UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Metod Južna

Pretvorba XML podatkov v OWL ontologijo

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marjan Krisper

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01518/2008
Datum: 15.10.2008

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **METOD JUŽNA**

Naslov: **PRETVORBA XML PODATKOV V OWL ONTOLOGIJO**
XML DATA CONVERSION INTO OWL ONTOLOGY

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V okviru diplomske naloge preglejte obstoječe pristope in orodja za preslikavo iz XML v OWL. Namen posameznih pristopov je zajem eksplicitnega znanja, skritega v XML dokumentih in njegov zapis v OWL ontologijo V nadaljevanju predstavite obstoječe pristope in jih primerjajte med seboj. Na podlagi te analize predlagajte izboljšave in dopolnitve pristopa k preslikavi. Pri tem primerjajte tudi preslikavo iz XML v RDF.

Mentor:

prof. dr. Marjan Krisper

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Franc Solina

ZAHVALA

Za pomoč in nasvete pri pisanju diplomske naloge se zahvaljujem izr. prof. dr. Marjanu Krisperju in asistentki Ani Šaši ter vsem ostalim, ki so kakorkoli pomagali pri izdelavi tega dela.
Ob dokončanju univerzitetnega študija se zahvaljujem družini za podporo in potrpežljivost v času študija.

Kazalo

Kazalo slik

Kazalo tabel

Uporabljene kratice in prevodi

Izraz	Prevod/Pomen
XML	Extensible Markup Language
OWL	Web Onlotogy Language
pretvornik	converter
stroj za sklepanje	reasoner
okvirni jezik	frame language
prvo-mestna logika	first-order logic
opisna logika	description logic
W3C	World Wide Web Consortium
DAMN	Darpa agent markup language
OIL	ontology inference layer
RDF	Resource Description Framework
lastnost objektov	object property
lastnost spremenljivke	datatype property
domena	domain
cilj	range
etiketa	tag
HTML	Hyper Text Markup Language
granularno popravljanje	granual updates
XSD	XML Schema Definition
XLST	Xtensible Stylesheet Language Transformations
XHTML	Extensible Hypertext Markap Language
PDF	Portable Document Format
XLST slog	XLST stylesheet
del nečesa	part-of
podtip nečesa	subtype-of
preslikava strukture	structure-mapping

Povzetek

Namen diplomske naloge je pregled obstoječih pristopov in orodij za preslikavo iz XML v OWL ter predlog moje ideje pristopa. Namen posameznih pristopov je zajem eksplicitnega znanja, skritega v XML dokumentih in njegov zapis v OWL ontologijo. V začetku dela je opisano področje teme, jeziki ter tehnologije povezane s tem. Osnovno znanje o teh pojmih je ključno za razumevanje teh pristopov. V nadaljevanju je vsak izmed pristopov natančneje opisan, pri čemer so navedena njegova pravila ter metode za preslikavo XML konstruktov v OWL koncepte, hkrati pa je podana dejanska kritika orodja. Poudarjene so dobre strani in slabosti. Pozornost je usmerjena na to, ali zna orodje samo ustvariti OWL ontologijo, ter kakšno obliko vhodne datoteke potrebuje (shema ali navadna XML datoteka), in seveda najpomembnejše: končni rezultat preslikave – tako sama ontologija kot tudi instance. Rezultat je ocenjen tudi glede na to, kako uspešen je nek pristop glede na že preizkušene. Nekatera pravila za preslikavo so dodatno ponazorjena s tem, da je prikazan rezultat, ki se ga dobi, če se uporabi to pravilo nad delom XML kode. Kot soroden pristop je omenjena preslikava iz XML-a v RDF. Med seboj se primerja RDF ter OWL. Na koncu so združeni vsi vtisi pristopov na enem samem mestu, na podlagi teh kritik pa je podan predlog, kakšen bi bil po mojem mnenju najboljši pristop za preslikavo, s katerim bi se izognili največjim pomanjkljivostim, ki so prisotne pri ostalih pristopih.

Ključne besede: preslikava iz XML v OWL, zajem eksplicitnega znanja, prenos znanja v semantični splet

Abstract

The purpose of the following thesis is to review current approaches and tools that serve as XML to OWL mapping. My idea of approach is also presented. The main goal of each approach is to capture explicit knowledge, hidden in XML document, and record it in OWL ontology. At the beginning of this thesis I describe object domain, languages and technologies related to it. Having basic knowledge about these concept is crucial for understanding of these approaches. The thesis continues with exact description of each approach. I describe its rules and methods for mapping XML constructs into OWL concepts. At the end of description I give my critique of the approach. I point out its advantages and disadvantages. I pay special attention if tool can create its own ontology and what kind of input file does it require (XML schema or just XML document) and of course, the most important: the final result of the mapping – ontology and its instances. I also pass my judgment by comparing each approach with others. Some mapping rules are further displayed by showing the result, witch is obtained if this rule is uses on XML code segment. As related approach, mapping from XML to RDF is mentioned. RDF with OWL languages are also compared. The thesis concludes with grouped impressions of approaches. On the basis of these, new approach is recommended, which would eliminate biggest disadvantages met during the reviews of other approaches.

Keywords: XML to OWL mapping, capturing of explicit knowladge, transfering knowladge into semantic web

1 Uvod

Na semantičnem spletu bodo programski agenti izvajali poizvedbe, iskali ter brskali med podatki, vse to za naše potrebe. Semantični splet omogoča računalnikom, da preiskujejo znanje porazdeljeno čez celoten splet, ga pridobijo in se nanj ustrezno odzovejo. Seveda najdemo na medmrežju ogromno znanja, a le to je izven dosega računalnikom. Na primer, določena enciklopedijska stran lahko človeškemu bralcu pove ogromno, a računalnik vidi le to, kako mora to stran prikazati na zaslonu. Datoteke skoraj vedno nastanejo z namenom, da predstavijo vsebino nekemu končnemu uporabniku. Resnična vsebina - znanje ki ga v sebi nosijo te datoteke so računalnikom popolnoma nevidne.

Z obstojem semantičnega spleta pa ni mišljeno, da bodo računalniki razumeli pomen česarkoli, ampak da se lahko logični podatki o pomenu stvari uporabijo s strani računalnika za pridobitev človeku uporabnejšim rezultatom.

Semantični splet dovoli programskim agentom razumeti in doumeti podatke, ki jih ponujajo spletne aplikacije. Na žalost, formalne ontologije, potrebne, da se izrazi podatkovna semantika, niso zlahka dostopne. Za izraznost semantike podatkov so potrebni formalni konceptualni modeli ali ontologije. A semantična informacija ni na voljo v takšni obliki, ampak je raztresena po celotni dokumentaciji in po različnih programskih komponentah. Ravno zaradi njihove formalne oblike predstavljajo podatki zapisani v XML-u široko zakladnico znanja, ki ga je le potrebno na nek način prenesti v obliko v kateri bi računalnikom kaj pomenilo. Z ontologijami je možno formalno predstaviti skupno domeno znanja, definiranega z koncepti, atributi, razmerji in instancami. Pojavila se je težnja, da se premosti vrzel med XML in ontologijo. Zaradi svoje razširjenosti, predvidenega napredka in označevalne sintakse pa se izkaže za zapis ontologije najprimerneje uporabiti ontološki jezik OWL. Jezik spletne ontologije ali tako imenovani OWL podpira zastopanost domenskega znanja, s pomočjo razredov, lastnosti in primerov (instancami) za uporabo v porazdeljenem okolju, kakršen je svetovni splet[10].

Zakaj sam XML ni dovolj? Ko se osebi nekaj pove, lahko združuje nova dejstva z že znanimi in na podlagi tega pove nekaj novega. Ko je sporočeno računalniku nekaj v XML-u, je ta morda v odziv sposoben sporočiti nekaj novega ampak le zato, ker nad podatki deluje programska oprema, ki ni del XML-a samega. Ta program je lahko implementiran drugače na drugih sistemih, kar pomeni, da lahko od različnih sistemov pričakujemo različne odzive. Če se računalniku pove nekaj novega v formalnem jeziku, kakršen je OWL, je zmožen podati nove informacije, v celoti osnovne nad OWL standardi.

Nabor OWL izrazov dovoljuje, da se iz njih pridobi nove OWL izraze, pri čemer se iz XML izrazov ne da priti do zaključka o novih XML izrazih. Da se v XML-u zapiše nove podatke, je potrebno znanje, ki mora biti nekje zapisano, in to ne le eksplicitno, kakor v OWL-u.
Na primer: Starševstvo je bolj splošna zveza kot materinstvo in izjava “Ana je Jankova mati” dovoljuje izpeljavo dejstva “Ana je Jankov starš”. Če uporabnik postavi vprašanje OWL iskalniku “Kdo so Jankovi starši?”, sistem odgovori, da je Ana eden izmed Jankovih staršev, četudi to dejstvo ni nikjer zapisano. Osnovano je le na logični definiciji, podani v OWL okolju, da je materinstvo podlastnost starševstva. Ista informacija, zapisana v XML-u ne dovoljuje izpeljave tega tretjega dejstva. Nekateri drugi jeziki, kakršen je RDF, sicer gredo korak dlje kot XML in bi lahko podprli zgornji primer, a OWL nudi nabor mnogih drugih lastnosti[1].

Vidi se, da je takšen način poizvedovanja uporaben na prav vsakem področju: v financah, kjer bi morda nekdo poizvedoval za vse bančne račune, ki so povezani z določeno osebo (pri

čemer sploh ni nujno, da ima ta oseba vse te račune v lasti), v logistiki, kjer nekdo povprašuje za stroške pošiljanja paketa v Srednjeevropsko mesto (pri čemer ta kategorija ni bila nikoli definirana, ampak je bil naveden le podatek za države Srednje Evrope). Imeti znanje, ki se dinamično uporablja za iskanje odgovora, je mnogo bolj napredno kot iskanje odgovora s posebnimi, vnaprej zapisanimi, procedurami.

Na žalost, tudi kadar uporabljamo XML za predstavitev podatkov, se pojavijo problemi, ko je potrebno integrirati različne vire podatkov, saj XML-u primanjkuje podpore za zadovoljivo skupno konceptualizacijo. XML odpove na skalabilnosti. Obstajata dva problema:

- Zaporedje v katerem se pojavijo elementi v XML dokumentu ni nepomembno in ima tudi svoj pomen. To je na pogled zelo nenaravno v svetu meta podatkov. Komu je mar, ali je, na primer, na prvem mestu ime ali priimek, pomembno je le, da sta oba dosegljiva.

- Vzdrževanje in popravljanje pravega zaporedja ogromne količine podatkov je v praksi težavno in predvsem zelo drago[8].

Mnogo bolj uporabno bi bilo imeti te podatke shranjene v obliki znanja – v ontologiji. Ker je razvijanje ontologije od samega začeta zelo težko in drago, bi bilo potrebno najprej v kar največji meri poizkusiti ponovno uporabiti semantične informacije, ki že obstajajo. XML sheme predstavljajo precej dobro osnovo za razvoj ali re-inženiring ontologije. Potrebno se je osredotočiti na eksplicitne semantične informacije, ki se skrivajo v teh dokumentih shem, in predlagati mehanizem, kako prenesti vse te podatke v formalno zapisano ontologijo.

Kljub podatkovni naravi XML-a so mnenja deljena. Nekateri avtorji, poznani v krogih semantičnega sveta, trdijo, da se v XML dokumentih resnično skriva nekaj semantike, ki se jo lahko odkrije v strukturi samega dokumenta, po drugi strani pa prav tako obstajajo imena, ki se s tem ne strinjajo. Pravijo, da XML predstavlja le strukturo dokumenta, vsebuje pa da ne nobenih informacij o pomenu njegove vsebine ali semantičnih omejitev.

Obstaja več strategij, kako lahko ustvarimo OWL ontologijo samodejno iz obstoječih XML podatkov. To se lahko ustvari s primerno preslikavo med različnimi podatkovnimi modeli elementov XML in OWL.

2 Predstavitev okolja, jezika in tehnologij

2.1 Ontologija

Ontologija je filozofski nauk o "bivajočem kot bivajočem", oziroma o temeljih in najsplošnejših lastnostih stvarnosti. Ontologija iz filozofije obravnava obstoj resničnosti na splošno, kot tudi osnovnih kategorij "bivajočega" in njihove medsebojne povezanosti. Tradicionalno uvrščena kot del večje veje filozofije poznane kot metafizika, se ontologija ukvarja z vprašanjem v zvezi katere entitete obstajajo, oziroma za katere se lahko reče, da obstajajo in kako se lahko te entitete združujejo, kako so povezane v hierarhijo in razdeljene glede na podobnosti in razlike. Mišljena je kot najsplošnejša filozofska teorija o najsplošnejših strukturah sveta, o temeljnih pojmih in kategorijah ter kot izhodišče za osnovna metodološka načela vseh znanosti.

Poznamo več vrst ontologij, ene izmed njih so ontologije, ki se uporabljajo v informacijski tehnologijah. Uporabljene so na različne načine in na različnih področjih. Srečamo jih, na primer, pri umetnih inteligencah, semantičnih omrežjih, biomedicinski informatiki, informacijski arhitekturi in tako dalje. Poznamo tudi ontologije za posebna področja, oziroma vrhnje ontologije[11] .

2.1.1 Ontologija v računalništvu

V informatiki so ontologije formalna predstavitev množice konceptov z nekega področja in razmerij med njimi. Ontologije tako rekoč združujejo zbirko objektov in pravil, ki so sestavljene tako, da se lahko združujejo. Pomembno je vedeti, da objekti niso zgolj besede s katerimi jih označujemo, temveč koncepti. Ontologije uporabljamo za predstavitev človeškega znanja in sklepanje. Z ontologijo zajamemo znanje o določeni domeni[12]. Ontologija vsebuje:

- Objekte, razrede, atribute in relacije
- Omejitve, pravila in aksiome

2.1.2 Ontološki jezik

Za pisanje ontologij potrebujemo posebna orodja in jezike. Računalniško razumljive ontologije morajo biti zastopane v logičnem jeziku, zato so v računalništvu ontološki jeziki formalni jeziki, ki se uporabljajo za konstrukcijo ontologij. Dovoljujejo kodiranje znanja o specifični domeni in večkrat vsebujejo pravila za sklepanje, ki podpirajo procesiranje tega znanja. Ontološki jeziki so po navadi deklarativni jeziki in so skoraj vedno generalizacija okvirnih jezikov in so najpogosteje osnovani na prvo-mestni logiki ali opisni logiki.

Različni ontološki jeziki prinašajo različne poglede. Seveda je treba vedeti, da so jeziki sami le sredstvo za izražanje vsebine, sami po sebi pa so brez informativne vsebine. To, na

primer, lahko ponazorimo s tem, kako se jezik slovenščina nanaša na podatke, izražene v slovenščini. Gre za besede s pomočjo katerih prenašamo podatke, ki pravzaprav narekujejo, katere besede bomo uporabili in v kakšnem zaporedju. Ni jezik (ali logika) ta, ki vse postori, ampak način kako ga uporabljamo. Ontologija je eden od načinov za uporabo jezika in logike bolj učinkovito.

Po mnenju mnogih je najprimernejši ontološki jezik, hkrati pa je tudi najbolj razširjen, OWL, ki priporočen tudi s strani konzorcija W3C. Ne smemo pozabiti da je jezik le logika, in da je na nas, da z njim nekaj naredimo in sami zagotavljamo pravilnost naših trditev izraženih s tem jezikom. Poleg OWL, obstajajo tu še DAMN in OIL, RDF ter mnogo drugih.

2.2 OWL

OWL, ontološki spletni jezik, predstavlja nov pristop k lažjemu in predvsem bolj smiselnemu iskanju poizvedb in povzetkov na določeno iskano tematiko. S tem jezikom bodo možne tudi aplikacije znotraj obsežnih intranetov in zelo obsežnih podatkovnih baz. Predpogoje za njegov razvoj sta omogočila XML in RDF. Temeljni cilj jezika OWL je, da bo iz neke zelo splošne, v stavku napisane, poizvedbe izpisal smiseln povzetek našega iskanja. To je verjetno še ena od priložnosti za izboljšanje sodelovanja med gospodarstvom, univerzami in posameznimi inštituti.

OWL je semantični označevalni jezik za objavljanje in izmenjavo ontologije na svetovnem spletu. Razvil se je, in se še razvija, kot slovarski podaljšek RDF-ja in je izveden iz DAML in OIL spletnih ontoloških jezikov. Razvoj se je začel leta 2000. Jezik OWL je zasnovan za uporabo v aplikacijah, ki naj procesirajo vsebino dokumentov, ne pa da le predstavijo informacije uporabniku. OWL omogoča večjo strojno prevedljivost spletnih vsebin kot zgolj XML, RDF, in RDF shema (RDF-S) s tem, da nas oskrbi z dodatnim slovarjem formalne semantike.

Ontologijo, izraženo v OWL jeziku, lahko predstavimo na več načinov – preko različnih kazal in uporabniškega vmesnika (program Protege – OWL) ali pa kar kot XML dokument, saj ima tovrstno sintakso[13].

OWL ima tri podjezike

- OWL-Lite
- OWL-DL
- OWL-Full

2.2.1 Izraznost podjezikov

Značilnost vsakega podjezika je njegova izraznost. Owl-Lite je najmanj izrazen podjezik. Owl-Full je najbolj izrazen podjezik. OWL-DL leži izrazno med OWL-Lite in OWL-Full. To, da

obstajajo trije jeziki, je neposreden rezultat poizkusa zadovoljitve velikega števila včasih konfliktnih si vplivov in zahtev.

