.

RDFS[8] določa preprost jezik za modeliranje RDF-dokumentov. RDF in RDFS skupaj označujemo RDF/S in izrazno nista zelo močna. Z njima lahko izrazimo koncepte, taksonomije[9] in binarne relacije.



Slika 7: Primer RDF-sheme.

OWL je postal priporočilo s strani W3C februarja 2004. Je razširitev XML, RDF in RDF-sheme in omogoča podrobnejšo predstavitev znanja na področju semantičnega spleta. Podrobneje je predstavljen v poglavju 2.3.1 OWL.

SPARQL[10] je postal priporočilo s strani W3C junija 2007. Je eden izmed najpomembnejših poizvedovalnih jezikov s katerim lahko poizvedujemo po podatkih zapisanih v RDF. Sintaksa je podobna kot pri SQL. Primer prikazuje preprosto poizvedbo, ki vrne vse oznake knjig in njihovih naslovov, ki se nahajajo v RDF-bazi.

[8] RDF Schema.
[9] V matematiki je taksonomija drevesna zgradba za dano množico objektov.
[10] SPARQL Protocol and RDF Query Language,

```
PREFIX books:   <http://example.org/book/>
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
SELECT ?book ?title
WHERE
{ ?book dc:title ?title }
```

Na najvišjih ravneh sklada se nahajajo metode za sklepanje in dokazovanje. Na samem vrhu se nahajajo aplikacije, ki uporabljajo spodaj ležeče tehnologije.

Kljub vsem prednostim, ki jih obljubljajo nove tehnologije, pa bo za njihov uspeh pomembna prav posebna pozornost pri naslednjih problemih [5]:

- **razpoložljivost semantične vsebine na spletu:** trenutno je na spletu prisotno še zelo malo podatkov s semantično vsebino. Semantično povezovanje informacij omogoča prehod iz današnjega spleta v izrazno bogatejši splet.

- **razpoložljivost ontologij, razvoj in evolucija:** potrebno bo zagotoviti primerno infrastrukturo za razvoj univerzalnih in domenskih ontologij, za preslikavo in njihovo združevanje ter za primeren nadzor nad evolucijo ontologij in označevalne vsebine.

- **razširljivost:** veliko truda bo potrebno vložiti za učinkovito organizacijo shranjevanja in iskanja vsebine semantičnega spleta. Vse te naloge morajo biti skrbno načrtovane in izvedene, saj se pričakuje velika rast semantičnega spleta.

- **večjezičnost:** problem se pojavlja že na obstoječem svetovnem spletu. S pomočjo semantičnega spleta pa bi bil lažje obvladljiv, saj naj bi pristopi semantičnega spleta omogočali dostop do informacij neodvisno od jezika.

- **vizualizacija:** intuitivna vizualizacija semantičnega spleta bo razreševala preveliko nasičenost s podatki. Uporabniki bodo zahtevali preprosto in učinkovito predstavitev njim relevantnih podatkov.

- **vzpostavitev in stabilnost jezikov semantičnega spleta:** s standardizacijo semantičnih jezikov se na podlagi priporočil konzorcija W3C ukvarja vrsta raziskovalnih skupin.

2.2 Ontologija

Ontologije so ključni element pri semantičnem spletu, kjer se prepleta človeško razumevanje simbolov z zmožnostjo računalniškega procesiranja. Ta pomembna lastnost, ki jo imajo ontologije, pospešuje tako skupno, kot ponovno uporabo ontologij med uporabniki, kot tudi računalniškimi sistemi. Uporaba ontologij in podpornih orodij prinaša možnosti za izboljšave obvladovanja znanja predvsem v večjih organizacijah. Ključni koraki pri uporabi ontologij so naslednji [6]:

- **Pridobivanje oz. učenje ontologij in povezovanje le-teh z velikimi količinami podatkov.**
  Zaradi prilagodljivosti mora biti ta proces avtomatiziran s pomočjo metod za luščenje informacij in procesiranje naravnega jezika.

- **Shranjevanje in vzdrževanje ontologije in njenih primerkov.**
  Potrebno je sprejeti odločitev, v kakšni obliki bo ontologija shranjena, tako da nam je na voljo repozitorij z lastnostmi podatkovne baze in enostavni obrazci za sklepanje na podlagi elementov ontologije.

- **Iskanje po semantično obogatenih informacijskih virih.**
  Sama predstavitev znanja in informacij ni dovolj, ampak morajo uporabniki ontologije uporabljati in po njih povpraševati. Pojavljajo se novi načini povpraševanja, ki nam omogočajo pridobivanje znanja iz porazdeljenih virov, vse raziskave pa se usmerjajo k semantično obogatenim iskalnikom.

Nekatere lastnosti ontologije:

- tvori hrbtenico semantičnega spleta
- določa terminologijo za semantično obogatitev elementov
- omogočajo sklepanje, osnovano na formalizmih
- računalnik se začne zavedati vsebine
- namenjene so za skupno rabo

2.2.1 Definicija Ontologije

Beseda ontologija izvira iz filozofije, kjer razlaga teorijo obstoja[11]. V zadnjem desetletju je beseda postala pomembna na različnih računalniških področjih, predvsem na področjih umetne inteligence, jezikovnih tehnologij, biomedicinske informatike, pri upravljanju z znanjem, itd. Ontologijo opredeljujejo besede in njihov pomen, ki se uporabljajo pri predstavitvi znanja na določenem področju. V literaturi obstaja množica definicij o tem, kaj naj bi ontologija bila, zato si oglejmo nekatere izmed njih.

- Filozofska disciplina, veja metafizike, ki obravnava osnovo, vzroke in najsplošnejše lastnosti stvarnosti.

- Določena teorija o naravi bitij in njihovem obstoju.

- Ontologija določa osnovne izraze in relacije, ki sestavljajo slovar in pravila za kombiniranje izrazov in relacij, ki določajo razširitve slovarja.

- Ontologija je eksplicitna specifikacija konceptualizacije.

Ta definicija je postala v literaturi ena izmed najbolj citiranih. Bistvo te definicije je, da so tipi uporabljenih konceptov, s katerimi opišemo abstraktni model določene domene iz realnega sveta in omejitve pri njihovi uporabi, natančno (eksplicitno) definirani [8].

- Ontologija je formalna, eksplicitna specifikacija skupne konceptualizacije.

Definicija je podobna zgornji. Beseda *skupno* odraža stališče, da ontologija zajema sporazumno znanje, torej znanje, ki ni vezano na posameznika, ampak ga podpira skupina ljudi. Beseda *formalna* se nanaša na dejstvo, da mora biti ontologija strojno berljiva. Ontologije lahko modeliramo z uporabo različnih formalizmov. Ustrezen formalizem izberemo na podlagi stopnje formalnosti in podrobnosti, ki jo zahtevamo pri predstavitvi problemske domene. Pri modeliranju bolj zapletenih ontologij bomo npr. izbrali formalizme, ki temeljijo na opisni logiki, logiki prvega reda, F-logiki, saj imajo večjo izrazno moč. V primeru modeliranja preprostejših ontologij pa lahko uporabimo formalizma UML in ER [4].

[11] http://www.formalontology.it/ontology-definitions-one.htm

2.2.2 Tri ravni predstavitve ontologije

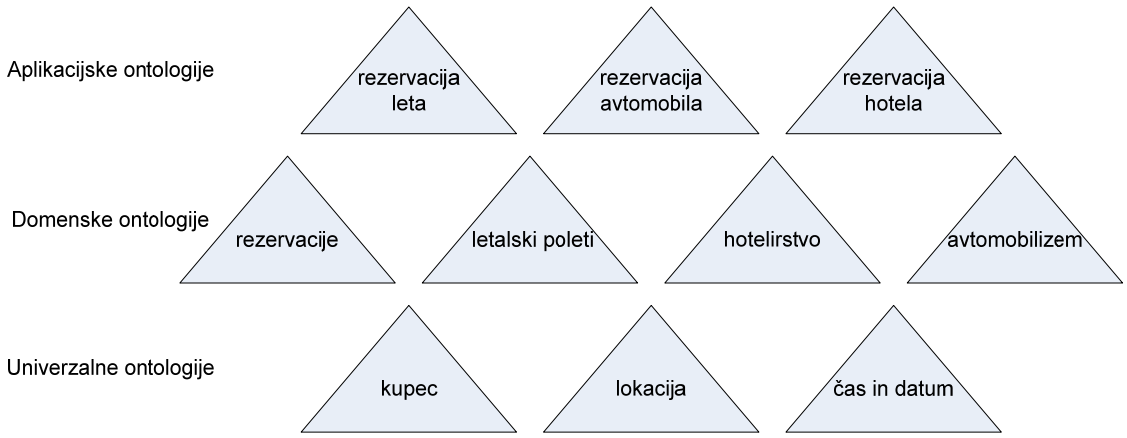
Glede na splošnost konceptov lahko ontologije razdelimo v tri skupine.

- **Univerzalne ontologije:** z njimi lahko opišemo zelo splošne koncepte, ki so skupni vsaki domeni. Cilj je doseči čim širše medsebojno sodelovanje ontologij, do katerih lahko dostopamo preko te univerzalne ontologije. V to skupino ontologij lahko štejemo ontologijo CyC, BFO (Basic Formal Ontology), DOLCE in ontologijo DnS, GFO (General Formal Ontology), WordNet, SUMO (Suggested Upper Merged Ontology), Biomedical Ontology. Trenutno nobena izmed naštetih ontologij ni dovolj razširjena, da bi bila sprejeta kot »de-facto« standard.

- **Domenske ontologije:** uporabimo jih pri modeliranju specifičnega problemskega področja iz realnega sveta. Z njimi predstavimo pomen izraza, ki se nanaša na določeno domeno, saj ima isti izraz lahko različen pomen kot npr. Dida na sliki 5.

    - **e-poslovanje** (UNSPSC, NAICS, SCTG, E-cl@ss, RosettaNet)
    - **medicinske** (GALEN, UMLS, ON9)
    - **inženirske** (EngMaht, PhysSys)
    - **poslovne** (Enterprise, TOVE)
    - **kemijske** (Chemicals, Ions, Pollutants)
    - **…**

- **Aplikacijske ontologije** se nanašajo na konkretno aplikacijo.



Slika 8: Prikaz ontologij glede na splošnost konceptov [10].

2.3 Pregled jezikov za zapis ontologij

Jeziki za zapis ontologij predstavljajo glavne nosilce informacije, ki jo hočemo objaviti in deliti z drugimi. V splošnem bodo jeziki za predstavitev pravil igrali ključno vlogo, saj bodo kot lingua franca za izmenjavo pravil med različnimi sistemi in orodji ter bodo omogočili razvijanje, izvajanje, objavljanje in medsebojno komunikacijo pravil na semantičnem spletu [11]. V poglavju si bomo ogledali tri jezike, s katerimi opišemo ontologije, ki se nanašajo na spletne storitve. Med seboj se ločijo predvsem po izrazni moči, ki jo nudijo, in s tem povezanimi formalizmi na katerih so osnovani.

2.3.1 OWL

OWL (Web Ontology Language) je semantični označevalni jezik za objavljanje in izmenjavo ontologije. OWL razširja RDFS in je izveden iz spletnih ontoloških jezikov DAML+OIL (Darpa Agent Markup Language + Ontology Interface Layer). OWL je zasnovan za uporabo v aplikacijah, ki procesirajo vsebino dokumentov in se ne ukvarjajo samo z predstavitvijo vsebine uporabniku. OWL omogoča večjo strojno prevedljivost vsebin, kot pa to omogočajo XML, RDF, RDF-S. V družino OWL spadajo trije jeziki, ki se razlikujejo glede na izrazno moč, ki jo nudijo:

- **OWL Lite** nudi najmanj izrazne moči. Uporabljajo ga predvsem uporabniki, ki potrebujejo hierarhično klasifikacijo konceptov in preproste omejitve. Npr. pri številnosti lahko uporabljamo vrednosti 0 ali 1.

- **OWL DL** nudi največjo izrazno moč, pri tem pa ohranja dovolj dobro izračunljivost. To pomeni, da se vse stvari lahko procesirajo v realnem času.

- **OWL Full** je namenjen uporabnikom, ki želijo največjo izrazno moč in sintaktično svobodo. Zaradi svoje kompleksnosti pa je zelo malo verjetno, da bodo v prihodnosti razvita orodja, ki bodo omogočala sklepanje na podlagi OWL Full.

Vsak izmed zgoraj naštetih jezikov predstavlja razširitev svojega predhodnika. Zato veljajo naslednje trditve (njihovi inverzi ne veljajo):

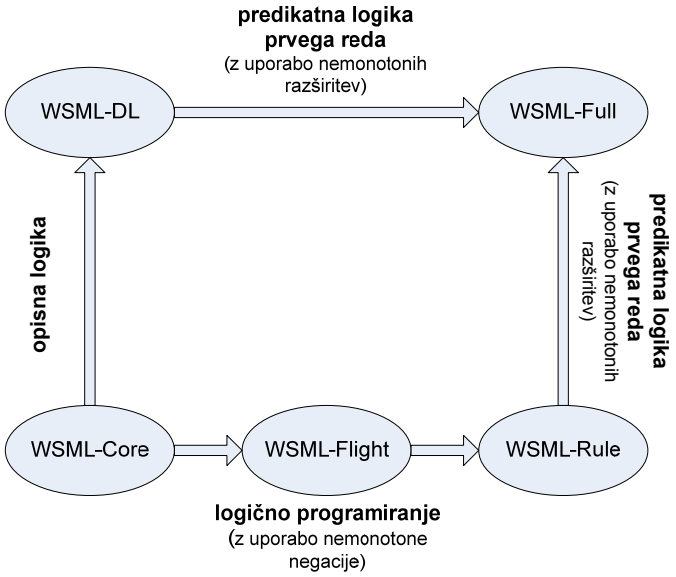
- Vsak zapis v ontologiji z OWL Lite lahko zapišemo tudi z OWL DL.
- Vsak zapis v ontologiji z OWL DL lahko zapišemo tudi z OWL Full.
- Vsak veljaven sklep zapisan v OWL Lite je veljaven tudi z OWL DL.
- Vsak veljaven sklep zapisan v OWL DL je veljaven tudi z OWL Full.

Gradnike, ki jih vsebuje OWL, lahko bralec najde na spletni strani http://www.w3.org/TR/owl-features/.

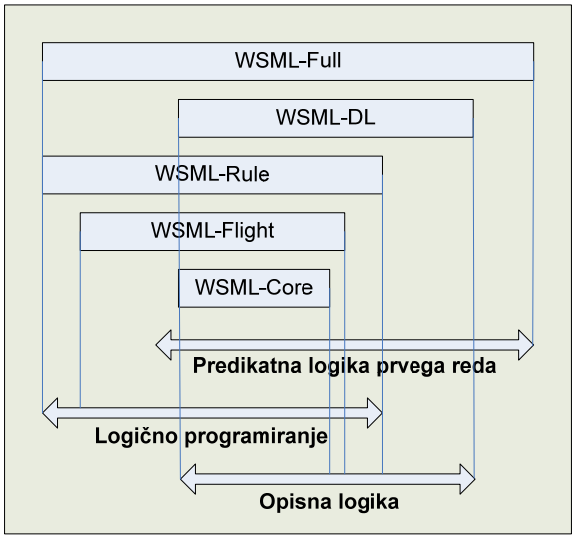
2.3.2 WSML

WSML (Web Service Modeling Language) poskrbi, da lahko formalno opišemo vse elemente, ki so definirani v WSMO. Osnovan je na množici različnih formalizmov kot so opisna logika, logika prvega reda, programska logika, ki se uporabljajo pri modeliranju semantičnih spletnih storitev. Formalizmi pa se dalje delijo na več ravni. Od WSML-Core, WSML-DL, WSML-Flight, WSML-Rule do WSML-Full, ki ima največjo izrazno moč. Največji razlog za nastanek tega jezika je bila šibkost OWL-S na konceptualni ravni, kot tudi lastnosti OWL.

Slika 9 prikazuje različice WSML-ja in razmerja med njimi, medtem ko slika 10 prikazuje različne ravni v WSML [12]. Različice se razlikujejo glede na izrazno moč, ki jo nudijo. S tem, ko je na voljo več različic, uporabnikom omogočimo, da kompenzirajo izrazno moč in kompleksnost pri klasificiranju. Puščice na sliki 9 prikazujejo smer širitve izrazne moči.



Slika 9: Različice WSML.



Slika 10: Ravni v WSML.

Ne-monotona negacija (bolje poznana *negacija kot neuspeh*) se večinoma uporablja v povezavi s podatkovnimi bazami. Princip *negacije kot neuspeha* je izločevanje posebnih primerov. Npr. Prolog pozna dva posebna cilja *true* in *fail*. Prvi vedno uspe, drugi pa nikoli. Z uporabo reza lahko enostavno izločimo nek posebni primer in priredimo celotnemu cilju vrednost fail[12].

WSML različice:

WSML-Core

Nudi najmanjšo izrazno moč izmed vseh jezikov, ki spadajo v družino WSML, vendar pa je zaradi tega sklepanje veliko lažje.

[12] www.ailab.si/tomaz/MOS/ vaje2005/8_vaja_25_4_2005.zip

WSML-DL
Razširja WSML-Core z uvedbo opisne logike.
WSML-Flight
Razširja WSML-Core in ima veliko lastnosti podobnih OWL Full. Omogoča meta-modeliranje, uvedbo omejitev in negacije kot neuspeha.
WSML-Rule
Razširja WSML-Core z uvedbo Hornove oblike pravil[13].
WSML-Full
Zaključuje vse različice z uporabo logike prvega reda.

Gradnike, ki jih jezik WSML vsebuje, lahko bralec najde na spletni strani http://www.wsmo.org/TR/d16/d16.1/v0.2/.

2.3.3 SWSL

SWSL (Semantic Web Service Language) je jezik s katerim lahko formalno opišemo koncepte in obnašanje spletne storitve. Tvorita ga dva različici:

- *SWSL-FOL*
  Temelji na predikatni logiki prvega reda. Z njim lahko popolno opišemo SWSO. Posebno pa pride do izraza pri opisovanju procesnega (storitvenega) modela.
- *SWSL-Rules*
  Je logični jezik na osnovi pravil. Poleg ostalih lastnosti priskrbi podporo pri objavi, odkrivanju in primerjanju storitev.

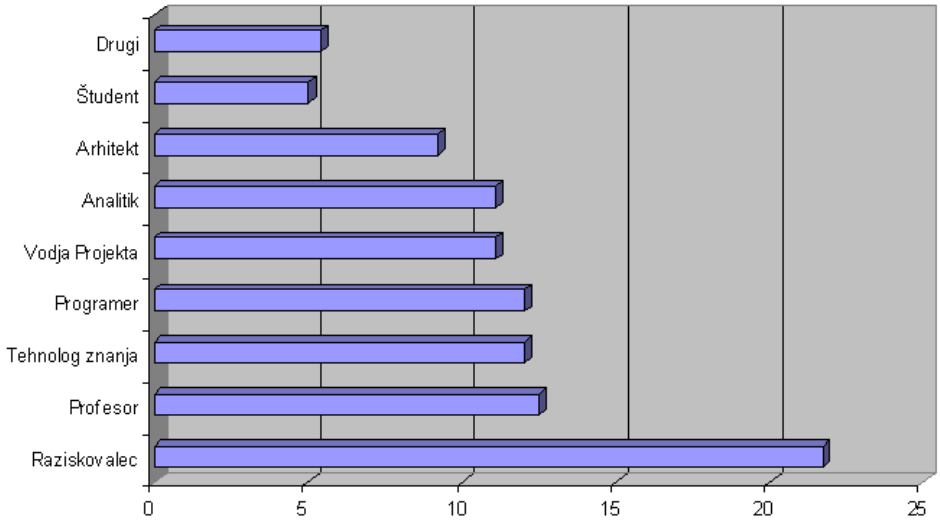
Oba, SWSL-Rules in SWSL-FOL, sta predstavljena kot nivojska jezika. Za razliko od OWL pa nivoji niso določeni glede na izrazno moč in izračunljivost, ampak vsak nivo vsebuje določeno število novih konceptov, s katerimi obogatimo modelirno moč jezika. Večina konceptov je skupna obema različicama, tako da je omogočeno preprosto prehajanje med njima.

Večina ravni, ki razširjajo jedro, je med seboj neodvisnih. Tak način omogoča, da lahko uporabimo samo določeno množico lastnosti jezika.

2.4 Stanje na področju semantičnega spleta

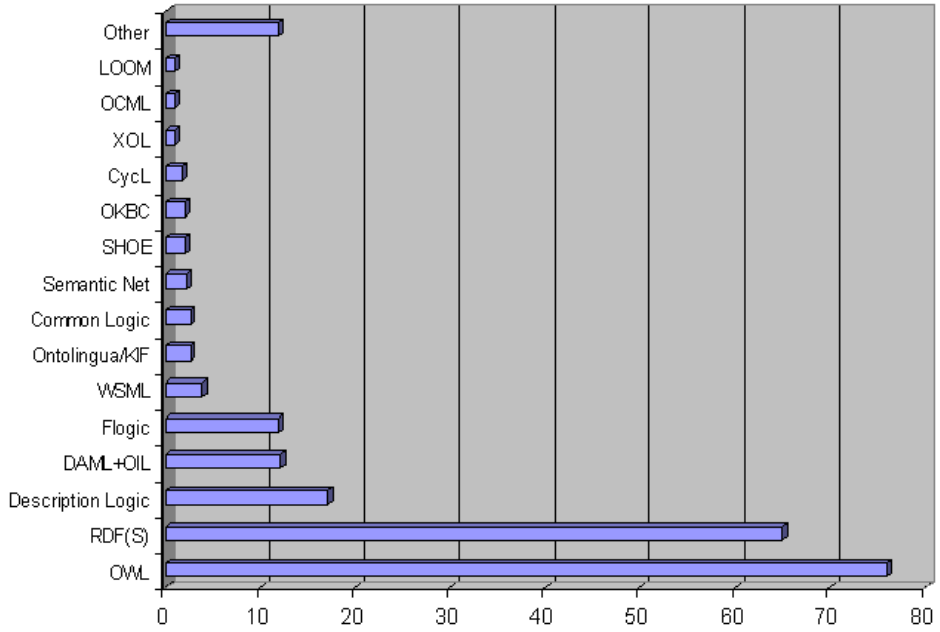
Raziskava, ki jo je opravil *Jorge Cardoso,* prikazuje stanje na področju ontologij in semantičnega spleta [13]. Temelji na 627 anketirancih, ki so preko spleta izpolnili vprašalnike zaprtega tipa v obdobju dveh mesecev (12. december 2006 – 26. januar 2007).

[13] Pravila so preprosto v obliki implikacije med vzroki (telo) in posledicami (glava). Pomen pravila pa lahko preberemo kot: vedno, ko je zadoščeno pogojem v telesu, morajo držati tudi posledice v glavi.



Slika 11: Vloga anketiranca [13].

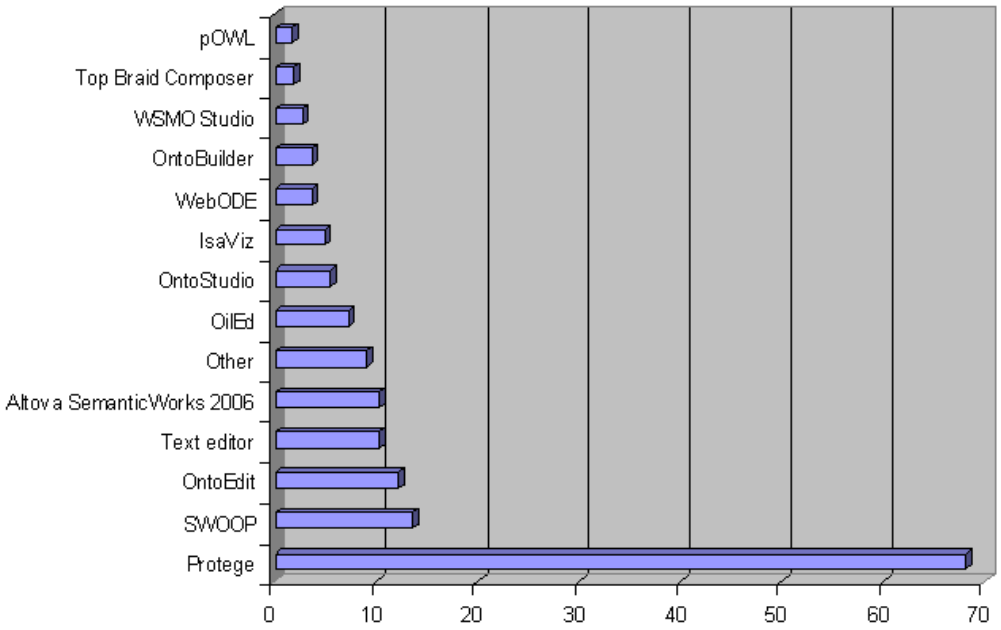
Ob izpolnitvi vprašalnikov je vsak anketiranec moral navesti tudi vlogo, ki jo ima v združbi. Slika 11 lepo prikazuje, da se trenutno s semantičnim spletom najbolj ukvarjajo v akademskih krogih. Ko bodo orodja postala dovolj zrela, pa bodo verjetno tehnologijo širše sprejela tudi IT podjetja.



Slika 12: Prisotnost ontologij med uporabniki[13].

Vidimo, da je uporaba OWL (75,9%) veliko bolj razširjena kot uporaba WSML (3,7%). To je pričakovan rezultat. OWL je preko DAML-S prisoten že od samih začetkov, za razliko od WSML-ja, ki je osnovan na novejših temeljih. Opozoriti moram tudi na to, da se OWL ne uporablja izključno v zvezi s spletnimi storitvami, medtem ko se uporaba WSML večinoma nanaša na semantične spletne storitve (to pa ne pomeni, da ga ne moremo uporabiti tudi za druge namene).

Čeprav je pristop WSMO boljši od pristopa OWL-S, moramo imeti v mislih dejstvo, da je trenutno OWL veliko bolj razširjen jezik. Iz tega sledi, da so problemske domene največkrat opisane z uporabo OWL. Torej lahko pri konceptualnem opisu uporabimo že obstoječe ontologije, kar pa poenostavi gradnjo in pohitri iskanje semantičnih spletnih storitev, saj ni potrebna uporaba posrednikov.

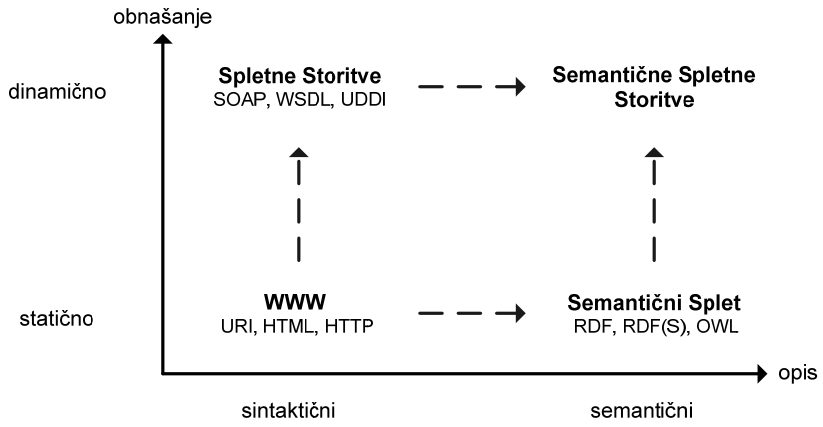


Slika 13: Razširjenost orodij za gradnjo ontologij med uporabniki[13].

Kot zanimivost lahko prikažemo tudi prisotnost orodij med uporabniki. Opazimo, da urejevalnik Protege občutno vodi pred ostalimi orodji. To je razumljivo, saj je orodje brezplačno, ontologijo modeliramo z OWL, ki je razširjen, lahko najdemo veliko dokumentacije, ki razlaga gradnjo ontologij, itd. Vendar je potrebno biti previden pri uporabi prikazanih deležev na sliki 13. Če bi raziskava temeljila samo na semantičnih spletnih storitvah, bi bila razlika v razmerju med urejevalnikoma Protege in WSMO Studio manjša. OWL in Protege bi kljub vsemu še vedno vodila pred WSLM in WSMO Studiom, saj lahko v orodju Protege modeliramo tudi OWL-S.

3 Semantične spletne storitve

Velik potencial predstavlja ideja povezovanja semantičnega spleta in spletnih storitev (slika 14). Ta dva dopolnjujoča se trenda obljubljata prehod iz spleta, ki je namenjen samo človeškemu uporabniku, v splet kjer bodo spletne storitve sposobne procesiranja računalniku razumljivih vsebin. Pojavili so se številni predlogi za semantično obogatitev spletnih storitev. Izredno veliko pozornosti se namenja ontologijam OWL-S, SWSO, WSDL-S / SAWSDL in WSMO, zato si jih bomo v naslednjih poglavjih ogledali podrobneje.



Slika 14: Evolucija svetovnega spleta.

3.1 Spletne storitve

Pričakujemo lahko, da bodo spletne storitve preplavile svetovni splet podobno kot so ga spletne strani v preteklosti [2]. Pri iskanju ustrezne spletne storitve pa bomo zaradi velike množice spletnih storitev naleteli na podobno težavo, kot pri izbiranju informacij iz preobsežne zbirke podatkov, kar trenutno predstavlja velik problem na svetovnem spletu.

Trenutni opisi spletnih storitev imajo veliko pomanjkljivost, saj v spletno storitev ne vnašajo dovolj semantike, ki je za avtomatizacijo spletnih storitev nujno potrebna. Omogočajo sicer opis kako lahko storitev obudimo, katere operacije lahko kličemo itd., vendar pa je ta opis v naravnem jeziku, tako da je berljiv le za človeške uporabnike. Da bi odpravili ta problem, se je začelo veliko dogajati tudi na področju semantičnih spletnih storitev. Razvilo se je več ontologij, s katerimi lahko opišemo spletno storitev in tako vanjo vnesemo potrebno semantiko.

3.2 Ontologije za opis semantičnih spletnih storitev

V grobem lahko semantičnim spletnim storitvam pripišemo dva pristopa. Prvi temelji na razvoju novih standardov za semantične spletne storitve, drugi pa vnaša semantiko med trenutno sprejete in že poznane standarde.

Kljub nekaterim temeljnim konceptualnim razlikam imajo vsi pristopi (WSMO, OWL-S, SWSO, WSDL-S) nekatere skupne točke.

- Splošna *klasifikacija storitve* je skupna vsem pristopom.
- Uporaba *pred-pogojev* in *po-pogojev* za uspešno izvedbo storitve je skupna vsem pristopom.
- Noben pristop ne zanemari nefunkcionalnih vidikov spletne storitve.

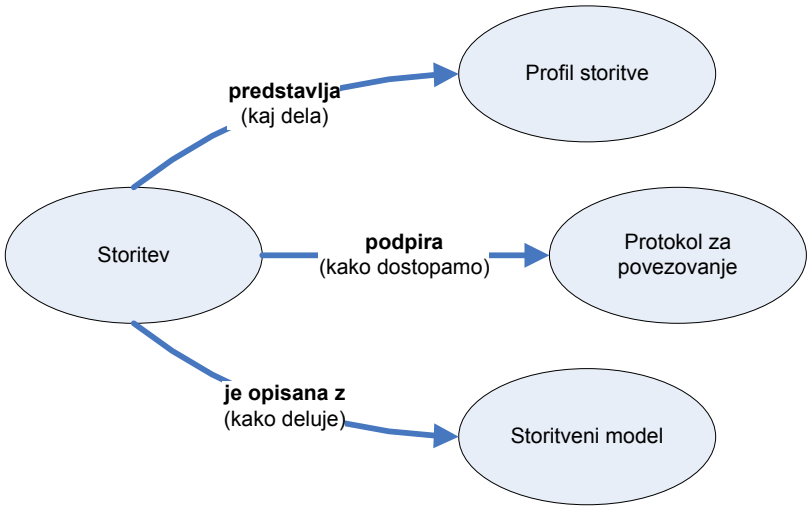
3.2.1 OWL-S

Pobuda OWL-S je nastala v krogih ameriških raziskovalnih inštitucij, ki vključujejo SRI International, BBN Technologies, Nokia, Yale University, Carnegie Mellon University, The MIND Laboratory of the University of Maryland itd. OWL-S poskuša odgovoriti na tri osnovna vprašanja, ki se postavljajo v zvezi s storitvijo [3]:

- **Kaj storitev omogoča potencialnemu odjemalcu?**
  Odgovor je podan v *profilu storitve*, ki se uporablja za oglaševanje storitve. Profil storitve vsebuje take informacije o delovanju storitve, da lahko inteligentni agent ugotovi ali storitev ustreza njegovim zahtevam. V profilu storitve zato navedemo kaj storitev omogoča, omejitve pri uporabi, kakovost storitve in pogoje, ki jih mora odjemalec izpolniti, da lahko storitev uspešno uporablja.

