Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Dejan Lavbič

MODEL ZA HITER RAZVOJ ONTOLOGIJ S PRISTOPOM OBVLADOVANJA POSLOVNIH PRAVIL V POSLOVNIH APLIKACIJAH

doktorska disertacija

mentor: prof. dr. Marjan Krisper

Ljubljana, 2010

POVZETEK

Vizija semantičnega spleta je predstavitev in medsebojna povezava podatkov na spletu tako, da jih računalniki ne uporabljajo zgolj za namene predstavitve (npr. uporabniku), ampak tudi za avtomatizacijo, integracijo in ponovno uporabo podatkov med različnimi aplikacijami. Od naslednje generacije svetovnega spleta tako pričakujemo avtomatizirane storitve, ki temeljijo na semantiki, izraženi v računalniku razumljivi obliki, uporabo teh podatkov pa bodo omogočali mehanizmi sklepanja in različne hevristične metode. Aplikacije semantičnega spleta, ki temeljijo na ontologijah, so trenutno omejene bolj na akademsko okolje, medtem ko je uspešnih implementacij v gospodarstvu še malo.

Enostavni in uporabniku prijazni pristopi za razvoj ontologij, skupaj s podpornim orodjem, predstavljajo zelo pomemben dejavnik, ki mu moramo nameniti še veliko pozornosti z vidika raziskovalnega dela. Obstoječi pristopi za razvoj ontologij so tehnično zelo zahtevni, saj je učna krivulja uporabnikov zelo dolga in tako niso najbolj primerni za poslovne uporabnike. Večina obstoječih pristopov zahteva pri procesu razvoja dodatno vlogo inženirja znanj, ki skrbi za prenos dejanskega znanja od poslovnih uporabnikov do načrtovalcev ontologij, ki znanje zapišejo v enem od izbranih formalizmov. Vpeljava več abstraktnih ravni obvladovanja kompleksnosti, kot predlagajo sistemi za obvladovanje poslovnih pravil in pristop MDA, se izkažejo za zelo učinkovite pri razvoju in obvladovanju razvoja ontologij za uporabo v poslovnih aplikacijah. Poleg poenostavitve procesa razvoja ontologij je pomemben vidik tudi popolnost razvite ontologije. Številne raziskave so identificirale pogoste napake in anomalije pri razvoju ontologij, ki jih lahko smiselno vključimo v sam proces razvoja in uporabniku pomagamo pri preprečevanju in odpravljanju pogostih načrtovalskih napak.

V okviru doktorske disertacije je bil glavni namen opredeliti inovativen model za hiter razvoj ontologij (ROD), ki je primeren za uporabnike brez poglobljenega tehničnega znanja in znanja načrtovanja ontologij. V okviru omenjenega procesa razvoja so podprte predrazvojne, razvojne in porazvojne aktivnosti od zajema poslovnega slovarja do uporabe razvite ontologije v obliki funkcionalne komponente v informacijskem sistemu. V predlaganem procesu ROD za razvoj ontologij poteka po posameznih korakih v procesu nenehna evaluacija razvite ontologije, ki uporabnika vodi skozi proces in mu daje navodila za izboljšanje kvalitete razvite ontologije. Omenjena funkcionalnost je implementirana s pomočjo indikatorja popolnosti ontologije (OC), ki se uporablja za lažje sledenje korakom procesa ROD in skrbi za nenehno evaluacijo razvite ontologije ter z dinamičnim prilagajanjem uteži v izračunu uporabniku pomaga s priporočili, s katerimi lahko le-ta izboljša ontologijo. Za boljšo podporo procesu ROD je bil razvit tudi prototip orodja IntelliOnto, ki predstavlja grafični vmesnik za razvoj ontologij in neposredno vključuje proces ROD ter indikator popolnosti ontologije. V okviru samega modela ROD so bili identificirani tudi različni načini uporabe funkcionalnih komponente ontologije v informacijskih sistemih, kjer je možna uporaba celotne razvite ontologije ali samo shematskega dela v različnih primerih uporabe. Za lažjo integracijo z obstoječimi podatki je eden od prispevkov tudi opredelitev vmesnikov za neposredno povezovanje delno strukturiranih virov v ontologijo, kjer lahko s pomočjo generičnega pristopa regularnih