2.2.1.1 OWL-lite

OWL-lite podpira uporabnike, ki v prvi vrsti potrebujejo neko hierarhično klasifikacijo in enostavne povzetke. Je sintaktično najbolj preprost podjezik. Uporabljen naj bi bil v situacijah, kjer se potrebuje le enostavna hierarhija razredov, uporabljajo pa se le preproste omejitve.

2.2.1.2 OWL DL

OWL-DL, pri čemer kratica DL predstavlja opisno logiko, je veliko bolj izrazen kot OWL-Lite. Kot tak je primeren za uporabnike, ki potrebujejo veliko izraznost, kjer pa ni toliko pomembna izračunljivost in odvisnost .Vse stvari se dajo in morajo procesirati v realnem času. Nad njim je zato možno izvajati avtomatsko sklepanje.

2.2.1.3 OWL Full

OWL-Full je namenjen uporabnikom, ki želijo maksimalno izraznost in sintaktično svobodo RDF-ja. V OWL Full je, recimo, razred lahko obravnavan kot zbirka posameznikov ali pa kot posameznik s svojimi pravicami in lastnostmi. Za zdaj še ni verjetno, da bi kakšno programsko orodje lahko podpiralo kompleten smisel za vsako besedo OWL Full. Ta podjezik naj bi bil uporaben v situacijah, kjer je zelo visoka izraznost bolj pomembna kot odločljivost ali izračunljivost celovitosti jezika. Avtomatskega sklepanja ni možno izvajati nad OWL-Full ontologijo[9].

2.2.2 Komponente OWL ontologije

OWL ontologija je skoraj v celoti določena z naslednjimi koncepti:

2.2.2.1 Razredi in instance

OWL razredi so interpretirani kot skupine, ki vsebujejo posameznike. Opisani so z uporabo formalnih (matematičnih) opisov, ki natančno sporočajo zahteve za članstvo v posameznem razredu. Na primer, razred `Mačka` bo vseboval vse posameznike, ki so mačke, seveda v okviru domene, ki nas zanima. Razredi so lahko organizirani v nadrazredno ozirom podrazredno hierarhijo, ki je poznana tudi kot taksonomija. Podrazredi specializirajo njihove nadrazrede. Na primer, razred `Mačka` je lahko podrazred razreda `Žival`. To pomeni tudi da so vse mačke živali, saj so vsi člani razreda mačka člani razreda žival. V OWL-u so razredi zgrajeni iz opisov, ki natančno opredelijo pogoje ki morajo biti zadoščeni za vsakega posameznika, da se smatra za člana tega razreda. Pogoji se nanašajo na to, katere in koliko lastnosti mora imeti posameznik oziroma v kakšni zvezi mora biti z ostalimi posamezniki. En posameznik lahko pripada večim razredom. Posamezni predstavniki razredov so instance, ki predstavljajo objekte v domeni, ki nas zanima.

OWL razred zgleda v XML sintaksi tako:

```
Dekleracija
<owl:Class rdf:about="#ImeRazreda">
      Uvrstitev v hiearhijo
      <rdfs:subClassOf rdf:resource="#ImeNadrazreda"/>
      ...
      Omejitve, v obliki nadrazreda
      <rdfs:subClassOf>
            Ena izmed omejitev
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#imeLastnosti"/>
                  Primer omejitve stevnosti za lastnost
                  <owl:minCardinality
                  rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality
                  >
            <owl:Class rdf:resource="&xsd;string"/>
            </owl:Restriction>
            ...
      </rdfs:subClassOf>
</owl:Class>
```

2.2.2.2 Lastnosti

Obstajata dve glavni vrsti lastnosti:

Lastnost objektov

Lastnosti objektov so binarne povezave med posamezniki – torej lastnost poveže med seboj dva objekta (instanci). Na primer, lastnost `jeVSorodu` lahko poveže `Janeza` in `Metko`. Lastnosti objetov so lahko tranzitivne ali simetrične. Za vsako lastnost je potrebno določiti domeno, torej razred, ki ima to lastnost in cilj, razred, na katerega se lastnost objektov nanaša. Večkrat ima posamezna lastnost njej inverzno lasnost, ki ima zamenjana cilj in domeno.

Lastnost objektov zgleda v XML sintaksi tako:

```
Deklaracija
<owl:ObjectProperty rdf:about="#imeLastnostiObjektov">
      <rdfs:domain rdf:resource="#Razred-cilj"/>
      Cilj
      <rdfs:range rdf:resource="#Razred-domena"/>
      Domena
</owl:ObjectProperty>
```

Lastnost spremenljivke

Lastnosti so lahko omejene tudi na eno samo vrednosti – so funkcionalne. Opisujejo zvezo med instanco in posamezno podatkovno vrednostjo. Na primer: `Janez jeStar "26"`. Za vsako lastnost spremenljivk je potrebno določiti domeno, torej razred, ki ima to lastnost, in tip podatkovne vrednosti.

Lastnost objektov zgleda v XML sintaksi tako:

```
Deklaracija
<owl:DatatypeProperty rdf:about="#imeLastnostiSpremenljivke">
      Domena
      <rdfs:domain rdf:resource="#Razred-Domena"/>
      Tip
      <rdfs:range rdf:resource="&xsd;tipSpremenljivke"/>
</owl:DatatypeProperty>
```

Obe vrsti lastnosti sta lahko urejeni hiearhično. Primer: lastnost objekotov `jeMati` izhaja iz lastnosti `jeStarš`.

2.2.3 Sklepanje

Ena od glavnih lastnosti ontologij, opisanih z uporabo jezika OWL-DL, je, da se jih lahko procesira s pomočjo stroja za sklepanje. Stroj za sklepanje je del programske opreme, zmožen sklepati logične zaključke iz niza dejstev in aksiomov.

Glavna funkcionalnost, izmed teh, ki jih stroj za sklepanje ponuja, je, da se preveri, če je en razred podrazred drugega. Ob zagonu takšnega testa nad vsemi razredi v ontologij je možno, da stroj za sklepanje izračuna logično razredno hierarhijo ontologije. Še ena izmed standardnih funkcionalnosti, ki jih stroj za sklepanje ponuja, je preverjanje konsistentnosti. Logični model dovoljuje uporabo stroja za sklepanje na način, da lahko preveri če so vse definicije in izjave v ontologiji medsebojno konsistentne in lahko tudi prepozna kateri koncepti spadajo pod posamezne definicije. Glede na opis (pogojev) razreda lahko stroj za sklepanje preveri ali je možno, da ima razred sploh lahko svoje instance. Razred se šteje kot nekonsistenten, če ne more imeti nobenega primerka. Stroj za sklepanje tako pomaga ohranjati pravilno razredno hierarhijo. To je še posebej uporabno, kadar delamo z primeri, kjer imajo razredi več kot le enega starša.

2.3 XML

XML je okrajšava za razširljiv označevalni jezik. XML je preprost računalniški jezik, podoben HTML-ju, ki nam omogoča oblikovanje formata za opisovanje strukturiranih podatkov ali arhitekturo za prenos podatkov in njihovo izmenjavo med več omrežji. XML je do sedaj spremenil in še vedno spreminja mnogo aspektov računalništva, še posebej na področju komuniciranja aplikacij in strežnikov. Da pa se ga tudi razširiti, saj ima namreč to možnost, da si lahko sami izmislimo imena etiket (< >). Zelo je uporaben za komunikacije, saj ima zelo preprosto in pregledno zgradbo. Razdeljen je na tri dele:

- Podatkovni: Vanj shranimo podatke v neki obliki z želenimi etiketami.
- Deklarativni: skrbi za to, da lahko pri dodajanju novih podatkov vidimo, kaj kakšna etiketa predstavlja.
- Predstavitveni: z njim oblikujemo izpis podatkov.

Razvijalci XML povečujejo vsebino tega jezika in s tem njegovih standardov tehnologije, ki vsebujejo podatke, ki se jih da enostavno preoblikovati in zamenjati v neenakih sistemih.

Obstaja več dobrih lastnosti uporabe jezika XML:

- XML razdeli podatke za lokalno obdelavo. Podatki so lahko brani v XML obliki, potem pa preneseni v lokalno aplikacijo, kot je na primer brskalnik za nadaljnje gledanje ali procesiranje. Podatki so lahko preneseni tudi skozi skript ali druge programske jezike s pomočjo XML objektnega modela.

- Uporabnikom da XML možnost primernega vpogleda v strukturirane podatke. Podatki preneseni na namizje so lahko predstavljeni v več možnih variantah. Lokalni podatki so lahko predstavljeni na takšen način, kot to najbolj ustreza uporabniku.

- Omogoča preproste ter logične vpoglede podatkov iz več virov. Običajno so bili uporabniki navajeni integrirati podatke iz strežniških baz in ostalih aplikacij na medmrežnih strežnikih tako, da so bili podatki uporabni za pošiljanje na ostale strežnike za nadaljnjo procesiranje, obdelavo in distribucijo.

- Opisuje podatke iz različnih aplikacij. Ker je XML obsežen jezik se lahko uporablja za opisovanje podatkov v široki variaciji aplikacij, od opisovanja kolekcij internetnih strani do podatkovnih zapisov. Ker so podatki samo-opisni, so lahko sprejeti in procesirani brez potrebe, da so še dodatno opisani.

- Omogoča boljši pretok skozi parcialno granularno popravljanje. Izvajalcem ni potrebno poslati celotnih strukturiranih podatkov vsakokrat, ko v njih pride do spremembe. Z granularnimi popravki se morajo distribuirati samo spremenjeni elementi, poslani od strežnika do klienta. Spremenjeni podatki so tako lahko predstavljeni brez ponovnega osveževanja celotne strani ali namizja.

XML dokument je instanca (primerek) posamezne XML sheme.

2.4 XML shema

XML shema se imenuje tudi XSD. Shema predstavlja "strukturo" in je dejanski dokument, podatki, ki so zastopani preko te sheme, pa se imenujejo instanca dokumenta. Kdor je, na primer, seznanjen z relacijskimi podatkovnimi bazami, si lahko predstavlja shemo kot tabelo, instanco dokumenta pa kot zapis v tej tabeli. Tisti, ki so seznanjeni z objektno orientirano

tehnologijo, pa si lahko preslikajo shemo v definicijo razreda ali instanco dokumenta v razred sam.

XSD določa sintakso in določa način, na kakršen so lahko zastopani elementi in atributi v XML dokumentu. Prav tako se zavzema za to, da morajo biti XML dokumenti v specifični obliki in točno določenih vrst podatkov. Ta opis se lahko uporablja ob preverjanju, če vsak del vsebine dokumenta ustreza opisu elementa, v katerega se vnaša vsebina. XSD je v celoti priporočen s strani W3C konzorcija kot standard za definiranje XML dokumentov. Na splošno velja, da je shema abstraktna predstavitev lastnosti objekta in njegov odnos do drugih objektov. XML shema predstavlja medsebojno delovanje med atributi in elementi XML predmeta (na primer dokument ali del dokumenta).

Kakšne so koristi v uporabi XSD sheme?

- XSD shema je že XML dokument, tako da ni dejanske potrebe po učenju nove sintakse.
- XSD shema podpira dedovanje, kjer je na primer ena shema dedovana iz druge sheme. To je odlična lastnost, saj zagotavlja priložnost za ponovno uporabnost.
- XSD shema določa možnost, da definiramo lastne vrste podatkov iz obstoječih podatkovnih tipov.
- XSD shema določa možnost, da opredelimo vrste podatkov za oboje: elemente in atribute.

2.5 RDF

RDF je bil ustvarjen leta 1999 kot dodatek k XML-u za kodiranje metapodatkov, kar bi dobesedno pomenilo podatki o podatkih. Metapodatki so, na primer, podatki o tem, kdo je avtor spletne strani ali katerega dne je bila objavljena neka novica. Gre torej za informacije, ki so v mnogih pogledih sekundarne glede na že nekatero ostalo vsebino, ki je na spletu. Po tem, ko je prišlo do RDF nadgradnje leta 2004 se je obseg le tega začel naglo širiti.

Sama ideja o nujni uporabi metapodatkov na spletu je bila od nekdaj zelo močna. Problem je bil edino v tem , da bi vsi začeli uporabljati iste, na enak način predstavljene, metapodatke. Alternativa je v uporabi lastnih poizvedovalnih ključev, vrednosti, programskih orodij in tako dalje. Metapodatki imajo precej skupnega, čeprav so različni. RDF - ali po slovensko “opisni okvir vira” predstavlja tak napor, da bi identificirali skupne niti in domislili ustrezno spletno arhitekturo. Na tak način bi bilo mogoče priskrbeti ustrezne metapodatke[15].

Semantični splet je decentralizirana platforma porazdeljenega znanja. RDF je standard konzorcija W3C za kodiranje znanja. Največja posebnost uporabe RDF-ja ni v kodiranju informacij, ampak informacij o relacijah med stvarmi v resničnem svetu, kot so ljudje, kraji, razni koncepti itd.

RDF je splošna metoda za dekompozicijo kakršnega koli znanja v manjše delce z nekaj pravili o semantiki oziroma pomenom teh delcev. Namen je, da imamo metodo, ki je tako preprosta, da lahko izrazi katero koli dejstvo, hkrati pa je vseeno tako strukturirana, da jo lahko računalniške aplikacije koristno uporabljajo[6]. Preprostost najlažje ponazorimo s primerom:

```
:Janez     :je          :Oseba.
:Janez     :jePorocen   :Ana.
:Janez     :imaOtroka   :Miha.
:Miha      :imaBrata    :Jure.
```

Vsak RDF je lahko definiran s tremi pravili:

- Vsako dejstvo je izraženo kot trojček oblike: subjekt, predikat, objekt. Rezultat zato zgleda kot navaden stavek v slovenskem jeziku.
  `:Janez  :je  :Oseba`

- Subjekti, predikati in objekti so imena entitet, konkretnih ali abstraktnih. Imena so lahko:
  - Globalna - nanašajo se na isto entiteto v katerem koli RDF dokumentu, kjer se pojavljajo – ta so vedno oblike URI (Uniform Resource Identifiers) npr: `http://www.primer.org/#Janez` To je skupno ime za imena in naslove, ki se nanašajo na objekte v World Wide Web-u. URL je ena izmed vrst URI-ja[23].
  - Lokalna - entiteta se ne more nanašati neposredno na entitete izven trenutnega RDF dokumenta.

- Objekti imajo lahko vrednost v obliki niza. To je še posebej pogosto, ko se RDF uporablja za meta-podatke. Npr:

  ```
  :<http://www.arnes.si/> :je :SpletnaStran
  ```

Standardna zgradba RDF-ja opisuje logične povezave med dejstvi in kako iskati dejstva v ogromni podatkovni bazi RDF znanja. Kar ga naredi primernega za porazdeljeno znanje je to, da RDF aplikacije povežejo skupaj RDF datoteke, objavljene s strani različnih internetnih posameznikov in se zlahka uči od njih, brez da bi imela posamezna datoteka prednost pred ostalimi. To počne na dva načina: prvič, da med seboj poveže dokumente s splošnim slovarjem, ki ga uporabljajo, in drugič, da dopusti dokumentom uporabo katerega koli slovarja. Ta fleksibilnost je unikatna[14].

Zgornji primer je napisan v stilu N3 (s tako imenovanimi trojčki), a za naše uporabo je veliko bolj uporaben XML zapis RDF-ja. Tukaj je XML zapis zgornjega primera:

```
<ns:Oseba rdf:about="http://www.primer.org/#Janez">
  <ns:jePorocen rdf:resource="http://www.primer.org/#Ana" />
  <ns:imaOtroka>
    <rdf:Description rdf:about="http://www.primer.org/#Miha">
      <ns:imaBrata rdf:resource="http://www.primer.org/#Jure" />
    </rdf:Description>
  </ns:imaOtorka>
</ns:Oseba>
```

Razlogi za uporabo RDF-ja:

- Dobro se obnese, če želimo integrirati podatke iz različnih virov, brez temu specifično namenjenega programiranja.

- Lažje ponudimo svoje podatke tretji stranki za ponovno uporabo.

- Lažje decentraliziramo podatke na način, da ni le ena oseba lastnik podatkov.

- Če želimo opravljati zahtevnejše posege nad veliko količino podatkov (brskanje, poizvedovanje, ujemanje…) lahko razvijemo generično orodje, ki dovoli opravljati te operacije nad RDF podatkovnim modelom, ki pa ima to prednost, da ni vezan na lastniško podatkovno shranjevalno/predstavitveno tehnologijo, kakršna je recimo pri podatkovnih bazah[8].

2.6 XLST

Ker lahko OWL dokument prikažemo z XML sintakso, je najbolj logična preslikava iz XML v OWL s pomočjo XLST procesorja. XLST je jezik, osnovan na XML-ju, uporabljen za pretvorbo XML dokumentov v druge XML ali kakšne druge človeku berljive dokumente (HTML, navaden tekst). Originalni dokument se pri tem ne spreminja ampak se naredi nov dokument, osnovan na vsebini izvirnika. XLST se najpogosteje uporablja za pretvorbo podatkov med različnima XML shemama ali pa za pretvorbo podatkov v HTML oziroma XHTML dokumente za spletne strani, pri čemer se ustvarja dinamične spletne strani; lahko pa tudi v druge XML formate, ki se lahko naprej pretvorijo v PDF dokumente.

V procesu preoblikovanja XSLT uporablja Xpath, da se določijo deli izvirnega dokumenta, ki se morajo ujemati enemu ali več vnaprej določenim predlogam. Ko pride do ujemanja, XSLT pretvori ujemajoči se del izvirnega dokumenta v nov (ciljni) dokument[26].

2.6.1 Xpath

Xpath je jezik za določanje vozlišč v XML dokumentu. Dodatno se lahko Xpath uporablja tudi za računanje spremenljivk (nizov, številčnih ter logičnih vrednosti). Xpath je bil, tako kot mnogo tovrstnih jezikov, definiran s strani WC3.