- **Kako se storitev uporablja?**
  Odgovor na to vprašanje podaja *storitveni model*, kjer je podrobno določena semantična vsebina zahtev in pogoji, pod katerimi dobimo rezultat. Storitveni model torej opisuje kako zahtevati storitev in kaj se zgodi, ko se storitev izvaja. Pri kompleksnejših storitvah ta opis lahko uporablja agent za različne namene: izvajanje podrobnejših analiz, ali storitev izpolnjuje vse zahteve agenta, pri kompoziciji opisov storitev za izvedbo specifične naloge, kontroliranje izvajanja storitve itd.

- **Kako do storitve dostopamo?**
  Protokol za povezovanje opredeljuje podrobnosti o tem, na kakšen način lahko avtonomni program oz. agent dostopa do storitve. Ponavadi vključuje komunikacijski protokol, format sporočil in ostale podrobnosti o storitvi.



Slika 15: Najvišja raven ontologije OWL-S.

Profil storitve

Vsaka storitev je lahko opisana z enim ali več profili storitve. Profil storitve se deli na tri dele:

- **Tekstualni opis**, **ime storitve**, **kontaktne informacije** ponudnika storitve, ki so namenjeni predvsem interakciji z ljudmi.
- **Funkcionalni opis** storitve, ki je predstavljen z vhodi in izhodi. Dodatno lahko določimo tudi pogoj, ki mora biti izpolnjen pred izvajanjem in učinek, ki predstavlja stanje po uspešni izvedbi storitve. Od štirih elementov VIPU (vhod, izhod, predpogoj, učinek) zgradba predpogojev in učinkov trenutno še ni natančno določena, zato jih primerjalni algoritmi še ne upoštevajo[15].
- **Dodatne lastnosti**, ki se uporabljajo pri opisu lastnosti storitev. Npr. kategorija storitve, kakovost storitve, čas izvajanja storitve, itd..

Storitveni model

Z namenom, da podamo podroben način interakcije s spletno storitvijo, lahko spletno storitev predstavimo kot proces. Ta proces ni mišljen kot program, ki naj bi se izvedel, ampak kot določitev poti, po katerih lahko odjemalec komunicira s storitvijo. OWL-S razlikuje med atomarnim procesom, preprostim procesom in sestavljenim procesom [16].

Atomaren proces ustreza zaporedju aktivnosti, ki jih storitev lahko izvede z natanko eno interakcijo. Lahko ga neposredno obudimo s tem, ko mu priskrbimo ustrezne vhodne podatke. Ne moremo ga razčleniti na podprocese in jih izvesti v enem koraku. Vsakemu atomarnemu procesu moramo določiti tudi ustrezen protokol za povezovanje, ki omogoča uporabniku storitve, da priskrbi procesu ustrezne podatke (sporočila).

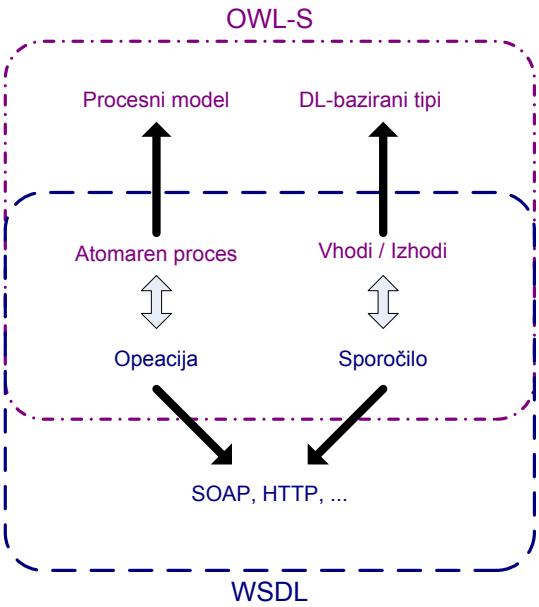
Preprostemu procesu ne določimo protokola za povezovanje, ampak ostanemo na stopnji abstrakcije. S preprostim procesom lahko ponazorimo potek atomarnega procesa kot tudi poenostavljen potek sestavljenega procesa (lažje razumevanje).

Sestavljen proces lahko razčlenimo v več sestavljenih, preprostih ali pa atomarnih procesov [2].

Protokol za povezovanje

Protokol za povezovanje preslika gradnike procesnega modela v podrobne opise sporočil, protokola, itd. OWL-S dovoljuje preslikavo tako gradnikov atomarnega procesa k ustrezni WSDL-operaciji, kot tudi opisov vhodov oz. izhodov k ustreznemu WSDL-sporočilu.

Protokol za povezovanje (OWL-S/WSDL grounding) temelji na podlagi treh soodvisnosti med OWL-S in WSDL. Slika 16 prikazuje prvi dve.



Slika 16: Preslikave med OWL-S in WSDL[14].

[14] http://www.w3.org/Submission/OWL-S

1. Atomaren proces v OWL-S ustreza operaciji v WSDL 1.1. Različni načini so lahko:
   - Atomaren proces z vhodi in izhodi ustreza operaciji *request-response*.
   - Atomaren proces z vhodi brez izhodov ustreza operaciji *one-way*.
   - Atomaren proces z izhodi brez vhodov ustreza operaciji *notification*.

2. Množica vhodov in izhodov v WSDL-ju ustreza konceptu sporočil. Natančneje izhodi OWL-S ustrezajo delom izhodnega WSDL-sporočila, vhodi OWL-S pa delom vhodnega WSDL-sporočila.

3. Tipi vhodov in izhodov v OWL-S ustrezajo konceptu *abstract type* v razširjeni WSDL-notaciji.

Profil storitve zagotavlja potrebne informacije za določitev ustrezne storitve. Storitveni model in protokol za povezovanje pa skupaj vsebujeta dovolj informacij, da lahko začne avtonomni program oz. inteligentni agent storitev, ki jo je našel učinkovito uporabljati. OWL-S med drugimi sledi tudi smernicam razvoja XML po nivojih, saj nadgrajuje XML sintaktični jezik na najnižji ravni in OWL na višji semantični ravni ter tako ohranja združljivost nazaj.

Omeniti je potrebno, da so orodja, ki trenutno podpirajo izgradnjo opisov v OWL-S, dokaj nezrela za uporabo. Večina orodij niti ni grajenih z namenom, da bi se uporabljala za produkcijske namene. Čeprav trenutno orodja niso najbolj kakovostna in uporaba v resnejše namene ni priporočljiva [18], pa lahko z gotovostjo pričakujemo, da se bo s časom stanje izboljšalo.

3.2.1.1 Primerjava med OWL-S in WSMO

OWL-S je z uporabo OWL jezika, ki je predlagan kot priporočilo s strani W3C, pridobil dodaten zagon pred WSMO, vendar pa ima OWL-S to pomanjkljivost, da se morajo jeziki z večjo izrazno močjo pretvoriti v OWL. Kot eno izmed najbolj opaznih razlik bi lahko navedli izrazno moč, ki jo nudita jezika, v katerih sta opisani obe ontologiji.

Pri OWL-S je tako metamodel kot opis konkretne storitve določen z uporabo istega jezika, čemer pa se WSMO, ki temelji na MOF[15] modelu, uspešno izogne.

Poleg razlik pa so med konceptualnima modeloma tudi podobnosti. *Profil storitve* pri ogrodju OWL-S lahko primerjamo s *funkcionalnostjo* s strani spletne storitve ali cilja pri WSMO ogrodju. Seveda pa moramo imeti v mislih, da WSMO loči med pogledom s strani ponudnika in s strani uporabnika spletne storitve, medtem ko OWL-S tega ne loči. Prav tako se pri OWL-S opis nefunkcionalnih lastnosti omeji samo na *profil storitve,* medtem ko lahko pri WSMO-modelu določimo opis nefunkcionalnih lastnosti za vsak element.

WSMO za razliko od OWL-S priporoča tudi uporabo širše sprejetih besednjakov (angl. vocabularies) (npr. Dublin Core Metadata Element Set, FOAF, itd.) pri specifikaciji nefunkcionalnih zahtev.

Storitveni model pri OWL-S je na konceptualni ravni podoben vmesnikom v WSMO. Vendar razlika med opisom obnašanja spletne storitve navzven (koreografija) in opisom medsebojne interakcije spletnih storitev (orkestracija) v OWL-S ni posebej poudarjena tako kot pri WSMO [2]. Prav tako WSMO dovoljuje določitev množice vmesnikov, OWL-S pa dovoli samo en *storitveni model* in s tem način, kako zahtevati storitev in kaj se zgodi, ko se storitev izvaja.

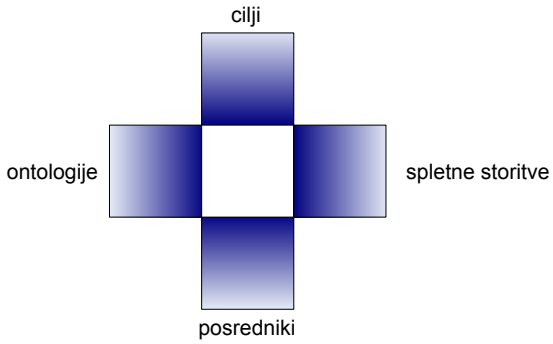
OWL-S ne določa posrednikov tako, kot jih WSMO. V ogrodju OWL-S na posrednike gledamo kot na posebne vrste spletnih storitev.

[15] MOF – Meta-Object Facility. MOF določa abstraktni jezik in ogrodje za opis, gradnjo in nadzor tehnološko neodvisnih meta-modelov.

Tako WSMO kot OWL-S pri protokolu povezovanja uporabljata podobne ideje, ki se dotikajo povezovanja z WSDL. Vendar pa je potrebno omeniti, da tako OWL-S kot WSMO nista omejena na WSDL kot edino tehnologijo za opis storitve.

3.2.2 WSMO

Predlog WSMO (Web Service Modeling Ontology) je nastal pod okriljem vodilnega raziskovalnega inštituta na področju semantičnega spleta in tehnologije semantičnih spletnih storitev v Evropi – Digital Enterprise Research Institute (DERI). Osnovni cilj avtorjev WSMO je bila opredelitev elementov za modeliranje, ki so potrebni za opisovanje semantičnih spletnih storitev izpeljanih iz konceptov Web Service Modelling Framework (WSMF). Slika 17 prikazuje elemente, ki pripadajo najvišji ravni ontologije [3].



Slika 17: Najvišja raven konceptov WSMO.

Ontologija je ključni element v WSMO, saj predstavlja osrednjo terminologijo za opis ostalih elementov. Poleg opredelitve formalne semantike informacij ima tudi vlogo povezovanja računalniške terminologije in terminologije, ki jo uporabljamo ljudje.

Spletna storitev omogoča povezovanje računalnikov in ostalih naprav s pomočjo spletno usmerjenih protokolov za izmenjavo podatkov. Zaradi svoje porazdeljene narave na svetovnem spletu jih odlikuje neodvisnost od računalniškega okolja. Vsaka spletna storitev predstavlja atomarni del funkcionalnosti, ki jo lahko uporabimo pri izgradnji bolj kompleksnih spletnih storitev. WSMO spletne storitve opisuje s treh vidikov: nefunkcionalnih lastnosti, funkcionalnosti in načina delovanja.

Cilj predstavlja zahteve, ki jih navede uporabnik storitve. Cilj si lahko predstavljamo tudi kot pričakovano funkcionalnost spletne storitve s strani odjemalca. Z uvedbo neodvisnih pojmov cilja in spletne storitve dosežemo razlikovanje med zahtevo in spletno storitvijo.

Posrednik opisuje elemente, s pomočjo katerih želimo odpraviti razlike med različnimi komponentami, ki jih uporabljamo pri WSMO opisovanju storitev. Posredniki nam omogočajo povezovanje heterogenih virov in odpravljajo nezdružljivost na različnih ravneh:

- *Raven podatkov* – posredujejo med različnimi terminologijami, ki so v uporabi in tako rešujejo problem integracije ontologij.
- *Raven procesa* – posredujejo pri različnih vzorcih komunikacije med heterogenimi viri. Takšna raznolikost se pojavi predvsem pri komunikaciji med spletnimi storitvami.

Ločimo 4 vrste posrednikov [19]:

- **OO-posredniki**

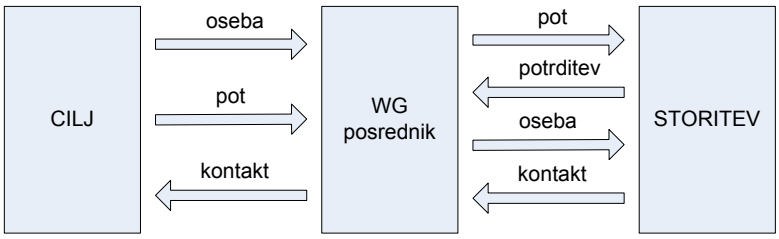
  Rešujejo razlike med različnimi ontologijami, ki opisujejo isto problemsko domeno. OO-posredniki so temeljne komponente, ki jih uporabljamo pri posredništvu, predlagane s strani WSMO. Premoščajo razlike med ontologijami, uporabljenimi za semantični opis vseh ostalih konceptov iz WSMO. OO-posrednike lahko uporablja katerakoli WSMO-komponenta (tudi posredniki), ko med ontologijami, uporabljenimi za konkretne semantične opise, pride do določenih neskladnosti.

- **GG-posredniki**

  Namen GG-posrednikov je povezati cilje in priskrbeti dodatno informacijo o njihovih razmerjih, da bi omogočili bolj dovršeno upravljanje z njimi. Poleg razreševanja možnih neujemanj na ravni podatkov, se GG-posredniki ukvarjajo tudi s funkcionalnimi razlikami dveh ciljev.

- **WG-posredniki**

  Opisujejo razmerja med spletnimi storitvami in cilji ter rešujejo neskladnosti med njimi z namenom, da priskrbijo dodatno podporo pri odkrivanju spletnih storitev. Te neskladnosti se lahko pojavijo med zahtevanimi in priskrbljenimi funkcionalnostmi, kot tudi med zahtevano in priskrbljeno koreografijo.



Slika 18: Primer posredovanja med storitvijo in ciljem.

V primeru, ki ga prikazuje slika 18, bo posrednik deloval na naslednji način:

1. v notranji repozitorij shrani parameter *oseba* za nadaljnjo uporabo
2. parameter *pot* posreduj storitvi
3. zadrži in izbriši parameter *potrditev* (ker ni navedena v cilju)
4. uporabi parameter *oseba* iz repozitorija in ga posreduj storitvi
5. parameter *oseba* posreduj storitvi

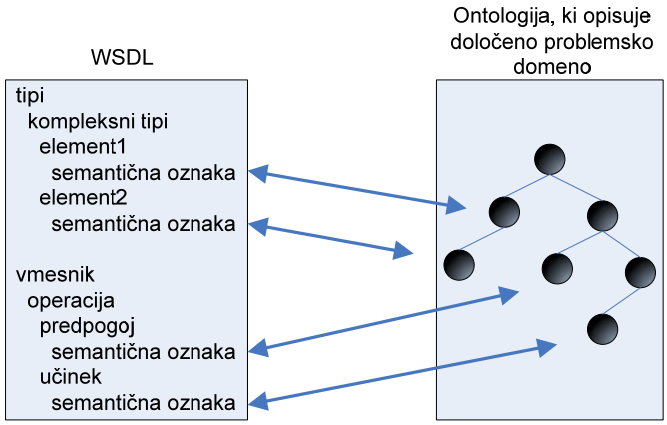
- **WW-posredniki**

  Rešujejo neujemanja med spletnimi storitvami, ki bi lahko ovirala interakcijo med spletnimi storitvami. Problem se lahko pojavi, ko mora spletna storitev za dosego želene funkcionalnosti združiti (obuditi) več različnih spletnih storitev.

3.2.3 WSDL-S / SAWSDL

Predlog WSDL-S je nastal pod okriljem laboratorijev LSDIS (Large Scale Distributed Information Systems). WSDL-S predpostavlja, da formalni semantični modeli, s katerimi lahko opišemo spletne storitve, že obstajajo in da do njih dostopamo preko referenc, ki jih navedemo v WSDL-dokumentu. S tem, ko ločimo WSDL-dokument in semantične modele, lahko za opis ontologije uporabimo poljuben jezik (WSML, OWL, UML,…). To omogoča uporabo že obstoječih modelov, ki opisujejo problemsko domeno v nekem jeziku (npr. UML).

WSDL-dokument predstavlja osnovo opisa spletne storitve. Semantične informacije, ki jih lahko vnesemo v WSDL dokument, so vhodi, izhodi, predpogoji in učinki operacij (VIPU). Prav tako pa lahko določimo tudi kategorijo storitve. S tem, ko vnesemo semantiko vhodom in izhodom, lahko na podlagi dodatno določenih predpogojev in učinkov omogočimo avtomatično odkrivanje spletne storitve. Predpogoji opisujejo zahteve, ki morajo biti izpolnjene preden se operacija lahko uspešno izvede, medtem ko učinki predstavljajo rezultat, do katerega bo pripeljala uspešno izvedena operacija. Prednost pristopa WSDL-S je, da gradi na obstoječih standardih, ki so širše sprejeti med uporabniki in se od njih ne oddalji preveč.



Slika 19: Semantično označevanje elementov v WSDL-dokumentu.

Dobra lastnost pristopa v primerjavi z uporabo OWL-S je ta, da WSDL-S ne podvaja opise (vhodi in izhodi), ki so sicer navedeni v WSDL dokumentu. Podvajanje opisov lahko pripelje do nekonsistentnosti.
Sicer so WSMO, OWL-S in SWSO teoretično boljši pristopi, vendar pa avtorji navajajo, da se v praksi pristop WSDL-S bolje obnese [20].

SAWSDL (Semantic Anntotations for WSDL) podobno kot WSDL-S določi mehanizem, s katerim lahko semantično obogatimo WSDL-dokument. SAWSDL ima z WSDL-S veliko skupnih lastnosti. Kot najbolj opazno razliko med SAWSDL in WSDL-S bi lahko navedli, da s prvim pristopom obogatimo dokument, ki je opisan v WSDL2.0, s slednjim pa dokument, ki je opisan v WSDL1.1. Koncepti so tako kot pri WSDL-S definirani zunaj WSDL-dokumenta in jih ustrezno vključimo v WSDL-dokument. WSDL-dokument lahko obogatimo na podlagi dveh osnovnih tipov:

- *modelReference*, s katerim lahko semantično obogatimo vmesnike, operacije, vhode, izhode ter elemente in atribute XML Sheme. Ta tip igra pomembno vlogo pri odkrivanju semantičnih spletnih storitev.

- *schemaMapping*, ki se deli na *liftingSchemaMapping* in *loweringSchemaMapping,* se uporablja pri razreševanju neenotnosti v podatkih in preslikavami iz enega načina predstavitve podatkov v drugega.

V prihodnosti nameravajo avtorji dodati še dva tipa. To sta predpogoj in učinek, ki sta v WSDL-S že realizirana.

3.2.3.1 Primerjava med WSDL-S / SAWSDL in WSMO

V primerjavi z WSMO, OWL-S in SWSF pri WSDL-S uporabimo manj kompleksen pristop vnosa semantike v opis spletne storitve. WSDL-S se zaradi svoje preprostosti ne more v celoti primerjati z ostalimi ontologijami za opis spletnih storitev. Vse koncepte, npr. tip sporočila, predpogoje, učinke in kategorijo storitve, lahko prav tako obogatimo v WSML/WSMO.

Razvijalec po svoji želji izbere jezik, ki opisuje ontologijo, s katero bo dodal semantiko spletni storitvi. Na račun tega je težko formalno določiti zahteve, poizvedbe ali pa ujemanja med zahtevano in ponujeno storitvijo. Omeniti moramo tudi, da so nefunkcionalne lastnosti (npr. kakovost storitve, cena storitve, dostopnost storitve) v WSDL-S upoštevane v standardih WS-*.

3.2.4 SWSO

Velik vpliv na konceptualni model SWSO (Semantic Web Services Ontology) je imela OWL-S. Obe ontologiji imata skupne koncepte *profil storitve, storitveni model* in *protokol za povezovanje*, ki smo jih spoznali v poglavju 3.2.1 OWL-S.

Kljub temu, da na SWSO lahko gledamo kot razširitev oz. izboljšavo OWL-S, pa SWSO prispeva tudi nekaj bistvenih razlik. Ena izmed najbolj očitnih je večja izrazna moč SWSL. Prav tako lahko bolj podrobno opišemo procesni model z uporabo PSL (Process specification language). PSL je modularna, razširljiva ontologija, ki zajema koncepte potrebne pri opisu poslovnega procesa. Temelji na logiki prvega reda (First-Order Logic) in je sprejeta kot mednarodni standard ISO 18629 [21].

Konceptualni model SWSO lahko formaliziramo na dva med seboj neodvisna načina:

- **FLOWS** (First-order Logic Ontology For Web Services), poznana tudi kot SWSO-FOL, predstavlja ontologijo, ki priskrbi konceptualno ogrodje za opis spletnih storitev. Zapišemo jo s SWSL-FOL. S FLOWS, ki temelji na PSL lahko zelo natančno opišemo procesni model. Zgrajena je modularno. Obsega tudi ontologijo za opis storitev, ki je v določeni meri podobna profilu storitve iz OWL-S, in protokol za povezovanje.

- **ROWS** (Rule Ontology For Web Services), ki je poznana tudi kot SWSO-Rules, zapišemo s SWSL-Rules. Izvira iz FLOWS, vendar je malo manj kompleksna, saj ima jezik SWSL-Rules manjšo izrazno moč kot SWSL-FOL.

Tak pristop je uporabljen zato, ker se različni pristopi pri različnih nalogah, ki so povezane s semantičnimi spletnimi storitvami, bolje obnesejo kot drugi.

3.2.4.1 Primerjava med SWSF in WSMO

SWSF (Semantic Web Services Framework) je ogrodje, ki vsebuje SWSL in SWSO. Kot smo že omenili, je SWSF nadgradnja OWL-S, zato si konceptualni model SWSF deli mnogo podobnosti z OWL-S. Vendar pa je bilo s prehodom iz OWL na SWSL odpravljenih mnogo pomanjkljivosti. Pri SWSF nismo več vezani na omejitve s katerimi se soočimo pri uporabi formalizma opisne logike. SWSF uporablja strogo določene definicije glede pogojev in dinamičnih vidikov semantičnih spletnih storitev z uporabo PSL ontologije, preko katere lahko bolj natančno opišemo storitveni model.

V teh pogledih je SWSF izredno podoben WSMO, saj so zgoraj navedene prednosti poglaviten cilj pri načrtovanju WSMO. Vendar pa je najpomembnejše dejstvo, da zaenkrat še ni prisotnih resnejših orodij, ki bi se ukvarjala z implementacijo SWSF. Literatura [2] navaja, da se tudi v prihodnje ne kažejo kakšni pomembni premiki v tej smeri. Zato obstaja velika nevarnost, da SWSF kljub dobri zasnovi ostane samo na teoretični ravni. V dobršni meri bo k uveljavitvi SWSF prispevalo tudi sprejetje standarda PSL med uporabniki.

3.3 Strnjen pregled ontologij za opis semantičnih spletnih storitev

Ker ontologija predstavlja enega izmed poglavitnih kriterijev, je izbira orodja najprej odvisna od ontologije v kateri so opisane spletne storitve. Tu je na mestu vprašanje, katera ontologija je najprimernejša za opise spletnih storitev? Tabela 1 predstavlja strnjen pregled lastnosti ontologij za opis spletnih storitev [7]. V tabeli 2 so predstavljene prednosti in slabosti določene ontologije za opis semantičnih spletnih storitev.

Predlog	Kratek Opis	Jezik za opis ontologije	Razširjenost jezika za opis ontologije*	Formalizem	Povezava z WSDL
OWL-S 1.x	Vrhnja ontologija, ki temelji na OWL	OWL	75,9%	Opisna logika (DL)	Preko protokola za povezovanje (service grounding)
WSDL-S	Uporablja razširitvene elemente, s katerimi semantično obogatimo opis spletne storitve	Uporabimo lahko OWL, WSMO, XML, UML ali katerikoli drug jezik, ki ga uporabljamo pri modeliranju	75,9%	Uporabnik lahko uporabi različne formalizme, saj nismo omejeni z določeno ontologijo	Elemente, ki vnašajo semantiko vstavljamo direktno v WSDL-dokument
WSMO	Ontologija, ki jo izrazimo s pomočjo WSML	WSML (z uporabo preslikav pa tudi OWL, XML, RDF)	3,7%	Opisna logika (DL), logika prvega reda (First-Order Logic), logično programiranje (Logic Programming)	Preko protokola za povezovanje (service grounding)
SWSO	Ontologija, ki jo izrazimo s pomočjo SWSL	SWSL	/	Logika prvega reda(First-Order Logic), logični jeziki na osnovi pravil (SWSL-Rules)	Preko protokola za povezovanje (service grounding)

* vir o razširjenosti jezika za opis ontologije: [13] Figure 5.

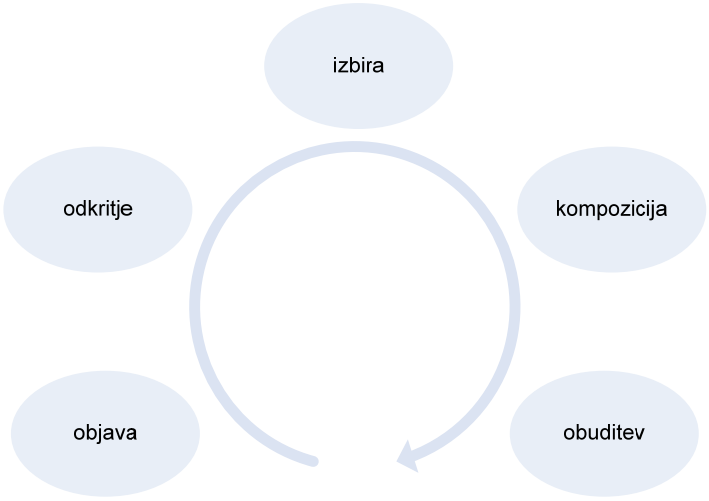
Tabela 1: Pregled ontologij za opis semantičnih spletnih storitev.

	PREDNOSTI	SLABOSTI
WSDL-S / SAWSDL (OWL)	• Preprostost v primerjavi z ostalimi pristopi • Minimalen poseg v že obstoječe opise spletnih storitev • Poznani standardi (zrelost standardov) • Razširjena uporaba OWL • OWL je predlagan kot W3C priporočilo	• V primerjavi z ostalimi pristopi vnese v model (spletno storitev) manj semantike • Manj podrobno iskanje • Ne omogoča opisa procesa
OWL-S 1.x (OWL)	• Razširjena uporaba OWL • OWL je predlagan kot W3C priporočilo • Omogoča opis procesa	• Šibkejša izrazna moč jezika v primerjavi z WSML in SWSL • Razlika med opisom obnašanja spletne storitve navzven (koreografija) in opisom medsebojne interakcije spletnih storitev (orkestracija) ni posebej poudarjena
SWSO (SWSL)	• Nadgradnja OWL-S • Uporaba formalizmov z veliko izrazno močjo • Uporaba standarda PSL • Enovito semantično ogrodje • Omogoča opis procesa	• Zahtevnost • Zaenkrat še ni prisotnih resnejših orodij, ki bi se ukvarjala z implementacijo SWSF-ja, in vse kaže, da bo tako ostalo tudi v prihodnje • Standard PSL se mora trdneje uveljaviti
WSMO (WSML, z uporabo preslikav pa tudi OWL, XML, RDF)	• Uporaba posrednikov, ki rešujejo raznovrstnosti in neujemanje uporabljenih ontologij • Uporaba formalizmov z veliko izrazno močjo • Omogoča opis procesa • Enovito semantično ogrodje • Edini pristop, ki loči med uporabnikovim in ponudnikovim pogledom na storitev • Priporoča uporabo širše sprejetih besednjakov (vocabularies) (npr. Dublin Core Metadata Element Set, FOAF, itd.)	• Zahtevnost • Velik odmik od dosedanjega pristopa gradnje spletnih storitev • Trenutno uporaba WSML še ni tako zelo razširjena

Tabela 2: Pregled prednosti in slabosti določene ontologije za opis semantičnih spletnih storitev.

4 Algoritmi za odkrivanje semantičnih spletnih storitev

Življenjski cikel spletne storitve vsebuje naslednje korake:



Slika 20: Življenjski cikel spletne storitve.

Izmed omenjenih korakov je v praksi najpomembnejši korak odkrivanja ustrezne storitve, oz. je pritegnil največ zanimanja s strani raziskovalnih skupin in industrije [7]. Ogledali si bomo deset algoritmov, ki se ukvarjajo z odkrivanjem semantičnih spletnih storitev. Pristopi se med seboj razlikujejo po elementih nad katerimi se izvaja primerjanje, načinu primerjanja, itd.

Uvrstimo jih v dve kategoriji:

- **Neposredno primerjanje** – zahtevana storitev se primerja z enim samim oglasom ponujene storitve.
- **Posredno primerjanje** – zahtevana storitev se primerja s storitvijo, ki je sicer sestavljena iz več posameznih storitev. Zato pri posrednem ujemanju zelo močno pridejo do izraza razne kompozicijske tehnike.

Preden pričnemo z opisi si oglejmo še notacijo s katero predstavimo določene elemente algoritma.

req … zahteva po storitvi
A … oglasi vseh storitev, ki so na voljo
adv … oglas storitve
$req.O$ … množica izhodov zahtevane storitve
$adv.O$ … množica izhodov ponujene storitve
$req.I$ … množica vhodov zahtevane storitve
$adv.I$ … množica vhodov ponujene storitve
$X.o$ … posamezen izhod iz množice $X.O$, kjer X predstavlja adv ali req
$X.i$ … posamezen vhod iz množice $X.I$, kjer X predstavlja adv ali req
$X.SC$ … opis kategorije storitve X
$X.par$ … poljubni parametri določeni s strani uporabnika

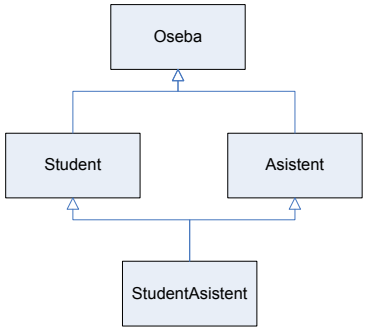
4.1 Ujemanje na podlagi funkcionalnosti

Osnovna ideja tega pristopa je, da primerjamo vhode objavljene storitve z vhodi zahtevane storitve in izhode objavljene storitve z izhodi zahtevane storitve. Vhodni in izhodni koncepti so predstavljeni z ontologijo (npr. v OWL-S so vhodi in izhodi opisani v profilu storitve). Stopnja ujemanja med vhodoma in izhodoma je odvisna od tega, v kakšnih razmerjih koncepti, ki predstavljajo vhode oz. izhode, nastopajo v ontologiji. Paolucci [23] je predlagal uporabo štirih stopenj ujemanja *DoM*. Stopnje so prikazane v tabeli 3 in se nanašajo na primerjavo med izhodi. Pri primerjavi vhodov namesto parametrov $req.o$ in $adv.o$ uporabimo parametra $req.i$ in $adv.i$. Končna stopnja ujemanja med storitvama je enaka najmanjšemu ujemanju vhodov oz. izhodov.

Algoritem prikazuje slika 23. Funkcija *degreeOfMatch* lahko zavzame stanja, ki so prikazana v tabeli 3 in v tabeli 4. Algoritem kot rezultat vrne seznam oglasov spletnih storitev, ki izpolnjujejo zahteve. Uporabnik določi kako natančno se bodo vrnjeni rezultati ujemali z zahtevano spletno storitvijo s tem, ko nastavi minimalno stopnjo ujemanja. Pristop uporablja zelo uveljavljeno primerjanje na podlagi relacije vsebovanosti.

Tabela 5 prikazuje štiri različne oblike relacije vsebovanosti. Relacija vsebovanosti se uporablja za prikaz hierarhične urejenosti razredov. Tipično je najbolj splošen razred (npr. Thing) najvišje v hierarhiji, medtem, ko se bolj specifični razredi (npr. Nissan, Lexus, itd.) nahajajo na nižjih nivojih.

Pomembna lastnost relacije vsebovanosti je, da izpeljani razred podeduje vse elemente svojega nadrazreda. Tem elementom pa lahko doda še svoje elemente in ponovno definira metode iz svojega nadrazreda. Zato lahko izpeljane razrede razumemo kot posebne primere svojih nadrazredov. Tako izpeljani razredi predstavljajo specializacijo, nadrazredi pa generalizacijo. Pri nekaterih ontologijah ima razred lahko največ enega predhodnika, vendar običajno ni tako. V tem primeru govorimo o večkratnem dedovanju. Slika 21 prikazuje primer razreda StudentAsistent, ki predstavlja osebe, ki so hkrati študenti in asistenti[16].