izrazov v ontologijo vključimo poljuben delno strukturiran vir; od dokumentov, spletnih strani, podatkovnih baz, datotek CSV ipd. Za potrditev omenjenega pristopa je bila razvita ontologija FITS za podporo trgovanju s finančnimi inštrumenti, ki s svojo modularno zasnovo omogoča ponovno uporabo. Pristop ROD se je v tem primeru izkazal za učinkovitega, saj je bilo za delujočo različico ontologije potrebnih manj iteracij, prav tako pa je kakovost razvite ontologije prej dosegla zahtevani prag potrditve.

S tem omenjena doktorska disertacija celostno opredeljuje model za hiter razvoj ontologij, ki uporabniku brez poglobljenih tehničnih znanj omogoča razvoj ontologij in uporabo številnih prednosti semantičnega spleta ter s tem izdelavo semantično obogatenih aplikacij, ki lahko na zelo intuitiven način uporabljajo številne podatkovne vire, tudi delno strukturirane.

Ključne besede: ontologije, semantični splet, metodologije za razvoj ontologij, poslovna pravila, evaluacija ontologij, hiter razvoj ontologij, ROD, IntelliOnto.

ABSTRACT

The Semantic Web vision is the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes (e.g. user), but for automation, integration and reuse of data across various applications. Next generation of the Web is expected to provide automated services based on machine processable semantics of data, reasoning techniques and heuristics that make use of these data. The applications of ontologies are mainly restricted to academia while successful employment in business environments is rare.

The simplicity of using approaches for ontology construction and accompanying tool support is an important issue which needs a lot of attention and further work. Current approaches in ontology development are technically very demanding and require long learning curve and are therefore inappropriate for business users. In majority of existing approaches an additional role of knowledge engineer is required for mediation between actual knowledge that business users possess and ontology engineers who encode knowledge in one of selected formalisms. Introduction of several abstraction layers as suggested in systems for business rules manipulation and MDA approach has turned out to be very effective in development of ontologies and using it in business applications. Besides simplifying the process of ontology construction we also have to focus on very important aspect of ontology completeness. Several researches have discussed error free ontologies and identified frequent errors and anomalies in ontology development, which is advised to be included in the development process and therefore aiding users at prevention and elimination of repeated design errors.

The main purpose of this thesis was to define innovative model for rapid ontology development (ROD) that is suitable for users without extensive technical knowledge and knowledge of ontology design. The proposed process includes pre-development, development and post-development activities, from business vocabulary acquisition to employment of developed ontology as a functional component in information system. ROD process also includes constant evaluation of developed ontologies which is conducted at every step of the process and gives user recommendations on how to improve the quality of developing ontology. This functionality is implemented using ontology completeness indicator (OC) that is used for following the steps of ROD process in simplified manner. Constant evaluation of developing ontology is also performed with dynamic adaption of weights in calculation which in turn aids user with recommendations on how to improve ontology. Along with the development of ROD model several possibilities of using ontology as a functional component was investigated. Ontology can be used as whole or just partly by using only schematic part. To improve integration with existing data sources, interfaces for direct linking of semi-structured data in ontology were developed. This was accomplished with a generic approach of regular expressions that enables us to connect to any semi-structured source of data, e.g. document, web page, data base, CSV file etc. For the evaluation purposes of proposed approach FITS ontology for trading with financial instruments was developed. Its generic design enables users very straightforward reuse. In this experiment ROD approach turned out to be very effective. It took less iterations to

develop a working version of ontology and the required confirmation level of ontology quality was also achieved earlier.

By this, this thesis represents integral model for rapid ontology development, that enables users without extensive technical knowledge development of ontology. By doing this they have an ability to employ the advantages of semantic web in building semantically enabled application that can very intuitively reuse data from several (also semi-structured) data sources.

Index terms: ontologies, semantic web, methodologies for ontology development, business rules, ontology evaluation, rapid ontology development, ROD, IntelliOnto.

IZJAVA O AVTORSTVU

doktorske disertacije

Spodaj podpisani Dejan Lavbič, z vpisno številko 63990260, sem avtor doktorske disertacije z naslovom

Model za hiter razvoj ontologij s pristopom obvladovanja poslovnih pravil v poslovnih aplikacijah.