Uporablja izraze v obliki poti, da se navigira po XML dokumentu. Ti izrazi so na las podobni onim, ki se uporabljajo pri delu z tradicionalnimi računalniškimi datotekami. Je glavni del XLST-ja in brez njega le-ta ne bi bil možen. Kako v osnovi Xpath deluje, je najbolje ponazoriti na primeru segmenta XML dokumenta:

```
<Druzina>
        <Otrok>
                <ImeOtroka>Miha</ImeOtroka>
                <Starost>13</Starost>
        </Otrok>
        <Otrok>
                <Ime>Jure</Ime>
```

```
            <Starost>15</Starost>
        </Otrok>
</Druzina>
```

Za izbiro vozlišč, na primer, vseh otrok lahko uporabimo izraz `Druzina/Otrok`, z izrazom `Druzina//Otrok` izberemo vse knjige, ki so potomec `Druzine`, ne glede na to, kje ležijo glede na element `Druzina` in tako dalje. Obstaja preko sto funkcij[25].

3 Obstoječa orodja in pristopi za preslikavo iz XML v OWL

XML, XLST, RDF, OWL ter druge tehnologije sem opisal, ker se uporabljajo v večini pristopov ali pa na njih temeljijo orodja za preslikavo. Obstoječe aplikacije za preslikavo so bile v večini narejene s strani posameznikov v okviru univerze in se uporabljajo v manjših projektih kot del bolj ali manj uspešnih poskusov vzpostavitve semantike v določeno okolje.

Na internetu je relativno veliko, predvsem teoretičnega, napisanega o tej temi, a hitro se v posameznih pristopih prepozna precejšno mero podobnosti. Tako želja po semantični obtežitvi podatkov, kot tudi namen avtomatiziranja preslikav podatkov v tovrstno okolje, še ni razširjeno do te mere, da bi imela večjo podporo s strani skupnosti. Zato tudi ne obstaja večjih forumskih diskusij na to temo, niti uporabnih ali nazornih primerov.

Zaradi neprofitne narave teh projektov je, na žalost, veliko spletnih strani zastarelih ter neosveženih. Obstajajo mrtve povezave do strani, kjer naj bi se nahajale dodatne informacije o temi ali celo orodje za preslikavo samo. Redke aplikacije za preslikavo spremlja pomanjkljiva uporabniška dokumentacija.

Orodja, ki so za posamezen pristop na voljo, sem tudi preizkusil. Najprej sem preveril, če resnično podpirajo vse, kar obljubljajo v dokumentaciji, nato pa tudi kvaliteto same preslikave glede na pričakovano in glede na ostala orodja. Izpostavil sem dobre in slabe lastnosti. Za ta namen sem ustvaril preprosto ontologijo ter XML datoteko s podatki, ki bi jih radi preslikali v to ontologijo in XML shemo te datoteke.

Pri orodjih, ki sama naredijo shemo iz obstoječega XML dokumenta, sem preveril, kakšna je ta shema v primerjavi z našo. Ta preizkus je zgolj obstranskega pomena, saj nas seveda najbolj zanima izgradnja nove OWL ontologije in preslikava njenih instanc. Pri tem sem primerjal rezultat, če jim shemo podamo ali pa dopustimo, da uporabijo pravkar ustvarjeno. Tako ontologijo kot njene instance sem primerjal z svojimi pričakovanimi, že prej ustvarjenimi. Posvetil sem se vprašanju, kako bi mogli podati XML dokument, oziroma, v kolikšni meri je potrebno spremeniti XML dokument, da pride do pravilne ali vsaj uporabnejše različice preslikave,če so te spremembe sploh potrebne.

3.1 Orodje XML2OWL

Je delo avtorjev Hannes Bohring in Sören Auer z univerze Leipzig. Orodje je razvito z XSLT-jem in transformira XML shemo v OWL ontologijo. To orodje gre korak dlje kot veliko drugih, saj je sposobno generirati svojo ontologijo kar iz XML instance ali sheme.

Zakaj je bolje pridobiti dokument OWL ontologije iz XML sheme in ne iz XML dokumenta? Odgovor je preprost. Shema predstavlja skupni imenovalec vseh svojih instanc. Če pri ustvarjanju OWL ontologije izhajamo le iz enega XML dokumenta, ta morda ne bo primerna za drug XML dokument, ki pa vseeno pripada isti shemi, kot model, iz katerega izhaja OWL ontologija. Problematični so predvsem elementi, ki jih shema predpisuje kot neobvezne. V tem primeru se pridobi ob preslikavi tudi nepopolno ontologijo. Poleg pomanjkljivih elementov lahko pride tudi do zapletov pri številnosti, saj se ob odsotnosti sheme ne da zagotoviti največje pojavitve elementov.

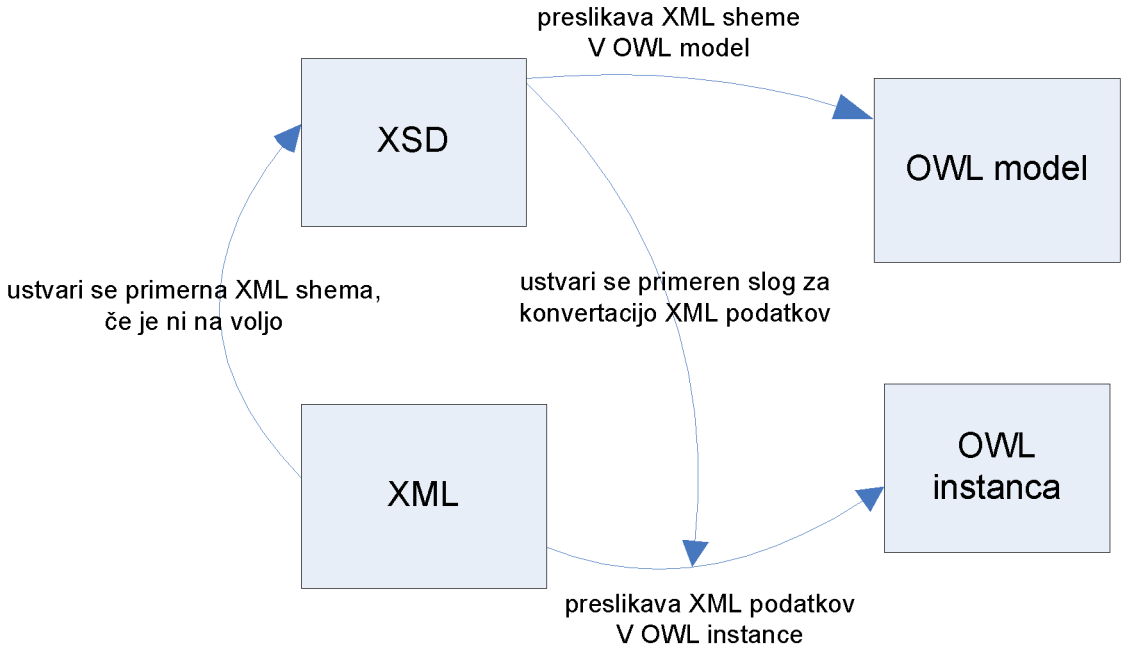
Preslikava med XML in OWL je implementirana v obliki večih XLST slogov. Za XML podatke, katerim ni predpisana nobena XML shema, se priredi primerna nadomestna XML

shema. Kadar je XML shema prisotna, se ustvari le model ontologije z razredi in lastnostmi. Če pa je na voljo le XML instanco, je potrebni narediti še en korak več.

V prvem koraku se pridobi XML shemo instance podatkov instance, da se lahko v drugem koraku naredi ontološki model. XLST slog, ki pretvori XML instanco v instanco ontologije je ustvarjen istočasno. Ta slog bo ustvarjen avtomatsko, da priredi proces spreminjanja instanc OWL modelu. Določi, kateri element postane razred ali lastnost. To je potrebno, saj so v XML instanci možni tudi opcijski elementi ali atributi, tako pa bo ustvarjeni XLST slog njihov skupni imenovalec.

Da sta podprta ločena model in podatki, je OWL model shranjen posebej od OWL instanc. OWL instance bodo vseeno povezane z modelom. Zato vsaka OWL instanca, ki je del določenega OWL modela, dobi imensko predpono.

Do tukaj implementacija vsebuje tri XLST sloge. Naslednji slog je narejen avtomatsko za pretvorbo XML instance v OWL instance. Ogrodje je narejeno tako, da se zlahka razširi in tako podpira, da se manjkajoče XSD komponente doda, prav tako pa je možno boljša integracija podpore za XML orientirane dokumente.

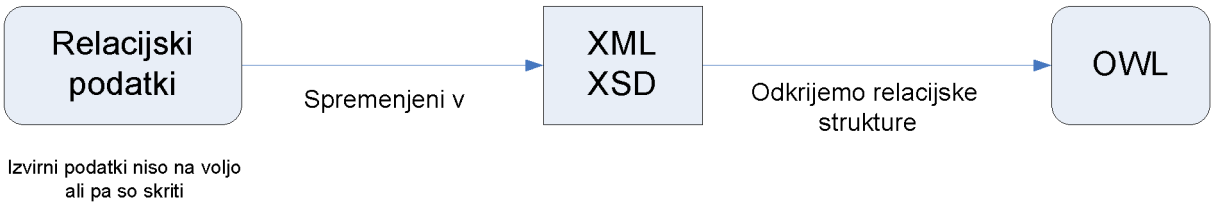


Slika 1: Potek sekvence aplikacije

OWL je semantično mnogo bolj izrazen, kakor pa od njega zahteva preslikava tega orodja. Transformacija je osnovana na hevristični metodi (hitro delujoča metoda, ki pride kar se da blizu najboljšemu možnemu odgovoru ali optimalni rešitvi) zato je možno, da optimalna rešitev ni možna. Prav iz tega razloga morajo procesu slediti poznejša ročna dela, ki bodo izboljšala in prilagodila ontologijo, da se prilega vsem zahtevam.

3.1.1 Preslikava

Recimo, da ima vsak XML dokument določene relacijske strukture. Te se poizkuša kar najbolje možno določiti in predstaviti preko OWL razredov, lastnosti in posameznih OWL instanc.



Slika 2: Podatkovni tok

Podatkovni model XML-ja opisuje posamezna vozlišča v drevesu, medtem ko je OWL podatkovni model osnovan na trojčku subjekt-predikat-objekt iz RDF-ja.

Vedno obstaja glavno vprašanje, kako se gleda na gnezdene etikete. Po eni strani se jih lahko dojema kot »del-nečesa« zvezo, lahko pa izražajo »podtip-nečesa« zvezo. Zaradi osredotočenja na podatkovno orientiran XML se lahko predpostavlja relacijsko strukturo in se za izboljšanje procesa transformacije uporabi to implicitno znanje o zgradbi izvirnega dokumenta.

Za gnezdene elemente je izbrana srednja pot. Za primere, kjer en element vsebuje drug element, ki ne vsebuje le vrednosti same, se predpostavja, da gre za »del-nečesa« zvezo, zato se lahko privzame 1:N povezava. To je preslikano v `owl:ObjectProperty`, ki vzpostavi odnos med dvema razredoma. A prav tako se lahko ustvari zveza »podtip-nečesa«, torej se med seboj povežejo `xsd:complexType` in od tam pridobljeni elementi. Tukaj obstaja tudi možnost večkratnega dedovanja (več kot ena domena).

Razredi (`owl:Class`) se bodo pojavili iz `xsd:complexType` in `xsd:Elements` glede na sledeča pravila: V primeru, da je element v XML drevesu vedno list, ki vsebuje samo vrednost in je brez atributov, bo ta element preslikan v `owl:DatatypeProperty`, ki bo imel domeno razred, ki predstavlja element. XML atributi pa bodo prav tako preslikani v `owl:DatatypeProperty`.

XML shema lahko prav tako vsebuje tudi različne omejitve, kot so `xsd:minOccurs` ali `xds:maxOccurs`, katere se preslikajo v ekvivalentne omejitve številnosti v OWL-u: `owl:minCardinality` in `owl:maxCardinality`. Tabela 1 povzame celotno zgoraj opisano preslikavo[3].

Iz primera XML kode

```
<Oce Ime="Janez">
      <Starost>52</starost>
      <Otrok Ime= »Marko«>
            <Starost>15</Starost>
      </Otrok>
</Oce>
```

torej dobimo OWL koncepte

```
<owl:Class rdf:about="#Oce">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
            <rdfs:subClassOf>
               <owl:Restriction>
                     <owl:onProperty rdf:resource="#PredponaO+Otrok"/>
                     <owl:onClass rdf:resource="#Thing"/>
                     <owl:minCardinality>0</owl:minCardinality>
               <owl:Restriction>
               <owl:Restriction>
                     <owl:onProperty rdf:resource="#predponaP+Starost"/>
                     <owl:onClass rdf:resource="&xsd:Literal"/>
                     <owl:minCardinality>0</owl:minCardinality>
                </owl:Restriction>
                <owl:Restriction>
                     <owl:onProperty rdf:resource="#predponaP+ime"/>
                     <owl:onClass rdf:resource="&xsd:Literal"/>
                     <owl:minCardinality>0</owl:minCardinality>
                </owl:Restriction>
            </rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:about="#Otrok">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
            <rdfs:subClassOf>
               <owl:Restriction>
                     <owl:onProperty rdf:resource="#predponaP+Starost"/>
                     <owl:onClass rdf:resource="&xsd:Literal"/>
                     <owl:minCardinality>0</owl:minCardinality>
                </owl:Restriction>
                <owl:Restriction>
                     <owl:onProperty rdf:resource="#predponaP+ime"/>
                     <owl:onClass rdf:resource="&xsd:Literal"/>
                     <owl:minCardinality>0</owl:minCardinality>
                </owl:Restriction>
            </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:about="#PredponaO+Otrok">
        <rdfs:range rdf:resource="#Oce"/>
        <rdfs:domain rdf:resource="#Otrok"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#PredponaP+ime">
            <rdfs:domain rdf:resource="#Oce"/>
            <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="# PredponaP+ime">
            <rdfs:domain rdf:resource="#Otrok"/>
            <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="# PredponaP+starost">
            <rdfs:domain rdf:resource="#OCe"/>
            <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:about="# PredponaP+starost">
            <rdfs:domain rdf:resource="#Otrok"/>
            <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

XSD	OWL
xsd:Elements, ki vsebujejo druge elemente ali pa imajo vsaj en atribut	owl:Class, povezan z owl:ObjectProperty
xsd:Elements, ki nimajo niti pod-elementov niti atributov	owl:DatatypeProperty
xsd:complexType	owl:Class
xsd:simpleType	owl:DataTypeProperty
xsd:minOccurs, xsd:maxOccurs	owl:minCardinality, owl:maxCardinality
xsd:sequence xsd:all	owl:intersectionOf
xsd:choice	kombinacija owl:intersectionOf, owl:union in owl:complementOf

Tabela 1: Preslikava XML2OWL je osnovana na teh korespondencah med elementi XML sheme in OWL razredi in lastnostmi

3.1.2 Uporaba

Orodje ni na voljo kot samostojna aplikacija, temveč ga je potrebno preizkusiti na spletni strani. Aplikacija za delovanje potrebuje le XML dokument, saj je sposobna shemo ustvariti kar sama. Ontologija ustvarjena z orodjem je avtomatično pridobljena iz podane sheme. Kot takšna lahko vsebuje le tiste semantične informacije, ki so bile že prej predstavljene v shemi – torej nekaj elementarnih razrednih definicij in omejitev ter lastnosti. Takšna precej preprosta ontologija, poznana tudi kot "lažje kategorna" ontologija ni zadostna za semantične aplikacije, ki zahtevajo prefinjeno sklepanje. Po drugi strani pa je potrebno omeniti, da je morda v nekaterih primerih zelo dobrodošlo pridobiti ontologijo, pa naj bo ta še tako siromašna. Če ta še ne obstaja, je to lahko zelo dobro izhodišče za ustvarjanje nove, čeprav je bolj v navadi, da se preslikuje podatke v že obstoječe ontologije. Pri tem pristopu vseeno ni možen vmesen poseg, s katerim bi se dalo tako pridobljeno ontologijo popraviti, saj pride do instanciranja takoj po pridobitvi OWL ontologije. To bi bilo možno, če bi bili na voljo posamezni segmenti orodja.

Preslikava naredi OWL razred za vsak tip elementa v vhodnem XML dokumentu, ki ni le list, torej ima atribute ali druge elemente. Selektivna preslikava za samo nekatere elemente ni možna. ID-ji ustvarjenih OWL instanc so sestavljeni iz sedem naključnih številk, kar je v praksi zelo nepregledno. Preslikava naredi lastnost spremenljivke za vsak atribut vsakega elementa v datoteki, kjer je domena razred, ki se nanaša na XML element, njegov tip pa je vedno `xsd:string`. Lastnost spremenljivke prevzame ime iz XML atributa, lahko pa se mu doda vsaj predpona. Če se OWL ontologija pridobi iz XML instance, se vseeno naredijo številske omejitve

nad razredom, le da je, kjerkoli se posamezna lastnost pojavi, njena najmanjša vrednost nič, s čimer se v bistvu ne pridobi veliko na semantiki. Vse omejitve podatkovnega tipa so tipa `Literal`, cilj objektnega tipa pa postane `Thing`, četudi bi lahko program navedel konkretni razred, na katerega se lastnost nanaša.

S preslikavo se ustvari lastnost objektov, kjer imajo elementi, ki se nanašajo na domeno in zalogo vrednosti zvezo starš-otrok. Vsaka starš-otrok zveza v XML datoteki je preslikana v lastnost objektov. Lastnost objekta dobi ime iz globalno določene predpone in imena ustreznega elementa, ki predstavlja otroka. Nadaljnja omejitev je to, da ni mogoče ustvariti lastnosti objektov med dvema instancama, če imata XML elementa, ki sta v zvezi prednik-potomec, kakršno koli drugo zvezo kot natančno starš-otrok.

3.2 TopBraid Composer

V skupino orodij, ki sama ponudijo ontologijo spada še dodatek za okolje Eclipsa TopBraid composer (ki se med drugim tudi uporablja za ustvarjanje OWL ontologij) imenovan XSDImport in Eclipse okolje TopBraidComposer Maestro Edition.

3.2.1 XSDImport

Je delo avtorja Michael Zimmermann-a. Kot že ime pove, program pričakuje kot vhodno datoteko le XSD shemo. XML-a niti ne potrebujemo, saj instanc ne podpira. O načinu kako posamezne XML konstrukte preslika ni napisano ničesar in se lahko o tem sklepa le iz nastalega končnega primerka. Obljublja preslikavo kompleksnih tipov, preprostih tipov, atributov in skupine atributov[19].

3.2.1.1 Preslikava in uporaba

Vse kompleksne tipe, pa tudi preproste z atributom, spremeni v razrede brez spicifičnega imena. Vsi postanejo `Class_1`, `Class_2`... Atribute prikaže kot lastnosti spremenljivke, a jim ne zna pripisati domene, četudi ve od kje jih vzame, saj hkrati naredi iz tega vozlišča OWL razred. Namesto tega prikaže na strani razrednih omejitev, da mora imeti razred eno spremenljivko (ime atributa) – omejitev števnosti. Kot razredno omejitev prikaže tudi tip te spremenljivke.

`<imeAtributa> max 1 Literal` - števnost
`<imeAtributa> only anySimpleType` – tip spremenljivke

Prav vsi podelementi kompleksnega tipa vozlišča, torej preprosti tipi, postanejo lastnosti objektov. Zopet brez domene in ciljne vrednosti in ponovno napisani kot razredne omejitve.

`predpona + <imePreprostegaTipa> only anyType` – oziroma - `only Class_1` - če privzame, da je podelement razred

`predpona +<imePreprostegaTipa> exactly 1 Thing` – številnost

Morda obstaja idealna struktura XML dokumenta nad katerim je bil ta program preizkušen, a nad večimi splošno znanimi XML primeri je dotična preslikava ustvarila podpovprečno ontologijo brez smiselnih povezav razredov in pomanjkljivimi lastnostmi.

Kot končni rezultat je pridobljena ontologija s slabo definiranimi lastnostmi, celim kupom razrednih omejitev, ki povzročajo le zmedo, ter slabo poimenovane razrede. S tako ontologijo je več dela, da se jo spravi v red, kot pa če se jo naredi "ročno", od začetka.

3.2.2 TopBraid Composer Maestro Edition

TobBraid composer Maestro Edition ima funkcijo za preslikovanje XML dokumentov v OWL ontologijo že vključeno v samo delovno okolje. Če se odpreme XML datoteko s tem programom, se OWL ontologija skupaj z instancami naredi avtomatsko. Ker je to orodje primarno namenjeno ustvarjanju OWL ontologije, lahko ustvarjeno spremenimo kar na licu mesta, brez kakršnekoli druge programske opreme.

Poleg tega, da orodje že samo pridobi instance, se lahko ročno popravi ontologijo ter ponovno uvozi instance iz istega izvirnega dokumenta, če dobljeni niso bili zadovoljivi, ali pa katerega koli drugega dokumenta, ki se nanaša na sorodne podatke[18].

3.2.2.1 Preslikava in uporaba

Popolnoma vsako vozlišče iz izvirne XML datoteke se preslika v svoj razred. Atributi tega vozlišča se preslikajo v lastnost spremenljivke z domeno pripadajočega elementa. Vrednost posameznega vozlišča pa se preslika v instanco označbe `genid#` tipa `TextNode` z lastnostjo spremenljivke `text`, ki ima vpisano vrednost. Veliko bolj logično je , da se tudi preprosti tipi preslikajo v lastnost spremenljivke posameznega razreda, vrednosti teh tipov pa v vrednost lastnosti posamezne instance, saj tako ne pride do prisotnih trivialnih instanc.

Instance so poimenovane na način `r-1-1-1-3`, pri čemer posamezne številke pomenijo globino v drevesu ter vrstni red v tej strukturi. Seveda se da v tem orodju vse popraviti ročno, a popravljanje imen pridobljenih instanc za veliko količino podatkov pač ne pride v poštev.

Lastnosti objektov obstaja le ena sama: starš-otrok imenovana `child`. Kjerkoli pride do take zveze med XML elementi bo instanca pridobila to lastnost. Na primer za primer