Slika 21: Večkratno dedovanje.

Omeniti moramo tudi to, da je ujemanje izhodov bolj pomembno kot ujemanje vhodov, saj uporabnik storitve ponavadi išče na podlagi tega, kaj mu lahko nudijo.

[16] http://www.najdi.si/redirect/index.jsp?redirect=http%3A//lisa.uni-mb.si/osebje/bonacic/predmeti/programiranje2_vs1/RAZREDI/p_Razredi4_dedovanje.ppt&sourcePage=search&pageIndex=0&linkIndex=3&isNkUrl=false&spCount=0&q=undefined&contenttype=undefined

Stopnja ujemanja DoM	Pogoji ujemanja
Natančno ujemanje (EXACT)	Če je $req.o$ ekvivalentna $adv.o$ ali če je $req.o$ direkten podrazred $adv.o$
Obratna vsebovanost - inkluzija (PLUGIN)	Če $adv.o$ vsebuje $req.o$ ($req.o$ je podrazred $adv.o$)
Vsebovanost (SUBSUMES)	Če $req.o$ vsebuje $adv.o$ ($adv.o$ je podrazred $req.o$)
Ni ujemanja (FAIL)	V primeru, da med $req.o$ in $adv.o$ ne nastopa relacija vsebovanosti

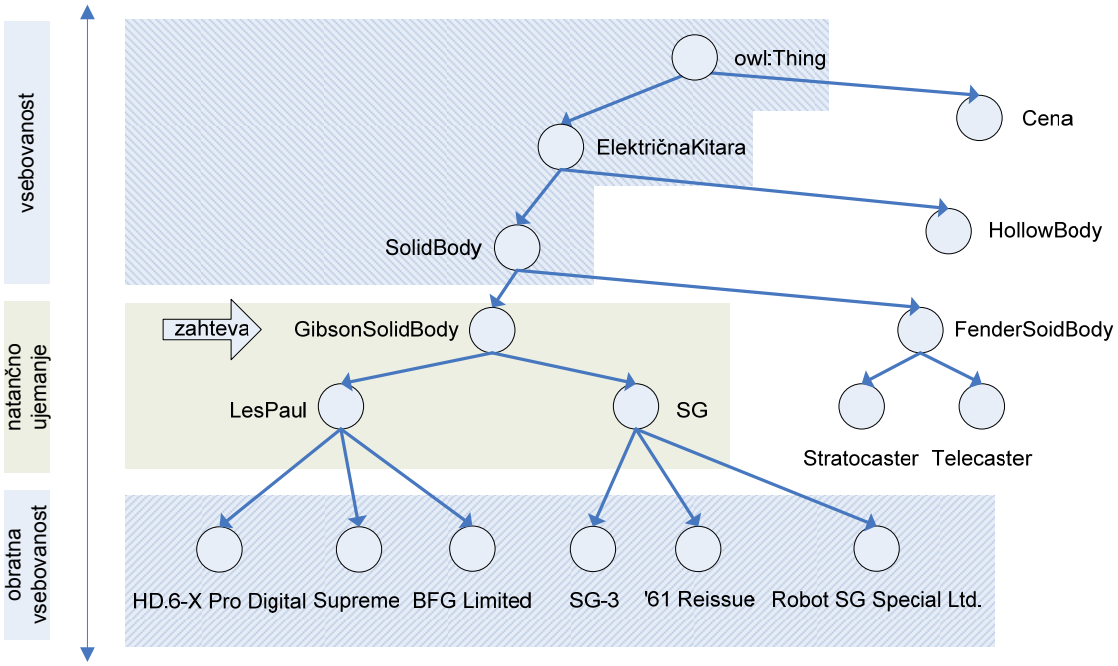
Tabela 3: Stopnje ujemanja pri izhodnih parametrih

Stopnja ujemanja DoM	Pogoji ujemanja
Natančno ujemanje (EXACT)	Če je $req.i$ ekvivalentna $adv.i$ ali če je $adv.i$ direkten podrazred $req.i$
Obratna vsebovanost - Inkluzija (PLUGIN)	Če $req.i$ vsebuje $adv.i$ ($adv.i$ je podrazred $req.i$)
Vsebovanost (SUBSUMES)	Če $adv.i$ vsebuje $req.i$ ($req.i$ je podrazred $adv.i$)
Ni ujemanja (FAIL)	V primeru, da med $req.i$ in $adv.i$ ne nastopa relacija vsebovanosti

Tabela 4: Stopnje ujemanja pri vhodnih parametrih.

Relacija vsebovanosti	Pomen
$req.i$ vsebuje $adv.i$	Obstaja možnost, da ponujena storitev za pravilno izvedbo potrebuje natančnejše vhodne informacije.
$adv.i$ vsebuje $req.i$	Zahtevana storitev nudi vse informacije vhodom ponujeni storitvi, da se ta lahko pravilno izvede.
$req.o$ vsebuje $adv.o$	Ponujena storitev sicer ustreza našim željam, vendar ne bomo zajeli vseh možnih rešitev.
$adv.o$ vsebuje $req.o$	Parameter, ki ga lahko priskrbi ponujena storitev bo mogoče preveč splošen za naše potrebe.

Tabela 5: Možni načini relacije vsebovanosti.



Slika 22: Primer določitve stopenj ujemanja ***vhodnih*** parametrov.

Main algorithm: match(req , A **)**
matchedServices = {}
For all adv in A do
 If ((DoMI = Imatch(req , adv) != FAIL) and (DoMO = Omatch(req , adv)) != FAIL))
  adv .DoM = min (DoMI, DoMO)
  matchedServices = matchedServices{ adv }
 return sortByDoM(matchedServices)

Function: Omatch(req,adv)
DoM = {}
For all $req.o$ in $req.O$ do
 For all $adv.o$ in $adv.O$ do
  DoM = DoM{degreeOfMatch($req.o$, $adv.o$)}
Return min(DoM)

Function: Imatch(req,adv)
DoM = {}
For all $adv.i$ in $adv.I$ do
 For all $req.i$ in $req.I$ do
  DoM = DoM{degreeOfMatch($adv.i$, $req.i$)}
Return min(DoM)

Slika 23: Algoritem – ujemanje na podlagi funkcionalnosti.

4.2 Večstopenjsko ujemanje

Drugi pristop opravi bolj podrobno primerjanje od prvega. Pri pristopu se držimo načela, da najboljšo primerjavo lahko dosežemo, če se ne omejimo samo na parametre, ki smo jih spoznali pri prvem pristopu, ampak na problem gledamo širše. Poleg primerjanja vhodnih in izhodnih parametrov objavljene in zahtevane storitve, algoritem primerja še dodatne parametre (npr. QoS) ter kategorijo storitve. Dodatno se tu srečamo s spajanjem stopenj ujemanja (*DoM aggregation)*. Zaradi bolj podrobnega primerjanja so tudi rezultati boljši. Poleg tega lahko različnim parametrom pripišemo različne uteži, ki določajo njihovo pomembnost. Algoritem je prikazan na sliki 24. Funkcijo *aggregate()* lahko realiziramo z obteženo vsoto vseh štirih DoM. Glavni problem tega pristopa je, da je težko najti optimalno funkcijo *aggregate()*. Funkcije Xmatch() pa so podobne funkcijam, ki jih uporablja algoritem opisan v poglavju 4.1 Ujemanje na podlagi funkcionalnosti.

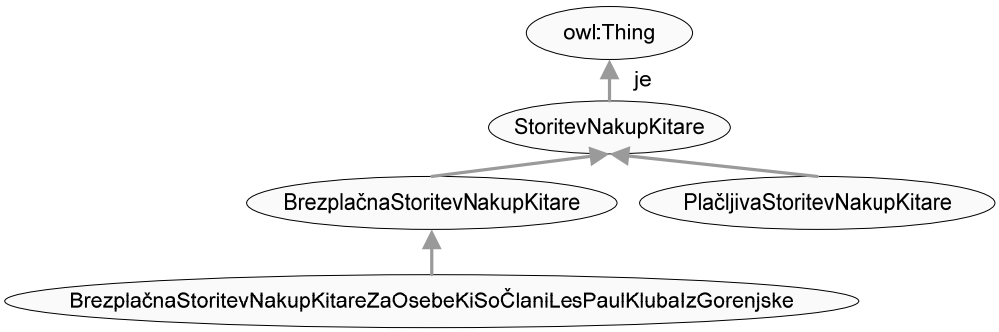
Main algorithm: match(req **,** adv)
DoMI = allImatch($req.I$, $adv.I$)
DoMO = allOmatch($req.O$, $adv.O$)
DoMSC = SCmatch($req.SC$, $adv.SC$)
DoMP = PARmatch($req.par$, $adv.par$)
If (DoMI == FAIL or DoMO == FAIL or DoMSC == FAIL or DoMP = FAIL)
  Return FAIL
Else
  Return (DoM = agregate (DoMI, DoMO, DoMSC, DoMP))

Slika 24: Algoritem – večstopenjsko ujemanje.

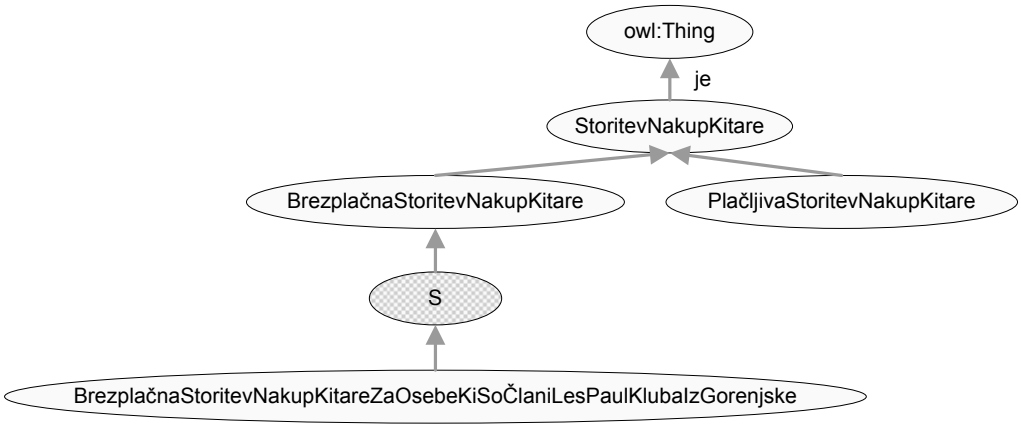
4.3 Ujemanje na podlagi profila storitve

Zgoraj opisana pristopa izvajata primerjanje na podlagi relacije vsebovanosti. Tretji pristop predlaga drugačen način primerjanja spletnih storitev. Podrobneje je opisan v knjigah [24] in [22]. Pristop temelji na uporabi opisne logike. Algoritem primerja opise profilov storitev, ki so opisani z ontologijo OWL-S. Vsak profil storitve je opisan v OWL. Dobra stran opisne logike je v tem, da so opisi lahko zelo podrobni, saj ima opisna logika zelo veliko izrazno moč.

Ontologija opisuje vse parametre profila storitve (vhod, izhod, itd.). Kot smo že omenili je vsaka storitev predstavljena z opisno logiko (slika 27). Če v pogled vzamemo domeno »Nakup kitare«, bi lahko spletne storitve, ki ji pripadajo, ponazorili tako kot jih prikazujeta sliki 25 in 26.



Slika 25: Ontologija s katero ponazorimo storitve, ki se ukvarjajo z nakupi kitar.



Slika 26: Umestitev storitve S.

Tako ontologijo si lahko predstavljamo kot neke vrste register, ki hrani oglase priskrbljene s strani ponudnikov storitve. Ko enkrat določimo ontologijo, ki vsebuje oglase, ki so jih priskrbeli ponudniki, lahko z opisno logiko izrazimo in v ontologijo umestimo zahtevo po spletni storitvi. Mehanizem sklepanja nato v ontologijo klasificira koncepte, ki jih zahtevana spletna storitev vsebuje.

Slika 27 prikazuje dva oglasa storitev opisanih z opisno logiko. *BrezplačnaStoritevNakupKitare* je opisana s profilom storitve, ki je skladen ontologiji za opis storitve, medtem ko ostali koncepti pripadajo problemski domeni.

BrezplačnaStoritevNakupKitareZaOsebeKiSoČlaniLesPaulKlubaIzGorenjske je bolj specifična od storitve *BrezplačnaStoritevNakupKitare*. Zahtevana storitev *S* je brezplačna, zajema osebe, ki živijo v Sloveniji ter so člani kluba LesPaul. Ker je storitev *S* bolj splošna kot storitev *BrezplačnaStoritevNakupKitareZaOsebeKiSoČlaniLesPaulKlubaIzGorenjske* in bolj specifična kot storitev *BrezplačnaStoritevNakupKitare* jo bo ustrezen mehanizem sklepanja (npr. Racer) na podlagi opisov prikazanih na sliki 27 umesti tako kot kaže slika 26.

ponujeni storitvi:

BrezplačnaStoritevNakupKitare ≡ StoritevNakupKitare ⊓ ∃ imaProfilStoritve.(ProfilStoritve ⊓ ∃ načinPlačila.Brezplačno)

BrezplačnaStoritevNakupKitareZaOsebeKiSoČlaniLesPaulKlubaIzGorenjske ≡ StoritevNakupKitare ⊓ ∃ imaProfilStoritve.(ProfilStoritve ⊓ ∃ načinPlačila.Brezplačno ⊓ ∃ imaOsebniProfil.(KontaktniPodatki ⊓ ∃ naLokaciji.Gorenjska) ⊓ ∃ imaČlanskePodatke.(ČlanskiPodatki ⊓ ∃ imaČlanskoIzkaznico.IzkaznicaLesPaul))

zahtevana storitev:

S ≡ StoritevNakupKitare ⊓ ∃ imaProfilStoritve.(ProfilStoritve ⊓ ∃ načinPlačila.Brezplačno ⊓ ∃ imaOsebniProfil.(KontaktniPodatki ⊓ ∃ naLokaciji.Slovenija) ⊓ ∃ imaČlanskePodatke.(ČlanskiPodatki ⊓ ∃ imaČlanskoIzkaznico.IzkaznicaLesPaul))

Slika 27: Opis storitve z uporabo opisne logike.

Če uporabimo ustrezno prirejen algoritem (primerjava bi se izvajala nad profili storitev in natančnemu ujemanju bi ustrezali samo ekvivalentni koncepti), ki smo ga spoznali v poglavju 4.1 vidimo, da natančno ujemanje ni mogoče. Še najbolje bi ustrezala storitev *BrezplačnaStoritevNakupKitare* (obratna vsebovanost). Opazimo tudi, da med storitvama *BrezplačnaStoritevNakupKitareZaOsebeKiSoČlaniLesPaulKlubaIzGorenjske* in *S* obstaja relacija vsebovanosti, ki je po definiciji šibkejša kot relacija obratne vsebovanosti.

4.4 Ujemanje na podlagi podobnosti iz pridobljenih informacij o spletnih storitvah

Vsi prejšnji pristopi imajo pomanjkljivost, da se ukvarjajo samo z relacijami vsebovanosti oz. logičnim sklepanjem. Pri določanju podobnosti med spletnimi storitvami pa v določenih primerih to ni dovolj. Predstavljajmo si, da uporabnik išče storitev, ki mu na podlagi priskrbljenih podatkov vrne kontaktne podatke osebe, ki bi jo lahko vključil v svoj orkester. Natančneje uporabnika zanimajo osebe, ki so prav tako kot on veliki ljubitelji igranja na katerikoli električni glasbeni instrument. Torej iščemo storitev, ki kot vhod sprejme koncept *{Hobiji* ⊓ ∃ *igraNaInstrument.ElektrofoniGlasbeniInštrument}*, ki je bolj specifičen kot koncept *Hobiji* in kot izhod vrne koncept *KontaktniPodatki*. Iz strani ponudnika sta na voljo dve storitvi, ki omogočata iskanje oseb.

Storitev1: VrniSpletnePodatkeOsebe

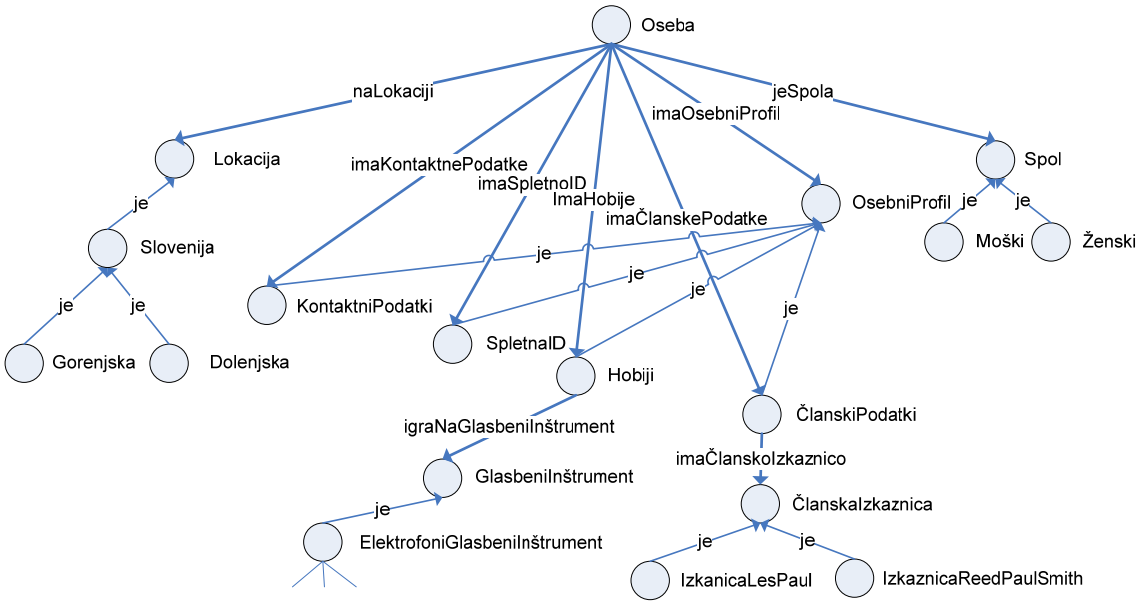
vhod={Oseba ⊓ ∃ jeSpola.Moški}

izhod={Hobiji, SpletnaID}

Storitev2: VrniOsebeGledeNaHobije

vhod={Hobiji}

izhod={KontaktniPodatki}



Slika 28: Prvi izsek ontologije s katero obogatimo opis spletne storitve.

V primeru, da uporabimo pristop, ki smo ga opisal v poglavju 4.1 se Storitev1 ne bo ujemala z željami uporabnika. To se zgodi zaradi tega, ker so koncepti *KontaktniPodatki, SpletnaID* in *Hobiji* med seboj terminološko nepovezani (oz. se izključujejo). Pri drugi storitvi pride do ujemanja (vhod: *obratna vsebovanost*, izhod: *natančno ujemanje*), saj je koncept *Hobiji* bolj splošen kot vhod, ki ga namerava podati uporabnik. Pristop, opisan v poglavju 4.1, bo uporabniku predlagal Storitev2 , medtem ko bo Storitev1 označena kot neustrezna.

Pomanjkljivost tega pristopa je v tem, ker Algoritem1 ne upošteva dejstva, da lahko uporabnik na podlagi *SpletnaID* kontaktira osebo in od nje naknadno pridobi vse ostale podatke, ki jih želi izvedeti. Način za premostitev teh težav je,

da pri odkrivanju storitve poleg eksplicitne semantike, ki jo predstavlja ontologija, s katero smo opisali problemsko domeno, uporabljamo tudi implicitno semantiko. To lahko dosežemo s tem, da uporabimo različne tehnike iz sorodnih področij, s katerimi se ukvarja informacijska tehnologija. Ena izmed takih tehnik je informacijsko poizvedovanje (information retrieval - IR). Temeljna ideja je v tem, da uporabimo tehniko informacijskega poizvedovanja, ko pristop na podlagi logičnega sklepanja odpove.

Ena izmed bolj priljubljenih in učinkovitih načinov informacijskega poizvedovanja je klasifikacijska metoda TFIDF. Kratica TFIDF pomeni "Term Frequency Inverse Document Frequency". Že ime nam pove, da delamo s številom pojavljanja besede v dokumentih (term frequency), medtem ko nam inverzna frekvenca dokumenta (inverse document frequency) pove, v koliko dokumentih se pojavi določena beseda. Bralec si metodo lahko podrobneje ogleda v [25] , [26].

Sedaj se stopnja ujemanja DoM izračunava malce drugače. Stopnja FAIL, ki smo jo spoznali že v prejšnjih algoritmih zamenjamo z atributom LOGICAL-FAIL. Dodamo novo stopnjo SIM_{IR}, ki pa ni tako pomembna kot LOGICAL FAIL. Stopnja SIM_{IR} zavzame vrednost *true*, če med vhodom in izhodom objavljene ter iskane storitve obstaja določena stopnja podobnosti. Algoritem vrne vrednost FAIL v primeru, ko si zahtevana oz. ponujena storitev nista dovolj podobni. Stopnji LOGICAL-FAIL in SIM_{IR} v takem primeru vrneta vrednosti *false*.

Algoritem prikazuje Slika 29. Funkciji *Imatch()* in *Omatch()* sta podobni funkcijam, ki so opisane v prejšnjih pristopih. Numerična spremenljivka *treshold* označuje najmanjšo še sprejemljivo mero podobnosti med zahtevano in ponujeno storitvijo.

```
Main algorithm: match( req , adv )
DoM = logicalMatch( req , adv )
If DoM == LOGICAL-FAIL
  If sim( req , adv ) threshold
    Return FAIL
  Else
    Return SIM_IR
Else
  Return DoM

Function: logicalMatch( req , adv )
  Return min( Omatch( req , adv ), Imatch( req , adv ) )

Function: sim( req , adv )
  Return similarityMeasureValue
```

Slika 29: Algoritem – ujemanje na podlagi podobnosti iz pridobljenih informacij o spletnih storitvah.

4.5 Pristop na podlagi semantičnega ujemanja grafov

Peti pristop temelji na podlagi semantičnega ujemanja grafov. Opis storitve je predstavljen kot usmerjeni graf. Graf vsebuje vozlišča, ki predstavljajo primerke določenega koncepta. Povezave v grafu predstavljajo lastnosti katere povezujejo vozlišča. Koren vsakega grafa predstavlja bodisi oglas storitve, bodisi zahtevo po storitvi. Ostala vozlišča pa se nanašajo na koncepte, kateri so lahko predstavljeni z različnimi ontologijami, ki opisujejo podano problemsko domeno. Ti koncepti opisujejo funkcionalnosti storitve (zmožnosti, omejitve ,…).

Prednost pristopa je v tem, da se primerjajo strukture danih grafov. Za razliko od prejšnjih pristopov se primerjanje izvaja nad vsemi lastnostmi konceptov in ne samo nad določenimi (npr. VIPU-elementi). Tako lahko bolj natančno določimo, ali se zahtevana in ponujena storitev ujemata. Vendar pa ima ta pristop pomanjkljivost, saj opisana različica ne podpira razvrščanja storitev glede na stopnjo ujemanja DoM. Zahtevana in ponujena storitev se lahko ali ujemata ali pa se ne ujemata. Algoritem prikazuje slika 30.

```
Main algorithm
N1 = root(graph1)
N2 = root(graph2)
Result = match(n1,n2)

Function match(n1,n2)
If (n1 susumes n2 or n2 subsumes n1)
  matching = TRUE
  For each property p1 or n1 do
    For each property p2 or n2 do
      If NOT [(p1 equvivalent-to p2 or p1 subproperty-of p2 or p2 subproperty-of p1) and
match(target(p1),target(p2))]
        matching = FALSE
  Return matching
Else
  Return FALSE
```

Slika 30: Algoritem – Pristop na podlagi semantičnega ujemanja grafov.

Orkestracija poslovnih procesov je ena izmed najpomembnejših aktivnosti pri realizaciji storitvenih arhitektur (SOA). Rešuje problematiko kompozicije spletnih storitev, ki se povezujejo v vedno večje in kompleksnejše dinamične poslovne procese.

V tej diplomski nalogi se s kompozicijo semantičnih spletnih storitev ne bomo podrobneje ukvarjali, vendar pa si bomo kljub temu ogledali dva možna pristopa, ki se ukvarjata z avtomatsko gradnjo spletnih storitev. S tem dosežemo želeno funkcionalnost, ki jo sicer samostojna spletna storitev ne omogoča.

4.6 Pristop na podlagi posrednega ujemanja grafov

Posredno ujemanje grafov uporabimo v primeru, ko nobena od ponujenih spletnih storitev ne ustreza našim željam, vendar pa bi v kombinaciji z ostalimi storitvami lahko dobili zahtevane izhode oz. vhode (zahtevano funkcionalnost). Pri preprosti kompoziciji si povezave vseh storitev, ki jih nudi ponudnik, lahko predstavljamo kot usmerjeni graf. Verige storitev, kot jih poimenujejo v [7], pa predstavljajo veljavne poti v grafu. Verige storitev naj vedno tvorijo aciklično pot. Obravnavan pristop je podrobneje opisan v knjigi [30]. Uporablja tehnike, ki izrabljajo prisotnost semantike v podatkih (primerjanje vhodno/izhodnih parametrov na podlagi relacije vsebovanosti) kot tehnike, ki se ukvarjajo z preiskovanjem grafov.

Glavna ideja pri kompoziciji preprostih storitev vsebuje dva temeljna koraka:

- Vhodi vsake storitve, ki sodeluje pri kompoziciji, se primerjajo z vhodi, podanimi bodisi s strani uporabnika, ki zahteva določeno funkcionalnost, bodisi z izhodi predhodne storitve, ki je že vključena v verigo.
- Izhodi želene storitve se primerjajo z izhodi zadnje storitve, ki nastopa v verigi.

Na začetku zgradimo graf storitev tako kot predlagata zgornja dva koraka. Elementi, ki tvorijo graf so definirani tako:

n_i - vozlišče, ki ustreza opisu storitve d_i
(n_i, n_j) - usmerjena povezava z začetnim vozliščem n_i in končnim vozliščem n_j. Povezava med storitvama je veljavna samo takrat, kadar se vsi vhodi storitve n_j ujemajo z:

- izhodi storitve n_i
- vhodi, ki jih namerava priskrbeti uporabnik storitve

V skladu z zgoraj predstavljeno definicijo predstavljajo vozlišča storitve, povezave med njimi pa tok podatkov.

Oglejmo primer izdelave grafa storitev. Zaradi večje preglednosti bomo vhodne in izhodne parametre predstavili kot črke abecede. Tabela 6 predstavlja storitve, ki so na voljo s strani ponudnikov. Storitev, ki jo želimo, pa naj bi kot vhodne podatke zahtevala A, B, C, D in kot izhodne vrnila Z in Y. Vozlišči povežemo, če se vhodi storitve ujemajo z izhodi druge. Ujemanje lahko določimo na podlagi katerega koli od algoritmov, ki primerja vhode in izhode dveh storitev. Ko je graf zgrajen moramo določiti, po katerih poteh bomo dosegli želene funkcionalnosti. V našem primeru bosta začetni vozlišči predstavljali storitvi S1 in S2. Izhodno množico tvorijo izhodni parametri storitev, ki pripadajo izbrani poti. Množica je veljavna samo takrat, ko velja trditev: *zahtevani izhodni parametri* $\subseteq$ *izhodni parametri storitev izbrane poti*. Dobljen graf prikazuje slika 31.

Zgrajene poti, ki ustrezajo naši zahtevi so naslednje [7]:
Veriga storitev 1: S1, S3, S4, S6, S7
Veriga storitev 2: S1, S3, S4, S5
Veriga storitev 3: S2, S4, S6, S7
Veriga storitev 4: S2, S4, S5

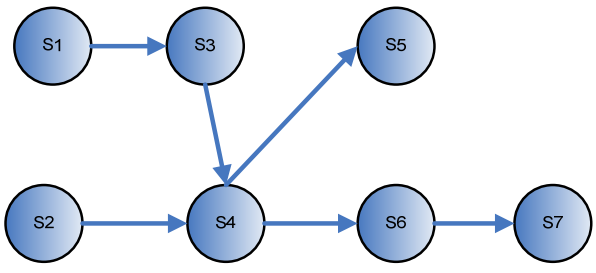
V proces odkrivanja lahko uvedemo dodatne omejitve. Ena izmed takih omejitev bi lahko bila izbira najkrajše poti v grafu. Taki zahtevi v našem primeru ustreza veriga storitev 4. Drugi način razširitve obstoječega algoritma bi lahko bil tudi ta, da obtežimo povezave med dvema zaporednima storitvama. Uteži določimo na podlagi stopnje ujemanja DoM. Lahko pa rahlo spremenimo pravila, po katerih gradimo graf. Tako dobimo novo zahtevo:

(n_i, n_j) - usmerjena povezava, z začetnim vozliščem n_i in končnim vozliščem n_j. Povezava med storitvama je ustvarjena, ko imata vhod storitve n_j in izhod storitve n_i vsaj en skupni parameter.

Taka definicija vodi do bolj zahtevnih grafov in s tem tudi algoritmov, ki se uporabljajo pri kompoziciji storitev.

Storitev	Vhodi	Izhodi
S1	A,B	E
S2	A,B,C	F,N
S3	E,C	F
S4	F	K,M
S5	K,D	Z,Y
S6	K	D,Z
S7	D	Y

Tabela 6: Vhodni in izhodni parametri storitev



Slika 31: Graf, ki ga dobimo z združitvijo potrebnih storitev.

4.7 Uporaba tehnike sklepanja: veriženje nazaj

Pristop temelji na podlagi logičnega sklepanja pri katerem uporabimo tehniko veriženja nazaj. Tehnika deluje tako, da se pomikamo nazaj od množice možnih sklepov oziroma ciljev in skušamo najti dokaze (dejstva v bazi znanja), s katerimi bi podprli in preverili njihovo pravilnost. To je deduktivno sklepanje. Podobno deluje jezik Prolog. Glavna ideja tega pristopa je v tem, da začnemo kompozicijo s storitvijo, katere izhodi se ujemajo z izhodi, ki jih zahteva uporabnik. Nato rekurzivno poizkušamo povezati storitev z ostalimi, dokler ne najdemo storitve, katere vhodi se ujemajo z vhodi (oz. dopolnijo množico vhodov, ki jih nudijo storitve v verigi), ki jih zahteva uporabnik storitve.

Poglejmo si uporabo tehnike veriženje nazaj. Storitve so predstavljene v tabeli 6. (S … storitev, C … cilj, VS … veriga storitev):

Korak 1: S = S5, C={ K }, VS = {S5}
Korak 2: S' = S4, C={ F }, VS = {S5, S4}
Korak 3: S' = S3, C={ E }, VS = {S5, S4, S3}
Korak 4: S' = S1, C={ }, VS = {S5, S4, S3, S1}
Korak 5: Izhod
Končni rezultat VS = {S1, S3, S4, S5}

Prikazani postopek lepo prikazuje osnovno idejo. Vendar pa vedno vrne samo eno izmed možnih verig storitev, ki pa ni nujno najboljša. Z ustreznim popravkom algoritem lahko vrne vse možne verige storitev. V našem primeru štiri.

```
Main algorithm
Find a service S whose outputs of the service request R
G= {all inputs of S except those matched by the inputs of R}
List SC = NULL
Insert S to SC
Repeat infinitely
If G == {}
  Return SC
Else
  Find a service S' with its outpust matching all elements of G
  G = {all inputs of S' except those matched by the inputs of R}
Insert S' to SC
```

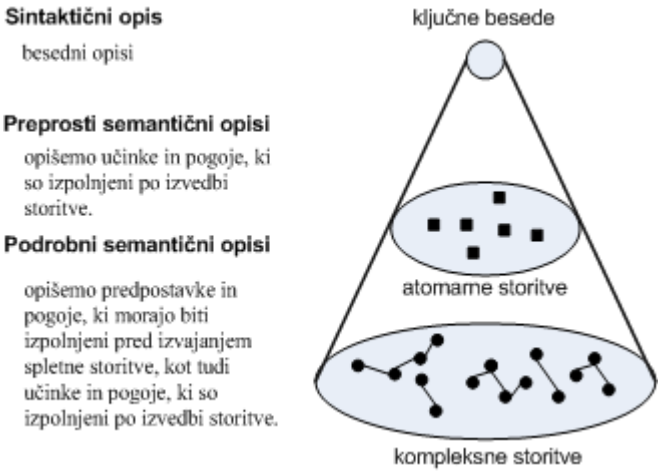
Slika 32: Algoritem – Uporaba tehnike sklepanja: veriženje nazaj.

Pomanjkljivost obeh pristopov VI, VII je ta, da osnovna različica ne podpira razvrščanja storitev glede na stopnjo ujemanja DoM. Tudi to pomanjkljivost lahko odpravimo z raznimi popravki algoritmov.