S svojim podpisom zagotavljam, da:

- sem doktorsko disertacijo izdelal samostojno pod vodstvom mentorja izr. prof. Marjana Krisperja,
- so elektronska oblika doktorske disertacije, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besed (slov., angl.) identični s tiskano obliko doktorske disertacije
- in soglašam z javno objavo elektronske oblike doktorske disertacije v zbirki "Dela FRI".

V Ljubljani, dne 22. 2. 2010. Podpis avtorja: _____

ZAHVALA

Najprej se zahvaljujem mentorju prof. dr. Marjanu Krisperju, ki me je sprejel za mladega raziskovalca in mi omogočil raziskovalno svobodo in me usmeril na izhodišča, na katerih sem začel graditi svojo doktorsko disertacijo. Hvaležen sem mu za pomoč pri podiplomskem študiju in za vodenje ter usmerjanje pri izdelavi doktorske disertacije.

Delo takšnega obsega seveda ne bi bilo mogoče opraviti brez pomoči naklonjenega okolja, ki sem ga našel na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Zahvaljujem se kolegom sodelavcem Laboratorija za informatiko in Laboratorija za podatkovne tehnologije. Poleg prijetnih konstruktivnih debat so mi s številnimi koristnimi nasveti in idejami dali dodaten motiv in pomoč pri delu.

Največja zahvala gre moji družini, ki ni izgubila upanja in se je morala marsičemu odreči in veliko potrpeti, da je to delo lahko nastalo. Iskrena hvala mojim staršem za vso skrb, spodbudo, moralno podporo in motivacijo. Še posebej pa se zahvaljujem Ani, ki je vedno verjela vame in me tudi ob trenutkih krize spodbujala, podpirala in mi stala ob strani. Posebna zahvala gre tudi moji mali Pii, ki v času pisanja doktorske disertacije še ni bila na tem svetu, a je bila vedno prisotna v mojih mislih.

Hvala tudi vsem ostalim, ki jih nisem omenil in so mi pomagali ter verjeli vame, tudi ko niso več razumeli, kaj točno počnem.

Not everything that can be counted counts, and not everything that counts can be counted.

(Albert Einstein)

KAZALO

1 Uvod **1**
1.1 Opis področja in problematike 1
1.2 Cilj 4
1.3 Delovne hipoteze 5
1.3.1 Hipoteza 1 ..... *6*
1.3.2 Hipoteza 2 ..... *6*
1.3.3 Hipoteza 3 ..... *6*
1.3.4 Hipoteza 4 ..... *6*
1.4 Nadaljnja vsebina disertacije 6
2 Raziskovalno področje **9**
2.1 Semantični splet 9
2.1.1 Svetovni splet in motivacija za semantični splet *9*
2.1.2 Opredelitev semantičnega spleta *9*
2.1.3 Sklad tehnologij semantičnega spleta *11*
2.1.4 Različni pogledi in pričakovanja semantičnega spleta *13*
2.2 Ontologije 16
2.2.1 Različni pogledi *16*
2.2.2 Osnovni gradniki in formalna opredelitev *19*
2.2.3 Jeziki za predstavitev ontologij *21*
2.3 Poslovna pravila 27
2.3.1 Opredelitev *27*
2.3.2 Primerjava s semantičnim spletom *29*
2.3.3 Pristop MDA *31*
2.4 Pregled metodologij za razvoj ontologij 33
2.4.1 Cyc *34*
2.4.2 Enterprise Ontology *35*
2.4.3 TOVE *36*
2.4.4 KACTUS *38*
2.4.5 METHONTOLOGY *39*
2.4.6 SENSUS *41*
2.4.7 CommonKADS *42*
2.4.8 DOGMA *42*