```
<Oce Ime="Janez">
      <Otrok>Miha</Otrok>
<Oce>
```

Instanca `r-1-1-1` (Oce Janez) pridobi lastnost objektov `child r-1-1-1-1` (Otrok Miha).

Ob uvozu XML podatkov se ustvarijo instance obstoječe ontologijo, če le imajo isto poimenovane etikete in istoimenske atribute. To je realizirano s tem, da dodajamo metapodatke

definiciji OWL razredom. Za ta namen sta ustvarjeni dve lastnosti: `sxml:attribute` in `sxml:element`. Lastnost element kaže na mesto v XML-u, ki se preslika v OWL razred. Podobno je za atribut. Na ta način lahko tudi rekonstruiramo XML strukturo[4].

3.3 Orodje JXML2OWL Mapper

Orodje JXML2OWL je delo avtorjev Toni Rudrigues ter Pedro Rosa. Namen tega programa je, da preslika elemente XML sheme v koncepte (razrede, lastnosti) ontologije. Potrebno je povedati, da JXML2OWL ne ustvari ontologije iz XML sheme. Ko uporabnik določi preslikavo, se ustvari XSLT slog, ki pretvori instance XML sheme v instance OWL ontologije.

Samo orodje je zelo podobno XML2OWL, a ima kar nekaj razlik. XML2OWL je orodje, ki ustvari novo ontologijo iz XML sheme, med ustvarjanjem le-te pa uporabnik nima prav nobene kontrole nad procesom. Tako podobno kot v XSD2OWL, zajame le implicitno semantiko, obstoječo znotraj strukture XML sheme. Zato je po navadi novo-nastala ontologija relativno primitivna – ni veliko semantično bogatejša kakor XML shema.

Cilj tega programa je malo drugačen. Ta pristop dovoli preslikavo XML sheme v že obstoječo ontologijo in temu primerno ustvari pravila, ki avtomatično pretvorijo instance XML sheme v instance preslikane ontologije. Rezultat je XLST datoteka preslikave oziroma datoteka OWL instance[17].

Res je, da je po navadi obstoječa (zunanja) ontologija semantično bogatejša kakor preslikana iz XML sheme, saj se lahko ontologija ustvari neodzivno od preslikanega XML vira podatkov, a dejstvo je, da že obstoječa ontologija v praksi ni vedno na voljo.

3.3.1 Preslikava

Orodje deluje na edinstven način, da preslika podan XML dokument v že obstoječo OWL ontologijo. Da pride do uspešne transformacije, potrebuje točno določene povezave, ki predstavljajo kaj se preslika v kaj. Uporabniku ponudi nabor vhodnih elementov (XML elementi) in zalogo vrednosti (OWL konceptov) na podlagi notacije, ki omogoča več različnih načinov preslikave: ena-v-ena, ena-v-mnogo, mnogo-v-ena in mnogo-v-mnogo. Predstavljena notacija torej omogoča preslikavo iz enega XML vozlišča v več OWL konceptov in preslikavo iz večih XML vozlišč v en sam OWL koncept.

Tako kot je OWL ontologija večinoma definirana z razredi, lastnosti objektov in lastnosti tipov podatkov, tako ločimo tri različne skupine preslikav:

- Preslikava v razrede: Preslika XML vozlišče v OWL razred.
- Preslikava v lastnosti spremenljivk: Preslika XML vozlišče v OWL lastnost spremenljivke.
- Preslikava v lastnosti objektov: Poveže med seboj dva preslikana razreda v OWL lastnost objektov.

Preslikava	Notacija
Razred	(OWL Razred URI, XPath izraz)
Lastnost tipa podatkov	(OWL Lastnost tipa podatkov URI, Preslikani razredi -domena, Xpath izraz)
Lastnost objekta	(OWL Lastnost objekta URI, Preslikasni razredi - domena , Preslikani razredi - cilj)

Tabela 2: Notacija preslikav

OWL koncepti se naslavljajo z uporabo njihovih URI referenc, medtem ko se XPath izrazi uporabljajo za naslavljanje preslikanih XML vozlišč. Uporaba XPath izrazov omogoča, da je možno razlikovanje med več različnimi XML vozlišči z istim imenom, a drugimi predniki in tako dovoljuje, da se jih preslika k njim pripadajočim OWL konceptom. XPath predikatov ni mogoče uporabljati in so možni le absolutni Xpath izrazi, kar onemogoča pogojno preslikavo.

Kot je vidno v tabeli 2, razredno preslikavo določa par, ki vsebuje URI referenco preslikanega OWL razreda in XPath izraz, ki identificira preslikavo XML vozlišče. Takšen par pomeni, da se bo naredila instanca OWL razreda za vsak XML vozlišče, ki se ujema z XPath izrazom. Kako so ustvarjene posamezne preslikave, najbolje prikaže sledeči primer: obstaja ontologijo z dvema OWL razredoma: `osebe:Oce` in `osebe:Otrok`. Na voljo ima lastnost objekta `osebe:imaOtroka` in obratna lastnost: `osebe:jeOtrok`. Na voljo so še sledeče `xsd:string` lastnosti tipov podatkov: `osebe:imeOtroka` in `osebe:imeOceta`

Preslikava se izvaja na primeru XML dokumenta:

```
<Druzinae>
      <Druzina>
            <Oce Ime="Janez"/>
            <Otrok Ime="Miha"/>
      </Druzina>
      <Druzina>
            <OCe Ime="Marko"/>
            <Otrok Ime="Ana"/>
      </Druzina>
</Druzine>
```

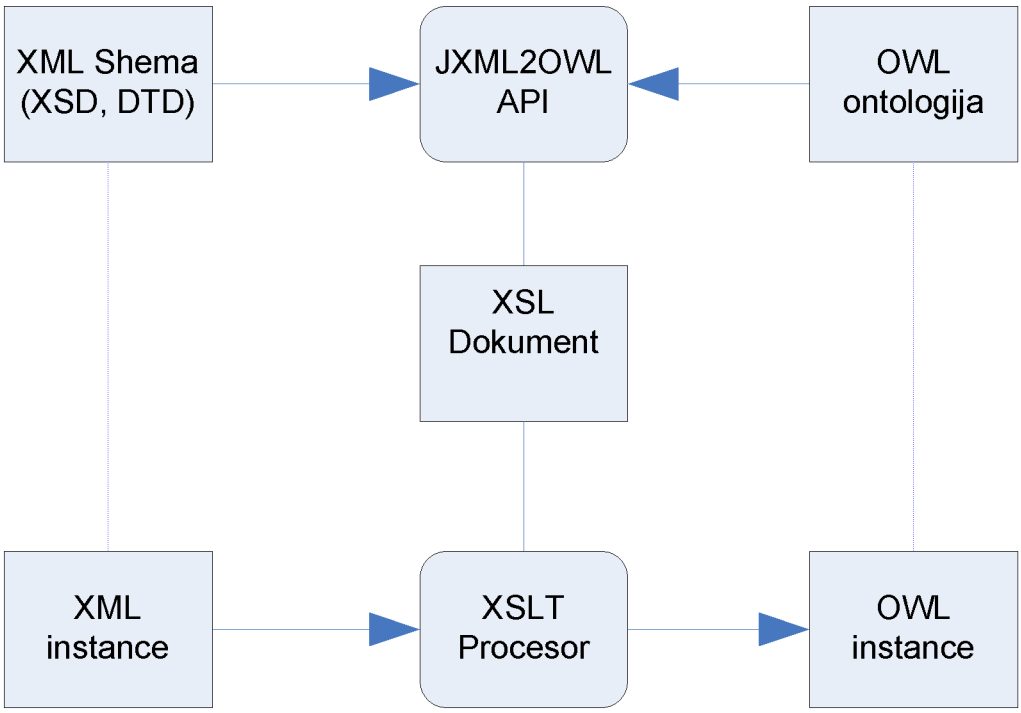
Glede na tabelo 2 so veljavne naslednje preslikave:

```
razredna preslikava
      rp1 = (osebe:OCe, /Druzine/Druzina/Oce)
      rp2 = (osebe:Otrok, /Druzine/Druzina/Otrok)
lastnost objekta - preslikava
      lop1 = (osebe:imaOtroka, rp1, rp2)
      lop2 = (osebe:jeOtrok, rp2, rp1)
lastnost tipa podatkov - preslikava
      ltpp3 = (osebe:imeOceta, rp1, /Druzine/Druzina/Oce/@Ime)
      ltpp4 = (osebe:imeOtroka, rp2, /Druzine/Druzina/Otrok/@Ime)
```

Pravilo preslikave `rp1` pomeni, da je ustvarjena instanca razreda `Oce` za vsako XML vozlišče, ki se ujema z XPath izrazom `/Druzine/Druzina/Oce`. V zgornjem primeru sta ustvarjena dva razreda za `Oce`, eden za Janeza in eden za Marka. Podobno sta ustvarjena dva razreda `Otrok`.

`lop1` preslikava pomeni, da je vsaka OWL instanca, ustvarjena iz razredne preslikave `rp1` domena lastnosti objekta `osebe:imaOtroka`, katerega ciljna vrednosti je posamezna instanca, ustvarjena z razredno preslikavo `rp2`. Ustvarjena so bila štiri razmerja ki med seboj povezujejo Janeza in Miho (in obratno) ter Marka z Ano.

`ltpp3` pomeni, da je za vsako instanco ustvarjeno iz razredne preslikave `rp1` prav tako ustvarjena lastnost tipa podatkov `osebe:imeOceta`. Ustvarjene so bile po štiri lastnosti tipov podatkov z imeni očetov in otrok[22].



Slika 3: Potek preslikave JXML2OWL

3.3.2 Uporaba

Za začetek preslikave je potrebno najprej naložiti želeno XML shemo ali pa primer XML dokumenta, ki se ga bo preslikalo ter OWL ontologijo. Pri tem je treba paziti, da obstoječa ontologija nima nobenih očitnih pomanjkljivosti, pa najsi bodo še tako trivialne kot je nedoločen tip lastnosti spremenljivke. V primeru, da je podan le XML dokument, norodje samo sestavi shemo, kakršna naj bi bila podlaga takšnemu dokumentu. Predlagana shema je v vseh pogledih identična originalni (z izjemo odsotnih podatkov), na žalost se sheme ne da shraniti v samostojen dokument, kar bi morda prišlo prav za kakšne druge namene, na primer pri kasnejši primerjavi.

Preko prijaznega uporabniškega vmesnika se povezuje elemente XML na levi strani in OWL koncepti na desni, kar vsaj v primeru preprostejših ontologij omogoča nadvse pregledno uporabo orodja. Vzpostavljenim razrednim povezavam se lahko pripisuje lastnosti spremenljivk in lastnosti objektov. Posebej dobrodošlo je, da za posamezno povezavo orodje samo ponudi možne tipe spremenljivke in lastnosti objektov, glede na to, v kateri razred OWL se je preslikal posamezen element, nato pa je potrebno le še označiti, na kateri element se nanaša posamezna lastnost.

Predstavitev preslikave pri zapletenejših in obsežnejših ontologijah (takšno stopnjo ontologije v resničnem življenju dosežejo precej hitro) postane nepregledna, zato je potrebno, da preslikavo opravlja oseba, ki je ontologijo ustvarjala v sodelovanju z osebo, ki ve veliko o izvirnih XML podatkih. Drugega dostopa do lastnosti spremenljivk in lastnosti objektov, kot preko povezav med razredi, ni, zato bi bilo dobrodošlo iskanje, kje se posamezna lastnost pojavlja, kar bi močno olajšalo delo z bolj zapletenimi ontologijami.

Poimenovanje instanc (ID) je omejeno na konkatenacijo globalne predpone, posameznega XML atributa oziroma vrednost podelementa od elementa preslikanega razreda, oziroma na vse vrednosti podelementov.

Če se poveža XML etiketa `Oce` z OWL razredom `Oce` z predhodno nastavljeno predpono `druzina`, in se to pravilo za preslikavo uporabi nad XML segmentom

```
<Oce Ime=”Janez”>
      <Ime>Janez</Ime>
      <Starost>52</Starost>
</Oce>
```

Se ustvari OWL instanco

```
<druzina:Oce rdf:ID="_druzinaOceJanez"/>
```

Vsekakor bi bila dobrodošla večja svoboda pri poimenovanju. Glede na unikatni identifikator, ki se ga izbere, orodje pregleduje, če se kje pojavljajo podvojene instance in jih na podlagi tega izničuje. Posamezne lastnosti večih istih instanc združi v eno samo, pri čemer jih v primeru iste vrednosti ne podvoji, v primeru polne in prazne pa vseeno zapiše tudi prazno, kar je nekoliko moteče.

Tako se iz XML segmenta

```
<Oce Ime="Janez">
      <Starost>52</starost>
      <Teža>88</Teža>
</Oce>
<Oce Ime="Janez">
      <Visina>172</Visina>
      <Starost></Starost>
      <Teža>88</Teža>
</Oce>
```

Ustvari OWL instanca