4.8 WSMO Discovery

Pri tem pristopu se primerjajo opisi spletnih storitev, ki jih priskrbi ponudnik, in cilji, ki jih želi doseči uporabnik spletne storitve. Rezultati, ki jih dobimo med primerjavo, so v veliki meri odvisni od podrobnosti opisa spletne storitve. Zato pri WSMO ločimo tri stopnje abstrakcije:

- Sintaktična raven (sintaktični opisi)
- Preprosta/lahka semantična raven (preprost semantični opis)
- Obogatena semantična raven (podroben semantični opis)



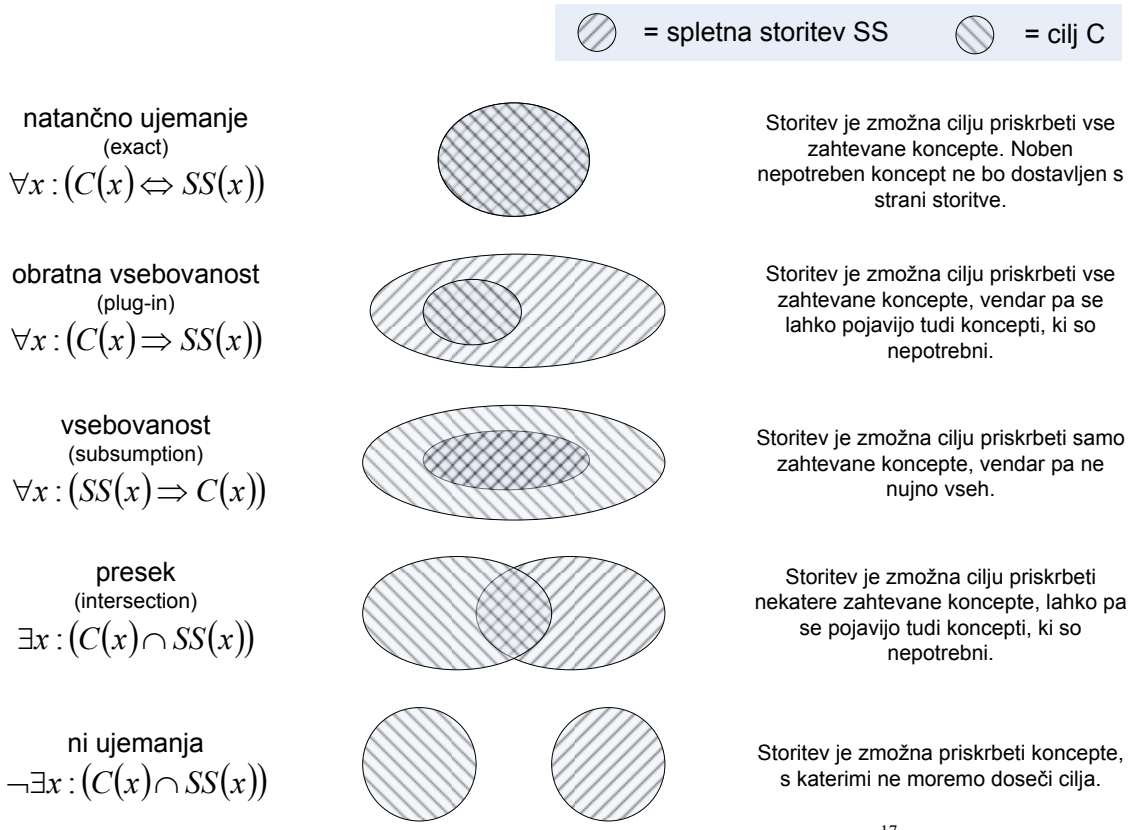
Slika 33: Različne stopnje abstrakcije opisa spletne storitve.

Odkrivanje na podlagi sintaktičnih opisov je najosnovnejše izmed omenjenih treh načinov. Na podlagi besednega opisa lahko v kratkem času procesiramo veliko število storitev, vendar pa je iskanje veliko bolj površno. Iskanje poteka tako, da iskalnemu pogonu priskrbimo množico besed, ta pa jih nato primerja z besedami, ki opisujejo spletno storitev. Dodatno lahko izboljšamo rezultate iskanja z uporabo leksikalnih podatkovnih zbirk npr. *WordNet, HowNet, FrameNet* [29].

Odkrivanje na podlagi preprostih semantičnih opisov je boljše, saj za razliko od prejšnjega pristopa za opis spletne storitve uporabi ontologijo. Ontologija v spletno storitev vnese točno določeno terminologijo, ki si jo uporabniki lahko delijo, prav tako pa pojasnjuje razmerja med koncepti, ki so prisotni v problemski domeni. Na tej stopnji abstrakcije so opisi spletnih storitev predstavljeni kot množica elementov (oz. konceptov z vidika ontologije). Opis funkcionalnosti na tem nivoju ne vsebuje nobenih dinamičnih faktorjev, kot so trenutno stanje in konkretni vhodi, ki jih mora uporabnik storitve priskrbeti. Odkrivanje na podlagi preprostih semantičnih opisov ne upošteva predpostavk in pogojev, ki morajo biti izpolnjeni pred izvajanjem spletne storitve, ampak samo učinke in pogoje, ki so izpolnjeni po izvedbi storitve [9].

Kljub temu, da s preprostimi semantičnimi opisi vnesemo v spletno storitev določeno mero semantike, pa se pri opisih spletnih storitev lahko spustimo v še večje podrobnosti. Tako lahko množico vrnjenih rezultatov dodatno prečistimo. Tak pristop imenujemo pristop na podlagi podrobnih semantičnih opisov. Slaba stran tega pristopa je v tem, da je opis spletne storitve veliko bolj zapleten. Vendar se moramo zavedati, da je tako podroben opis nujen, če hočemo doseči visoko stopnjo avtomatizacije pri uporabi spletnih storitev. Temeljit opis entitet (kot so funkcionalnosti in cilji), ki so vključene v proces odkrivanja, lahko dosežemo z izpopolnitvijo konceptualne stopnje. Tak pristop zahteva bogat modelirni jezik, ki omogoča opise spremenljivk, konstant, simbolov itd. Ta pogoj lahko dosežemo z uporabo logike prvega reda.
Na tej stopnji opisujemo kako vhodni parametri, ki jih priskrbi uporabnik, dejansko vplivajo na izhode in učinke, ko kličemo storitev.

Pri primerjanju ločimo pet stopenj ujemanja:



Slika 34: Grafična predstavitev stopenj ujemanja pri WSMO[17].

Primeri, ki prikazujejo različne stopnje ujemanja[18]:

CiljPotovanjeAU2NEM ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.AvstrijskoMesto ⊓ ∃ konec.NemškoMesto)
CiljPotovanjeAU ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek. AvstrijskoMesto ⊓ ∃ konec. AvstrijskoMesto)
CiljPotovanjeEU ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.EvropskoMesto ⊓ ∃ konec.EvropskoMesto)

CiljPotovanjeIBK2FRA ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.{Innsbruck} ⊓ ∃ konec.{Frankfurt})
CiljPotovanjeIBK2SZG ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.{Innsbruck} ⊓ ∃ konec.{Salzburg})

StoritevVTA ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.(NemškoMesto ∨ AvstrijskoMesto) ⊓ ∃ konec.(NemškoMesto ∨ AvstrijskoMesto))
StoritevDB ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.NemškoMesto ⊓ ∃ konec.NemškoMesto)
StoritevOEBB ≡ ∃ hasPostCondition.(Potovanje ⊓ ∃ začetek.AvstrijskoMesto ⊓ ∃ konec.AvstrijskoMesto)
StoritevDBNočno ≡ ∃ hasPostCondition.(NočnoPotovanje ⊓ ∃ začetek.{Innsbruck ∨ Salzburg} ⊓ ∃ konec. NemškoMesto)

Natančno ujemanje: **CiljPotovanjeAUT ≡ StoritevOEBB**

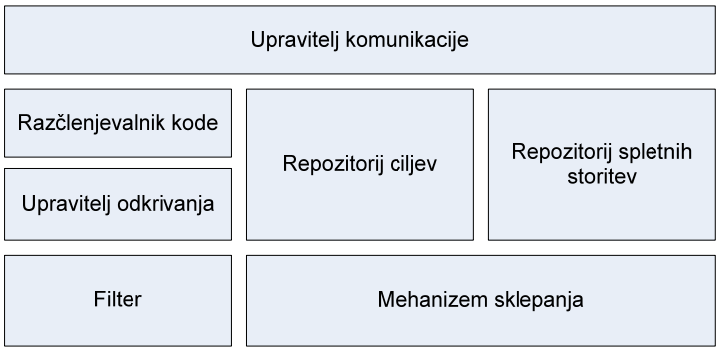
[17] http://www.wsmo.org/2004/d5/d5.1/v0.1/20040913/d5.1v0.1_20040913.pdf
[18] http://www.wsmo.org/TR/d10/v0.2/d10.pdf

Obratna vsebovanost: **CiljPotovanjeAU2NEM** $\subseteq$ **StoritevVTA**
CiljPotovanjeIBK2FRA $\subseteq$ **StoritevVTA**
CiljPotovanjeAU $\subseteq$ **StoritevVTA**
CiljPotovanjeIBK2SZG $\subseteq$ **StoritevVTA**
CiljPotovanjeIBK2SZG $\subseteq$ **StoritevOEBB**

Vsebovanost: **StoritevVTA** $\subseteq$ **CiljPotovanjeEU**
StoritevDB $\subseteq$ **CiljPotovanjeEU**
StoritevOEBB $\subseteq$ **CiljPotovanjeEU**
StoritevDBNočno $\subseteq$ **CiljPotovanjeEU**

Presek: $\neg$ (**CiljPotovanjeIBK2FRA** $\sqcap$ **StoritevDBNočno** $\subseteq \bot$ [19])

Glavni del komponente, ki se ukvarja z odkrivanjem, je *Upravitelj odkrivanja*, ki nadzira proces odkrivanja semantičnih spletnih storitev. *Upravitelj komunikacije* pošlje zahtevo *Upravitelju odkrivanja*, ta pa iz *Repozitorija ciljev* dvigne zahtevani cilj. *Upravitelj odkrivanja* predloži zahtevo *Razčlenjevalniku kode (angl. parser)*. *Upravitelj odkrivanja* nato pošlje koncepte, ki jih je vrnil *Razčlenjevalnik kode, Filtru*, ki vrne seznam vseh spletnih storitev katerih koncepti se ujemajo s koncepti cilja. Postopek se izvede izključno na podlagi besednega primerjanja. Ko *Filter* vrne ustrezne storitve, *Upravitelj odkrivanja* obudi *Mehanizem sklepanja* za izvedbo odkrivanja na podlagi preprostih semantičnih opisov. Pri tem koraku je zelo pomemben odzivni čas, saj lahko *Filter* vrne izredno veliko storitev. Ko enkrat *Mehanizem sklepanja* določi spletne storitve, ki se ujemajo z zahtevanim ciljem, jih vrne *Upravitelju odkrivanja*. *Upravitelj odkrivanja* sedaj vsebuje množico ustreznih storitev, ki jih lahko posreduje *Upravitelju komunikacije.*



Slika 35: Komponenta preko katere odkrivamo spletne storitve v WSMX.

4.9 Algoritem SM-T (Semantic Matching Web Services using Tversky's model)

Algoritem izračunava stopnjo ujemanja med dvema vhodnima in dvema izhodnima konceptoma.
Prav tako primerja tudi funkcionalnost, ki jo želi zahtevana storitev in funkcionalnost, ki jo nudi ponujena storitev. Tako funkcionalnost, kot tudi vhodni in izhodni koncepti, so predstavljeni z ustrezno ontologijo.

Model je osnovan na ideji, da skupne lastnosti dveh konceptov povečujejo mero podobnosti, medtem ko različne lastnosti nižajo mero podobnosti. Podobnost koncepta c_1 s konceptom c_2 je določena s funkcijo $S^{=}$. Funkcija

[19] disjunkcija

upošteva lastnosti, ki so skupne konceptoma c_1 in c_2, potem lastnosti ki jih ima c_1 in jih c_2 nima in tiste lastnosti, ki jih ima c_2 in jih c_1 nima.

V modelu, ki ga je predlagal Tversky, so predstavljene funkcije, nad katerimi izvajamo primerjanje $S_i^{=}(c_R, c_A)$, $S_o^{=}(c_R, c_A)$ in $S_f^{=}(c_R, c_A)$. Funkcije predstavljajo število lastnosti, ki so skupne konceptoma c_R in c_A (c_R prestavlja koncept, ki ga vsebuje zahteva po storitvi, c_A pa predstavlja koncept, ki ga vsebuje ponujena storitev. S črko i je predstavljen vhod, s črko o pa izhod spletne storitve. Funkcija $p(c)$ vrne vse lastnosti, ki so povezane z konceptom c, funkcija $|M|$ pa ustreza številu elementov v množici M.

Simbol '=' nakazuje, da sta koncepta enaka $(c_R = c_A)$. Simbola '>' in '<' med konceptoma nakazujeta generalizacijo oz. specializacijo. Npr. za primer $c_R > c_A$ velja, da je koncept c_R bolj specifičen kot c_A, medtem ko v primeru $c_R < c_A$ velja, da je c_R bolj splošen od c_A.

Izračuni mere podobnosti:

$$S_i^{=}(c_R, c_A) = \begin{cases} 1, & c_R = c_A \\ 1, & c_R > c_A \\ \dfrac{|p(c_R)|}{|p(c_A)|}, & c_R < c_A \\ \dfrac{|p(c_R) \cap p(c_A)|}{|p(c_A)|}, & c_R \neq c_A \end{cases} \qquad (0)$$

$$S_o^{=}(c_R, c_A) = S_f^{=}(c_R, c_A) = \begin{cases} 1, & c_R = c_A \\ \dfrac{|p(c_A)|}{|p(c_R)|}, & c_R > c_A \\ 1, & c_R < c_A \\ \dfrac{|p(c_R) \cap p(c_A)|}{|p(c_R)|}, & c_R \neq c_A \end{cases} \qquad (1)$$

Slika 36: Enačbe, s katerimi izračunamo mero podobnosti.

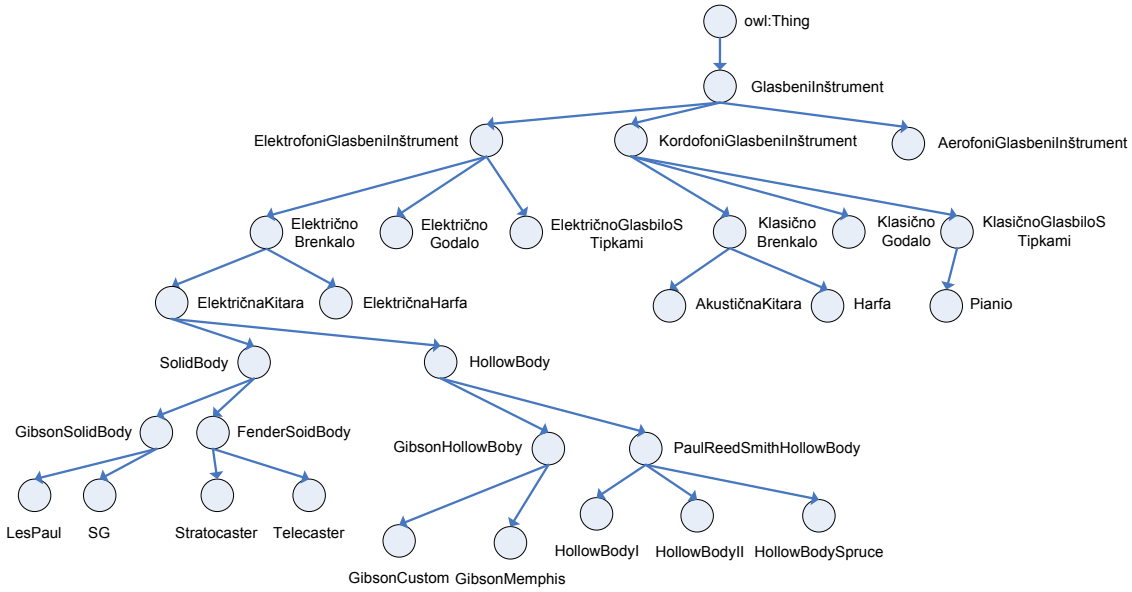
Zaradi velike podobnosti med funkcijami bomo opisali samo $S_i^{=}(c_R, c_A)$.

V prvem primeru, ko sta koncepta enaka $(c_R = c_A)$, je podobnost med njima maksimalna, zato je stopnja ujemanja enaka 1.

V drugem primeru koncept c_R predstavlja specializacijo koncepta c_A $(c_R > c_A)$. Stopnja ujemanja je prav tako 1, ker je koncept c_R podrazred koncepta c_A in ima zato vsaj takšne lastnosti, kot jih ima koncept c_A (lahko pa ima še kakšne dodatno določene lastnosti).

V tretjem primeru pa koncept c_R predstavlja generalizacijo koncepta c_A $(c_R < c_A)$. Iz tega sledi, da koncept c_A vsebuje nekatere lastnosti, ki jih c_R nima.

V četrtem primeru med konceptoma c_R in c_A ne obstaja relacija vsebovanosti. V takem primeru se ocena ujemanja izračuna na podlagi lastnosti, ki so skupne oz. različne obema konceptoma.



Slika 37: Drugi izsek ontologije s katero obogatimo opis spletne storitve.

Zatheva c_R	Oglas c_A	$S_i^{=}(c_R, c_A)$
ElektričnoBrenkalo	ElektričnoBrenkalo	1
ElektričnoBrenkalo	ElektrofoniGlasbeniInštrument	1
ElektričnoBrenkalo	ElektričnaKitara	0,75
ElektričnoBrenkalo	Piano	0,54

Tabela 7: Izračuni mere podobnosti, ki se navezujejo na sliko 37.

Oglejmo si kako smo prišli do rezultata v tretji vrstici tretjega stolpca in četrti vrstici tretjega stolpca . Za izračune uporabimo enačbe prikazane na sliki 36.

Lastnosti konceptov:
ElektrofonGlasbeniInštrument = {Elektronika, UporabniškiVmesnik, Naziv, Cena, Proizvajalec, ŠteviloOktav}

ElektričnoBrenkalo = {Elektronika, UporabniškiVmesnik, Naziv, Cena, Proizvajalec, ŠteviloOktav, ŠteviloStrun, TipStrun}

ElektričnaKitara = {Elektronika, UporabniškiVmesnik, Naziv, Cena, Proizvajalec, ŠteviloOktav, ŠteviloStrun, TipStrun, ŠteviloPoljNaVratu, ŠteviloMagnetov, VrstaLesa, TipMagnetov}

Piano = {UporabniškiVmesnik, Naziv, Cena, Proizvajalec, ŠteviloOktav, VrstaLesa, ŠteviloPedal, LegaStrun, VrstaTipk, VrstaKladivc, Teža }

Iz grafa na sliki 37 je razvidno, da je koncept *ElektričnoBrenkalo* (c_R) nadrazred koncepta *ElektričnaKitara* (c_A). Prav tako lastnosti *ElektričnoBrenkalo = {Elektronika, UporabniškiVmesnik, Naziv, Cena, Proizvajalec,*

ŠteviloOktav, ŠteviloStrun, TipStrun} tvorijo podmnožico lastnosti koncepta *ElektričnaKitara = {Elektronika, UporabniškiVmesnik, Naziv, Cena, Proizvajalec, ŠteviloOktav, ŠteviloStrun, TipStrun, ŠteviloPoljNaVratu, ŠteviloMagnetov, VrstaLesa, TipMagnetov}*. Opazimo, da bodo ob uporabi te storitve nekatere lastnosti (*ŠteviloPoljNaVratu, ŠteviloMagnetov, VrstaLesa, TipMagnetov*) ostale neizpolnjene. Ker je koncept c_R bolj splošen kot c_A $(c_R < c_A)$, uporabimo enačbo:

$$S_i^{=}(c_R, c_A) = \frac{|p(c_R)|}{|p(c_A)|} = \frac{|8|}{|12|} = 0{,}667$$

Zanimivejši je primer, kjer nastopata koncepta *ElektričnoBrenkalo* (c_R) in *Piano* (c_A) . Koncept *ElektričnaBrenkala* je predstavljen z množico lastnosti, ki smo jo navedli že zgoraj, medtem ko koncept *Piano* vsebuje lastnosti *{UporabniškiVmesnik, Naziv, Cena, Proizvajalec, ŠteviloOktav, VrstaLesa, ŠteviloPedal, LegaStrun, VrstaTipk, VrstaKladivc, Teža }*. Koncepta nista v relaciji "vsebuje", torej velja razmerje $(c_R \neq c_A)$. Iz tega sledi, da uporabimo enačbo:

$$S_i^{=}(c_R, c_A) = \frac{|p(c_R) \cap p(c_A)|}{|p(c_A)|} = \frac{|p(c_R) \cap p(c_A)|}{|p(c_A)|} = \frac{|5|}{|11|} = 0{,}455$$

$REQ(c_i, c_o, c_f) =$ *Zahteva po spletni storitvi*
$ADV(c_{ji}, c_{jo}, c_{jf}) =$ *Seznam storitev s strani ponudnikov*

For all j get $ADV_j(c_{ji}, c_{jo}, c_{jf})$
If *same_ontology* (c_i, c_{ji}) $i = S_i^{=}(c_i, c_{ji})$
Else $i = S_i^{\neq}(c_i, c_{ji})$

If *same_ontology* (c_o, c_{jo}) $o = S_o^{=}(c_o, c_{jo})$
Else $o = S_o^{\neq}(c_o, c_{jo})$

If *same_ontology* (c_f, c_{jf}) $f = S_f^{=}(c_f, c_{jf})$
Else $f = S_f^{\neq}(c_f, c_{jf})$

$match[j] = \frac{(i + o + f)}{3}$

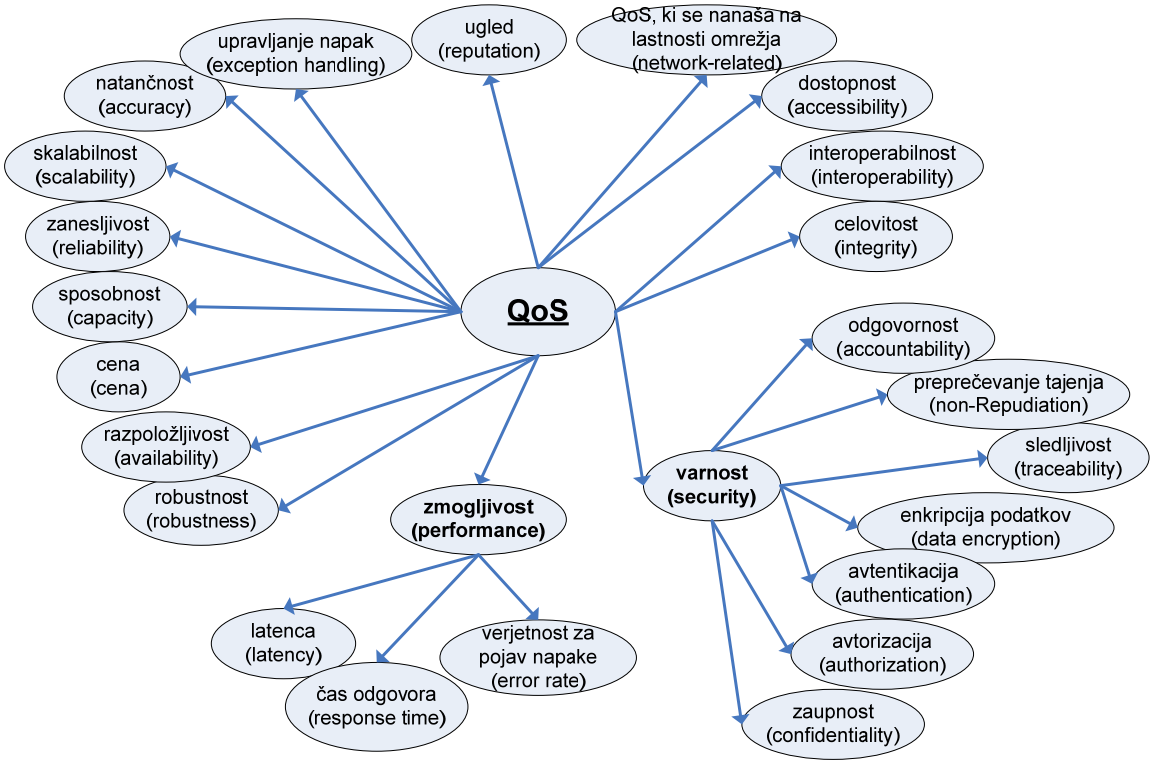
For all
Sort $match[j]$

Slika 38: Algoritem – SM-T algoritem.

Opis, kako postopati kadar je problemska domena opisana v različnih ontologijah, je kompleksnejši. Bralec si lahko nazoren prikaz ogleda v [28].

4.10 QoS Discovery

QoS temelji na opisni logiki in razširja WSMO-model. Parametri, ki odražajo kakovost storitve, so predstavljeni na sliki 39.



Slika 39: QoS-parametri.

QoS service discovery običajno nastopa skupaj z WSMO Discovery. Najprej se vrši primerjava nefunkcionalnih in funkcionalnih lastnosti. Šele nato, ko izločimo neustrezno množico storitev, se izvrši primerjava na podlagi QoS-parametrov. WSMO obravnava kakovost kot eno izmed nefunkcionalnih lastnosti, ki opisujejo spletno storitev.

Izbira storitve na podlagi QoS parametrov je dokaj zahtevna zaradi različnih metrik, s katerimi je lahko storitev opisana (npr, različni podatkovni tipi, razpon vrednosti,…).
Preden nadaljujemo si oglejmo in razložimo nabor spremenljivk, ki jih bomo uporabljali v nadaljevanju:

Q_I … nabor QoS parametrov.

Q_R … nabor QoS parametrov, ki pripadajo profilu zahtevane storitve.

Q_A … nabor QoS parametrov, ki pripadajo profilu objavljene storitve.

Q_N … obvezen nabor QoS parametrov. $Q_N \subseteq Q_I$.

Q_O … neobvezen nabor QoS parametrov. $Q_O = Q_I \setminus Q_N$.

Q_D … standarden nabor QoS parametrov, če uporabnik parametrov ne določi. Ponavadi $Q_N \subseteq Q_D$.

npr. če velja $Q_R = \{\} \wedge Q_D \subseteq Q_I$, potem bo $Q_R = Q_D$.

$s_R = \{NF_R, F_R, Q_R, C_R\}$ … profil zahtevane storitve (nefunkcionalne lastnosti, funkcionalne lastnosti, kakovost, cena).

$s_A = \{NF_A, F_A, Q_A, C_A\}$… profil objavljene storitve (nefunkcionalne lastnosti, funkcionalne lastnosti, kakovost, cena).

Uporabnik lahko po želji določi Q_N, Q_D, Q_I .

Profil, ki obsega kakovostno zahtevo uporabnika, opišemo kot množico $Q_R = \{r_1, r_2, ..., r_k\}$. Zahteva, ki jo izrazi uporabnik ima k kakovostnih parametrov. Podobno opišemo profile m storitev, ki tvorijo množico S. Torej lahko zapišemo $Q_S = \{Q_{A1}, Q_{A2}, ..., Q_{Am}\}$, kjer $Q_{Ai} = \{q_{i1}, q_{i2}, ..., q_{ij}\}$, $i, j \in N$. Oglas storitve S_i ima j kakovostnih parametrov. Nastopita lahko dve situaciji:

- $Q_R = \{\}$, potem velja $Q_R = Q_D$
- $Q_R \neq \{\}$, potem velja $Q_R = Q_R \cup Q_N$. Q_R se primerja z vsako $Q_{Ai}, i \in N$

V praksi se malokrat zgodi, da bosta imela Q_A in Q_R enako število parametrov. Za primer, kjer imamo podan Q_R, postopamo po naslednjih korakih:

- parametre iz Q_{Ai} prerazporedimo v enako zaporedje
- če Q_{Ai} ne vsebuje parametra, ki ga vsebuje Q_R, temu parametru iz Q_{Ai} dodelimo vrednost 0
- parametre, ki jih Q_R ne vsebuje, vsebuje pa jih Q_{Ai}, ignoriramo

Iz tega sledi, da ima QoS matrika kjer primerjamo storitve, ki tvorijo množico $M_Q = \{Q_R, Q_{A1}, Q_{A2}, ..., Q_{Am}\}$ naslednjo obliko:

$$M_Q = \begin{Bmatrix} r_1 & r_2 & r_3 & \dots & r_k \\ q_{11} & q_{12} & q_{13} & \dots & q_{1k} \\ q_{21} & q_{22} & q_{23} & \dots & q_{2k} \\ \dots & \dots & \dots & \dots & \dots \\ q_{m1} & q_{m2} & q_{m3} & \dots & q_{mk} \end{Bmatrix}_{(m+1)\times k}$$

V prvi vrstici so navedeni QoS parametri, ki jih uporabnik zahteva, v ostalih vrsticah pa QoS parametri, ki jih nudijo najdene storitve. Matriko M_Q je potrebno normalizirati, tako da preslikamo vse realne vrednosti v interval $[0,1]$. Novo normalizirano matriko bomo poimenovali Q'. Matrika ne vsebuje prve vrstice $[r_1, r_2, ...r_k]$.Parameter z maksimalno oz. minimalno vrednostjo dobi vrednost 1 ali 0. Dodelitev je odvisna od tega, kakšno vrednost predstavlja določen parameter. Npr. pri parametru *cena* težimo k temu, da bo imel čim manjšo vrednost. Torej bomo tisti storitvi, ki je predstavljena z vrstico i in katere parameter *cena* (predstavljen v stolpcu j) je v primerjavi s ceno, ki jo ponujajo druge storitve najmanjši, priredili vrednost $q_{ij} = 1$.

Poglejmo kako izračunamo vrednosti v normirani matriki:

1. če za podani parameter težimo k tem, da ima minimalno vrednost:

$$q'_{ij} = \left(1 - \frac{q_{ij} - q_{\min}}{q_{\max} - q_{\min}}\right) \tag{2}$$

2. če za podani parameter težimo k temu, da ima maksimalno vrednost:

$$q'_{ij} = \left(1 - \frac{q_{\max} - q_{ij}}{q_{\max} - q_{\min}}\right) \tag{3}$$

3. če za podani parameter težimo k temu, da ima parameter vrednost najbližje zahtevani:

$$q'_{ij} = \begin{cases} 1 - \dfrac{q_{\max} - q_{ij}}{q_{\max} - q_{\min}} & r_j \geq q_{\max} \\ \dfrac{q_{ij} - q_{\min}}{q_{\max} - q_{\min}} & r_j \leq q_{\min} \\ 1 - \left(\left|\dfrac{|q_{ij} - r_j| - o}{n - o}\right|\right) & r_j \in (q_{\min}, q_{\max}) \end{cases} \tag{4}$$

$$q_{\max} = \max\{q_{ij}\},\ q_{\min} = \min\{q_{ij}\},\ n = \max\{|q_{ij} - r_{ij}|\},\ o = \min\{|q_{ij} - r_{ij}|\},\ i \in m,\ j \in k$$

Končno lahko izračunamo, katera storitev najbolj ustreza navedenim zahtevam. W predstavlja uteži parametrov.

$$Q'' = Q' \times W = \sum_{i=1}^{m} \left(q'_{ij} \times w_i\right) \tag{5}$$

Za lažje razumevanje si oglejmo praktičen primer izračuna stopnje ujemanja med želeno in ponujenimi storitvami. Parametre bom predstavil v matriki.

	Price [$]	Transaction [0,1]	Time Out [µs]	Compensation Rate [%]	Peanlty Rate [%]	Execution Duration [µs]	Reputation [0,5]
Zahteve	**30**	**1**	**80**	**40**	**80**	**120**	**4,0**
Storitev 1	25	1	60	50	50	100	2,0
Storitev 2	40	1	200	80	10	40	2,5
Storitev 3	28	1	140	20	80	200	3,0
Storitev 4	55	1	180	60	40	170	4,0

Tabela 8: Testni primeri storitev [27].

V prvi vrstici so navedene zahteve QoS-parametrov, v ostalih vrsticah pa so QoS-parametri, ki jih nudijo storitve. V realnem svetu težimo k temu, da bosta parametra *Price* in *Execution Duration* imela čim nižje vrednosti, *Compensation Rate*, *Penalty Rate*, and *Reputation* pa čim višje vrednosti, parameter *Time Out* pa naj bi bil čim bližje zahtevani vrednosti.