2.4.9 OTK Methodology ... *43*
2.4.10 HCONE ... *45*
2.4.11 DILIGENT ... *46*
2.4.12 UPON ... *47*
2.5 Pregled pristopov za evaluacijo ontologij ... 49
3 Model za hiter razvoj ontologij ... 51
3.1 Uvod ... 51
3.2 Proces razvoja ... 52
3.2.1 Predrazvoj ... *53*
3.2.2 Razvoj ... *54*
3.2.3 Porazvoj ... *67*
3.3 Indikator popolnosti ontologije ... 77
3.3.1 Uvod ... *77*
3.3.2 Komponenta TBox ... *85*
3.3.3 Komponenta RBox ... *98*
3.3.4 Komponenta ABox ... *113*
4 Evaluacija ... 142
4.1 Orodje IntelliOnto ... 142
4.1.1 Podrobnosti implementacije ... *144*
4.1.2 Podprti jeziki za zapis ontologije ... *179*
4.2 Primer trgovanja na finančnih trgih ... 179
4.2.1 Metoda ... *179*
4.2.2 Rezultati in diskusija ... *180*
5 Zaključek ... 190
5.1 Povzetek ... 190
5.2 Rezultati ... 191
5.2.1 Hipoteza 1 ... *192*
5.2.2 Hipoteza 2 ... *192*
5.2.3 Hipoteza 3 ... *193*
5.2.4 Hipoteza 4 ... *193*
5.3 Prispevki znanosti ... 194
5.4 Možnosti za nadaljnje delo ... 194
5.4.1 Razširitev indikatorja popolnosti ontologije ... *194*
5.4.2 Izboljšanje podpore orodja ... *195*

5.4.3 Integracija modela za hiter razvoj ontologij s socialnimi omrežji 195
5.4.4 Integracija pristopa s pristopi samodejnega učenja ontologij 195
Literatura 197
Priloga 1: Primer trgovanja s finančnimi inštrumenti 202

SEZNAM SLIK

Slika 1: Sklad semantičnega spleta 11
Slika 2: Kaj je glavno mesto Slovenije? 15
Slika 3: Ideja semantične podatkovne baze 15
Slika 4: Različni pogledi na ontologije 16
Slika 5: Kompleksnost ontologije 16
Slika 6: Homonim "jaguar" v trikotniku pomena 17
Slika 7: Vloga slovarja in ontologije v trikotniku pomena 18
Slika 8: Pogled razvijalcev na podatke 19
Slika 9: Načini formalizacije jezikov za predstavitev ontologij 21
Slika 10: Kronološki pregled razvoja jezikova za predstavitev ontologij 22
Slika 11: Označevalni jeziki za predstavitev ontologij 23
Slika 12: Primer izjave RDF 25
Slika 13: Ravni jezika OWL 26
Slika 14: Vloga SBVR pri pristopu MDA 31
Slika 15: Meta model pristopa SBVR 32
Slika 16: Proces razvoja ontologij ROD v primerjavi z obstoječimi pristopi 34
Slika 17: Razvojne faze metodologije TOVE 38
Slika 18: Zapis ontologije v metodologiji METHONTOLOGY 39
Slika 19: Razvojni proces in življenjski cikel metodologije METHONTOLOGY 40
Slika 20: Pristop SENSUS k razvoju ontologij 42
Slika 21: Pristop DOGMA k razvoju ontologij 43
Slika 22: Ortogonalna procesa razvoja metodologije OTK 43
Slika 23: Koraki meta procesa razvoja metodologije OTK 44
Slika 24: Proces razvoja znanja pri metodologiji OTK 44
Slika 25: Decentraliziran razvoj ontologij HCONE 45
Slika 26: Proces decentraliziranega razvoja ontologij metodologije DILIGENT 47
Slika 27: Razvojni proces metodologije UPON 48
Slika 28: Različna vidika na proces razvoja ontologij ROD 51
Slika 29: Poenostavljen pogled na model za hiter razvoj aplikacij 52
Slika 30: Proces hitrega razvoja ontologij 53
Slika 31: Gradnja ontologije s pomočjo miselnih vzorcev 55
Slika 32: Gradnja ontologije s pomočjo analize besedila 56