```
<druzina:Oce rdf:ID="_druzinaOceJanez">
```

```
        <druzina:Starost rdf:datatype="integer">52</druzina:Starost>
        <druzina:Starost rdf:datatype="integer"/>
        <druzina:Ime rdf:datatype="string">Janez</druzina:Ime>
        <druzina:Visina:datatype="integer">172</druzina:Visina>
        <druzina:Teža:datatype="integer">88</druzina:Teža>
 </druzina:Oce>
```

Dobro bi bilo, če bi aplikacija preverjala, če se že preslikana instanca pojavlja v razredu, nadrejenemu pravkar preslikani instanci, in le to izbriše. Recimo, da obstaja razredno hierarhija `Oseba/Druzinskiclan/Oce`. Z ontološkega vidika bi bilo popolnoma dovolj, da je instanca uvrščena le v en sam razred – `Oce`, tako pa lahko pride do primera, kjer obstaja ista instanca v treh izvedbah, kar povzroča le zmedo.

Zaradi velike svobode pri preslikovanju je dobro, da aplikacija opozarja na najbolj očitne morebitne napake, kakršne so neujemanje preslikave elementa tipa spremenljivke v XML dokumentu in tipa, ki ga pričakuje ontologija, ter podvojene lastnosti in številnost.

Tovrsten način dela seveda pride v poštev, kjer že obstaja vzpostavljena OWL ontologija, vanjo pa se vnaša podatke – dela se na trdni osnovi, kar naj bi pravzaprav lahko tudi bil glavni namen takšnega pristopa. Nujno je da se preslikava izvaja na končni in preverjeni ontologiji, saj najmanjše spremembe ontologije zahtevajo ponovno vzpostavitev vseh povezav. Prav zaradi povezovalnega načina ujemanja se lahko na pogled popolnoma neprimerne podatke uvrsti v ciljno ontologijo.

3.4 COMA++

Orodja, ki podpirajo preslikavo XML sheme v že obstoječe OWL ontologije, so relativno redka. Zato velja omeniti še COMA++, ki pa je na žalost le ujemevalno orodje. Je prilagodljivo in generično orodje za ujemanje obojega, tako shem in ontologij prikazanih v jezikih kot so SQL, XML ali OWL. Ima uporabniški vmesnik in podpira kombinirano uporabo različnih ujemevalnih algoritmov, ki sami vzpostavijo povezavo med posameznimi koncepti, možni pa so tudi ročni popravki. Kljub temu, da aplikacija ni bila narejena izključno za ujemanje ontologije, se ponaša z dobrimi rezultati[16]. Preizkušena je bila nad primeri na spletni strani, kjer se nahaja množica testov, s katerimi se ugotavlja stopnja pravilnosti ujemanja ontologij[2]. COMA++ lahko vzpostavi preslikavo med XML shemo v OWL, ampak ne z namenom, da bi olajšala preslikavo instance sheme v OWL instance, tako kot to počne JXML2OWL, zato je z našega vidika dokaj neuporabna.

3.5 Orodje XML2OWL s pravili

Orodje je delo avtorja Muhammada Awais Amina, osnovano na okolju *Another Tool for Language Recogniiton*. Napisano je v jeziku *C++* in namenjeni izključno avtomatski pretvorbi iz XML v OWL instance. Pravila za preslikavo morajo biti napisano v natančno določeni sintaktični obliki in shranjena v datoteki s končnico `.rules`. Rezultat preslikave je OWL datoteka, ki pa

vsebuje le podatke o instancah. Privzame se, da ontologija, ki specificira razrede in lastnosti, katere se vežejo na posamezne instance, že obstaja[20].

3.5.1 Preslikava

3.5.1.1 Preslikava XML elementov v OWL instance

Pravilo:
`IND(owl_razred,xml_element, "predpona_niz" + xml_element@atribut + "koncnica_niz")`

To pravilo se uporablja za ustvarjanje instanc v izhodni OWL datoteki. ID OWL instance vsebuje vrednost XML atributa. Za boljše razumevanje: atribut je vedno podan skupaj z elementom, ki ga vsebuje - torej `xml_element@atribut`. Za vsak XML element ki se imenuje `xml_element` se naredi instanca v izhodni datoteki. Predpona in končnica sta opcijski del pravila in se lahko uporabita, da spremenita ID ustvarjene instance (v OWL datoteki zapisano kot `rdf:ID`).

Pravilo `IND(Oce,Oce,Oce@Ime)` bo spremenilo segment XML

```
<Oce Ime =”Janez”>
      <Starost>52</Starost>
</Oce>
```

v OWL izjavo

```
<Oce rdf:ID="Janez"/>
```

Obstaja več variacij tega pravila.

Po pravilu `IND(owl_razred,xml_element,"identifikator")` se za vsak pojavljeni `xml_element` naredi OWL instanca, kjer je `rdf:ID` enak identifikatorju. Če pride do enakosti pri večih XML elementih, se doda identifikatorju številka z namenom da se naredi unikaten ID za vsako novo OWL instanco. To je še posebej uporabno, če ne želimo vključiti vrednost atributa kot del ID-ja

Pravilo `IND(Otrok,Otrok,”Potomec_”)` bo spremenilo XML segment

```
<Otrok Ime = “Metka” />
<Otrok Ime = “Janko” />
```

v OWL izjavo

```
<Otrok rdf:ID="Potomec_1"/>
```

```
<Otrok rdf:ID="Potomec_2"/>
```

Pretvornik lahko preslika XML element s tekstovno vsebino ali celo mešano vsebino (torej XML atribut in tekstovno vsebino) z vrednostjo ID-ja novo nastalih instanc glede na njihovo tekstovno vsebino. Ta preslikava poteka po pravilu

```
IND(owl_razred,xml_element, "predpona_niz" + xml_element@TEXT +
"koncnica_niz")
```

`owl_razred` je ime razreda instance ki se bo ustvarila. `xml_element` je ime elementa s tekstno ali mešano vsebino. rdf_ID-ji ustvarjenih instanc so osnovani na tekstni vsebini elementov `xml_element`-a. Atribut TEXT (`xml_element@TEXT`) se uporablja da loči to pravilo od zgornjega, kjer `rdf:ID` vsebuje vrednost XML atributa. Potrebno je vedeti, da je to pravilo primerno le za pretvorbo XML elementov, katerih tekst sestoji iz ene same besede, saj pretvornik ne preverja, če so prisotni presledki, kar pa pripelje v to, da OWL ID-ji niso pravilni.

Pravilo `IND(Oce,Oce,Oce@Ime)` bo tako spremenilo segment XML

```
<Oce>
      <Ime>Janez</Ime>
      <Starost>52</Starost>
</Oce>
```

v OWL izjavo

```
<Oce rdf:ID="Janez"/>
```

Orodje pozna tudi selektivno pretvorbo.

```
IND(ime_razreda,xml_element, "predpona_niz" + xml_element@atribut1 +
"koncnina_niz", xml_element@atribut2="atribut_vrednost")
```

Če vhodna XML datoteka vsebuje element imenovan xml_element z atributom imenovanim atribut2, in če ta atribut vsebuje vrednost definirano v nizu atribut_vrednost, potem se naredi OWL instanca in njegova ID vsebuje vrednost atrbuta1 XML elementa. Atributa 1 in 2 sta lahko tudi enaka.

Pravilo `IND(Moski,Oseba,Oseba@Spol,Oseba@Spol="m")` bo tako spremenilo segment XML

```
<Oseba spol="m">
      <Ime>Janez</Ime>
      <Starost>52</Starost>
</Oseba>
```

v OWL izjavo

```
<Oseba rdf:ID="m"/>
```

3.5.1.2 Preslikava objektnih lastnosti

OWL objektna lastnost poveže med seboj instanco izmed domenskih razredov in instanco, ki pripada razredu iz ciljnih vrednosti te lastnosti. Instance morajo biti na voljo že vnaprej (ali so ustvarjene s pretvornikom ali pa definirane v drugih OWL datotekah). Če je instanca narejena s pretvornikom, ima le-ta njen ID skupaj z imeni XML elementov in atributov, ki so bili uporabljeni za ustvarjanje te instance, shranjen v spominu. V tem primeru obstajata dve vrsti ukazov za preslikavo.

Prvi ukaz

```
OTP(ime_lastnosti,domena_razred,ciljV_razred)
```

To pravilo definira lastnost objekta imenovano `ime_lastnosti` med instancami. XML elementa, ki sovpadata z `domena_razred` in `ciljV_razred instancami`, morata v sami XML datoteki biti nujno urejena v zvezi prednik-potomec (torej gre za zvezo starš-otrok ali otrok-starš). Kot poseben primer lahko `domena_razred` in `ciljV_razred` sovpadata z istimi XML elementom.

Pravila

```
IND(Oce,Oce,Oce@Ime)
IND(Otrok,Otrok,Otrok@Ime)
OTP(imaOtroka,Oce,Otrok)
```

bodo tako spremenila segment XML

```
<Oce Ime=”Janez”>
      <Otrok Ime=”Metka” />
</Oce>
```

v OWL izjavo

```
<Oce rdf:ID="Janez"/>
<Otrok rdf:ID="Metka"/>
<Oce rdf:about="#Janez">
      <imaOtroka rdf:resource="#Metka"/>
</Oce>
```

Drugi ukaz

```
OTP(ime_lastnosti, domena_razred, ciljV_razred, domena_xml_element@atribut,
ciljV_xml_element@atribut)
```

Zopet pravilo definira objektno lastnost ime_lastnosti, med instancama, ki pripadata OWL razredom `domena_razred` in `ciljV_razred`. Tokrat XML elementa, ki sovpadata z instancami nista povezana med seboj kot prednik-potomec. Namesto tega je zveza med XML elementov vzpostavljena z nedvoumno vrednostjo atributa.

Pravila

```
IND(Oce,Oce,Oce@Ime)
IND(Otrok,Otrok,Otrok@Ime)
OTP(imaOtroka, Oce, Otrok, Oce@Ime, Otrok@Oce)
```

bodo tako spremenila segment XML

```
<Druzina>
      <Oce Ime ="Janez">
            <Starost>52</Starost>
      </Oce>
      <Mati Ime="Mojca">
            <Starost>50</Starost>
            <Otrok Oce="Janez" Ime="Metka">
                  <Starost>15</Starost>
            </Otrok>
      </Mati>
</Druzina>
```

v OWL izjavo

```
<Oce rdf:ID="Janez"/>
<Otrok rdf:ID="Metka"/>
<Oce rdf:about="#Janez">
      <imaOtroka rdf:resource="#Metka"/>
</Oce>
```

3.5.1.3 Preslikava lastnosti tipa podatkov

To pravilo se uporablja za definiranje lastnosti tipa podatkov za instance posameznega razreda. Podobno kot pri objektnih lastnostih, morajo udeležene instance biti ustvarjene pred določitvijo pravila. Podatek je lahko je lahko tipa float ali niz. Obstajata dva različna pristopa.

Prvi ukaz

```
DTP(ime_lastnosti, domena_razred, xml_element@atribut, tip)
```

V tem primeru dobi lastnosti tipa podatka vrednosti iz XML atributa definiranega v `xml_element@atribut`. Če obstaja primer, kjer xml_elemet v XML datoteki nima atributa, pretvornik vstavi "-0" za tip float oziroma znak '-' za nize.

Pravili

```
IND(Oce,Oce,Oce@Ime)
DTP(sePise, Oce, Oce@Priimek, String)
```

bosta tako spremenili segment XML

```
<Oce Ime ="Janez" Priimek="Kranjski"/>
```

v OWL izjavo

```
<Oce rdf:ID="Janez"/>
<Oce rdf:about="#Janez">
      <sePise rdf:datatype="&xsd;#String">Kranjski</sePise>
</Oce>
```

Drugi ukaz

```
DTP(ime_lastnosti, domena_razred, xml_element/otrok_element, tip)
```

V tem primeru je vrednost lastnosti tipa podatkov enaka vsebini XML tekst elementa, ki je definiran z Xpath izrazom `xml_element/otrok_element`. Če ima XML element več kot le enega otroka, ki se ujema z Xpath izrazom, potem pride do kombiniranja vsebine, ki je na koncu predstavljena kot enojna lastnost tipa podatkov, ki za ločitveni znak uporablja novo vrstico[7] .

Pravili

```
IND(Oce,Oce,Oce@Ime)
DTP(sePise, Oce, Oce@Priimek, String)
```

bosta tako spremenili segment XML

```
<Oce Ime ="Janez">
      <Priimek> Kranjski <Priimek>
</Oce>
```

v OWL izjavo

```
<Oce rdf:ID="Janez"/>
<Oce rdf:about="#Janez">
      <sePise rdf:datatype="&xsd;#String">Kranjski</sePise>
</Oce>
```

3.5.2 Uporaba orodja

Program potrebuje dva argumenta: definicijo pravil (`*.rules`), ki definira XML preslikavo v OWL in XML datoteko, ki vsebuje potrebne podatke za preslikavo. Rules datoteka se prevede preko prevajalnika, ki je avtomatično generiran v jezikovnem orodju ANTLR-u.

Za pravilno uporabo tega pretvornika je pomembno razumevanje gramatične strukture rules datoteke. Sintaksa preslikovnih pravil je sestavljena iz nizov, ki popolnoma definirajo na kakšen način se izvede določena akcija za to pravilo. V primeru napake v XML datoteki ali sintaktične napake se prikaže skopo opozorilo prevajalnika. Ta pristop seveda zahteva nekaj dodatnega truda a je mnogo bolj fleksibilen.

Ta pretvornik preslika XML podatke v že obstoječo ontologijo, zato ni nobene vsiljene omejitve semantične obogatenosti ciljne ontologije. Če je potrebno, se lahko podatki preslikajo v tako imenovano “težko kategorno” ontologijo, ki vsebuje kompleksne lastnosti, omejitve teh lastnosti, aksiome, ki jih lahko izkorišča sofisticirana programska oprema. Potrebno je vedeti, da

je orodje primerno za že obstoječo ontologijo, kljub temu da se ob sami preslikavi prav nič ne naslanja nanjo. Imena lastnosti ter razredov se lahko izbere in prav zaradi tega je potrebno veliko znanja o obstoječi OWL ontologiji, potrebna pa je jasna predstava o tem kako naj izgleda rezultat, saj se lahko v nasprotnem primeru pridobi instance, ki niso kompatibilne z ciljno ontologijo. Je pa res, da je v primeru želenih popravkov zaradi sprememb ontologije kar najmanj dela z vnosom novih sprememb.

Selektivna preslikava je nadvse dobrodošla, a kaj, ko je vezana na atribut. Na primer, kolikokrat je podan spol osebe kot atribut, če bi ga radi glede na to uvrstili v razred moški ali ženska.

Podano imamo

```
<Oseba>
      <Spol>m</Spol>
      <Ime> …
```

Potrebujemo

```
<Oseba spol="m">
      <Ime> …
```

Nasploh se veliko pravil nanaša samo na XML atribute, kar pomeni, da morajo ti biti na voljo v izvirnem XML dokumentu, kar pa seveda ni vedno res. Zato bi bilo dobro dodati marsikje še dodatno obliko pravila, ki dovoljuje namesto atributa XML element, tako kot je to možno pri poimenovanju razreda, kjer se lahko izbere ime iz atributa ali katerega koli podelementa. Pravilo za selektivno preslikavo bi bilo lahko oblike