Rezultat, ki ga dobimo z enačbami (1),(2),(3) je matrika Q'.

$$Q' = \begin{pmatrix} 1 & 1 & 1 & 0,5 & 0,571 & 0,625 & 0 \\ 0,5 & 1 & 0 & 1 & 0 & 1 & 0,25 \\ 0,9 & 1 & 0,6 & 0 & 1 & 0 & 0,5 \\ 0 & 1 & 0,2 & 0,667 & 0,429 & 0,188 & 1 \end{pmatrix}$$

Če določimo uteži $W =$ (4; 0; 0; 2; 1; 1 ;2), potem dobimo končni rezultat $Q'' =$ (6,196; 5,5; 5,6; 3,951).
Opazimo, da *Storitev1* najbolj ustreza našim zahtevam.

4.11 Strnjen pregled značilnosti algoritmov za odkrivanje semantičnih spletnih storitev

V tabeli 9 so predstavljeni pristopi opisani v poglavjih 4.1 – 4.10. Stolpci tabele predstavljajo naslednje značilnosti algoritma.

- **Elementi nad katerimi se izvaja primerjanje:** predstavljajo elemente ponujene oz. zahtevane storitve nad katerimi se izvaja primerjanje. Možno je primerjanje na podlagi profila storitve, VIPU oz. katere koli njihove podmnožice (npr. VI) ali pa nad raznimi dodatnimi parametri (npr. nefunkcionalni opis storitve, QoS parametri). Primerjanje se lahko izvaja tudi nad kategorijo storitve ali pa njenim tekstovnim opisom.

- **Rangiranje rezultatov primerjanja:** ta stolpec prikazuje ali algoritem omogoča razvrščanje storitev po stopnji ujemanja.

- **Posredno ujemanje:** algoritem podpira posredno ujemanje, kadar je možna preprosta kompozicija samostojnih storitev v verige storitev.

- **Način primerjanja:** v tem stolpcu je prikazano, kakšen pristop uporabi algoritem, ko izvaja primerjanje med dvema storitvama. Rezultat lahko dobimo na podlagi izključno logičnega ujemanja (npr. ujemanje na podlagi relacije vsebovanosti), kombinacije logičnega ujemanja in ujemanja na podlagi podobnosti ali pa kombinacije logičnega ujemanja in tehnik, ki jih lahko izvajamo nad usmerjenimi grafi.

Zgoraj predstavljeni algoritmi so samo del možnih pristopov pri odkrivanju spletnih storitev, saj je področje, ki se ukvarja z odkrivanjem semantičnih spletnih storitev, še zelo mlado. Ker bodo v nadaljevanju predstavljena tudi orodja za odkrivanje semantičnih spletnih storitev, sem izmed algoritmov večinoma izbral tiste, ki so v orodjih bolj ali manj uspešno realizirni. Večina primerjalnikov izkorišča profil semantične spletne storitve, natančneje elemente IOPE[20], ki so opisani v profilu storitve. Tudi intuitivno je tak pristop dober, saj ponavadi drži, da ujemanje elementov IOPE nadalje pomeni ujemanje storitev. Vendar pa ne smemo pozabiti tudi na druge načine primerjanja, ki ne temeljijo izključno na logičnem sklepanju.
Ne glede na to, nad katerimi elementi se vrši primerjava, je pri obsežnejših problemih zelo naivno pričakovati, da se bosta zahtevana in ponujena storitev ujemali do potankosti.

[20] IOPE – vhod, izhod, predpogoj, učinek (tudi VIPU).

PRISTOP	ZNAČILNOSTI			
	Elementi nad katerimi se izvaja primerjanje	**Rangiranje rezultatov primerjanja**	**Posredno ujemanje**	**Način primerjanja**
I	V/I	da	ne	logično
II	V/I , kategorija storitve , dodatni parametri (npr. QoS-parametri)	da	ne	logično
III	profil storitve	da	ne	logično
IV	tekstualen opis, VIPU	da	ne	logično + merjenje podobnosti (Similarity Measure)
V	profil storitve	ne	ne	logično + ujemanje grafa
VI	V/I	ne**	da	logično + tehnike preiskovanja grafa
VII	V/I	ne**	da	logično
QoS Discovery*	QoS-parametri	da	ne	logično
algoritem SM-T	V/I, na podlagi funkcionalnosti	da	ne	merjenje podobnosti med konceptoma na podlagi njunih lastnosti
WSMO Discovery	cilj storitve primerjamo z funkcionalnostjo objavljene storitve	ne	ne	logično

* QoS discovery običajno uporabljamo kot nadgradnjo komponente WSMO Discovery
** osnovni različici algoritma

Tabela 9: Strnjen pregled algoritmov za odkrivanje semantičnih spletnih storitev.

5 Pregled orodij za odkrivanje semantičnih spletnih storitev

Množica orodij, ki se ukvarjajo z odkrivanjem spletnih storitev, je velika. Raznolikost orodij je močno povezana z dejstvom, da obstaja množica različnih algoritmov, ki uporabljajo različne pristope, prav tako pa so različne spletne storitve opisane z različnimi ontologijami. V tem poglavju bom opisal nekaj najbolj poznanih orodij, ki se ukvarjajo z odkrivanjem semantičnih spletnih storitev. Opozoriti moram, da to niso edina orodja. Prav tako pa je potrebno spremljati novosti spodaj predstavljenih orodij, saj je razvoj na področju semantičnih spletnih storitev izredno hiter.

Izmed orodij, ki omogočajo realizacijo pristopa WSDL-S oz. SAWSDL si bomo ogledali IBM STWS, Radiant in Lumino. Orodja, ki omogočajo opise in iskanje na podlagi OWL-S, so OWL-S TUB Matchmaker, OWL-S/UDDI Matchmaker in Hybrid OWL-S Web Service Matchmaker, ki poleg logičnega primerjanja omogoča tudi primerjanje na podlagi merjenja podobnosti. WSMO-opise in iskanje le-teh podpirata komponenta WSMX Discovery in komponenta QoS Discovery.

5.1 Lumina in Radiant

Projekt METEOR-S, osnovan s strani LSDIS, je poizkus vnosa semantike k trenutnim standardom na področju spletnih storitev. MWSDI (METEOR-S Web Service Discovery Infrastucture) je ogrodje, zasnovano za polavtomatsko označevanje opisov, objavljanje in odkrivanje spletnih storitev. Opisi semantičnih spletnih storitev so narejeni v skladu s specifikacijami WSDL-S. WSDL-dokument semantično obogatimo z orodjem Radiant, medtem ko storitve odkrivamo z orodjem Lumina. Oba sta integrirana v orodje Eclipse. Pri objavi se opis semantične spletne storitve preslika v UDDI-imenik. Ontologija, s katero obogatimo opise v spletnih storitvah, je običajno opisana z OWL. Algoritem, ki ga uporablja orodje, smo opisali v poglavju 4.9. Radiant je trenutno edino orodje, ki podpira opise storitev v načinu SAWSDL[7].

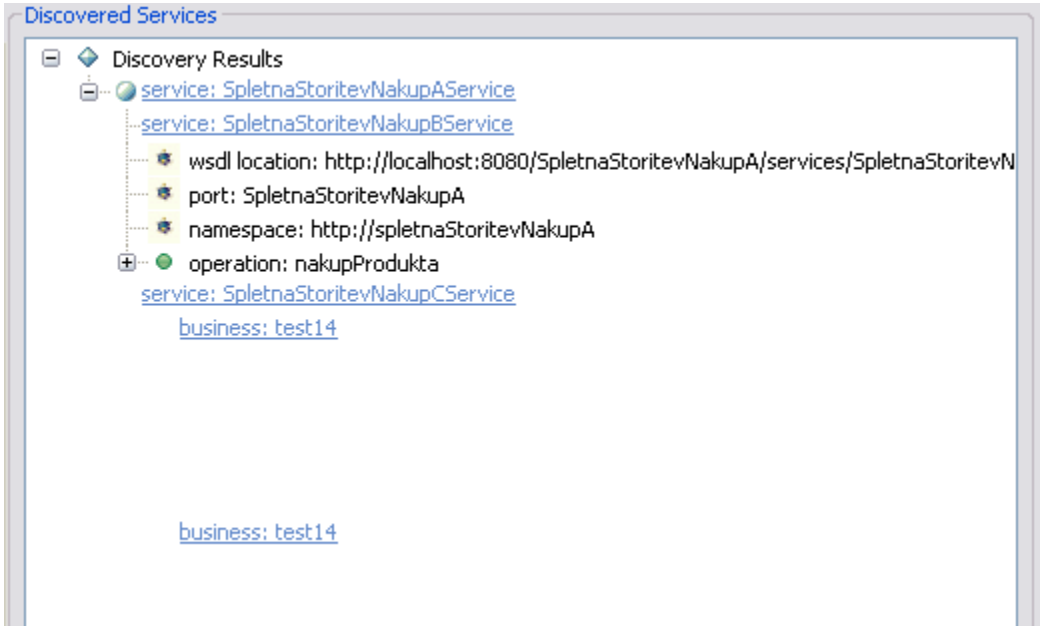
Drži, da je namestitev tako Lumine, kot Radianta izredno preprosta, vendar moramo upoštevati, da je potrebno poleg omenjenih orodij namestiti tudi UDDI-imenik. Ponudnik priporoča postavitev jUDDI-imenika, ki je verzije 0.9rc4 Sicer sama postavitev jUDDI-imenika ni zahtevna, vendar se moram obregniti ob dejstvo, da bi lahko razvijalci opozorili, da Java 1.6 jUDDI-imeniku povzroča resne probleme in je zato potrebno namestiti Javo 1.5. Poleg tega bi bilo dobro, če bi se spletna vadnica na ponudnikovi spletni strani povsem ujemala z implementacijo orodja v Eclipsovem okolju, saj bi bila tako prihranjena marsikatera neprijetnost.

Oba Radiant kot Lumina sta dokaj »hroščata«, zato se uporaba teh dveh orodij za resnejše namene trenutno ne zdi najbolj primerna. Pri uporabi orodja Radiant skupaj z novejšo verzijo orodja Eclipse3.3.2 prihaja do problemov pri objavi obogatenega dokumenta v jUDDI-imenik. Ko se pojavi taka napaka, je potrebno zapreti in ponovno pognati Eclipse3.3.2, zato priporočam uporabo starejše verzije Eclipse3.2, saj se omenjena napaka tam ne pojavi.

Z orodjem Radiant lahko semantično obogatimo operacije, vhode in izhode, predpogoje un učinke. V teoriji je pri Luimini stopnja ujemanja med zahtevano in ponujeno storitvijo prikazana kot vrednost, ki pripada intervalu [0..1]. Zanimivo je, da pri primerjavi istih konceptov, ki pripadata isti ontologiji, orodje pri vhodnih in izhodnih parametrih ne določi stopnje ujemanja. Ta ostane enaka nič, vendar pa se tu porodi vprašanje zakaj potem orodje na podlagi vhodnega (ali izhodnega) parametra lahko odkrije semantično spletno storitev. Ker orodje prikaže, da je stopnja ujemanja vhodnih (ali izhodnih) parametrov enaka nič, bi bilo smiselno trditi, da se spleti storitvi ne ujemata, vendar pa orodje pravilno odkrije želeno spletno storitev. Paziti je potrebno tudi, da imamo pri načinu »drag and drop« odprt pogled »Outline«, kamor dodajamo semantiko v WSDL, saj drugače spremembe v WSDL-dokumentu ne bodo opazne.

Baza MySQL, ki jo uporabljamo v zvezi z orodjem Radiant in baza MySQL, ki jo uporablja OWL-S UDDIMatchmaker se razlikujeta med seboj zato moramo paziti, da pri namestitvi drugega ne povozimo nastavitve prvega orodja.

V primeru, da želimo ponoviti iskanje storitev je priporočljivo orodje osvežiti. To storimo z uporabo gumba »reset«. Zaradi tega moramo parametre ponovno navesti, vendar pa se izognemo težavi, ki jo prikazuje slika 40. Na sliki bi kot rezultat iskanja morala biti prikazana samo storitev *spletnaStoritevNakupA*.



Slika 40: Težave, ki se pojavijo pri ponovnem iskanju.

V praksi običajno storitev vrača samo en parameter (npr. String), ki ima XML-obliko. Nato običajno ta niz razčlenimo in tako pridobimo želene podatke. Ta pristop je dober, če uporabnik pozna izhodne parametre, ki jih vsebuje XML-dokument. Problem nastane, ko hočemo tak WSDL-dokument semantično obogatiti. V takem primeru ima WSDL-dokument samo en izhodni parameter, orodje Radiant pa ne omogoča, da bi enemu izhodnemu parametru pripisali več konceptov. Za razliko od ostalih pristopov je tu problem najbolj opazen, saj semantiko vnašamo direktno v WSDL dokument in zato v zgoraj opisanem primeru ne moremo dobro opisati izhodnih parametrov storitve.

Če hočemo, da v WSDL-dokumentu navedemo izhodni parameter kompleksnega tipa, mora metoda, ki pripada razredu iz katerega kreiramo spletno storitev in s tem WSDL-dokument, vračati objekte. Taka storitev vrne parameter, ki je kompleksnega tipa. Slika 41 prikazuje tak WSDL-dokument, ki ga kreiramo z orodjem Eclipse WTP. Problemi se pojavijo, ko hočemo obogatiti vhode in izhode. Ko to storimo tako, da semantično obogatimo elemente, ki sestavljajo kompleksni tip, orodje Radiant brez težav objavi tak WSDL-dokument v imenik, vendar pa Lumina semantično obogatenih vhodnih in izhodnih parametrov ne prepozna. Lumina zazna vhodne oz. izhodne parametre, če jih označimo na drugem mestu v WSDL-dokumentu. Ko sem si natančneje ogledal WSDL-dokument, ki je predstavljen v spletni vadnici in WSDL-dokument, ki sem ga sam ustvaril, sem opazil, da se dokumenta rahlo razlikujeta. Z zeleno barvo sem označil kje v Radiantu semantično obogatimo operacije. Modra barva označuje mesto, kjer Lumina prepozna vhodne in izhodne parametre. Z oranžno barvo pa so prikazani koncepti iz ontologije Rosetta.owl, ki označujejo vhodne in izhodne parametre ter operacije.

Na sliki 41 vidimo, da storitev vrača več izhodnih parametrov, ki pa jih zaradi načina delovanja orodja Lumina, v tem dokumentu ne bomo mogli pravilno semantično obogatiti. Če hočemo, da bo iskanje uspešno tudi na podlagi vhodnih in izhodnih parametrov, moramo WSDL-dokument preoblikovati iz oblike prikazane na sliki 41 v obliko, ki jo prikazuje slika 42.

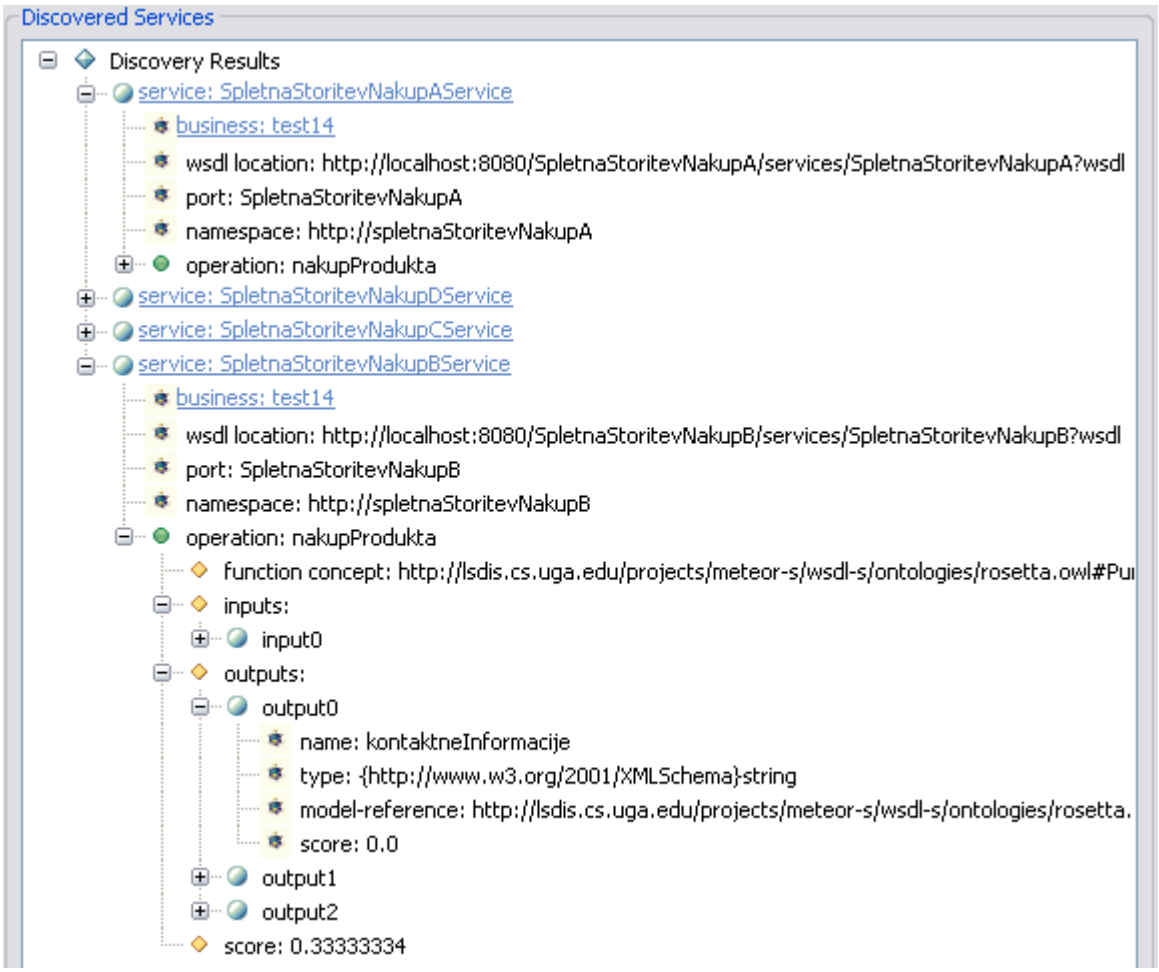
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://spletnaStoritevNakupB" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://spletnaStoritevNakupB" xmlns:intf="http://spletnaStoritevNakupB"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://spletnaStoritevNakupB"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="nakupProdukta">
    <complexType>
     <sequence>
      <element name="produkt" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="nakupProduktaResponse">
    <complexType>
     <sequence>
      <element name="nakupProduktaReturn" type="impl:VrniZalogoVrednostiStoritveB"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="VrniZalogoVrednostiStoritveB">
    <sequence>
     <element name="celotnaCena" nillable="true" type="xsd:string"/>
     <element name="kontaktneInformacije" nillable="true" type="xsd:string"/>
     <element name="osnovnaCenaNaEnoto" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
 </wsdl:types>

   <wsdl:message name="nakupProduktaResponse">

      <wsdl:part element="impl:nakupProduktaResponse" name="parameters"/>

   </wsdl:message>

   <wsdl:message name="nakupProduktaRequest">

      <wsdl:part element="impl:nakupProdukta" name="parameters"/>

   </wsdl:message>

   <wsdl:portType name="SpletnaStoritevNakupB">

      <wsdl:operation name="nakupProdukta">

         <wsdl:input message="impl:nakupProduktaRequest" name="nakupProduktaRequest"/>

         <wsdl:output message="impl:nakupProduktaResponse" name="nakupProduktaResponse"/>

      </wsdl:operation>

   </wsdl:portType>
    <wsdl:binding> … </wsdl:binding>
  <wsdl:service> … </wsdl:service>
```

Slika 41: WSDL-dokument, ki ga dobimo ob kreiranju spletne storitve.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://spletnaStoritevNakupB" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://spletnaStoritevNakupB" xmlns:intf="http://spletnaStoritevNakupB"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wssem="http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-
s/examples/WSSemantics.xsd" xmlns:Ontology3="http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-
s/ontologies/rosetta.owl#">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->

  <wsdl:message name="nakupProduktaResponse">

    <wsdl:part name="celotnaCena" type="xsd:string" wssem:modelReference="Ontology3#TotalPrice"/>
    <wsdl:part name="kontaktneInformacije" type="xsd:string" wssem:modelReference="Ontology3#ContactInformation"/>
    <wsdl:part name="osnovnaCenaNaEnoto" type="xsd:string" wssem:modelReference="Ontology3#UnitPrice"/>

  </wsdl:message>

  <wsdl:message name="nakupProduktaRequest">

      <wsdl:part name="produkt" type="xsd:string" wssem:modelReference="Ontology3#ProductLineItem"/>

  </wsdl:message>

  <wsdl:portType name="SpletnaStoritevNakupB">

    <wsdl:operation name="nakupProdukta" wssem:modelReference="Ontology3#PurchaseOrderRequest">

      <wsdl:input message="impl:nakupProduktaRequest" name="nakupProduktaRequest"/>

      <wsdl:output message="impl:nakupProduktaResponse" name="nakupProduktaResponse"/>

    </wsdl:operation>

  </wsdl:portType>
  <wsdl:binding> … </wsdl:binding>
  <wsdl:service> … </wsdl:service>
```

Slika 42: WSDL-dokument, ki smo ga priredili za delo z orodjem Lumina.

Problem se pojavi tudi, ko hočemo vnesti kategorijo storitve. Kljub temu, da sledim postopkom, ki so navedeni v spletni vadnici, orodje Lumina pri iskanju na podlagi dodatnega parametra *kategorija storitve* ne odkrije nobene storitve, čeprav bi jo moralo. Prav tako nobeden od testnih primerov, ki jih na strani navaja ponudnik orodja, nima realizirane semantične obogatitve s predpogoji, učinki in kategorijo storitve, ampak ima realizirane samo vhode, izhode in operacije.



Slika 43: Prikaz pravilnega izpisa rezultatov iskanja.

5.2 OWL-S TUB Matchmaker OWLSM

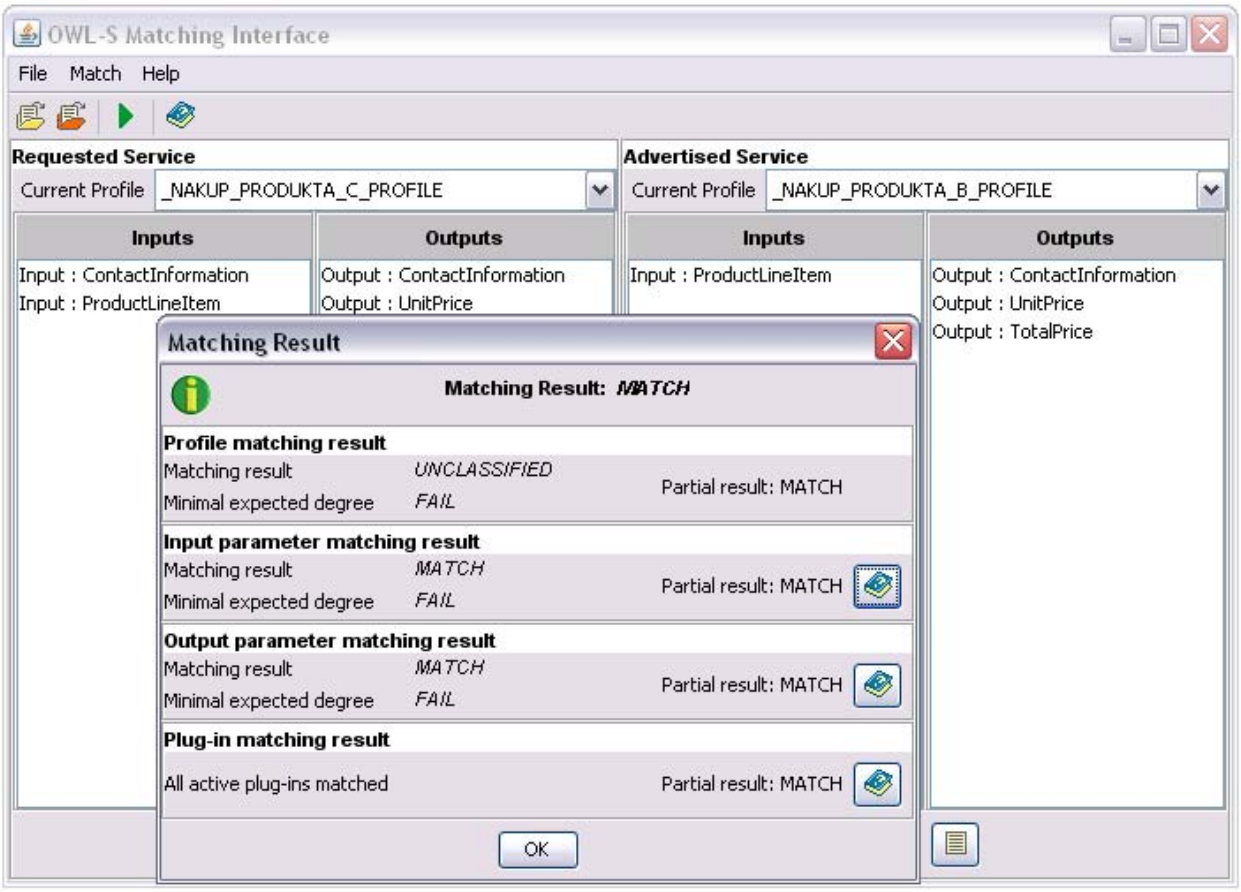
Orodje je bilo razvito pod okriljem Tehnične univerze v Berlinu TUB. Orodje izvaja primerjavo na podlagi pristopa, ki smo ga spoznali v poglavju 4.2 [7]. Zgrajeno je v Javi in namenjeno odkrivanju semantičnih spletnih storitev, ki so opisane v OWL-S 1.0. Zato je orodje iz vidika ontologije v rahlem zaostanku za ostalimi ponudniki, vendar pa ima izredno dober uporabniški vmesnik (slika 44), ki je zelo pregleden in v primerjavi z nekaterimi drugimi orodji zelo lepo prikaže stopnjo ujemanja zahtevane in ponujene spletne storitve. Orodje deluje tako, da lahko določi stopnjo ujemanja samo med dvema storitvama, ki ju navedemo. Ostala orodja, ki sem jih pregledal, lahko primerjajo zahtevano storitev z več ponujenimi storitvami, kar je bolje.

Pri tem orodju sem naletel na izredno veliko problemov že pri sami namestitvi, vendar pa moram omeniti, da so se razvijalci odzvali na vsako moje vprašanje. Pozorni moramo biti, da pred gradnjo *.jar* datoteke, s katero bomo poganjali orodje, dodamo datoteki *owljesskb.jar* in *jess.jar*. Slednja je na voljo 30 dni kot poizkusna verzija, potem pa jo je potrebno kupiti. Brez njiju je program neuporaben. Navodila, kako zgraditi aplikacijo, se nahajajo na ponudnikovi spletni strani[21]. Kljub temu, da se program uspešno zgradi, pa se ob nalaganju spletnih storitev, ki naj bi

[21] http://kbs.cs.tu-berlin.de/ivs/Projekte/owlsmatcher/

jih primerjali med sabo, prožijo izjeme. Ta problem sem odpravil tako, da razred `MatchingGUI.java`, ki se nahaja v paketu `de.tuberlin.flp.owlsm.gui`, poženem direktno v orodju Eclipse. Odkril sem tudi, da potrebujem *jaxen-1.1.1.jar*, čeprav to v navodilih ni navedeno. Do zapletov pride tudi pri uporabi primera, ki ga navaja ponudnik. OWLSM ga ne upošteva kot pravilno napisanega in zaradi tega zopet proži izjemo ob nalaganju spletne storitve. Ker ta primer ni deloval, sem se odločil, da bom uporabil testne primere iz paketa *owls-tc2_1*. Ko sem uporabil storitve, ki so opisane v ontologiji OWL-S 1.0, sem naletel na proženje iste izjeme. Preveril sem, ali so vsi navedeni URI-ji veljavni. Po manjši spremembi opisa je OWLSM začel ustrezno delovati. Potrebno je bilo samo še kreirati *.jar* datoteko, saj je poganjanje orodja neodvisno od orodja Eclipse veliko bolj elegantno.

Čeprav je orodje namenjeno odkrivanju semantičnih spletnih storitev, ki so opisane v OWL-S 1.0, pri storitvah, ki so opisane z ontologijo OWL-S 1.1, orodje prav tako deluje, vendar prihaja do manjših razlik. Ena izmed njih je ta, da orodje vhodne in izhodne parametre poimenuje z lastnimi imeni in ne s tistimi, ki so navedeni v profilu spletne storitve. Prav tako pa je potrebno upoštevati tudi dejstvo, da so trenutno opisi iz testne množice *owls-tc2_1* zelo preprosti in med njimi ne obstajajo neke večje razlike.



Slika 44: Uporabniški vmesnik orodja OWLSM.

5.3 Komponenta WSMX Discovery

Celotno WSMX-ogrodje kamor spada tudi komponenta WSMX Discovery, je bilo večinoma razvito v inštitutih DERI Galway in STI Innsbruck. Algoritem, na katerem temelji odkrivanje spletnih storitev, je opisan v poglavju 4.8. Komponento lahko uporabimo pri odkrivanju na podlagi besednega opisa, preprostega semantičnega opisa in na podlagi dodatnega preverjanja QoS-parametrov. Vsi opisi so modelirani v WSMO. Ker komponenta WSMX 0.4

podpira odkrivanje na podlagi preprostih semantičnih opisov, posledično ne upošteva predpostavk in pogojev, ki morajo biti izpolnjeni pred izvajanjem spletne storitve[9], ampak samo učinke in pogoje, ki so izpolnjeni po izvedbi storitve. Nova verzija komponente obljublja odkrivanje na podlagi podrobnih semantičnih opisov.

Namestitev izvajalnega ogrodja WSMX je preprosta. Najnovejšo različico prenesemo iz splete strani in jo razširimo. WSMX poženemo z ukazom *java -Djava.security.policy=policy.all -jar wsmx.core* (zraven je priložena datoteka *start.bat*, ki poenostavi poganjanje komponente). V spletnem brskalniku vpišemo naslov *http://localhost:8080/*, preko katerega dostopamo do konzole za upravljanje WSMX. Kliknemo na zavihek »Server view« kjer opazimo komponento *components:name=DiscoveryFramework*, s katero lahko odkrivamo in objavljamo spletne storitve.

V starejši različici izvajalnega ogrodja WSMX je potrebno QoS-komponento dodati naknadno. V novejši različici je ta komponenta že dodana. Uporabnika lahko zavede dokumentacija, ki jo nudi ponudnik QoS-komponente, saj se ta nanaša na starejšo različico WSMX0.3, kjer je potrebno ročno dodati QoS-komponento. Prav tako je potrebno paziti, da v datoteki *qosdisc.properties* ustrezno spremenimo nastavitve glede na lokacijo, kjer se nahaja WSMX-ogrodje. Ta problem najpreprosteje rešimo tako, da WSMX-ogrodje namestimo v mapo *c:/WSMX*, saj tako ni potrebno spreminjati nastavitev. Rezultati se poleg izpisa v konzoli zapišejo tudi v datoteko *output.wsml*. Če hočemo izklopiti odkrivanje spletnih storitev na podlagi dodatnih QoS-parametrov, je potrebno spremeniti vrednost parametra *wsmx.discovery.qosdiscovery* v datoteki *config.properties*. Ob spremembi nastavitev je potrebno ogrodje ponovno zagnati.

5.4 Komponenta QoS Discovery

Komponenta QoS discovery nudi avtomatično odkrivanje spletnih storitev, ki dodatno temelji na ujemanju parametrov, s katerimi opišemo kakovost storitve. Bila je razvita v okviru projekta DIP (Data, Information and Process Integration with Semantic Web Services). Pri odkrivanju ustreznih storitev komponenta uporablja algoritem, ki je opisan v poglavju 4.10. Pri opisih kakovostnih parametrov spletnih storitev uporabljamo WSML.

Komponenta primerja parametre, ki se dotikajo storitve, iz vidika njene kakovosti, vendar pa lahko uporabnik dodatno izbere opcijo, ki omogoči iskanje storitev na podlagi funkcionalnih zahtev. Iskanje na podlagi funkcionalnosti je izredno pomembno, saj morajo storitve v prvi vrsti zadostiti funkcionalnim zahtevam. Na parametre, ki opisujejo kakovost, pa lahko gledamo kot izboljšavo odkrivanja storitev. Iz vidika funkcionalnosti samostojna QoS-komponenta nudi odkrivanje na podlagi preprostih semantičnih opisov. Zato ne upošteva predpostavk in pogojev, ki morajo biti izpolnjeni pred izvajanjem spletne storitve[9], ampak samo učinke in pogoje, ki so izpolnjeni po izvedbi storitve. Komponenta izbere in rangira spletne storitve tako, da primerja seznam ponujenih storitev s ciljem, ki ga je navedel uporabnik storitve. Storitve nad katerimi nameravamo izvajati primerjavo določimo v datoteki *qosdisc.properties*.