```
IND(ime_razreda,xml_element, "predpona_niz" + xml_element@TEXT +
"koncnina_niz", xml_element@TAG="tag_vrednost")
```

Orodje za prevajanje je občutljivo na veliko začetnice, zato je treba biti še posebej previden pri zapisu pravil. Prav tako orodje odpove, če se kjerkoli pojavijo šumniki, kar pomeni dodatno nepotrebno prečiščevanje podatkov.

Na takšen način kot so realizirane preslikave OWL instanc bi lahko z vpeljavo novih pravil omogočil celo izgradnjo nove ontologije, na primer s pravili za preslikovanje razredov, določanje podrazredov…

Če se preslikuje primeren XML, se lahko zadovolji skoraj vse zahteve, ki jih omogoča ciljna OWL ontologija. Pri tem pa mora izvirna datoteka imeti ključne XML atribute, kar bi pomenilo spremembo vhodne datoteke, da je primerna za delo s pravili, ali pa so , manj verjetno, orodju primerno strukturirani podatki že na voljo.

3.6 XSD2OWL

Orodje XSD2OWL je sorodno orodju XML2OWL, a le v nekaterih pogledih. Prav tako je zmožno samo ustvariti svojo OWL ontologijo po načelu rigoroznih pravil, a kot že nakazuje samo ime, le iz XML sheme. Preslikava se tako, kot pri XML2OWL realizira z uporabo XSL slogov. Kar je zanimivo pri tem pristopu je, da XSD2OWL služi le za ustvarjanje OWL

ontologije, njene instance pa se v ontologijo preslikajo z drugim orodjem - XML2RDF, pri čemer pride do preslikave iz XML formata v RDF[21].

3.6.1 Preslikava

Preslikava mora zajeti shemino implicitno semantiko. Ta semantika je določena s kombinacijo konstruktov XML sheme. XSD2OWL preslikava je osnovana na prevedbi teh konstruktov v OWL, ki kar najbolje zajamejo njihovo semantiko. Ta prevedba je prikazana v tabeli 2.

XML shema	OWL	Semantika, ki si jo delita
`Element/atribut`	`owl:DatatypeProperty` `owl:ObjectProperty`	Poimenske zveze med vozlišči ali med vozlišči in vrednostmi
`kompleksenTip|skupinaElementov|skupinaAtributov`	`Owl:Class`	Relacije in kontekstne omejitve
`komplesenTip/element`	`owl:Restriction`	Kontekstne omejitve relacije
`@maxOccurs` `@minOccurs`	`owl:maxCardinality` `owl:minCardinality`	Omeji število pojavitev relacij
`sequence` `choice`	`owl:intersectionOf` `owl:union`	Kombinacije relacij v kontekstu

Tabela 3: XSD2OWL prevedba konstruktov XML sheme in skupna semantika z OWL konstrukti

XSD2OWL preslikava zajame velik del semantike XML sheme. Imena uporabljena za XML konstrukte so uporabljena tudi za tiste v OWL, četudi v novem imenskem prostoru, definiranem za ontologijo. Naredi se OWL ontologija, ki eksplicitno izraža semantiko ustrezne XML sheme. Edine omejitve so implicitni vrstni red ki izhaja iz `xsd:sequence` in ekskluzivnosti `xsd:choice`.

Problem: `owl:intersectionOf` ne ohrani vrstnega reda operandov. Ni jasne rešitve ki ohranja visok nivo preglednosti ki je bila dosežena. Uporaba RDF seznama naj bi vnesla red ampak predstavlja ad-hoc (rešitev, specifična le tej nalogi) konstrukt, ki ni prisoten v originalnih metapodatkih. Povrhu vsega, kot je že bilo dokazano v praksi, vrstni red elementov ne prispeva kaj veliko iz zornega kota semantike. Drugi problem `Owl:union` je inkluzivna unija. Rešitev bi bila v uporabi OWL konstrukta, `owl:disjointWith`, med vsemi operandi unije, da jih naredi ekskluzivne.

Za predefinirane tipe, ki so vključeni v OWL specifikacijo kot sta `xsd:string` in `xsd:boolean` je preslikava direktna. Za vse ostale, ki niso del standarda, se preslikajo v `xsd:string`, da ostane njihova leksična vrednost nedotaknjena. Kljub temu, da to povzroči izgubo semantične informacije, je še vedno možno preveriti XML instance glede na izvirno XML pred preslikavo v RDF.

Če se primerjajo pravila za preslikavo med tabelama 1 in 3, bi se lahko na prvi pogled reklo, da so skoraj enaka, a temu ni tako. Medtem, ko so razredi in lastnosti definirni pri XML2OWL vsak zase in morda povezani kot domene, cilji in omejitve, orodje XSD2OWL

vstavlja definicije razredov in latsnosti znotraj definicij drugih latsnosti – uporablja bolj RDF-jevski pristop grafa. Če naleti na element, ki ni list, ga privzame kot objektno lastnost in rekruzivno napolni njegove omejitve- to pomeni, da naredi isto z vsemi nelistnatimi elementi, ki jih sreča pri tem polnenju. Začne pri korenskem elementu, torej je v bistvu celoten opis ontologje zbran v eni sami lastnosti. Na koncu doda samostojne OWL koncepte brez razredov (ti prodobijo k imenu končnico Type), a brez definicij. Razred prav tako ne zapolni domene posameznih objektnih lastnosti, ko jih definira na tak način.

Tako iz XSD segmenta kode

```
<xs:element>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="Otrok">
                        <xs:complexType>
                              <xs:attribute name="Ime"/>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="Ime"/>
      </xs:complexType>
</xs:element>
```

Nastane sledeči OWL zapis

```
<owl:ObjectProperty rdf:ID="Oce">
   <rdfs:range>
      <owl:Class rdf:ID="OceType">
         <rdfs:subClassOf>
            <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                  <owl:Restriction>
                    <owl:onProperty rdf:resource="#Otrok"/>
                       <owl:allValuesFrom>
                          <owl:Class rdf:ID="otrokType">
                            <rdfs:subClassOf>
                               <owl:Class>
                                  <owl:Restriction>
                                     <owl:onProperty
rdf:resource="#ime"/>
                                        <owl:allValuesFrom
rdf:resource=#ime"/>
                                     </owl:Restriction>
                                     <owl:Restriction>
                                       <owl:onProperty
rdf:resource="#ime"/>
                                        <owl:maxCardinality
rdf:datatype="nonNegativeInteger">1</owl:maxCardinality>
                                     </owl:Restriction>
                                  </owl:Class>
                              </rdfs:subClassOf>
```

```
                                    </owl:allValuesFrom>
                                </owl:Restriction>
                            </owl:intersectionOf>
                        </owl:Class>
                    </rdfs:subClassOf>
                </rdfs:range>
            </owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Otrok"/>
<owl:DatatypeProperty rdf:ID="ime"/>
```

Takšen zapis je nestanderden glede na ostale, ki so bili do sedaj opisani, a načeloma opisuje isto stvar. Po končani preslikavi so potrebni nekateri ročni popravki z namenom, da se razreši nasprotujoča si imena med OWL razredi in RDF lastnostmi. Do tega pride ker ima XML neodvisno imensko domeno za elemente in kompleksne tipe, medtem ko ima OWL unikatno za vse konstrukte v svoji domeni. Kot rezultat se pridobi OWL ontologija, ki pa je lahko celo izraznosti OWL-Full, ker je preslikava vključila `rdf:Property` za tiste elemente, ki imajo tako objektno lastnost, kot tudi podatkovno[24].

3.7 Preslikovanje XML v RDF

V prejšnjih primerih so vidni zapisi v OWL datoteki oblike: `rdf:about=` ter `rdf:ID`. Gre za RDF zapise, saj je OWL zgrajen kot slovarski podaljšek RDF-ja. Rečeno je, da OWL uporablja XML sintakso, v resnici pa uporablja hibridno XML ter RDF sintakso. Zakaj bi potemtakem sploh uporabljali OWL?

OWL in RDF sta na prvi pogled videti zelo podobna, a OWL je močnejši jezik z večjim slovarjem in močnejšo sintakso, kot tak pa veliko večji potencial računalniške interpretacije kot RDF. OWL in RDF sta osnovana na navajanju dejstev, določanja strukture razredov ter lastnosti. OWL te možnosti v precejšni meri razširi.

Tako OWL, kot tudi RDF, lahko deklarirata razrede in umestita te razrede v hierarhijo. Kar pa presega zmožnosti RDF-ja, je zmožnost OWL-a, da lahko te razrede specificira kot logične kombinacije (intersekcija, unija, kompliment) drugih razredov. Oba lahko tudi deklarirata lastnosti, jih umestita v hierarhijo, jim določita domeno ter ciljni razred in jim določita tip, a OWL lahko razširi ta opis s tem da razglasi lastnost za tranzitivno, simetrično, funkcionalno ali pa obratno neki drugi lastnosti.

OWL lahko izrazi kateri posameznik (instanca) pripada kateremu razredu in kakšne so vrednosti lastnosti tega posameznika. Med razredi in lastnostmi so lahko izražene izjave o ekvivalenci ter disjunktnosti, med posamezniki pa se lahko vzpostavi enakost ali neenakost. Največja razširitev OWL-a nad RDF-jem je njegova zmožnost vzpostavljanja omejitev nad tem, kako so posamezne lastnosti prisotne v razredu. OWL lahko definira razrede, v katerih je lastnost omejena na način, da morajo posamezniki tega razreda imeti spremenljivke, katerih vrednosti pripadajo točno določenemu razredu; da mora vsaj ena izmed vrednosti biti iz določenega razreda; da mora biti vsaj določeno število ali pa le največje število različnih vrednosti[5].

Iz tega razberemo, da RDF nikakor ne more nadomestiti opisa ontologije z OWL. Morda le najpreprostejše. Bi pa morda prišel prav pri opisu instanc ontologije, kjer določamo le katerim razredom pripadajo in katere lastnosti imajo. Prav zato si velja posebej pogledati pristop, soroden

do sedaj omenjenim, ki temelji na preslikavi iz XML v RDF. Tako pridobljene instance pa lahko z manjšim trudom uporabimo v OWL okolju.

3.7.1 XML2RDF

Ko je enkrat na voljo XML shema v obliki OWL ontologije, je možno vanjo uvoziti XML podatke, ki predstavljajo instance te sheme. Namen je ustvariti kar najbolj pregledne RDF metapodatke, zato je bil izbran pristop preslikave strukture. Preglednost je dosežena z strukturno preslikanimi modeli, ker ti samo poizkušajo predstaviti strukturo XML podatkov, torej drevesa z uporabo RDF-ja. RDF model je osnovan na grafu in ga je zato lahko modelirati, saj so drevesa le specializirani grafi. S tem pristopom ni potrebno skrbeti za izgubo semantike. Semantika se je formalizirala v ustrezno ontologijo, in se ji bo pripelo RDF metapodatke z uporabo relacije `rdf:type`.

Pristop preslikave strukture je osnovan na prevodu XML podatkov instanc v RDF instance, ki predstavljajo ustrezne OWL konstrukte. Najosnovnejša preslikava je med relacijo instanc, iz `xsd:elements` in `xsd:attrubtes` v `rdf:Properties`. Konkretneje, `owl:ObjectProperites` za razmerja vozlišče-vozlišče in `owl:DataypeProperties` za razmerja vozlišče-vrednost. Vseeno je v nekaterih primer potrebno uporabiti `rdf:Properites` za `xsd:elements`, ki imajo obe vrednosti, tako objektnega tipa, kot tudi podatkovnega.

Posamezne vrednosti so med preslikavo shranjene kot preprosti tipi. Prazna RDF vozlišča so v RDF modelu z namenom da služijo kot vir in cilj za posamezne lastnosti. Ostali bodo prazni dokler ne bodo dopolnjeni z semantično informacijo.

Končno rezultat je RDF model grafa, ki vsebuje vse kar se je dalo pridobili iz XML drevesa. Je že semantično dopolnjen zaradi `rdf:type` relacije, ki povezuje vsako RDF lastnost z `owl:ObjecttProperty` ali `owl:DatatypeProperty`, ki jih predstavlja. Lahko je še nadaljnjo dopolnjen, če se prazna vozlišča poveže z `owl:Class`, ki definira skupino lastnosti in z njimi povezane omejitve,torej ustreznim `xsd:complexType`. Ta semantična dopolnitev grafa je formaliziran z uporabo relacije `rdf:type` iz praznih vozlišč v ustrezne OWL razrede.

3 Zaključek

3.1 Ocena posameznih orodij in pristopov

Orodje	Prednosti	Slabosti
XML2OWL	-Sposobno ustvariti OWL svojo ontologijo. -Sposobno ustvariti OWL ontologije iz XML instance. -Možnost dodajanja globalne predpone za poimenovanje lastnosti.	-Pridobljena ontologija je preprosta. -Lastnost spremenljivke vedno tipa `xsd:string`. -Dodajanje trivialnih omejitev nad razredi ob preslikavi iz XML instance: najmanjša številnost 0. -Omejitve imajo vedno cilje: `Literal` in `Thing`. -Slabo poimenovanje instanc – sedem naključnih številk.
XSD2OWL	-Sposobno ustvariti svojo OWL ontologijo.	-Potrebuje dodatno orodje za preslikavo instanc v svojo ontologijo -Pridobljena ontologija je lahko izraznosti OWL-Full. -Nestandarden zapis ontologije, ki je zato nekompatibilna z nekaterimi urejevalniki ontologij. -Slabo definirane lastnosti.
TopBraid – XSDIMport	-Ustvari svojo ontologijo in instance iz XML dokumenta.	-Poimenovanje instanc na način `Class_1`, `Class_2` ne zadošča. -Lastnostim ne predpiše domene ali ciljne vrednosti. -Vsi podelementi kompleksnega tipa vozlišča postanejo lastnosti objektov, kar ni vedno res.
TopBraid - Maestro Edition	-Ustvari svojo ontologijo in instance iz XML dokumenta. -Možna sprememba ontologijo v istem programu. -Možen uvoz instanc v obstoječo OWL ontologijo.	-Vsako vozlišče se preslika v razred, kar ni vedno res. -Ustvarjeno veliko število trivialnih instanc. -Slabo poimenovanje instanc: `r-1-1-3`. -Vsako razmerje starš-otrok se

		spremeni v lastnost objektov, kar ni vedno res.
JXML2OWL Mapper	-Orodje sposobno ustvariti shemo iz XML dokumenta. -Dober grafični uporabniški vmesnik. -Orodje samo ponudi možne lastnosti, glede na prej določene razredne preslikave. -Združevanje podvojenih instanc v eno samo.	- Nujno potrebujemo obstoječo OWL ontologijo. -Pri velikih ontologijah postane iskanje lastnosti precej zahtevno, saj niso samostojne, temveč vezane na posamezno razredno preslikavo. -Relativno pomanjkljive možnosti poimenovanja instanc. -Glede na to, da se orodje v tolikšni meri naslanja na obstoječo ontologijo, bi lahko bolje izrabljalo razredno hierarhijo (uvrstitev instance le v en sam razred).
COMA++	-Vzpostavi relativno dobro avtomatsko povezavo med XML konstrukti ter obstoječimi OWL koncepti.	-Ni namenjena preslikavi instanc.
XML2OWL s pravili	-Inkrementalni način poimenovanja točno določenih instanc. -Poimenovanje instanc glede na atribut ali vsebino XML elementa. -Selektivna preslikava. -Možnost vzpostavitve objektne lastnosti ne nujno na razmerju starš-otrok. -Primerno za delo z obstoječo “težko-kategorno” ontologijo. -V primeru spremenjene ontologije so spremembe pravil minimalne.	-Zahteva po dodatni `.rules` datoteki. -Poznavanje nove sintakse. -Potrebna dodatna previdnost pri ujemanje naših pravil z obstoječo ontologijo. -Samo ena oblika pravil – po navadi se veže pravilo le na atribute in ne na vrednosti. -Orodje občutljivo na veliko začetnico. -Ne podpira šumnikov.

Tabela 4: Pregled prednosti in slabosti posameznih orodij

3.2 Orodje za preslikavo XML v OWL po naši meri

V tem pristopu sem združil večino dobrih lastnosti posameznih orodij in pristopov, ki sem jih opisal do sedaj, njihove pomanjkljivosti pa odpravil v kar največji meri. Glavna prednost mojega pristopa je univerzalnost: preslikava bi bila možna in učinkovita za prav vsak XML dokument. Preslikava bi na naš način potekala v dveh oziroma treh korakih:

- Določanje pravil za preslikavo XML dokumenta v obstoječo ontologijo. Nekatera pravila se določijo avtomatsko.
- Izvajanje preslikave - ustvarjanje OWL instanc.
- Pogojni korak: Če obstoječa ontologija ne obstaja, se ustvari avtomatsko iz izvirnega XML dokumenta.

3.2.1 Ustvarjanje nove ontologije

Za namen ustvarjanja nove ontologije bi moj pristop sprejel tako shemo kot tudi navaden XML dokument. Če delamo le z navadnim XML dokumentom, potem se shema ustvari sama. Ne glede na to, da je novo ustvarjena ontologija preprostejše narave, je vsekakor dobrodošla, če obstoječa ni na voljo. Nanjo je potrebno gledati kot osnovo, ki bo služila kot dober kažipot za ročni razvoj ontologije do zadovoljive stopnje. Pri preslikavi se držim pravila, da je dobra osnova za ontologijo taka, da uporabnik nima z njo več dela kot koristi.

Iz tega razloga so izbrana tudi pravila za preslikavo. Največja verjetnost je, ne glede na to, kakšen vhodni dokument pridobimo, da so tako pridobljeni koncepti ontologije primerni, oziroma takšni, kakršne bi ustvarili sami, če bi ontologijo izdelovali ročno, od začetka. Če bi natančno vedel za katero XML instanco ali shemo bi izdeloval ontologijo, bi bila temu primerno prirejana pravila za preslikavo, a na račun avtomatiziranosti. Tovrstna pravila bi bolj malo pripomogla pri ostalih primerih.

Najprimernejše bi bilo, če bi se dalo popravljati in dopolnjevati tako pridobljeno ontologijo kar v isti aplikaciji, brez potrebe po dodatni programski opremi. Prav zaradi tega bi bilo dobro, da bi bil moj pristop implementiran kot orodje, izdelano kot dodatek (plugin) obstoječemu orodju za delo z OWL ontologijami (na primer Protégé-OWL). Tako bi bila vsa potrebna funkcionalnost zbrana na enem mestu.

Predstavljeno preslikavo se bo izvajalo na segmentu XML sheme:

```
<xs:element name="Oseba">
      <xs:complexType>
         <xs:sequence>
               <xs:element name="SpolOsebe" minOccurs="0"/>
               <xs:element name="StarostOsebe"/>
               <xs:element name="Otrok" minOccurs="0" maxOccurs="10">
                     <xs:complexType>
                           <xs:sequence>
                                 <xs:element name="SpolOtroka"/>
                                 <xs:element name="StarostOtroka"/>
                           </xs:sequence>
```

```
                        <xs:attribute name="ImeOtroka"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Znanec" minOccurs="0"/>
                <xs:complexType>
                    <xs:attribute name="ImeZnanca"/>
                </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="ImeOsebe"/>
    </xs:complexType>
</xs:element>
```

Preslikava iz XML dokumenta v novo ontologijo naj poteka na sledeči način:

- Kompleksni tip se preslika v OWL razred, njegovi morebitni atribut in njegovi preprosti otroci brez atributov postanejo lastnost spremenljivke, pravkar določen razred pa postane domena te lastnosti. Ime razreda je ime kompleksnega vozlišča in se mora začeti z veliko začetnico v skladu z OWL standardi. Ime lastnosti spremenljivke postane ime atributa oziroma preprostega vozlišča, oboje z malo začetnico. Prav tako se mora določiti tip lastnosti – iz sheme se ga pridobi iz podanega tipa, iz XML instance pa določimo ali gre za `xsd:integer` ali `xsd:string` na podlagi zapisanega podatka. Vsi novi razredi so podrazredi univerzalnemu razredu `Thing` (stvar), saj jih na na podlagi podatkov pridobljenih z XML ne moremo razvrstiti drugače, četudi vemo, na primer, da je `Otrok` podrazred razreda `Oseba`. Po teh pravilih se iz zgornjega primera pridobi naslednje razrede in lastnosti spremenljivk.

  Razredi:

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

  <owl:Class rdf:about="#Otrok">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>

  <owl:Class rdf:about="#Znanec">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  </owl:Class>

  <owl:Class rdf:about="&owl;Thing"/>
  ```

  Lastnostni spremenljivk

  ```
  <owl:DatatypeProperty rdf:about="#imeOsebe">
        <rdfs:domain rdf:resource="#Oseba"/>
        <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:about="#imeOtroka">
  ```

```
      <rdfs:domain rdf:resource="#Otrok"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>


<owl:DatatypeProperty rdf:about="#imeZnanca">
      <rdfs:domain rdf:resource="#Znanec"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#spolOsebe">
      <rdfs:domain rdf:resource="#Oseba"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#spolOtroka">
      <rdfs:domain rdf:resource="#Otrok"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>


<owl:DatatypeProperty rdf:about="#starostOsebe">
      <rdfs:domain rdf:resource="#Oseba"/>
      <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#starostOtroka">
      <rdfs:domain rdf:resource="#Otrok"/>
      <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

Lastnost `Ime` se pojavlja večkrat. Istoimenske lastnosti istega tipa bi pristop združil in le dodal več ustreznih razredov v domeno:

```
<owl:DatatypeProperty rdf:about="#ime">
      <rdfs:domain rdf:resource="#Oseba"/>
      <rdfs:domain rdf:resource="#Otrok"/>
      <rdfs:domain rdf:resource="#Znanec"/>
      <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

- Če ima posamezno vozlišče potomce, ki so se izkazali za OWL razrede (preslikava poteka v večih iteracijah, kjer se najprej določijo razredi ter nato vse drugo), se med vozliščem in neposrednim potomcem (starš-otrok) vzpostavi lastnost objektov, kjer je domena starš, otrok pa postane ciljna vrednost. Ime dobi na podlagi globalne predpone za lastnosti objektov, ki jo določi uporabnik (na primer `ima`).

```
<owl:ObjectProperty rdf:about="#imaOtrok">
   <rdfs:domain rdf:resource="#Oseba"/>
      <rdfs:range rdf:resource="#Otrok"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:about="#imaZnanec">
      <rdfs:domain rdf:resource="#Oseba"/>
```

```
        <rdfs:range rdf:resource="#Znanec"/>
 </owl:ObjectProperty>
```

Lahko bi se vzpostavila tudi inverzna lastnost, kar pride večkrat prav: Domena in ciljna vrednost se zamenjajta, nastavljiva predpona za takšne primere pa bi bila najprimernejša `je`.

```
<owl:ObjectProperty rdf:about="#jeOtrok">
     <rdfs:range rdf:resource="#Oseba"/>
     <rdfs:domain rdf:resource="#Otrok"/>
 </owl:ObjectProperty>
```

Torej se preprosti tipi otrok kompleksnega XML elementa ne spremenijo nujno v lastnost tega razreda, če se izkaže da je istoimenski element v bistvu OWL razred, sicer nedefiniran v razmerju starš-otrok. Na primer:

```
<Oce>
    <Ime></Ime>
    <Otrok></Otrok>
</Oce>
<Otrok>
    <Ime></Ime>
    <Starost><Starost>
<Otrok>
```

- Vsekakor ni vseeno če se preslikava izvaja iz XML sheme ali XML instance. Po shemi se lahko v novo ontologijo vstavi več omejitev nad razredi.

- Če je v shemi podano številnost za posamezni element, se določi minimalno (`min`) oziroma maksimalno (`max`) število pojavitev posamezne lastnosti – objektna lastnost, če je element vezan na element, ki je OWL razred. Če je element obvezen, se privzame, da je njegova številnost natančno ena. Isto velja tudi za atribute.

  Vsi ti pogoji morajo biti izpolnjeni za posamezno instanco, da se šteje kot član tega razreda, zato se omejitve združujejo zaradi preglednosti kot intersekcijo (`owl:intersection`) razredov, ki bo nadrazred temu razredu. To se bo pojavilo vedno, kjer je v shemi oblika `xsd:sequence` saj gre tu vedno za preslikan razred z lastnostmi. Zgornji primer razreda `Oseba` z dodanimi omejitvami tako postane:

```
<owl:Class rdf:about="#Oseba">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
       <rdfs:subClassOf>
          <owl:intersectionOf rdf:parseType="Collection">
             <owl:Restriction>
                <owl:onProperty rdf:resource="#imaOtrok"/>
                <owl:onClass rdf:resource="#Otrok"/>
                <owl:minCardinality>0</owl:minCardinality>
             </owl:Restriction>
             <owl:Restriction>
                <owl:onProperty rdf:resource="#imaOtrok"/>
                <owl:onClass rdf:resource="#Otrok"/>
                <owl:maxCardinality>10</owl:maxCardinality>
```

```
            </owl:Restriction>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#StarostOsebe"/>
                <owl:onClass rdf:resource="&xsd:integer"/>
                <owl:cardinality>1</owl:cardinality>
            </owl:Restriction>
            <owl:Restriction>
                <owl:onProperty rdf:resource="# ImeOsebe"/>
                <owl:onClass rdf:resource="&xsd:string"/>
                <owl:cardinality>1</owl:cardinality>
            </owl:Restriction>
        </owl:intersectionOf>
    </rdfs:subClassOf>
</owl:Class>
```

Neobveznih elementov se ne zapiše kot omejitev s številnostjo nič.

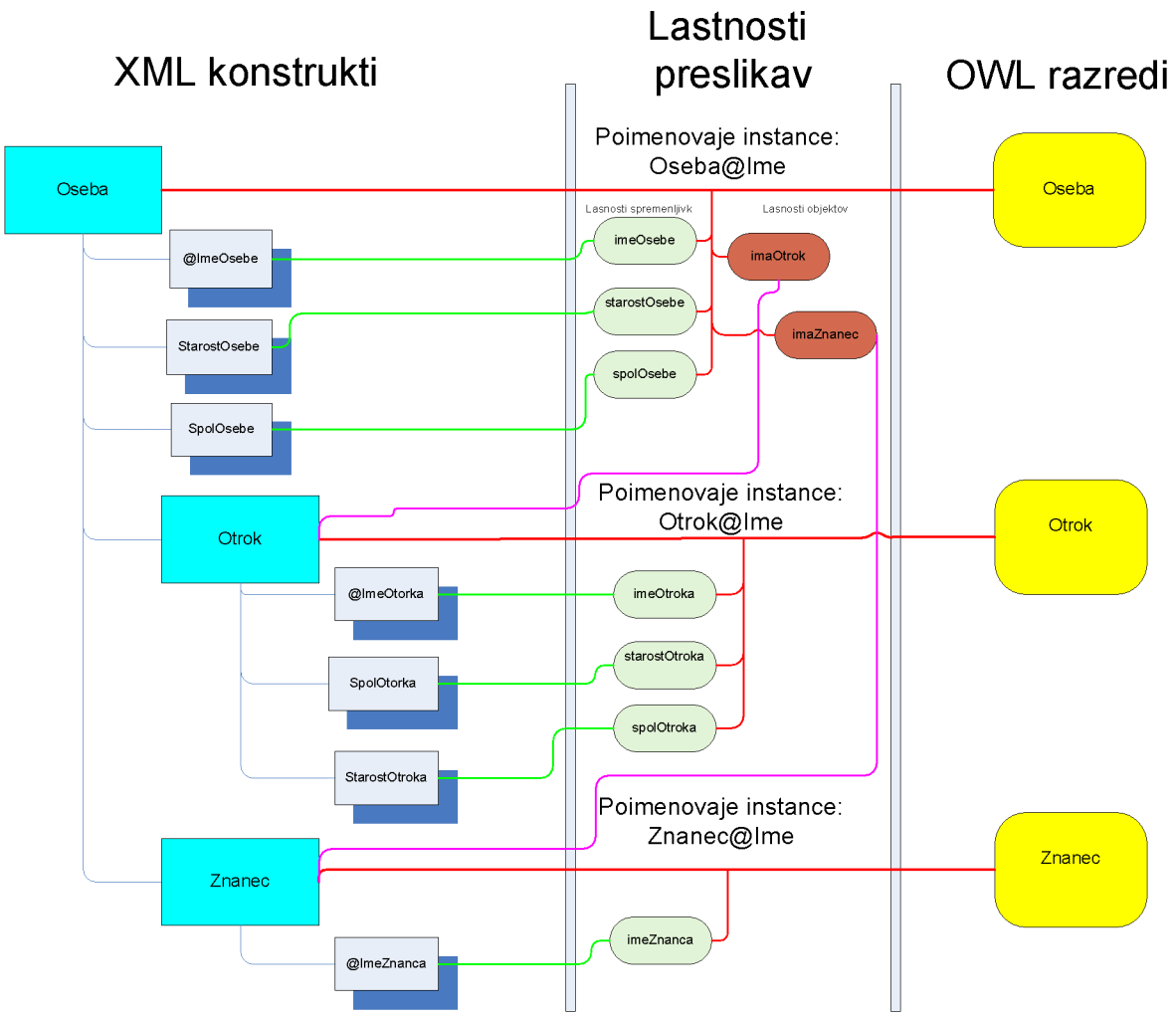
- V primeru, da obstja v shemi zapis `xsd:choice`, je dovolj, da je izpolnjena le ena izmed lastnosti, zato se omejtive združujejo kot unija (`owl:union`) razredov, ki bo nadrazred temu razredu.

Niti iz sheme pa se ne da sklepati o nadrazredih ter podrazredih, disjunktivnosti ter ekvivalenci ter ali je morda lastnost funkcijska, tranzitivna…

3.2.2 Pravila za preslikavo instanc

Pravila za preslikavo bi narekovala kako se ustvarijo posamezne instance v ciljni OWL ontologiji. Povezave, ki so se vzpostavile med preslikavo se najbolje spremlja v grafični obliki, kakršen je pri orodju XMLJ2OWL. Če bi se delala z že obstoječo ontologijo, bi pristop moral na podlagi zgornjih pravil znati vzpostaviti povezave med XML konstrukti ter OWL koncepti na podlagi poimenovanja vozlišč ter OWL konceptov. Vozlišče `<Otrok>` bi se povezalo z razredom `Otrok` in tako dalje. Podobno bi se zgodilo z lastnostmi spremenljivk. Težje je tej zahtevi ugoditi pri lastnosti objektov, zaradi globalne predpone. Za bolj dovršeno ujemanje bi se lahko uporabilo algoritme, kakršni so prisotni pri orodju COMA++.

Obstoječim, avtomatsko dodanim, pravilom za preslikavo se lahko dodajo naknadna. Nad njimi se določijo tudi posamezne lastnosti objektov in lastnosti spremenljivk. Možna mora biti pogojna preslikava. Za posamezni razred je možno dodati vse tiste lastnosti, v katerih ta razred nastopa kot domena (opiranje na ciljno ontologijo). Nujno potrebna bi bila možnost iskanja, kje se posamezna lastnost pojavlja, kar ni prisotno pri pri prejšnjih orodjih. Prikaz, kako bi izgledala preslikava za zgornji primer, če se ontologija ni spreminjala, najbolje prikaže slika 4.



Slika 4: Pravila za preslikavo med XML konstrukti ter OWL koncepti za zgornji primer. Lastnosti preslikav naj ne bi bile vidne na takšen način, ampak le kot dodatna orodna vrstica.

V primeru da pride do sprememb v XML shemi oziroma instanci ali ontologiji, se mora dati te spremembe prikazati na način, ki ne zahteva ponovitev prejšnjih korakov od začetka. Če izginejo ključne preslikave v razrede skupaj z njimi izginejo tudi vse povezane lastnosti. Novi razredi in elementi pa se le dodajo obstoječim in so brez povezav, oziroma se povežejo, če tako določi ujemevalni algoritem, potrebno pa je tudi sporočilo o spremembi.

3.2.3 Uvoz instanc

Na podlagi pravil se uvrsti XML podatke v instance OWL ontologije. Za vsako razredno preslikavo posebej se določi način poimenovanja instance. Lahko je poimenovana po vrednosti atributa, vrednosti podvozlišča preprostega tipa, konkatenacij večih podvozlišč ali pa kar določen niz, seveda z možnostjo inkrementacije, da se zagotovi unikatnost instanc. V primeru podvojene instance, bi se ustvarila le ena sama ter združila vse lastnosti. Če ima ontologija podano razredno

hierarhijo ter ugotovi, da pripada instanca dvema razredoma ki sta v nadrazred-podrazred razmerju, se obdrži le podrazred, seveda z združitvijo lastnosti.

Pravilo za preslikavo iz slike 4 se uporabi nad nad XML dokumentom, ki je bil pridobljen iz zgornjega primera XML sheme:

```
<Oseba Ime="Janez">
      <Spol>M</Spol>
      <Starost>52</Starost>
      <Otrok Ime="Janko">
            <Spol>M</Spol>
            <Starost>15</Starost>
      </Otrok>
      <Otrok Ime="Metka">
            <Spol>Ž</Spol>
            <Starost>13</Starost>
      </Otrok>
      <Znanec Ime="Iztok"/>
      <Znanec Ime="Jože"/>
      <Znanec Ime="Marko"/>
</Oseba>
```

Ustvarjene so instance z lastnostmi:

```
<Oseba rdf:about="#Janez">
      <starostOsebe>52</starostOsebe>
      <imeOsebe>Janez</imeOsebe>
      <spolOsebe>M</spolOsebe>
      <imaZnanec rdf:resource="#Iztok"/>
      <imaOtrok rdf:resource="#Janko"/>
      <imaZnanec rdf:resource="#Jože"/>
      <imaZnanec rdf:resource="#Marko"/>
      <imaOtrok rdf:resource="#Metka"/>
</Oseba>

<Otrok rdf:about="#Janko">
      <spolOtroka>M</spolOtroka>
      <imeOtroka>Janko</imeOtroka>
      <starostOtroka>15</starostOtroka>
</Otrok>

<Otrok rdf:about="#Metka">
      <starostOtroka>13</starostOtroka>
      <imeOtroka>Metka</imeOtroka>
      <spolOtroka>Ž;</spolOtroka>
</Otrok>

<Znanec rdf:about="#Iztok">
      <imeZnanca>Iztok</imeZnanca>
</Znanec>


<Znanec rdf:about="#Jože">
      <imeZnanca>Jože</imeZnanca>
</Znanec>
```

```
<Znanec rdf:about="#Marko">
      <imeZnanca>Marko</imeZnanca>
</Znanec>
```

3.3 Pogled v prihodnost

Glede na prizadevanja konzorcija W3C, nekaterih najbolj znanih pionirjev v razvoju svetovnega spleta in razvijalcev ontoloških spletnih jezikov, je ontološki svetovni splet stvar relativno bližnje prihodnosti. Največji svetovni proizvajalci programske opreme že nekaj časa zelo resno delajo na razvoju orodij, ki bodo lahko znotraj zelo velikih intranetov sama pripravila na ontologiji temelječe izsledke, vedno bolj smiselno dognane povzetke in predloge za poslovni sistem. Gotovo bodo taka orodja, na podlagi vedno bolj dognanega ontološkega jezika OWL, lahko predlagala določene rešitve glede trenutnih odločitev, terminskih, srednjeročnih in dolgoročnih planov za celoten poslovni sistem in za njegove posamezne poslovne funkcije. V kombinaciji izsledkov ontoloških orodij znotraj poslovnega sistema in izsledkov pridobljenimi na svetovnem spletu, bodo potem odločitve vodstev hitrejše in bolj natančne[8].

Vzporedno z rastjo ontološkega spleta bo rasla tudi potreba po premostitvi mostu med podatki kot takimi in njihovi ontološki obliki z namenom širitve baze znanja, pri tem pa niti ni nujno, da bo šlo za preslikavo iz XML v OWL.

4 Priloge

V prilogi so zbrane datoteke, ki so služile za preizkušanje orodij.

4.1 Primer ontologije

Primer ontologije je bil narejen z orodjem Protégé-OWL verzija 4.0. Ontologija je izraznosti DL-Lite.

Imamo osebe, kjer je lahko posamezna oseba starš, otrok ali znanec. Vsaka oseba lahko pozna več znancev. Vsak otrok ima lahko največ dva starša. Vsak starš mora imeti vsaj enega otroka. Vse osebe imajo natančno en spol in eno ime. Družina ima največ po eno mamo ali očeta. Družina živi na naslovu znotraj države. Družina ima priimek. Oseba je znanec, če ga pozna vsaj ena oseba.