Komponento lahko uporabljamo na dva načina. Pri prvem načinu deluje samostojno. Če hočemo poleg odkrivanja spletnih storitev na podlagi QoS parametrov uporabljati tudi odkrivanje na podlagi funkcionalnosti, moramo v datoteki *qosdisc.properties* ustrezno spremeniti parameter *functional*. Potem moramo komponento nekajkrat pognati, da nove nastavitve obveljajo[22]. To se mi zdi rahlo moteče, saj ne vemo po koliko poizkusih bo komponenta začela upoštevati tudi funkcionalne lastnosti.
Namestitev samostojne komponente je preprosta. Najlažje je, če uporabimo datoteko *qosdisc.zip,* v katero so vključene pomembne datoteke. Komponento poženemo s pomočjo datoteke *run.bat*, ki je vključena v datoteko *qosdisc.zip*. Rezultati so prikazani v konzoli, poleg tega pa se izpišejo tudi v izhodno datoteko *output.wsml*. Parameter, ki določa kam se bodo izpisali rezultati, lahko po želji spremenimo v datoteki *qosdisc.properties.* Paziti moramo, da obstaja datoteka, v katero se izpisujejo rezultati, ker komponenta sama ne more kreirati izhodne datoteke.

[22] http://lsirpeople.epfl.ch/lhvu/download/qosdisc/qosdisc.properties

Pri drugem načinu je komponenta integrirana v WSMX-ogrodje. Kot smo omenili v poglavju 5.3 ima različica WSMX v0.4 že vgrajeno komponento QoS Discovery. V starejše verzije WSMX-ogrodja pa je treba QoS-discovery komponento dodati. To storimo tako, da v mapo, kjer se nahaja WSMX ogrodje, dodamo datoteki *qosdisc.wsmx in qosdisc.properties*. Testni primeri, ki so ponujeni skupaj s starejšo verzijo, v novi verziji WSMX v0.4 ne delujejo, čeprav se mi zdijo napisani veliko bolj logično. Poleg tega se dokumentacija v zvezi s komponento Qos Discovery nanaša na starejšo verzijo WSMX v0.3. Zaradi tega sem se odločil, da s primeri uporabe počakamo na bolj zanesljivo verzijo. Bralec si podrobnejše opise lahko ogleda na spletni strani
http://lsirpeople.epfl.ch/lhvu/download/qosdisc/, kjer se nahajajo tudi testni primeri.

5.5 OWL-S/UDDI Matchmaker

Orodje je eno izmed prvih, ki se je ukvarjalo s primerjanjem dveh semantično opisanih storitev. Algoritem, na katerem temelji orodje, uporablja pristop, ki smo ga opisali v poglavju 4.1 [7]. Na voljo je v dveh različicah. Kot samostojna različica, ki jo poganjamo na lokalnem računalniku in kot verzija, ki ima vmesnik dostopen na spletu. OWL-S/UDDI Matchmaker uporablja UDDI-imenik, ki je zmožen hraniti opise spletnih storitev opisanih v OWL-S. Imeniku je dodan modul, ki izvaja primerjanje opisov storitev OWL-S, ki se nahajajo v njem. Pošiljanje in pridobivanje odgovorov poizvedb poteka preko vrat *capability port.* Ko UDDI-imenik sprejme zahtevo po objavi opisa spletne storitve, se sprva vrši procesiranje v UDDI-komponenti. Nato se preveri, ali oglas vsebuje kakršne koli informacije, ki se nanašajo na OWL-S. Če jih oglas vsebuje, ga imenik poda modulu, ki poskrbi za opis OWL-S[23].
Preizkusil sem verzijo, ki ne nudi grafičnega vmesnika, tako da za uporabnike, ki niso vešči programiranja ni najbolj primerna. Prav tako orodje ni združljivo s testnimi primeri iz paketa *owls-tc2_1*, pa čeprav so primeri opisani tako v OWL-S1.0, kot tudi v OWL-S1.1. Včasih OWL-S/UDDI Matchmaker iz neznanega razloga preneha delovati, dokler ga ponovno ne poženemo.

5.6 IBM STWS

Orodje so razvili v IBM alphaWorks in je del Emerging Tehnologies Toolkit (ETTK). ETTK je nabor tehnologij, ki so bile razvite v laboratorijih podjetja IBM. Orodje je zasnovano tako, da v primeru, ko ne najdemo storitve, ki bi ustrezala našim željam, orodje uporabi tehniko veriženja nazaj in ostale pristope, ki jih nudi področje umetne inteligence [7]. Tehniko veriženja nazaj smo podrobneje spoznali v poglavju 4.7. Rezultat, ki ga dobimo samo s primerjanjem ali pa kompozicijo, v kolikor je le ta potrebna, je predstavljen kot množica storitev. Vsaki storitvi pripada ustrezna stopnja ujemanja. Množico vrnjenih storitev, ki zadoščajo pogojem, pa lahko razvijalec oz. uporabnik dodatno prečisti. Opisi semantičnih spletnih storitev so narejeni v skladu z specifikacijami WSDL-S. Prav tako kot pri orodju Radiant, ontologijo, ki opisuje problemsko področje, opišemo z OWL, ki je de-facto standard (priporočen tudi s strani W3C). Orodje ni samostojno, ampak je implementirano kot dodatek (plug-in) k orodju WebSphere Integration Developer (WID) 6.0.1. Za uspešno namestitev potrebujemo tudi WordNet 2.1 in WebSphere Integration Developer (WID) 6.0.1[24].

Orodje poleg odkrivanja semantičnih spletnih storitev omogoča tudi semantično obogatitev WSDL-dokumenta, vendar IBM STWS, za razliko od Radianta, ne podpira opisov storitev z WSDL 2.0, ampak samo z WSDL 1.1. Prav tako je zadnja verzija orodja Lumina novejša od IBM STWS.

Ker je orodje plačljivo, se z njim nisem podrobneje ukvarjal. Za izčrpnejši opis naj se bralec obrne na ponudnika.

[23] http://www.daml.ri.cmu.edu/matchmaker/details.htm
[24] http://www.alphaworks.ibm.com/tech/wssem

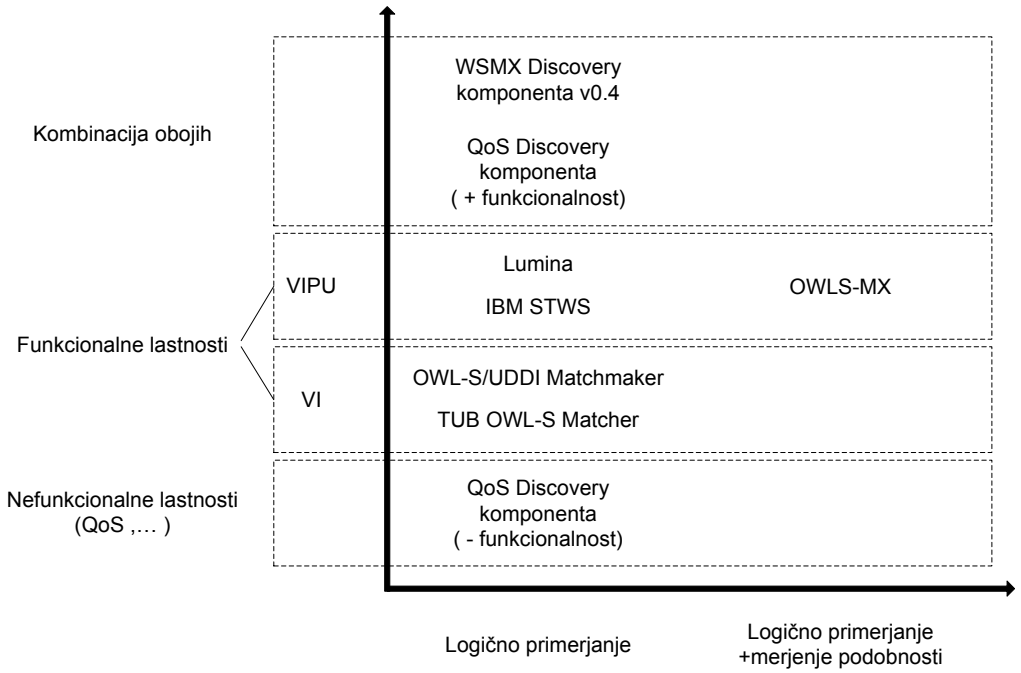
5.7 Hybrid OWL-S Web Service Matchmaker OWLS-MX

V nekaterih primerih je dobro, da se poleg logičnega primerjanja izvaja tudi primerjanje na podlagi sintaktične podobnosti. To omogoča orodje OWLS-MX, ki je bilo razvito v okviru projekta SCALLOPS, vodenega s strani nemškega raziskovalnega centra za umetno inteligenco DFKI Saarbruecken. Algoritem, ki ga uporablja orodje, temelji na pristopu, ki smo ga opisali v poglavju 4.4. Pri merjenju podobnosti lahko izbiramo med različnimi metodami. OWLS-MX podpira opise spletnih storitev v OWL-S [7].

Namestitev orodja je preprosta, prav tako pa lahko pohvalim tudi uporabniški vmesnik. Pri testiranju orodij sem se sprva zanašal na testni paket *owls-tc2_1,* saj vsebuje veliko opisov v OWL-S. Primeri so namenjeni lokalni uporabi, vendar ne na strežniku Apache Tomcat. Priporočena je uporaba XAMPP-okolja, saj imajo vsi primeri URI naveden tako, da jih lahko uporabljamo skupaj s strežnikom Apache HTTPD 2.2.8. Če hočemo primere uporabljati s Tomcatovim strežnikom, jim moramo ustrezno spremeniti URI-je, kar pa je zamudno opravilo. Pri razčlembi (angl. parse) semantičnih spletnih storitev, ki so opisane z OWL-S1.0, prihaja do problemov, zato testnih primerov *owls-tc1_1*, ki temeljijo na opisu v OWL-S1.0, ne moremo uporabljati skupaj z OWLS-MX v1.1, ampak s starejšo verzijo orodja OWLS-MX v1.0.

5.8 Strnjen pregled orodij za odkrivanje semantičnih spletnih storitev

Slika 45 prikazuje umestitev orodij po elementih nad katerimi se izvaja primerjanje. Natančnejši pregled in primerjava orodij je prikazana v tabelah 10-14.



Slika 45: Umestitev orodij glede na način primerjanja.

Pri predstavitvi orodij sem se odločil za tabelaričen prikaz. V tabelah 10-12 sem predstavil kriterije, ki se mi zdijo bistveni pri izbiri orodja. Prav tako pa sem v tabeli 13 in 14 predstavil povzetek prednosti in slabosti vsakega orodja.

Oceno, s katero bi podal katero izmed orodij se najbolje obnese je zelo težko določiti, saj vsako orodje podpira natanko eno ontologijo, s katero so opisane spletne storitve. Zato je izbira orodja močno odvisna od ontologije, s

katero je opisana spletna storitev. Lahko bi se sicer odločil, da ocenim orodja znotraj posamezne ontologije vendar množica orodij, ki sem jih obravnaval ni tako zelo velika. Orodji, ki podpirata opise v WSMO sta komponenta WSMX Discovery in komponenta QoS Discovery. Moje mnenje je, da lahko že iz same vsebine 5. poglavja razberemo, da je prvo orodje boljše od drugega. Orodji, ki omogočata realizacijo pristopa WSDL-S oz. SAWSDL sta IBM STWS in Radiant/Lumina. Ker je prvo orodje plačljivo se v podrobnejšo analizo nisem spuščal. Zato se mi zdi neupravičeno izpostavljati kateregakoli izmed njih. Malo večja množica orodij se ukvarja z odrivanjem spletnih storitev, ki so opisane z OWL-S. Tu bi določitev kriterijev in končna ocena orodja prišla bolje do izraza. Vendar pa moram omeniti, da sem med pregledom orodja Hybrid OWL-S Web Service Matchmaker imel najmanj problemov. Prav tako pa sem mnenja, da že samo tabele 10-14 postavijo omenjeno orodje pred ostali dve.

Orodja se med seboj zelo razlikujejo in omogočajo različne pristope. Če uporabnik želi vnesti semantiko in nadgraditi že obstoječo spletno storitev, pri tem pa noče preveč odstopati od dosedanjega načina dela in poznanih ter uveljavljenih tehnologij je priporočeno, da izbere orodje IBM STWS, ki omogoča obogatitev in iskanje opisov spletnih storitev, ali pa orodje Radiant in posledično išče spletno storitev z orodjem Lumina. Zaenkrat še ni prisotnih resnejših orodij, ki bi se ukvarjala z implementacijo SWSF. V kolikor želi uporabnik slediti teoretično najboljšemu pristopu s katerim lahko opišemo spletne storitve in ga lahko realiziramo, se bo odločil za uporabo WSMO in s tem izbral orodje, ki tak pristop podpira. Največ orodij za odkrivanje semantičnih spletnih storitev temelji na uporabi OWL-S za opis semantičnih spletnih storitev, saj je trenutno tudi najbolj uporabljena ontologija. Prav tako je izredno pomembno, kakšen algoritem orodje uporablja pri odkrivanju spletnih storitev, saj nekateri algoritmi omogočajo natančnejše odkrivanje semantičnih spletnih storitev, a so zaradi tega kompleksnejši.

Kriteriji, ki sem jih izbral pri pregledu orodij :

- **Ponudnik**
- **Najnovejša verzija / datum izdaje**
- **Odprtokodno orodje** – kriterij prikazuje, ali je orodje na voljo kot brezplačna različica ali pa ga je potrebno kupiti.
- **Spletna stran**
- **Ontologija za opis spletnih storitev** – kriterij označuje, v kateri ontologiji je predstavljen opis semantične spletne storitve.
- **Algoritem** – kriterij označuje, kateri algoritem za odkrivanje semantičnih spletnih storitev uporablja orodje.
- **Dokumentacija** – kriterij označuje, ali je na voljo zadosti dokumentacije, ki se nanaša na uporabo in namestitev orodja.
- **Aktualnost orodja** – kriterij prikazuje, ali orodje temelji na uporabi novejših tehnologij ter ali razvijalci še vedno bdijo nad projektom in je z njimi možno vzpostaviti kontakt.
- **Primerjanje z množico storitev** – določa, ali lahko navedemo več opisov ponudnikovih storitev nad katerimi se bo vršila primerjava (ti opisi so lahko hranjeni v UDDI-imeniku, v datoteki, ki vsebuje URI-je opisov, itd.) ali pa lahko opis zahtevane storitve primerjamo samo z enim od opisov ponujenih storitev.
- **Primernost za poslovne uporabnike** – kriterij označuje, ali je orodje dovolj preprosto, da ga bodo brez težav lahko uporabljali tudi uporabniki, ki niso ravno eksperti na področju IT.
- **Uporabniški vmesnik** – kriterij označuje, kakšen uporabniški vmesnik nudi orodje.
- **Dostop do opisov spletnih storitev** – kriterij prikazuje, kje so shranjeni opisi spletnih storitev, katere potem primerjamo z opisom zahtevane spletne storitve.

ORODJE	LASTNOSTI			
	Ponudnik	**Najnovejša verzija / datum izdaje**	**Odprtokodno orodje**	**Spletna stran**
OWL-S/UDDI Matchmaker	**DIP** *Data, Information and Process Integration with Semantic Web Services*	v1.3 / 15. januar 2007	da	http://www.daml.ri.cmu.edu/matchmaker/
IBM STWS	**IBM alphaworks**	Plugin za WebSphere Integration Developer (WID) 6.0.1 / 23. oktober 2006	ne	http://www.alphaworks.ibm.com/tech/wssem
Lumina	**LSDIS** *Large Scale Distributed Information Systems*	v1.0.1 / 29. maj 2007	da	http://lsdis.cs.uga.edu/
Hybrid OWL-S Web Service Matchmaker	**DFKI** *Das Deutsche Forschungszentrum für Künstliche Intelligenz*	v1.1c / 6. oktober 2006	da	http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/
The TUB OWL-S Matcher	**TU Berlin** *Technische Universität Berlin*	- / 29. december 2005	delno (brezplačna uporaba knjižnice Jess traja 30 dni)	http://owlsm.projects.semwebcentral.org/
WSMX Discovery Component	**DIP** *Data, Information and Process Integration with Semantic Web Services*	v0.4 / 6. februar 2007	da	http://www.wsmx.org/
Komponenta za odkrivanje spletnih storitev na podlagi njihove kakovosti (QoS)	**DIP** *Data, Information and Process Integration with Semantic Web Services*	Prototip1 / 22. junij 2006	da	http://lsirpeople.epfl.ch/lhvu/download/qosdisc/

Tabela 10: Strnjen pregled lastnosti orodij za odkrivanje semantičnih spletnih storitev 1/3.

ORODJE	LASTNOSTI				
	Ontologija za opis spletnih storitev	**Algoritem**	**Dokumentacija**	**Aktualnost orodja**	**Primerjanje z množico storitev**
OWL-S/UDDI Matchmaker	OWL-S 1.1	Pristop I	pomanjkljivo	da	da
IBM STWS	WSDL-S	/	/	da	da
Lumina	WSDL-S / SAWSDL	algoritem SM-T	dobro	da	da
Hybrid OWL-S Web Service Matchmaker	OWL-S 1.1	Pristop IV	odlično	da	da
The TUB OWL-S Matcher	OWL-S 1.0	Pristop II	dobro	delno	ne
WSMX Discovery Component	WSMO	WSMO Discovery	dobro	da	da
Komponenta za odkrivanje spletnih storitev na podlagi njihove kakovosti (QoS)	WSMO	QoS Discovery	odlično	da	da

Tabela 11: Strnjen pregled lastnosti orodij za odkrivanje semantičnih spletnih storitev 2/3.

ORODJE	LASTNOSTI		
	Primernost za poslovne uporabnike	**Uporabniški vmesnik**	**Dostop do opisov spletnih storitev**
OWL-S/UDDI Matchmaker	ne	ne	UDDI-imenik
IBM STWS	/	grafični	UDDI-imenik
Lumina	da	grafični	UDDI-imenik
Hybrid OWL-S Web Service Matchmaker	da	grafični	navedemo URI
The TUB OWL-S Matcher	delno	grafični	navedemo URI
WSMX Discovery Component	da	grafični	WSMX-ogrodje
Komponenta za odkrivanje spletnih storitev na podlagi njihove kakovosti (QoS)	delno	datoteka	navedemo URI

Tabela 12: Strnjen pregled lastnosti orodij za odkrivanje semantičnih spletnih storitev 1/3.

ORODJE	Problemi na katere sem naletel med pregledom orodja
OWL-S/UDDI Matchmaker	• Poizvedba po ustrezni spletni storitvi in objava spletne storitve v UDDI imeniku se naj izvaja v ločenem projektu. • Prodje ni kompatibilno s testnimi primeri iz paketa *owls-tc2_1*. • Včasih orodje iz neznanega razloga preneha delovati.
IBM STWS	• Primerljivo z orodjema Radiant in Lumina, vendar ni brezplačno, saj je potrebno imeti nameščen WebSphere Integration Developer (WID) 6.0.1. • Razliko od Radianta, ki ga uporabljamo v kombinaciji z Lumino, orodje ne podpira opisov storitev z WSDL2.0 ampak samo z WSDL1.1. • Za podrobnejši opis naj se bralec obrne na ponudnika.
Lumina	• Potrebna je postavitev lastnega UDDI-imenika. UDDI-imeniki, ki so navedeni na strani http://www.lsdis.cs.uga.edu niso javno dostopni ali pa ne dovoljujejo objave spletne storitve. • Za novejši jUDDI 2.0 nisem našel zadovoljive dokumentacije, ki bi omogočala njegovo postavitev. • Ponudnik navaja, da se pri uporabi orodja Lumina priporoča uporaba verzije jUDDI 0.94, ne opozori pa, da moramo striktno uporabljati Javo JDK1.5 in ne JDK1.6, saj pri uporabi slednje register jUDDI 0.94 ne deluje. • V spletni vadnici (tutorial-u) so predstavljene določene funkcije, ki pa so bile z novejšo verzijo orodja spremenjene in niso več na voljo (npr. navadno iskanje po UDDI-imeniku je bilo z novejšo verzijo orodja ukinjeno). • Pomanjkljivo je predstavljen način, kako naj objavimo semantično obogaten opis WSDL-dokumenta v registru jUDDI. • Po vsakem iskanju priporočam uporabo gumba »Reset«, ker se okno »Discovery Service« velikokrat ne osveži samo. • WSDL-dokument, ki pridobimo s tem, ko kreiramo spletno storitev v orodju Eclipse WTP, je potrebno rahlo prirediti, da ga lahko uspešno uporabimo v Lumini.
Hybrid OWL-S Web Service Matchmaker OWLS-MX	• Primeri, ki jih vsebuje paket *owls-tc2_1,* so namenjeni lokalni uporabi, vendar ne na strežniku Apache Tomcat. Priporočena je uporaba XAMPP-okolja, ker imajo vsi primeri URI naveden tako, da jih lahko uporabljamo skupaj s strežnikom Apache HTTPD 2.2.8. Če hočemo primere uporabljati s Tomcatovim strežnikom, jim moramo ustrezno spremeniti URI-je, kar pa je zamudno opravilo. • Pri razčlembi (angl. parse) semantičnih spletnih storitev, ki so opisane z OWL-S1.0, prihaja do problemov, zato testnih primerov OWLS-TC v1, ki temeljijo na opisih v OWL-S1.0 ne moremo uporabljati z OWLS-MX v1.1, ampak z OWLS-MX v1.0.

Tabela 13: Problemi na katere sem naletel med pregledom orodij 1/2.

ORODJE	Problemi, na katere sem naletel med pregledom orodja
The TUB OWL-S Matcher	• Dodatno moramo uvoziti dve knjižnici. To sta owljesskb.jar ter jess.jar. Slednja je na voljo 30 dni kot poizkusna verzija, potem pa jo je potrebno kupiti. • Orodje je namenjeno predvsem primerjanju storitev, ki so opisane v OWL-S 1.0, katero pa je nadomestila ontologija OWL-S 1.1. To je zadosten razlog, da se vprašamo, ali je uporaba tega orodja smiselna z ozirom, da obstajajo brezplačna orodja, ki temeljijo že na OWL-S 1.1. • Orodje je namenjeno primerjavi ujemanja dveh storitev in ne množice storitev. • Avtor navaja, da je koda s programerskega vidika polna napak, prav tako naletimo med izvajanjem na veliko opozoril, ki pa ne ogrožajo pravilnosti delovanja.
WSMX Discovery Component	• Pomanjkanje spletnih storitev, ki so opisane v WSMO (WSMO Working Group nudi nekatere primere opisane v WSMO, ki pa so po večini nepopolni in napačni). • Dokumentacija, na katero naletimo pri QoS-komponenti, opisuje, kako QoS-komponento vgradimo v komponento WSMX Discovery, kar je rahlo moteče, saj ima najnovejša verzija že vgrajeno QoS-komponento. • Verzijam, ki so starejše od WSMX 0.4, moramo ročno dodati opcijo iskanja po dodatnih QoS-parametrih. • Verzija WSMX 0.4 ne podpira odkrivanja na podlagi predpostavk in pogojev, ki morajo biti izpolnjeni pred izvajanjem spletne storitve.
Komponenta za odkrivanje spletnih storitev na podlagi njihove kakovosti (QoS)	• Samostojno komponento je treba nekajkrat pognati, preden začne upoštevati tudi iskanje po funkcionalnosti spletne storitve. • Pomanjkanje spletnih storitev, ki so opisane z dodatnimi QoS-parametri. • Komponenta ne podpira odkrivanja na podlagi predpostavk in pogojev, ki morajo biti izpolnjeni pred izvajanjem spletne storitve.

Tabela 14: Problemi na katere sem naletel med pregledom orodij 1/2.

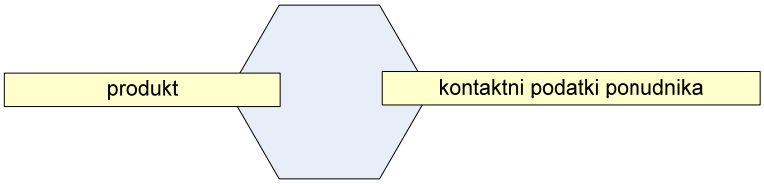
6 Primeri uporabe: Iskanje in opisi spletnih storitev

V sledečih poglavjih si bomo najprej ogledali opise storitev, nad katerimi bomo izvajali primere uporabe. Kot prvi premer uporabe bom predstavil navaden pristop objave v UDDI-imeniku. Sledili bodo pristopi WSDL-S, OWL-S, WSMO. Pri vsakem primeru bom najprej storitev ustrezno opisal, nato pa si bomo ogledali še način iskanja.

6.1 Opisi spletnih storitev

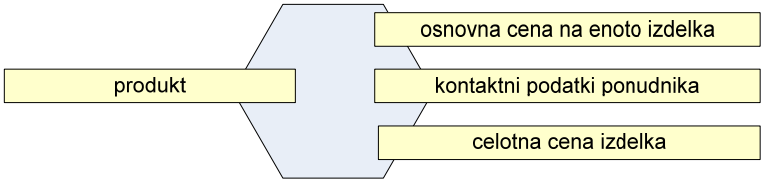
Bralec se mora zavedati, da spodaj navedene storitve niso primerne za dejansko uporabo, ampak služijo le kot primeri preko katerih si bomo ogledali primerjavo med navadnim opisom storitve v UDDI-imeniku in opisom spletne storitve s pomočjo ontologij.

SpletnaStoritevNakupA – Storitev omogoča, da glede na iskani produkt pridobimo kontaktne podatke o ponudnikih.



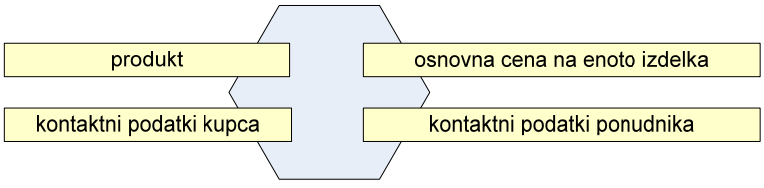
Slika 46: SpletnaStoritevNakupA.

SpletnaStoritevNakupB – Storitev omogoča, da glede na iskani produkt pridobimo kontaktne podatke ponudnika, osnovno ceno na enoto izdelka in celotno ceno izdelka.



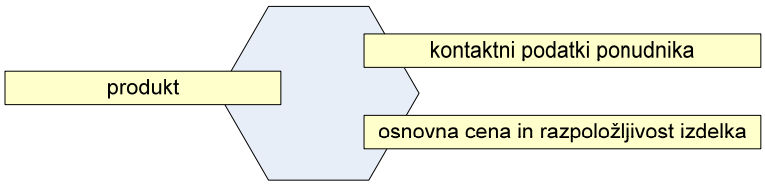
Slika 47: SpletnaStoritevNakupB.

SpletnaStoritevNakupC – Storitev omogoča, da glede na iskani produkt pridobimo kontaktne podatke ponudnika in osnovno ceno na enoto izdelka. Poleg iskanega produkta pa moramo kot vhodni parameter navesti tudi kontaktne podatke kupca.



Slika 48: SpletnaStoritevNakupC.

SpletnaStoritevNakupD – Storitev omogoča, da glede na iskani produkt pridobimo kontaktne podatke ponudnika, osnovno ceno na enoto izdelka in ali je želeni produkt trenutno na zalogi.



Slika 49: SpletnaStoritevNakupD.

6.2 Primer uporabe1

Testno okolje sistema:
- **Strežnik Apache Tomcat** 5.5.26 z izvajalnim okoljem Java 1.5
- **Imenik jUDDI** 0.9rc4
- **Orodje EclipseWTP** v3.3.2

6.2.1 Navaden pristop opisa spletne storitve

Preden objavimo spletno storitev v UDDI-imeniku moramo najprej priskrbeti opis poslovnega partnerja[25]. V našem primeru je poslovni partner *Baza50*, ki nudi različno izbiro knjig, športne opreme in glasbenih izdelkov. Skladno s tem sem storitve, ki jih nudi partner *Baza50* po klasifikacijskem sistemu NAICS, umestil v kategorijo *Retail trade*, natančneje *Sporting Goods, Hobby, Book, and Music Stores*. Slika 50 prikazuje objavo spletne storitve v jUDDI-imenik.

Get or find a business and then provide the WSDL URL and a name for the service to be published. Other parameters may also be specified. Authentication may be required.

Publish URL

http://localhost:8080/juddi/publish

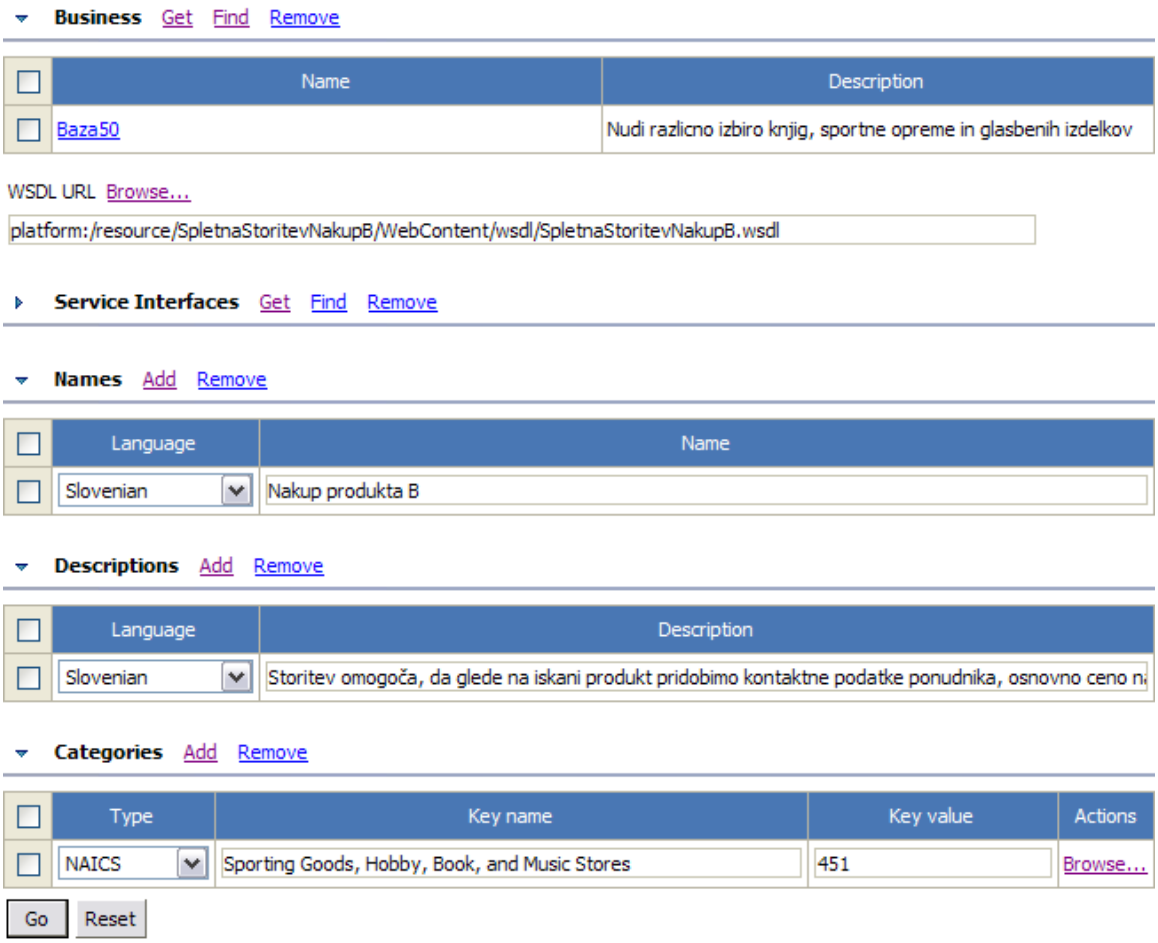
User ID

juddi

Password

•••••

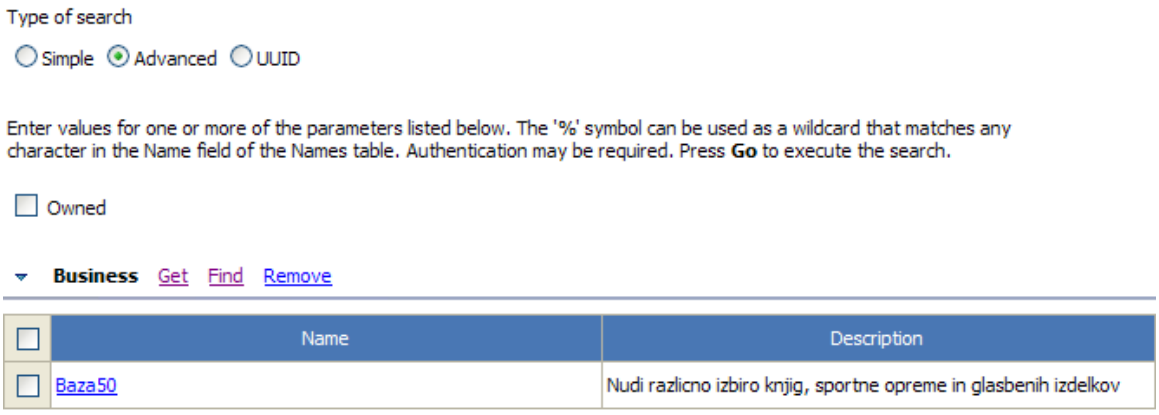
[25] Business description

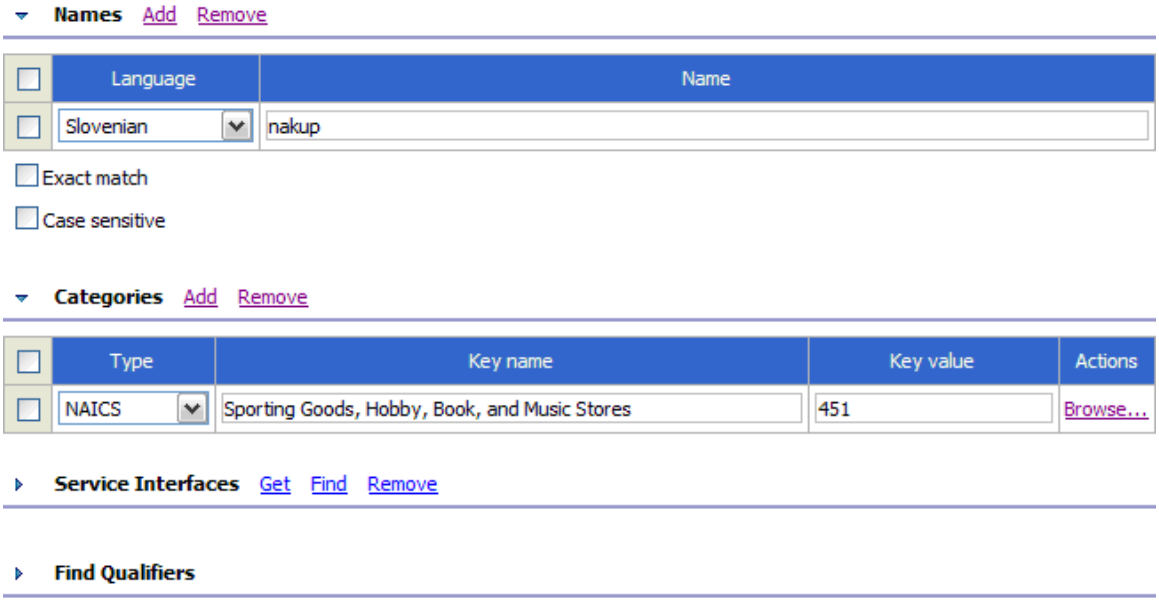


Slika 50: Objava spletne storitve v UDDI-imeniku.