Ontologija.owl

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY ontologija "http://www.semanticweb.org/ontologija.owl#" >
]>


<rdf:RDF
xmlns="http://www.semanticweb.org/ontologies/2009/3/Ontology1239258138828.owl#
"

xml:base="http://www.semanticweb.org/ontologies/2009/3/Ontology1239258138828.o
wl"
     xmlns:ontologija="http://www.semanticweb.org/ontologija.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about=""/>



    <!--

////////////////////////////////////////////////////////////////////////////
////////
    //
    // Object Properties
```

```
    //

////////////////////////////////////////////////////////////////////////////////
/////////
     -->



    <!-- http://www.semanticweb.org/ontologija.owl#imaClana -->

    <owl:ObjectProperty rdf:about="#imaClana">
        <rdfs:domain rdf:resource="#Druzina"/>
        <rdfs:range rdf:resource="#Otrok"/>
        <rdfs:range rdf:resource="#Stars"/>
        <owl:inverseOf rdf:resource="#jeClan"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#imaGospodarja -->

    <owl:ObjectProperty rdf:about="#imaGospodarja">
        <rdfs:range rdf:resource="#Druzina"/>
        <rdfs:domain rdf:resource="#Oce"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#imaGospodinjo -->

    <owl:ObjectProperty rdf:about="#imaGospodinjo">
        <rdfs:range rdf:resource="#Druzina"/>
        <rdfs:domain rdf:resource="#Mati"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#imaOtroka -->

    <owl:ObjectProperty rdf:about="#imaOtroka">
        <rdfs:range rdf:resource="#Otrok"/>
        <rdfs:domain rdf:resource="#Stars"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#imaOtrokaVDruzini -->

    <owl:ObjectProperty rdf:about="#imaOtrokaVDruzini">
        <rdfs:range rdf:resource="#Druzina"/>
        <rdfs:domain rdf:resource="#Otrok"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#jeClan -->
```

```
    <owl:ObjectProperty rdf:about="#jeClan">
        <rdfs:range rdf:resource="#Druzina"/>
        <rdfs:domain rdf:resource="#Otrok"/>
        <rdfs:domain rdf:resource="#Stars"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#jeOtrok -->

    <owl:ObjectProperty rdf:about="#jeOtrok">
        <rdfs:domain rdf:resource="#Otrok"/>
        <rdfs:range rdf:resource="#Stars"/>
        <owl:inverseOf rdf:resource="#imaOtroka"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#jePoznanOd -->

    <owl:ObjectProperty rdf:about="#jePoznanOd">
        <rdfs:range rdf:resource="#Oseba"/>
        <rdfs:domain rdf:resource="#Znanec"/>
        <owl:inverseOf rdf:resource="#pozna"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#pozna -->

    <owl:ObjectProperty rdf:about="#pozna">
        <rdfs:domain rdf:resource="#Oseba"/>
        <rdfs:range rdf:resource="#Znanec"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologija.owl#ziviV -->

    <owl:ObjectProperty rdf:about="#ziviV">
        <rdfs:domain rdf:resource="#Druzina"/>
        <rdfs:range rdf:resource="#Drzava"/>
    </owl:ObjectProperty>



    <!--

///////////////////////////////////////////////////////////////////////////////////
/////////
    //
    // Data properties
    //

///////////////////////////////////////////////////////////////////////////////////
/////////
```

```
 -->



<!-- http://www.semanticweb.org/ontologija.owl#ime -->

<owl:DatatypeProperty rdf:about="#ime">
    <rdfs:domain rdf:resource="#Oseba"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologija.owl#imeDrzave -->

<owl:DatatypeProperty rdf:about="#imeDrzave">
    <rdfs:domain rdf:resource="#Drzava"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologija.owl#naslov -->

<owl:DatatypeProperty rdf:about="#naslov">
    <rdfs:domain rdf:resource="#Druzina"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologija.owl#prihodek -->

<owl:DatatypeProperty rdf:about="#prihodek">
    <rdfs:domain rdf:resource="#Oce"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologija.owl#priimek -->

<owl:DatatypeProperty rdf:about="#priimek">
    <rdfs:domain rdf:resource="#Druzina"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologija.owl#spol -->

<owl:DatatypeProperty rdf:about="#spol">
    <rdfs:domain rdf:resource="#Oseba"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
    <!--

///////////////////////////////////////////////////////////////////////////////
/////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////////////
/////////
     -->



    <!-- http://www.semanticweb.org/ontologija.owl#Druzina -->

    <owl:Class rdf:about="#Druzina">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#imaGospodinjo"/>
                <owl:onClass rdf:resource="#Mati"/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#naslov"/>
                <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#imaGospodarja"/>
                <owl:onClass rdf:resource="#Oce"/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#ziviV"/>
                <owl:allValuesFrom rdf:resource="#Drzava"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Drzava -->

    <owl:Class rdf:about="#Drzava">
```

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



    <!-- http://www.semanticweb.org/ontologija.owl#Hcerka -->

    <owl:Class rdf:about="#Hcerka">
        <rdfs:subClassOf rdf:resource="#Otrok"/>
        <owl:disjointWith rdf:resource="#Sin"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Mati -->

    <owl:Class rdf:about="#Mati">
        <rdfs:subClassOf rdf:resource="#Stars"/>
        <owl:disjointWith rdf:resource="#Oce"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Oce -->

    <owl:Class rdf:about="#Oce">
        <rdfs:subClassOf rdf:resource="#Stars"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#prihodek"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Oseba -->

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

```
    <!-- http://www.semanticweb.org/ontologija.owl#Otrok -->

    <owl:Class rdf:about="#Otrok">
        <rdfs:subClassOf rdf:resource="#Oseba"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#jeOtrok"/>
                <owl:onClass rdf:resource="#Stars"/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Sin -->

    <owl:Class rdf:about="#Sin">
        <rdfs:subClassOf rdf:resource="#Otrok"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Stars -->

    <owl:Class rdf:about="#Stars">
        <rdfs:subClassOf rdf:resource="#Oseba"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#imaOtroka"/>
                <owl:onClass rdf:resource="#Mati"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologija.owl#Znanec -->

    <owl:Class rdf:about="#Znanec">
        <rdfs:subClassOf rdf:resource="#Oseba"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#jePoznanOd"/>
                <owl:onClass rdf:resource="#Oseba"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

```
    <!-- http://www.w3.org/2002/07/owl#Thing -->

    <owl:Class rdf:about="&owl;Thing"/>



    <!--

///////////////////////////////////////////////////////////////////////////////
/////////
    //
    // Individuals
    //

///////////////////////////////////////////////////////////////////////////////
/////////
     -->



    <!-- http://www.semanticweb.org/ontologija.owl#Ana -->

    <Mati rdf:about="#Ana">
        <ime rdf:datatype="&xsd;string">Ana</ime>
        <spol rdf:datatype="&xsd;string">&#381;</spol>
        <jeClan rdf:resource="#DruzinaNovak"/>
        <imaOtroka rdf:resource="#Janko"/>
        <imaOtroka rdf:resource="#Maja"/>
    </Mati>



    <!-- http://www.semanticweb.org/ontologija.owl#DruzinaNovak -->

    <Druzina rdf:about="#DruzinaNovak">
        <priimek rdf:datatype="&xsd;string">Novak</priimek>
        <naslov rdf:datatype="&xsd;string"
            >Slovenska cesta 33</naslov>
        <imaGospodinjo rdf:resource="#Ana"/>
        <imaGospodarja rdf:resource="#Janez"/>
        <ziviV rdf:resource="#Slovenija"/>
    </Druzina>



    <!-- http://www.semanticweb.org/ontologija.owl#Janez -->

    <Oce rdf:about="#Janez">
        <prihodek rdf:datatype="&xsd;float">20000</prihodek>
        <ime rdf:datatype="&xsd;string">Janez</ime>
        <spol rdf:datatype="&xsd;string">M</spol>
        <jeClan rdf:resource="#DruzinaNovak"/>
        <imaOtroka rdf:resource="#Janko"/>
        <pozna rdf:resource="#Lojze"/>
```

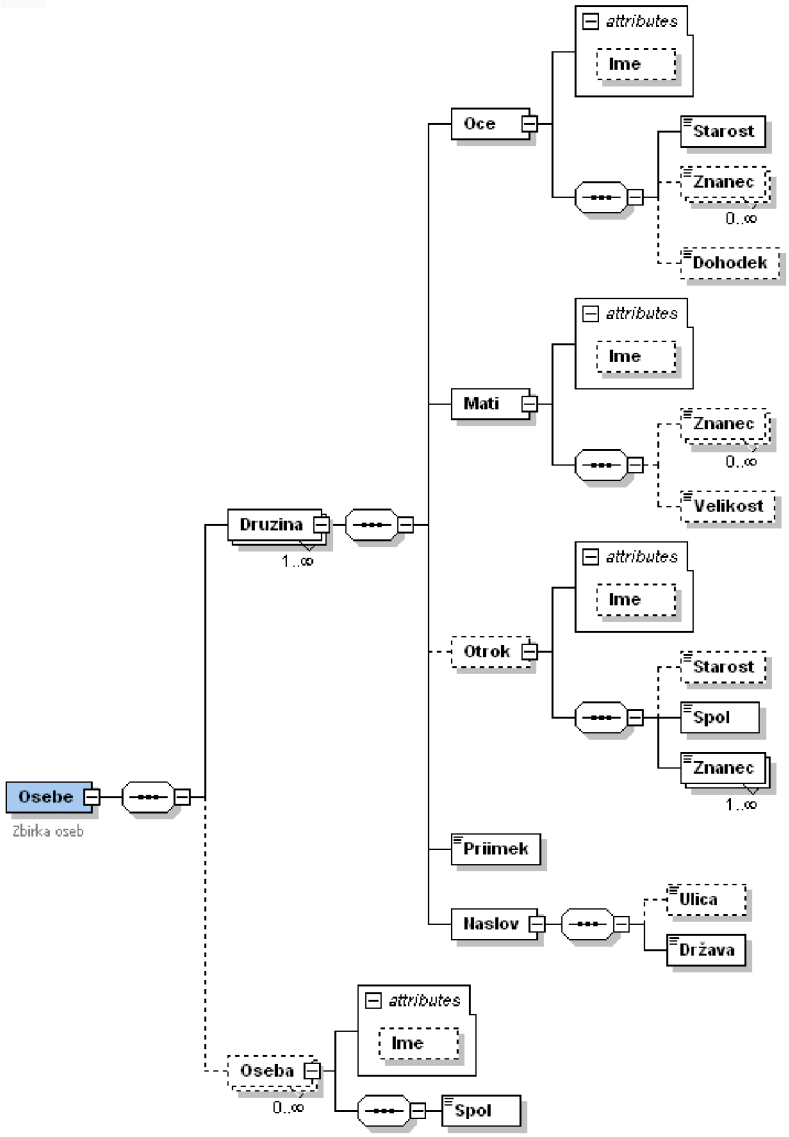
```
        <imaOtroka rdf:resource="#Maja"/>
    </Oce>



    <!-- http://www.semanticweb.org/ontologija.owl#Janko -->

    <Sin rdf:about="#Janko">
        <ime rdf:datatype="&xsd;string">Janko</ime>
        <spol rdf:datatype="&xsd;string">M</spol>
        <jeOtrok rdf:resource="#Ana"/>
        <jeClan rdf:resource="#DruzinaNovak"/>
        <jeOtrok rdf:resource="#Janez"/>
    </Sin>



    <!-- http://www.semanticweb.org/ontologija.owl#Lojze -->

    <Znanec rdf:about="#Lojze">
        <ime rdf:datatype="&xsd;string">Lojze</ime>
        <spol rdf:datatype="&xsd;string">M</spol>
        <jePoznanOd rdf:resource="#Janez"/>
    </Znanec>



    <!-- http://www.semanticweb.org/ontologija.owl#Maja -->

    <Hcerka rdf:about="#Maja">
        <ime rdf:datatype="&xsd;string">Maja</ime>
        <spol rdf:datatype="&xsd;string">&#381;</spol>
        <jeOtrok rdf:resource="#Ana"/>
        <jeClan rdf:resource="#DruzinaNovak"/>
        <jeOtrok rdf:resource="#Janez"/>
        <pozna rdf:resource="#Stojko"/>
    </Hcerka>



    <!-- http://www.semanticweb.org/ontologija.owl#Slovenija -->

    <Drzava rdf:about="#Slovenija">
        <imeDrzave rdf:datatype="&xsd;string">Slovenija</imeDrzave>
    </Drzava>



    <!-- http://www.semanticweb.org/ontologija.owl#Stojko -->

    <Znanec rdf:about="#Stojko">
        <spol rdf:datatype="&xsd;string">M</spol>
        <ime rdf:datatype="&xsd;string">Stojko</ime>
        <jePoznanOd rdf:resource="#Maja"/>
    </Znanec>
</rdf:RDF>
```

```
<!-- Generated by the OWL API (version 2.2.1.1042)
http://owlapi.sourceforge.net -->
```

4.2 XML shema

Primer XML sheme je bil narejen z orodjem XML Spy verzija 2008.

`Shema.xsd`



4.3 XML dokument

Primer XML dokumenta je bil narejen z orodjem XML Spy verzija 2008. Za podlago mu je služila zgornja XML shema.

Instanca.xml

```
<Osebe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Shema.xsd">
      <Druzina>
            <Oce Ime="Janez">
                  <Starost>45</Starost/>
                  <Znanec>Lojze<Znanec/>
                  <Dohodek>20000<Dohodek/>
            </Oce>
            <Mati Ime="Ana">
                  <Velikost>163</Velikost>
            </Mati>
            <Otrok Ime=”Janko”>
                  <Starost>15</Starost>
                  <Spol>M</Spol>
            </Otrok>
            </Otrok Ime=”Maja”>
                  <Spol>Ž</Spol>
                  <Znanec>Stojko<Znanec>
            </Otrok>
            <Priimek>Novak</Priimek>
            <Naslov>
                  <Ulica>Slovenska cesta 33</Ulica>
                  <Država>Slovenija</Država>
            </Naslov>
      </Druzina>
      <Oseba Ime="Stojko">
            <Spol>M</Spol>
      </Oseba>
           <Oseba Ime="Lojze">
            <Spol>M</Spol>
      </Oseba>
</Osebe>
```

5 Viri in literatura

[1] Adam Pease, Why use OWL? Dostopno na: http://www.xfront.com/why-use-owl.html

[2] EON Ontology Alignment Contest. Dostopno na: http://oaei.ontologymatching.org/2004/Contest/

[3] Hannes Bohring, Sören Auer, Mapping XML to OWL Ontologies, Nemčija. Dostopno na: http://www.docstoc.com/docs/3756385/Mapping-XML-to-OWL-Ontologies-Hannes-Bohring-and-S-ren

[4] Holger Knublauch, Semantic XML: Mapping arbitrary XML documents to OWL. Dostopno na: http://composing-the-semantic-web.blogspot.com/2007/11/xmap-mapping-arbitrary-xml-documents-to.html

[5] Ian Horrocks, Peter F. Patel-Schneider, Frank van Harmelen, , From SHIQ and RDF to OWL:The Making of a Web Ontology Language, University of Manchester. Dostopno na: http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/HoPH03a.pdf

[6] Joshua Tauberer, What is RDF. Dostopno na: http://www.xml.com/pub/a/2001/01/24/rdf.html?page=2#rdf

[7] M.A. Amin, J. Morbach, Rules Definition File, Lehrstuhl für Prozesstechnik, RWTH Aachen University, Oktober 2006. Dostopno na: http://www.avt.rwth-aachen.de/AVT/fileadmin/redaktion/ontocape/xml2owl.zip

[8] Marjan Čufer, ONTOLOŠKI SPLET IN ONTOLOŠKI SPLETNI JEZIK (OWL), Jesenice. Dostopno na: http://www.telesat.si/~user239/owl.pdf

[9] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, A Practical Guide To Building OWL Ontologies Using TheProt´eg´e-OWL Plugin and CO-ODE ToolsEdition 1.0, University Of Manchester, August 2004. Dostopno na: http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf

[10] Matthias Ferdinand1, Christian Zirpins1, and David Trastour, Lifting XML Schema to OWL, Hewlett-Packard Laboratories Bristol, Anglija. Dostopno na:

http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/204/fzt-lxs-04.pdf

[11] Ontology. Dostopno na:
http://en.wikipedia.org/wiki/Ontology

[12] Ontology (Computer science). Dostopno na:
http://en.wikipedia.org/wiki/Ontology_(computer_science)

[13] OWL Web Ontology Language. Dostopno na:
http://www.w3.org/TR/owl-features

[14] Quick intro to RDF. Dostopno na:
http://www.rdfabout.com/quickintro.xpd

[15] RDF Primer. Dostopno na:
http://www.w3.org/TR/REC-rdf-syntax/

[16] Resource Description Framework. Dostopno na:
http://en.wikipedia.org/wiki/Resource_Description_Framework

[17] Sabine Massmann, Daniel Engmann, Erhard Rahm, COMA++: Results for the Ontology Alignment Contest OAEI 2006, University of Leipzig. Dostopno na:
http://www.dit.unitn.it/~p2p/OM-2006/9-coma-OAEI%2706.pdf

[18] Spetna stran orodja JXML2OWL Mapper. Dostopno na:
http://jxml2owl.projects.semwebcentral.org/

[19] Spetna stran orodja TopBraid Maestro Edition. Dostopno na:
http://www.topquadrant.com/topbraid/composer/tbc-me.html

[20] Spretna stran XDSImport.Dostopno na:
http://www.incunabulum.de/projects/it/xsdimport/xsdimport-import-xsd-schemas-into-owl

[21] Splošno o orodju XML2OWL s pravili. Dostopno na:
http://www.avt.rwth-aachen.de/AVT/index.php?id=524

[22] Spletna stran orodja XSD2OWL in XML2RDF. Dostopno na:
http://rhizomik.net/redefer

[23] Toni Rodrigues, Pedro Rosa, Jorge Cardoso, MAPPING XML TO EXISTING OWL ONTOLOGIES, University of Madeira, Portugalska.

[24] Uniform Resource Identifier. Dostopno na:

http://en.wikipedia.org/wiki/Uniform_Resource_Identifier

[25] XML Semantic reuse Methodology. Dostopno na:
http://rhizomik.net/~roberto/thesis/html/Methodology.html#XMLSemanticsReuse

[26] Xpath. Dostopno na:
http://www.w3.org/TR/xpath

[27] XSLT. Dostopno na:
http://en.wikipedia.org/wiki/XSLT

IZJAVA

Spodaj podpisani Metod Južna izjavljam, da sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marjana Krisperja.

Ljubljana, april 2009

Metod Južna