6.2.2 Iskanje spletne storitve v UDDI-imeniku

Iskani oglas storitve pridobimo z navedbo ključnih besed v UDDI-imeniku, glede na kategorijo storitve, lahko pa tudi z navedbo ostalih parametrov, ki jih vidimo na sliki 51. Vendar pa nobeden od neštetih načinov ne omogoča zelo podrobnega iskanja s katerim bi bilo mogoče avtomatično najti in uporabiti storitev brez posredovanja uporabnika.





Slika 51: Iskanje spletne storitve po UDDI-imeniku.

Ker lahko z ontologijami opišemo spletne storitve bolj podrobno, so v nadaljevanju predstavljeni pristopi, ki smo jih spoznali v poglavju 3.2. Spletnim storitvam, ki so opisane v OWL-S ali pa imajo obogaten WSDL-dokument, dodatno uvozimo ontologijo *rosetta.owl*, s katero označimo operacije, vhodne in izhodne koncepte, itd. Celoten opis ontologije se nahaja na spletnem naslovu http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/rosetta.owl. Levi del slike 52 zaradi preglednosti prikazuje le izsek celotnega opisa.
Pri opisih, ki temeljijo na WSMO, je bila uporabljena ontologija *purcasheOntlogy.wsml*. Del ontologije *purcasheOntology.wsml* sem pretvoril iz ontologije *rosetta.owl*, nahaja pa se na spletnem naslovu www.pintar.tusek.info/SWS/ontologije/purcasheOntology.wsml. Pri pretvorbi jezika za opis ontologije iz OWL v WSML ali obratno si lahko pomagamo tudi z orodjem OWL - WSML Translator v1.0, ki ga dobimo na strani http://tools.sti-innsbruck.at/wsml/owl2wsml-translator/v0.1/. Dodatno primer uporabe storitve opisane v WSMO uporabljaja tudi ontologijo, ki se nahaja na naslovu http://www.gsmo.org/dip/travel/domainOntology.wsml.

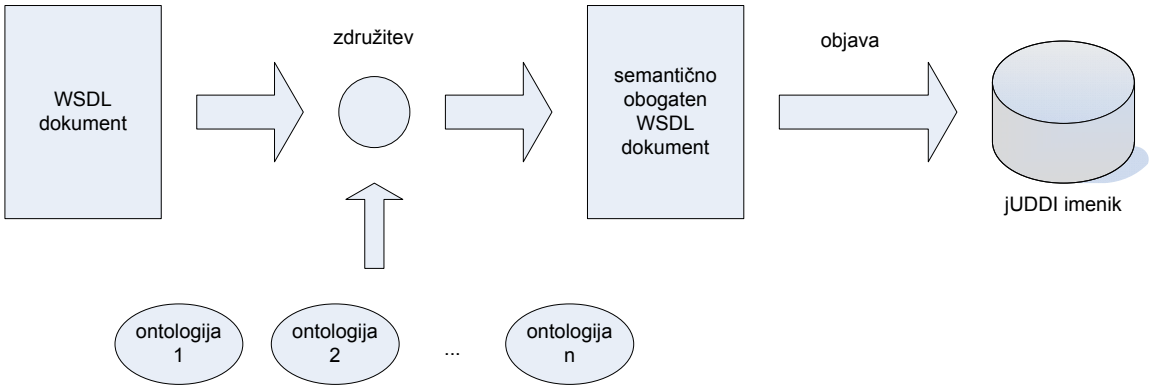
Slika 52: Grafični prikaz in izsek iz ontologije rosetta.owl

6.3 Primer uporabe2

Testno okolje sistema:
- **Podatkovna baza MySQL Server** 5.0
- **Strežnik Apache Tomcat** 5.5.26 z izvajalnim okoljem Java 1.5
- **Imenik jUDDI** 0.9rc4
- **Orodje Radiant** v0.9.4beta
- **Orodje Lumina** v1.0.1

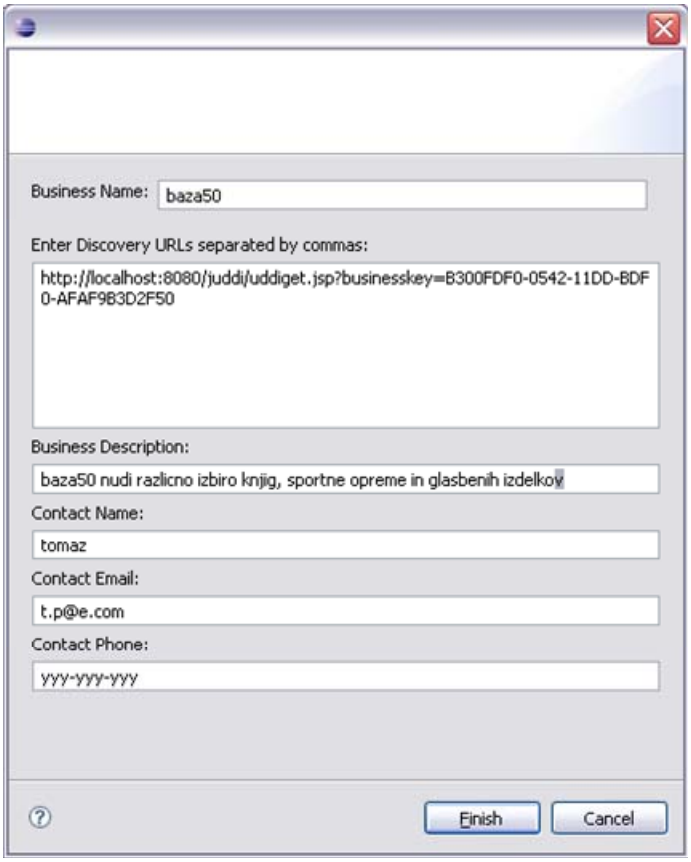
6.3.1 Semantična obogatitev WSDL-dokumenta

Celoten postopek objave spletne storitve je izveden v orodju Radiant (slika 53). Spletno storitev lahko objavimo šele potem, ko smo na podlagi *Discovery URL-ja* našli poslovnega partnerja. Slika 54 prikazuje opis poslovnega partnerja.

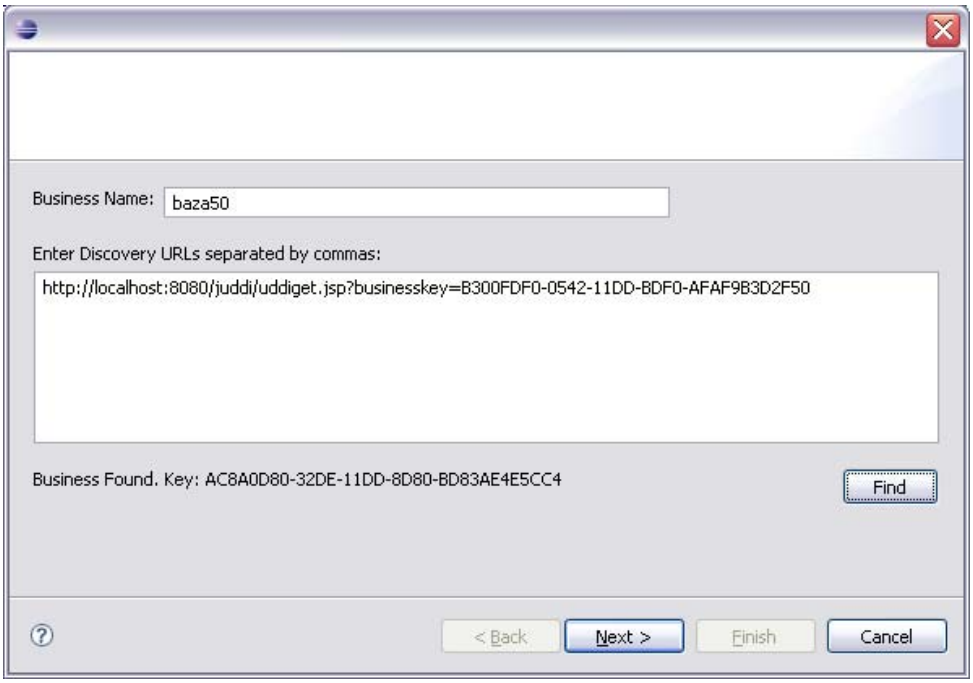


Slika 53: Postopek objave semantično obogatenega opisa z orodjem Radiant.

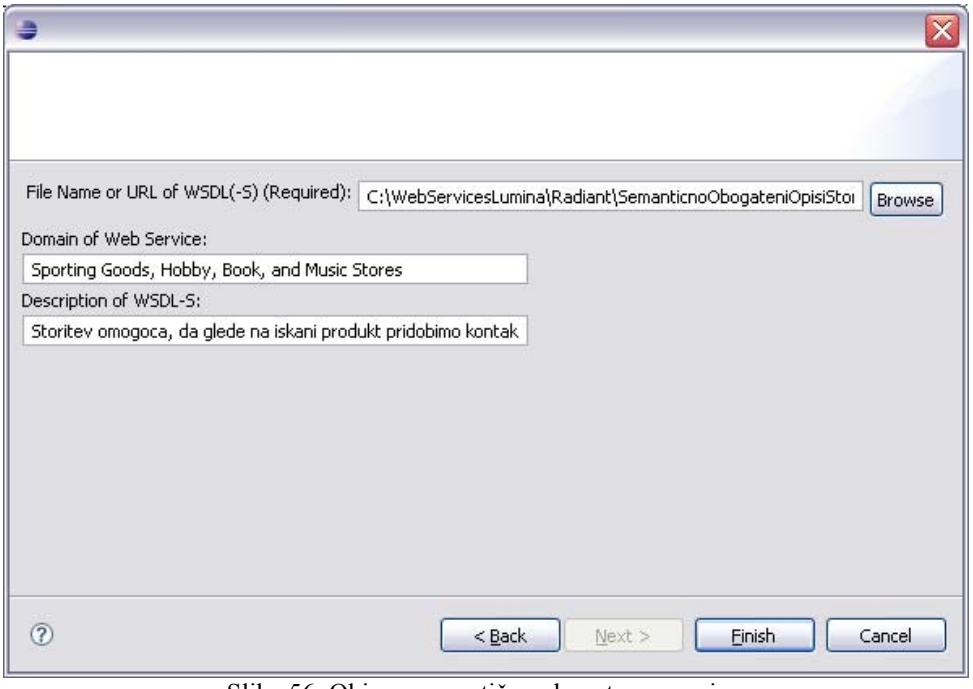
Koraki objave semantično obogatenega opisa v jUDDI-imeniku so dodatno prikazani na sliki 54, sliki 55 in sliki 56.



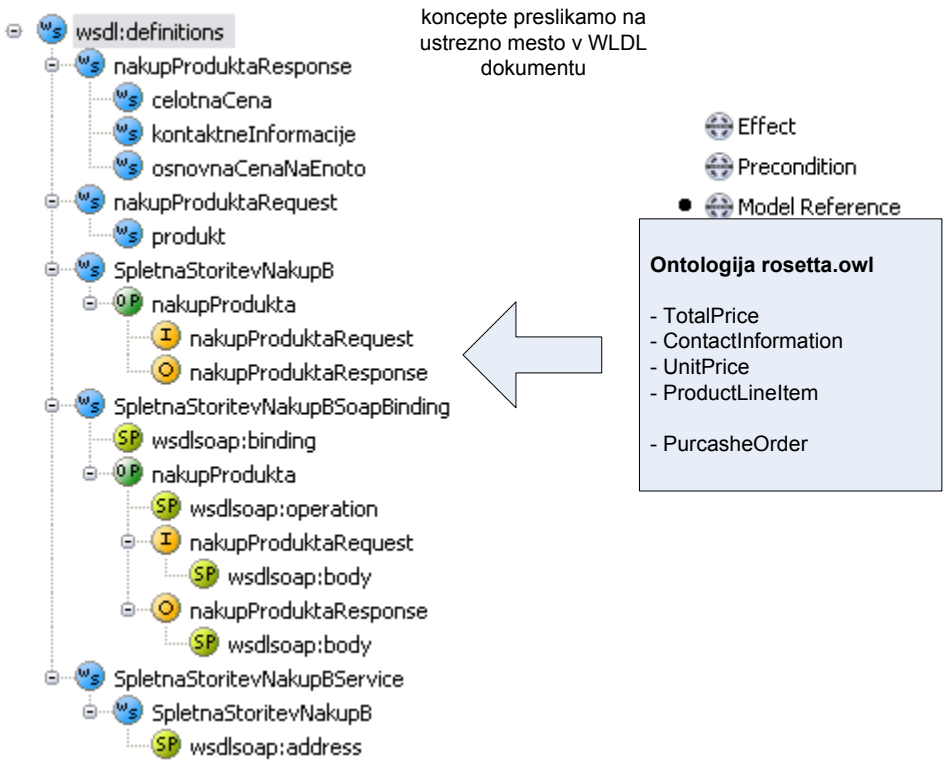
Slika 54: Opis in objava poslovnega partnerja.



Slika 55: Izbira poslovnega partnerja za objavo spletne storitve.



Slika 56: Objava semantično obogatenega opisa.



Slika 57: Grafični prikaz WSDL-opisa.

Zavihek *Outline* prikazuje grafični pogled na WSDL-dokument (slika 57 levo). V WSDL-dokument vnesemo semantiko tako, da povlečemo koncepte, ki pripadajo izbrani ontologiji, na ustrezno mesto v grafični hierarhiji. Paziti moramo, da pri WSDL1.1 uporabljamo način WSDL-S, pri WSDL2.0 pa način SAWSDL. Slika 58 prikazuje semantično obogaten WSDL dokument, ki opisuje storitev SpletnaStoritevNakupB.

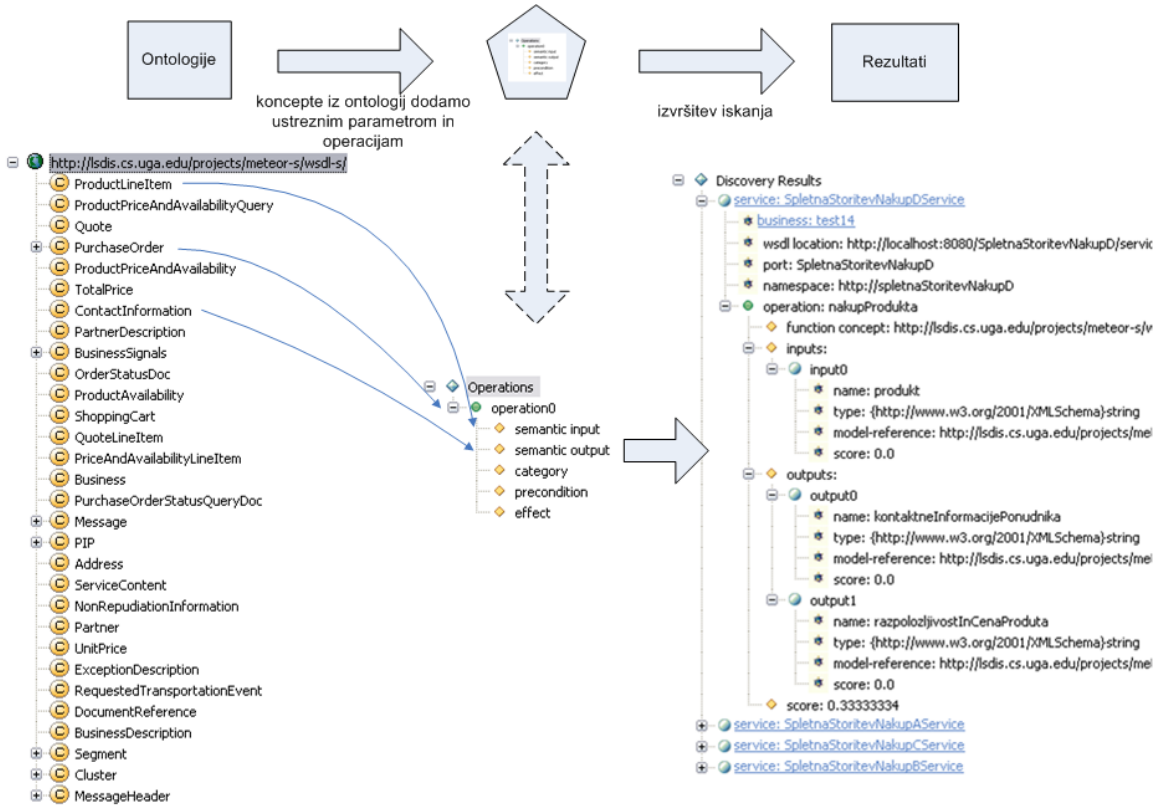
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://spletnaStoritevNakupB"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://spletnaStoritevNakupB" xmlns:intf="http://spletnaStoritevNakupB"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wssem="http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/examples/WSSemantics.xsd"
xmlns:Ontology3="http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-
s/ontologies/rosetta.owl#">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->

   <wsdl:message name="nakupProduktaResponse">
      <wsdl:part name="celotnaCena" type="xsd:string"
wssem:modelReference="Ontology3#TotalPrice"/>
       <wsdl:part name="kontaktneInformacije" type="xsd:string"
wssem:modelReference="Ontology3#ContactInformation"/>
       <wsdl:part name="osnovnaCenaNaEnoto" type="xsd:string"
wssem:modelReference="Ontology3#UnitPrice"/>
   </wsdl:message>

   <wsdl:message name="nakupProduktaRequest">
       <wsdl:part name="produkt" type="xsd:string"
wssem:modelReference="Ontology3#ProductLineItem"/>
   </wsdl:message>
```

```
   <wsdl:portType name="SpletnaStoritevNakupB">
      <wsdl:operation name="nakupProdukta"
wssem:modelReference="Ontology3#PurchaseOrder">
         <wsdl:input message="impl:nakupProduktaRequest" name="nakupProduktaRequest"/>
         <wsdl:output message="impl:nakupProduktaResponse" name="nakupProduktaResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="SpletnaStoritevNakupBSoapBinding"
type="impl:SpletnaStoritevNakupB">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="nakupProdukta">
         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="nakupProduktaRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>

         <wsdl:output name="nakupProduktaResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>

      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="SpletnaStoritevNakupBService">
      <wsdl:port binding="impl:SpletnaStoritevNakupBSoapBinding"
name="SpletnaStoritevNakupB">
         <wsdlsoap:address
location="http://localhost:8080/SpletnaStoritevNakupB/services/SpletnaStoritevNakupB"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

Slika 58: Obogaten WSDL-dokument, ki opisuje spletno storitev *SpletnaStoritevNakupB*.

6.3.2 Iskanje spletne storitve s semantično obogatenim WSDL-dokumentom

Iskanje z orodjem Lumina je preprosto, saj ni potrebno ustvariti nobenega dokumenta, ki bi predstavljal uporabnikove zahteve. Namesto tega uporabnik, podobno kot v Radiantu, elegantno povleče koncepte, ki pripadajo izbrani ontologiji, do ustreznega semantičnega parametra, ki je prikazan na sredini slike 59.
Rezultati iskanja so prikazani v oknu *Discovery Results* (slika 59 desno). Pri vsaki storitvi je naveden opis poslovnega partnerja in lokacija WSDL-dokumenta, preko katerega potem dostopamo do izbrane spletne storitve.



Slika 59: Iskanje z orodjem Lumina.

6.4 Primer uporabe3

Testno okolje sistema:
- **Strežnik Apache Tomcat** 5.5.26 z izvajalnim okoljem Java 1.5
- **Orodje OWLS-MX Matchmaker** 1.1c

6.4.1 Opis spletne storitve v OWL-S

Slika 60 prikazuje opis spletne storitve SpletnaStoritevNakupB v OWL-S. Kot vidimo je v profilu storitve navedeno, kaj storitev omogoča. V storitvenem modelu je prikazan atomaren proces s pripadajočimi vhodi in izhodi, protokol za povezovanje pa prikazuje dostop do storitve.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF  xmlns:owl = "http://www.w3.org/2002/07/owl#"
        xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
        xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:service = "http://www.daml.org/services/owl-s/1.1/Service.owl#"
        xmlns:process = "http://www.daml.org/services/owl-s/1.1/Process.owl#"
        xmlns:profile = "http://www.daml.org/services/owl-s/1.1/Profile.owl#"
        xmlns:grounding = "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
        xml:base = "http://localhost:8080/services/1.1/NakupProduktaB.owl">
```

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl" />
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl" />
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl" />
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Grounding.owl" />
  <owl:imports rdf:resource="http://localhost:8080/ontology/rosetta.owl" />
</owl:Ontology>

<service:Service rdf:ID="NakupProduktaBService">
  <service:presents rdf:resource="#NakupProduktaBProfile" />
  <service:describedBy rdf:resource="#NakupProduktaBProcess" />
  <service:supports rdf:resource="#NakupProduktaBGrounding" />
</service:Service>

<profile:Profile rdf:ID="NakupProduktaBProfile">
  <service:presentedBy rdf:resource="#NakupProduktaBService" />
  <profile:serviceName xml:lang="en">NakupProduktaB</profile:serviceName>
  <profile:contactInformation>
    <actor:Actor rdf:ID="Baza50">
      <actor:name>Tomaz Pintar</actor:name>
          <actor:title>
            Posredovanje Storitev
          </actor:title>
          <actor:phone>111 222 333 </actor:phone>
          <actor:fax>111 222 334 </actor:fax>
          <actor:email>baza50@baza50.si</actor:email>
          <actor:physicalAddress>
              Jamova XX,
              1000 Ljubljana,
              SLO
          </actor:physicalAddress>
          <actor:webURL>
        http://baza50.com
      </actor:webURL>
    </actor:Actor>
  </profile:contactInformation>
  <profile:textDescription xml:lang="en">Storitev omogoca, da glede na iskani produkt pridobimo kontaktne podatke ponudnika,
osnovna cena na enoto izdelka in celotna cena izdelka.</profile:textDescription>
  <profile:serviceCategory>
    <addParam:NAICS rdf:ID="NAICS-category">
      <profile:value>
          Sporting Goods, Hobby, Book, and Music Stores
      </profile:value>
      <profile:code>
          451
      </profile:code>
    </addParam:NAICS>
  </profile:serviceCategory>
  <profile:hasInput  rdf:resource="#ProductLineItem" />
  <profile:hasOutput rdf:resource="#UnitPrice" />
  <profile:hasOutput rdf:resource="#TotalPrice" />
  <profile:hasOutput rdf:resource="#ContactInformation" />
</profile:Profile>

<process:AtomicProcess rdf:ID="NakupProduktaBProcess">
  <service:describes rdf:resource="#NakupProduktaBService" />
  <process:hasInput  rdf:resource="#ProductLineItem"/>
  <process:hasOutput rdf:resource="#UnitPrice"/>
  <process:hasOutput rdf:resource="#TotalPrice"/>
  <process:hasOutput rdf:resource="#ContactInformation"/>
</process:AtomicProcess>

<process:Input rdf:ID="ProductLineItem">
  <process:parameterType
```

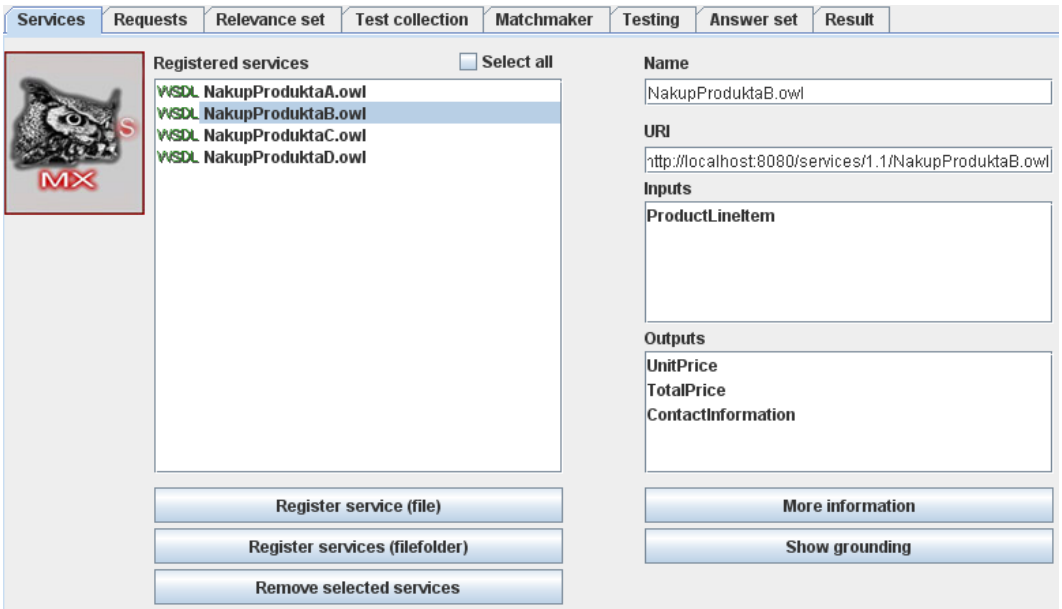
```
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/rosetta.owl#ProductLineItem</proces
s:parameterType>
    <rdfs:label>Naziv produkta, ki ga iscemo</rdfs:label>
  </process:Input>

  <process:Input rdf:ID="IncomingMessage">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/rosetta.owl#IncomingMessage</pro
cess:parameterType>
    <rdfs:label>Predstavlja vhodno sporocilo</rdfs:label>
  </process:Input>

  <process:Output  rdf:ID="UnitPrice">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/rosetta.owl#UnitPrice</process:para
meterType>
    <rdfs:label>Parameter predstavlja osnovno ceno na enoto izdelka</rdfs:label>
  </process:Output>

  <process:Output  rdf:ID="TotalPrice">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/rosetta.owl#TotalPrice</process:par
ameterType>
    <rdfs:label>Parameter predstavlja celotno ceno izdelka.</rdfs:label>
  </process:Output>

  <process:Output  rdf:ID="ContactInformation">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/rosetta.owl#ContactInformation</pr
ocess:parameterType>
    <rdfs:label>Parameter predstavlja kontaktne informacije ponudnika.</rdfs:label>
  </process:Output>

  <process:Output  rdf:ID="OutgoingMessage">
    <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/rosetta.owl#OutgoingMessage</proc
ess:parameterType>
    <rdfs:label>Predstavlja izhodno sporocilo</rdfs:label>
  </process:Output>

  <grounding:WsdlGrounding rdf:ID="NakupProduktaBGrounding">
    <service:supportedBy rdf:resource="#NakupProduktaBService" />
    <grounding:hasAtomicProcessGrounding rdf:resource="#NakupProduktaBProcessGrounding" />
  </grounding:WsdlGrounding>

  <grounding:WsdlAtomicProcessGrounding rdf:ID="NakupProduktaBProcessGrounding">
    <grounding:owlsProcess rdf:resource="#NakupProduktaBProcess" />
    <grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB.wsdl</grounding:
wsdlDocument>
    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
        <grounding:portType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB/SpletnaStoritevNa
kupB</grounding:portType>
        <grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB/nakupProdukta</
grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>

    <grounding:wsdlInputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB/nakupProduktaRe
quest</grounding:wsdlInputMessage>
```

```
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
            <grounding:owlsParameter rdf:resource="#IncomingMessage" />
        <grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB/parameters</gro
unding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>

    <grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB/nakupProduktaRe
sponse</grounding:wsdlOutputMessage>
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
            <grounding:owlsParameter rdf:resource="#OutgoingMessage" />
        <grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/wsdl/SpletnaStoritevNakupB/parameters</gro
unding:wsdlMessagePart>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>

  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

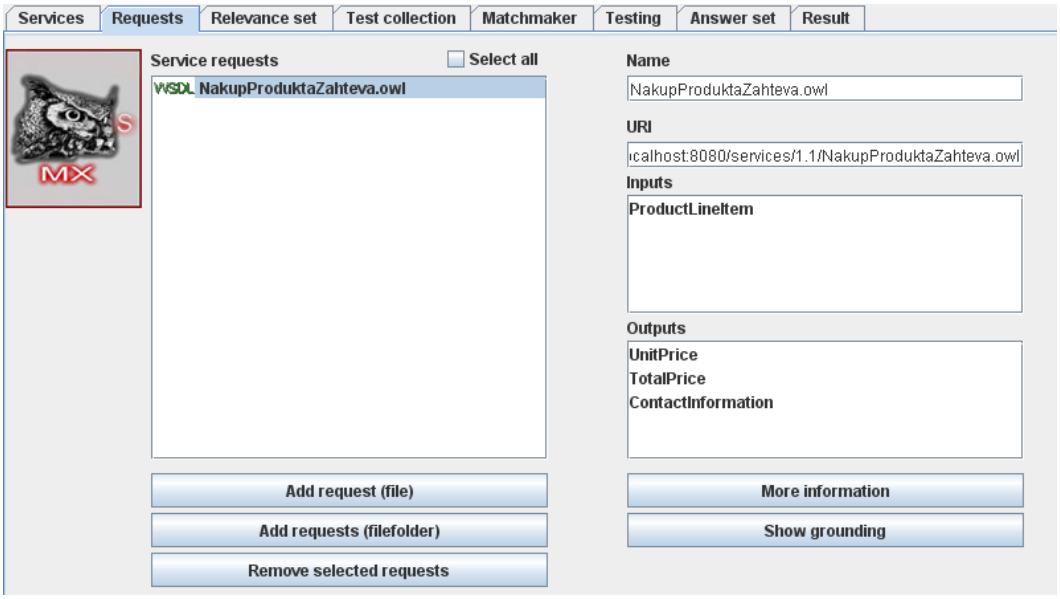
Slika 60: Opis spletne storitve SpletnaStoritevNakupB v OWL-S.

6.4.2 Iskanje na podlagi opisa spletne storitve v OWL-S

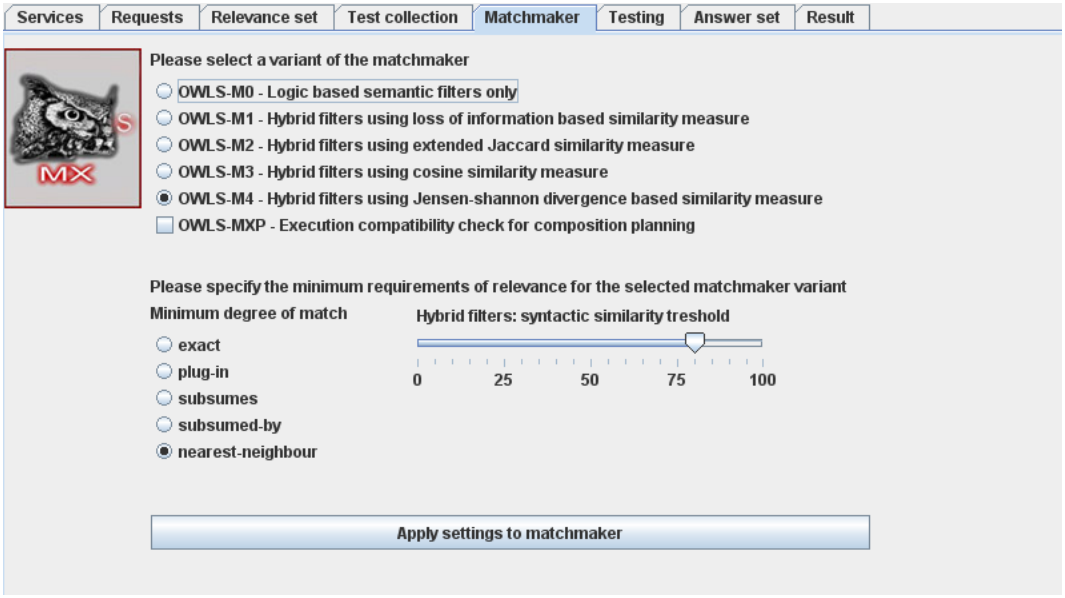
Pri predstavitvi iskanja bom uporabil orodje OWSL-MX, saj se mi zdi njegov način odkrivanja spletnih storitev boljši kot pri ostalih dveh orodjih. Storitve, ki so na voljo, navedemo v zavihku *Services,* kot prikazuje slika 61. Opazimo, da je poleg vsake storitve tudi zelena oznaka *WSDL*, ki nakazuje, da imajo storitve urejen protokol za povezovanje. Desno so navedeni vhodni in izhodni parametri, ki jih posamezna storitev nudi. Vmesnik kjer navedemo storitve, ki jih zahtevamo, se nahaja v zavihku *Requests,* ki ga prikazuje slika 62.



Slika 61: Storitve, ki so na voljo s strani ponudnika.



Slika 62: Zahteve s strani uporabnika storitve.

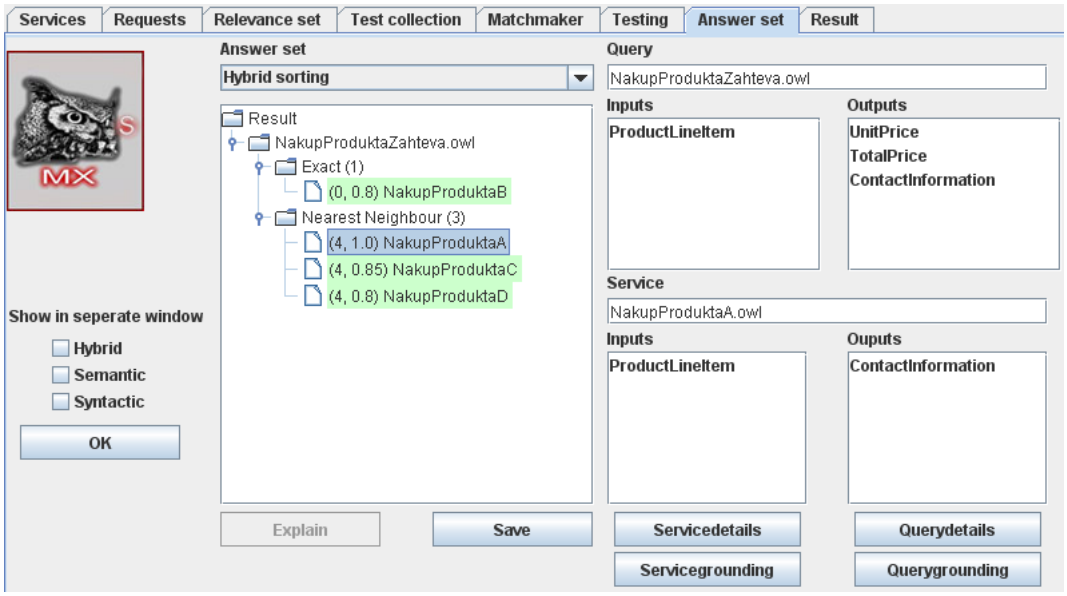


Slika 63: Nastavitve primerjalnika v orodju OWLS-MX.

Slika 63 prikazuje vmesnik, kjer lahko izbiramo različne načine preko katerih orodje izbere ustrezno spletno storitev. Ko je primerjanje končano, se rezultati izpišejo na način, kot ga prikazuje slika 64. Zavihek *Result* omogoča izvajanje tudi različnih analiz uspešnosti iskanja (recall[26]/precision[27], poraba pomnilnika, povprečen odzivni čas). V kolikor se odločimo izvajati analizo, moramo glede na vrnjene rezultate določiti tudi množico storitev, ki bi dejansko

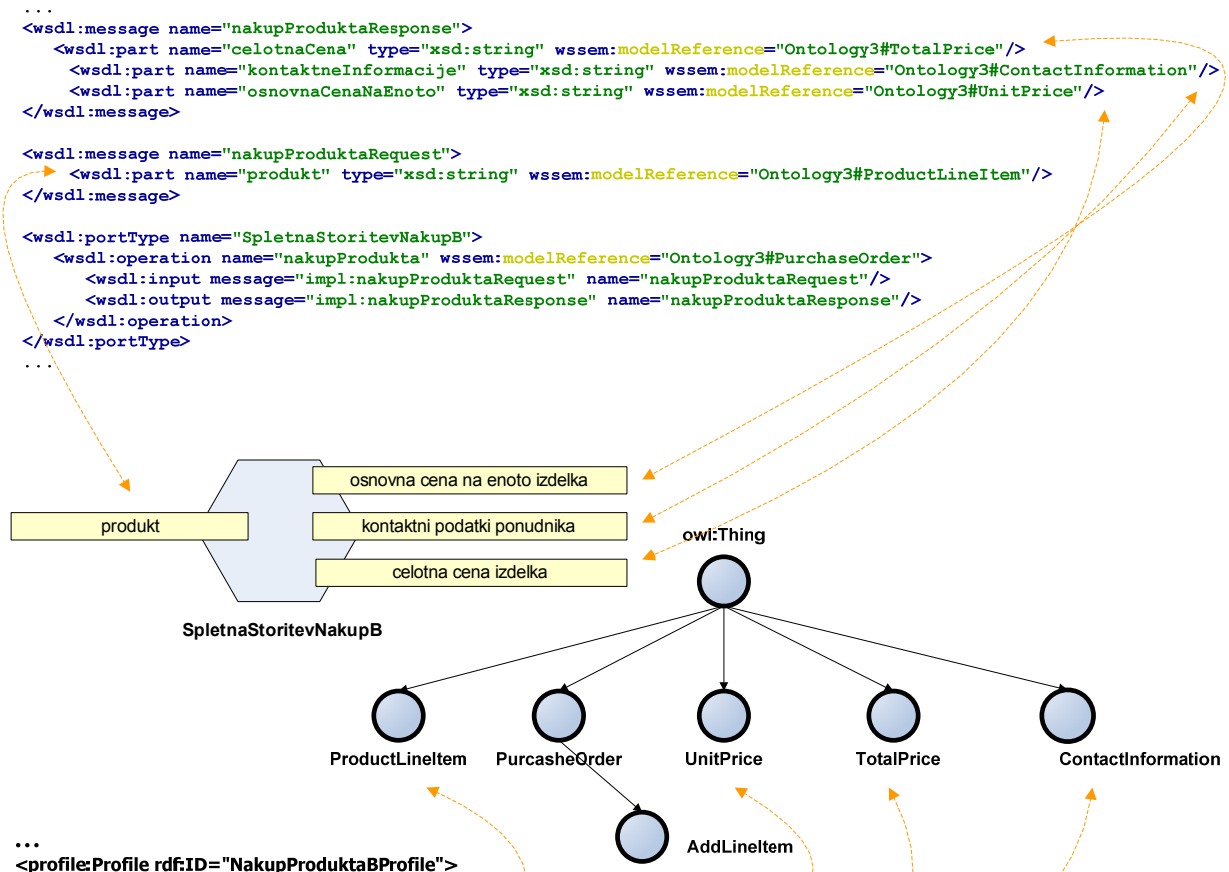
[26] recall – število ustreznih storitev, ki so vrnjene / število obstoječih ustreznih storitev.
[27] precision – število ustreznih storitev, ki so vrnjene / število vrnjenih storitev.

morale biti predlagane kot ustrezne. To storimo v zavihku *Relevance set*. Natančnejše opise analize bralec lahko najde v [17].



Slika 64: Rezultati, ki jih vrne OWLS-MX.

Zaradi zanimive primerjave med dvema različnima pristopoma so spodaj prikazani pomembnejši izseki dokumentov, kjer je uporabljen WSDL-S pristop in način, ko opišemo spletno storitev z OWL-S.




```
...
<profile:Profile rdf:ID="NakupProduktaBProfile">
   <service:presentedBy rdf:resource="#NakupProduktaBService" />
   <profile:serviceName xml:lang="en">NakupProduktaB</profile:serviceName>
   <profile:textDescription xml:lang="en">Storitev omogoca, da glede na iskani produkt pridobimo kontaktne podatke ponudnika,
osnovna cena na enoto izdelka in celotna cena izdelka.</profile:textDescription>
   <profile:hasInput  rdf:resource="#ProductLineItem" />
   <profile:hasOutput rdf:resource="#UnitPrice" />
   <profile:hasOutput rdf:resource="#TotalPrice" />
   <profile:hasOutput rdf:resource="#ContactInformation" />
 </profile:Profile>
 ...
  <process:Input rdf:ID="ProductLineItem">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/
rosetta.owl#ProductLineItem</process:parameterType>
   <rdfs:label>Naziv produkta, ki ga iscemo</rdfs:label>
  </process:Input>
  <process:Output  rdf:ID="UnitPrice">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/
rosetta.owl#UnitPrice</process:parameterType>
   <rdfs:label>Parameter predstavlja osnovno ceno na enoto izdelka</rdfs:label>
  </process:Output>
  <process:Output  rdf:ID="TotalPrice">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/
rosetta.owl#TotalPrice</process:parameterType>
   <rdfs:label>Parameter predstavlja celotno ceno izdelka.</rdfs:label>
  </process:Output>
  <process:Output  rdf:ID="ContactInformation">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost:8080/ontology/
rosetta.owl#ContactInformation</process:parameterType>
   <rdfs:label>Parameter predstavlja kontaktne informacije ponudnika.</rdfs:label>
  </process:Output>
...
```

Slika 65: Izsek iz dokumentov *SpletnaStoritevNakupB.wsdl* (zgoraj) in *SpletnaStoritevNakupB.owl* (spodaj).

6.5 Primer uporabe4

Testno okolje sistema:
- **WSMX - Web Service Execution Environment** (wsmx_dist-0.4)
- **WSMX discovery component + QoS discovery component** (obe komponenti sta vgrajeni v WSMX)

Pri tem primeru bomo storitev SpletnaStoritevNakupB dodatno omejili, da bo posredovala ustrezne podatke samo za izdelke, ki se nanašajo na Gibsonove kitare, prav tako pa bomo lokacijo ponudnikov omejili na Evropo.

6.5.1 Opis spletne storitve v WSMO

Odkrivanje preprostih semantičnih opisov nima vgrajenega iskanja na podlagi predpogojev in predpostavk. V svojih storitvah sem predpogoje kljub vsemu navedel, saj nova verzija WSMX 0.5 obljublja odkrivanje podrobnejših semantičnih opisov tudi z upoštevanjem predpogojev in predpostavk[28].

Slika 66: Odkrivanje storitve, ki ustreza cilju GoalSpletnaStoritevNakupB1 v WSMX-okolju.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"file:///c:/WSMX/resources/Storitve/SpletnaStoritevNakupB1.wsml#",
        dO _"http://www.gsmo.org/dip/travel/domainOntology#",
        pO _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/purcasheOntology.wsml#",
        tO _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/travelOntology.wsml#",
        dc _"http://purl.org/dc/elements/1.1#"}
/*
 * Spletna Storitev B1
 */
webService SpletnaStoritevNakupB1

        nonFunctionalProperties
                dc#title hasValue "NakupProduktaB1"
                dc#contributor hasValue "Tomaz Pintar"
          dc#description hasValue "Storitev omogoca, da glede na iskani produkt Gibson pridobimo kontaktne podatke
evropskega proizvajalca, osnovno cena na enoto izdelka in celotno cena izdelka."
        endNonFunctionalProperties

        importsOntology {_"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/purcasheOntology.wsml",
                         _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/travelOntology.wsml"}
```

[28] http://www.wsmx.org:8080/wsmxsite/.

```
	capability SpletnaStoritevNakupB1Capability
		precondition
			nonFunctionalProperties
			dc#description hasValue "Pogoji, ki so izpolnjeni pred izvajanju: Naziv produkta"
			endNonFunctionalProperties
			definedBy
				?item[pO#GlasbeniInstrument hasValue ?glasbeniInstrument] memberOf
pO#productLineItem and ?glasbeniInstrument memberOf pO#Gibson.

		postcondition
			nonFunctionalProperties
			dc#description hasValue "Pogoji, ki so izpolnjeni po izvajanju: Osnovna ceno na enoto
izdelka, celotna cena izdelka, kontaktne informacije ponudnika"
			endNonFunctionalProperties
			definedBy
				?item[pO#GlasbeniInstrument hasValue ?glasbeniInstrument] memberOf
pO#productLineItem and ?glasbeniInstrument memberOf pO#Gibson and
				?totalp memberOf pO#totalPrice and
				?contact[pO#location hasValue ?location] memberOf pO#contactInformation and
?location memberOf tO#EuropeanCity and
				?unitp memberOf pO#unitPrice.

	interface SpletnaStoritevNakupB1Interface
		choreography SpletnaStoritevNakupB1Choreography
			stateSignature SpletnaStoritevNakupB1StateSignature
				importsOntology
{_"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/purcasheOntology.wsml",

	_"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/travelOntology.wsml" }

					in pO#productLineItem

					out pO#totalPrice
					out pO#unitPrice
					out pO#contactInformation
```

Slika 67: Opis spletne storitve SpletnaStoritevNakupB1 v WSMO.

6.5.2 Iskanje na podlagi opisa spletne storitve v WSMO

Rezultati iskanja so prikazani v konzoli na način, ki ga prikazuje slika 69. Iskanje na podlagi QoS-parametrov nisem prikazal zaradi problemov, ki sem jih opisal v poglavjih 5.3 in 5.4.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"file:///c:/WSMX/resources/Cilji/GoalSpletnaStoritevNakupB1.wsml#",
	dO _"http://www.gsmo.org/dip/travel/domainOntology#",
	pO _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/purcasheOntology.wsml#",
	tO _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/travelOntology.wsml#",
	dc _"http://purl.org/dc/elements/1.1#"}
/*
 * Cilj B1
 */
goal GoalSpletnaStoritevNakupB1

	nonFunctionalProperties
		dc#title hasValue "Cilj B1"
		dc#contributor hasValue "Tomaz Pintar"
	  dc#description hasValue "Vrne spletno storitev, ki omogoca, da glede na iskani produkt Gibson pridobimo
kontaktne podatke avstrijskega ponudnika, osnovna cena na enoto izdelka in celotna cena izdelka."
	endNonFunctionalProperties
```

```
importsOntology {_"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/purcasheOntology.wsml",
                 _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/travelOntology.wsml"}

capability goalCapability
        precondition
                nonFunctionalProperties
                dc#description hasValue "Zeljeni pogoji, ki so izpolnjeni pred izvajanju: Naziv produkta"
                endNonFunctionalProperties
                definedBy
                        ?item[pO#GlasbeniInstrument hasValue ?glasbeniInstrument] memberOf
pO#productLineItem and ?glasbeniInstrument memberOf pO#Gibson.

        postcondition
                nonFunctionalProperties
                dc#description hasValue "Zeljeni pogoji, ki so izpolnjeni po izvajanju: Osnovna ceno na
enoto izdelka, celotna cena izdelka, kontaktne informacije ponudnika"
                endNonFunctionalProperties
                definedBy
                        ?item[pO#GlasbeniInstrument hasValue ?glasbeniInstrument] memberOf
pO#productLineItem and ?glasbeniInstrument memberOf pO#Gibson and
                        ?totalp memberOf pO#totalPrice and
                        ?contact[pO#location hasValue ?location] memberOf pO#contactInformation and
?location memberOf tO#AustrianCity and
                        ?unitp memberOf pO#unitPrice.



interface GoalSpletnaStoritevNakupB1Interface
        choreography GoalSpletnaStoritevNakupB1Choreography
                stateSignature GoalSpletnaStoritevNakupB1StateSignature
                        importsOntology
{_"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/purcasheOntology.wsml",
 _"file:///c:/WSMX/resources/qosdiscovery/ontologies/bankinter/travelOntology.wsml" }

                                        in pO#productLineItem

                                        out pO#totalPrice
                                        out pO#unitPrice
                                        out pO#contactInformation
```

Slika 68: Opis cilja, ki ga želimo doseči v WSMO.

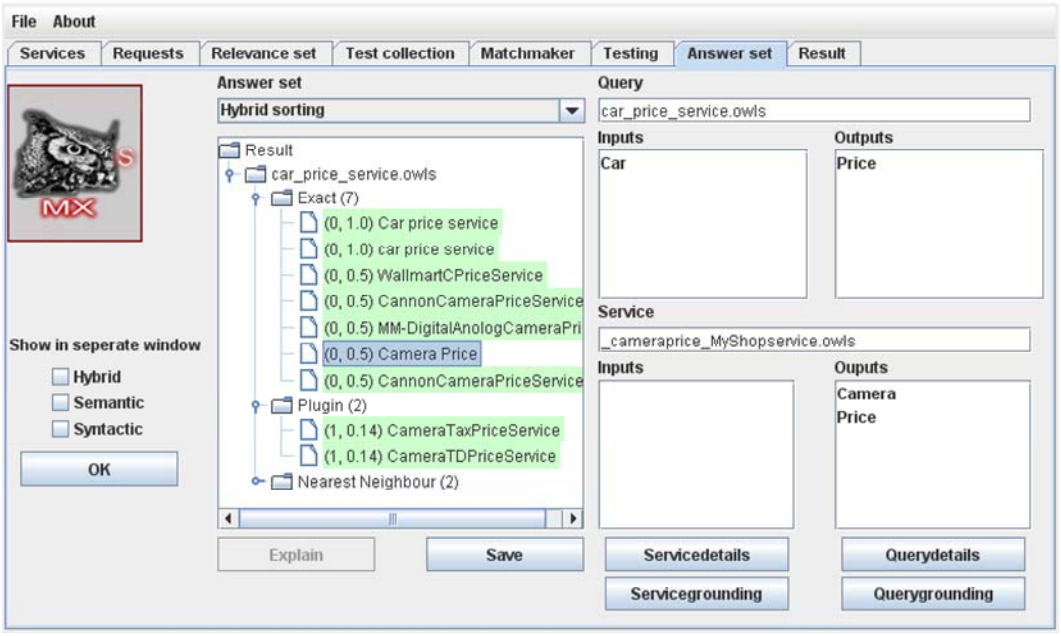
Slika 69: Prikaz spletnih storitev, ki ustrezajo želenim ciljem.

6.6 Povzetek odkrivanja na podlagi sintaktičnega in semantičnega opisa

V Eclipsovem iskalniku »Web Service Explorer« lahko iščemo le na podlagi kategorije storitve in ne na podlagi podrobnejšega opisa storitve. To pa v večini primerov in ob veliki množici objavljenih storitev ni dovolj učinkovito, saj je premalo natančno pa tudi vrnjenih rezultatov je občutno preveč. V primeru, da storitvi določimo kategorijo *Sporting Goods, Hobby, Book, and Music Stores,* nam iskalnik ne bo vrnil storitev, pri katerih smo za kategorijo določili *Retail Trade*. Čeprav je po klasifikaciji **NAICS** kategorija *Goods, Hobby, Book, and Music Stores* specializacija kategorije *Retail Trade*.

Problemi pri odkrivanju spletnih storitev lahko nastanejo, če opise storitev izdelamo površno in vanje ne vnesemo dovolj semantike. Ko iščemo na podlagi vhodov in izhodov, lahko dobimo storitve, ki nimajo nobene povezave z našim željami. Oglejmo si primer, ki je vzet iz testne množice owl-tc2_1. Imamo ontologijo, ki poleg ostalih vsebuje tudi koncepte »car«, »price« in »camera«. Iščem storitev, ki na podlagi modela avtomobila vrne njegovo ceno. Ponudnik objavi tudi drugo storitev, ki vrne model in ceno fotoaparata. V primeru, da bomo iskali spletno storitev samo na podlagi zgoraj omenjenih vhodno/izhodnih parametrov, bosta kot rezultat iskanja podani obe storitvi. Problem prikazuje slika 70.

S preprostimi semantičnimi opisi ne izkoristimo celotnega potenciala, ki ga semantični splet ponuja, vendar pa so kompleksni semantični opisi spletnih storitev za neizurjenega uporabnika lahko prezahtevni.
Ideja, da bi zgradili aplikacijo, ki bi razbrala in zgradila opise spletnih storitev iz tekstualnega zapisa, je dobra. Prav tako pa bi lahko bolj avtomatizirali gradnjo semantičnih opisov spletnih storitev.



Slika 70: Storitev, ki pomensko ne ustreza našim zahtevam.

7 Sklep

Eno izmed glavnih vodil na področju semantičnih spletnih storitev je večja stopnja avtomatizacije. Z večjo stopnjo avtomatizacije bi tako razbremenili uporabnike ter pohitrili izvajanje nekaterih poslovnih procesov.

Trenutno z WSDL ne moremo opisati nefunkcionalnih lastnosti spletne storitve, običajen pristop nudi samo neformalne opise funkcionalnosti spletne storitve, uporabnik pridobi želeni oglas storitve z iskanjem ključnih besed po UDDI imeniku ali glede na kategorijo storitve, omejena je izrazna moč klasifikacijskih shem (NAICS, UNSPSC, GEO,…), UDDI imenik ne podpira kompozicije storitev, prav tako ni mogoče avtomatično najti in obuditi storitev. To so nekatere izmed slabosti na katere naletimo pri iskanju storitve z uporabo WSDL in UDDI, ki pa bi jih z uporabo semantičnih spletnih storitev elegantno odpravili.

V zadnjih letih je bilo na področju semantičnih spletnih storitev vloženega veliko truda in denarja. Trenutno so semantične spletne storitve in tehnologije povezane z njimi v začetni fazi razvoja, vendar pa se stvari na tem področju odvijajo z veliko naglico. Skladno s povedanim so postopki odkrivanja semantičnih spletnih storitev vedno bolj kompleksni in izpopolnjeni.

Podal sem pregled najvidnejših orodij, ki se ukvarjajo z odkrivanjem semantičnih spletnih storitev in predstavil tehnologije semantičnega spleta, ki so neposredno povezane s spletnimi storitvami. Kljub hitremu razvoju, orodja trenutno še niso dovolj izpopolnjena za resnejšo uporabo. Vendar pa lahko pričakujemo, da se bo v prihodnosti to spremenilo. Hiter razvoj potrjuje tudi dejstvo, da se je v času zaključevanja diplomske naloge pojavilo nekaj novih orodij. Prav tako je bila objavljena nova verzija komponente WSMX discovery v0.5, ki omogoča iskanje na podlagi podrobnih semantičnih opisov, tako da orodja ne napredujejo le v smeri odpravljanja obstoječih napak.

V prihodnosti bo prav tako potrebno veliko pozornosti nameniti testnim primerom, ki so trenutno premalo kompleksni in tako premalo odražajo primere iz realnega sveta, da bi omogočali realne in bolj natančne primerjave med različnimi orodji in pristopi. Trenutno je omembe vredna edino zbirka *owls-tc2*, vendar pa uporablja preveč nerealne primere[29]. Z gotovostjo si upam trditi, da navadnega uporabnika (razen nekaterih izbrancev širom sveta) ne bo zanimala naslednja storitev: *government_funding_Missileservice.owls* ali pa *Government BallisticMissile Lending Service.owls*[30].

Zavedati se moramo, da je odkrivanje v veliki meri odvisno tudi od ponudnika oz. uporabnika spletnih storitev. Če je opis zahtevane ali ponujene spletne storitve slab, nikakor ne moremo pričakovati dobrih rezultatov. Tako se zahtevnost opisa spletne storitve stopnjuje, ko se pomikamo iz sintaktične ravni proti obogateni semantični ravni. Tako moramo skleniti razumen kompromis med tem, ali se bomo zadovoljili s slabšimi rezultati iskanja storitve in bomo svoje želje prilagodili najdeni spletni storitvi, ki delno ustreza našim zahtevam (v takem primeru bomo mogoče morali spremenili VI atribute, morda celo funkcionalnost), ali pa bomo vložili več truda v opis želene (oz. ponujene) storitve in zaradi tega dodatni posegi v (aplikacijo, ki bo uporabljala to storitev) ne bodo potrebni.

[29] Sicer ni nujno, da samo z realnimi primeri lahko izpostavimo prednosti ali slabosti pristopov, vendar pa uporaba realnih primerov utrdi zaupanje.
[30] http://hnsp.inf-bb.uni-jena.de/opossum/index.php?action=listservices&serviceid=5350

Seznam uporabljenih virov

[1] D. Martin, J. Domingue, "Semantic Web Services, Part 1," *IEEE Intelligent Systems*, zv. 22, št. 5, str. 12-17, 2007. Dostopno na: http://www.computer.org/portal/cms_docs_intelligent/intelligent/homepage/2007/x507/x5012.pdf

[2] D. Fensel, H. Lausen**,** A. Polleres, J.d. Bruijn, M. Stollberg, D. Roman, J. Domingue, *Enabling Semantic Web Services: The Web Service Modeling Ontology*, Springer, 2007.

[3] D. Lavbič, »Semantične Spletne Storitve,« *13. posvetovanje Dnevi slovenske informatike 2006* Portorož, Slovenija, apr. 2006. Dostopno na: http://amor.fri.uni-lj.si/dejan/delo/bibliografija/(2006,%20DSI)%20Semanticne%20spletne%20storitve%20(Dejan%20Lavbic).pdf

[4] A. Pivk, »Avtomatska gradnja ontologij iz spletnih tabel,« Doktorska disertacija, Univerza v Mariboru, Fakulteta za organizacijske vede, Kranj, 2005. Dostopno na http://dis.ijs.si/sandi/docs/phd.pdf.

[5] V.R. Benjamins, J. Contreras, O. Corcho, A. Gómez-Pérez, »Six Challenges for the Semantic Web,« *KR2002 Workshop on Semantic Web,* Toulouse, France, apr. 2002. Dostopno na: http://www.cs.man.ac.uk/~ocorcho/documents/KRR2002WS_BenjaminsEtAl.pdf

[6] D. Lavbič, M. Krisper, »Semantika podatkov in ontologije,« Uporabna informatika, št. 3, letnik XIV, 2006.

[7] J. Cardoso, *Semantic Web Services: Theory, tools and applications*, IGI Global, 2007.

[8] T. Gruber, »Ontology,«, 2007. Dostopno na: http://tomgruber.org/writing/ontology-definition-2007.htm

[9] L.H. Vu, S. Gerlach, F. Porto, O. Tajmouati, M. Hauswirth, »QoS-enabled Service Discovery Component Prototype 1 Report,« 2006. Dostopno na: http://lsirpeople.epfl.ch/lhvu/download/qosdisc/doc/D4.18-PrototypeReport.pdf.

[10] José-Manuel López-Cobo, Alejandro López-Pérez, James Scicluna, "A Semantic Choreography-driven Frequent Flyer Program," v zborniku *Future Research and Challenges for Software and Services Workshop,*Vienna, Austria, apr. 2006, str. 6-7.

[11] D. Lavbič, M. Bajec, M. Krisper, »Pravila na semantičnem spletu,« Elektrotehniški vestnik, št. 5, letnik 73, 2006.

[12] (2005) D16.1v021 The Web Service Modeling Language WSML. Dostopno na: http://www.wsmo.org/TR/d16/d16.1/v0.21/20051005/

[13] J. Cardoso, "The Semantic Web Vision: Where are We?," IEEE Intelligent Systems, št. 5, zv. 22, str. 22-26, 2007.

[14] U. Küster, H. Lausen, B. König-Ries, »Evaluation of Semantic Service Discovery - A Survey and Directions for Future Research,« v zborniku *Post-Proceedings of the 2nd Workshop on Emerging Web Services Technology (WEWST07) in conjunction with the 5th IEEE European Conference on Web Services (ECOWS07)*, Halle, Germany, November 2007, str. 37-53.

[15] M. C. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. Mühl, K. Geihs, *Ranked Matching for Service Descriptions Using OWL-S*, Springer, 2005, str. 91-102.

[16] (2004) OWL-S: Semantic Markup for Web Services. Dostopno na: http://www.w3.org/Submission/OWL-S

[17] M. Klusch, B. Fries, K. Sycara, »Automated Semantic Web Service Discovery with OWLS-MX,« v zborniku *5th International Conference on Autonomous Agents and Multi-Agent Systems*, Hakodate, Japan, 2006, str. 915-922. Dostopno na: http://www-ags.dfki.uni-sb.de/~klusch/owls-mx/owlsmx-06-final.pdf.

[18] (2008) Ontology Web Language for Services (OWL-S). Dostopno na: http://www.sei.cmu.edu/isis/guide/technologies/owl-s.htm

[19] (2005) D29v0.1 WSMO Mediators. Dostopno na: http://www.wsmo.org/TR/d29/

[20] (2005) Web Service Semantics – WSDL-S. Dostopno na: http://www.w3.org/Submission/WSDL-S/

[21] (2005) FLOWS: A First-Order Logic Ontology for Web Services, str. 15-16. Dostopno na: http://www.w3.org/2005/04/FSWS/Submissions/59/Flows.pdf

[22] L. Li, I. Horrocks, »A software framework for matchmaking based on Semantic Web technology,« v zborniku *International Journal of Electronic Commerce*, št. 4, zv. 8, str. 39-60, 2004.

[23] M. Paolucci, T. Kawamura, T.R. Payne, K.P. Sycara, »Semantic matching of Web services capabilities,« v zborniku *First International Semantic Web Conference on the Semantic Web*, Sardinia, Italy, 2002, str. 333-347.

[24] J. Gonzales-Castillo, D. Trastour, C. Bartolini, »Description logics for matchmaking of services,« v zborniku *KI-2001 Workshop on Applications of Description Logics*, Vienna, Austria, 2001, str. 74-85.

[25] A. Bratko, »Hierarhično razvrščanje elektronske pošte z metodami strojnega učenja,« Diplomsko delo, Univerza v Ljubljani, FRI, Trzin, 2003, pogl. 2.4.2. Dostopno na: http://magix.fri.uni-lj.si/blaz/diplomske/diplome/bratko.pdf

[26] B. Jerko, »Samodejno indeksiranje povzetkov, « v zborniku *7. mednarodne multikonference Informacijska družba IS 2004*, Ljubljana, Slovenija, okt. 2004, str. 13-18. Dostopno na: http://www.mf.uni-lj.si/ibmi-english/raziskovanje/tagger.pdf.

[27] X. Wang, T. Vitvar, M. Kerrigan, I. Toma, »A QoS-Aware Selection Model for Semantic Web Services,« v zborniku *International Conference on Service Oriented Computing*, Chicago, USA, 2006, str.390-401. Dostopno na: www.vitvar.com/doc/ICSOC2006-WangVKT.pdf

[28] J. Cardoso, J. A. Miller, »Discovering Semantic Web Services using the Tversky Model with a Case. Study based upon Semantically Annotated WSDL,« 2007. Dostopno na: http://www.cs.uga.edu/~jam/home/theses/cardoso_dissert/final/Tverskymodel-6-Sep-07c.pdf.

[29] D. Fišer, »Pristopi k izdelavi leksikalnih podatkovnih zbirk,« *Jezik in slovstvo*, št. 6, zv. 50, str. 17-32, 2005. Dostopno na: http://nl.ijs.si/slownet/bib/slown-JinS.pdf.

[30] R.D. Giv, B. Kalali, S. Zhang, N. Zhong, *Algorithms for direct and indirect dynamic matching of Web services*, Waterloo, Ontairo, Canada, 2004.

Izjava

Izjavljam, da sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marjana Krisperja. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Tomaž Pintar