Slika 33: Gradnja ontologije s pomočjo enostavnih pravil 56
Slika 34: Gradnja ontologije s pomočjo odločitvenega drevesa 57
Slika 35: Gradnja ontologije s pomočjo odločitvene tabele 57
Slika 36: Medsebojno povezani spletni viri, ki so nastali v okviru projekta W3C SWEO Linking Open Data (LOD) 58
Slika 37: Pretvorba sheme podatkovne baze v ontologijo 59
Slika 38: Pretvorba podatkov CSV v ontologijo 59
Slika 39: Ponovna uporaba obstoječih ontologij 60
Slika 40: Primer obrazca za zajem atoma 63
Slika 41: Zajem pravila v kaj-če obliki 64
Slika 42: Primer pravila v kaj-če obliki 64
Slika 43: Primer skupine pravil v obliki odločitvene tabele 65
Slika 44: Zajem pravila v obliki odločitvenega drevesa 66
Slika 45: Primer pravila v obliki odločitvenega drevesa 67
Slika 46: Integracija podatkov iz obstoječih ontologij 68
Slika 47: Integracija podatkov iz relacijske podatkovne baze 69
Slika 48: Integracija primerkov iz datoteke CSV 70
Slika 49: Integracija primerkov iz delno strukturirane spletne strani 71
Slika 50: Primer opredelitve vhoda in izhoda funkcionalne komponente 73
Slika 51: Primer izbire konceptov, relacij, omejitev, pravil in dinamičnih uvozov 74
Slika 52: Primer izvoza funkcionalne komponente v obliki BPEL 76
Slika 53: Drevo pogojev indikatorja popolnosti OC s semantičnimi testi v listih in pripadajočimi utežmi 79
Slika 54: Vpliv uteži komponent na 1. nivoju (TBox, RBox in ABox) korakov v procesu ROD 82
Slika 55: Vpliv uteži komponent na 2. nivoju (opis, razdelitev, odvečnost, doslednost in anomalija) korakov v procesu ROD 83
Slika 56: Uporabniški vmesnik prikaza rezultata izračuna OC 83
Slika 57: Izračun popolnosti ontologije 84
Slika 58: Pomembnost preverjanja obstoja razreda (opis komponente TBox) pri izračunu OC 85
Slika 59: Primer preverjanja obstoja razreda (opis komponente TBox) pri izračunu OC 85
Slika 60: Pomembnost preverjanja opisa razreda v angleškem naravnem jeziku (opis komponente TBox) pri izračunu OC 87

Slika 61: Primer preverjanja opisa razreda v angleškem naravnem jeziku (opis komponente TBox) pri izračunu OC 87
Slika 62: Pomembnost preverjanja opisa razreda v naravnem jeziku, drugačnem od angleškega (opis komponente TBox) pri izračunu OC 88
Slika 63: Primer preverjanja opisa razreda v naravnem jeziku, drugačnem od angleškega (opis komponente TBox) pri izračunu OC 89
Slika 64: Pomembnost preverjanja skupnih razredov (razdelitev komponente TBox) pri izračunu OC 90
Slika 65: Primer preverjanja skupnih razredov (razdelitev komponente TBox) pri izračunu OC 91
Slika 66: Pomembnost preverjanja identične opredelitve razreda (odvečnost komponente TBox) pri izračunu OC 92
Slika 67: Primer preverjanja identične opredelitve razreda (odvečnost komponente TBox) pri izračunu OC 93
Slika 68: Pomembnost preverjanja odvečnosti podrazreda (odvečnost komponente TBox) pri izračunu OC 94
Slika 69: Primer preverjanja odvečnosti podrazreda (odvečnost komponente TBox) pri izračunu OC 94
Slika 70: Pomembnost preverjanja cikličnih napak (doslednost komponente TBox) pri izračunu OC 95
Slika 71: Primer preverjanja cikličnih napak (doslednost komponente TBox) pri izračunu OC 96
Slika 72: Pomembnost iskanja verige dedovanja (anomalija komponente TBox) pri izračunu OC 97
Slika 73: Primer iskanja verige dedovanja (anomalija komponente TBox) pri izračunu OC 98
Slika 74: Pomembnost preverjanja obstoja lastnosti (opis komponente RBox) pri izračunu OC 99
Slika 75: Primer preverjanja obstoja lastnosti (opis komponente RBox) pri izračunu OC 99
Slika 76: Pomembnost preverjanja opisa lastnosti v angleškem naravnem jeziku (opis komponente RBox) pri izračunu OC 100
Slika 77: Primer preverjanja opisa lastnosti v angleškem naravnem jeziku (opis komponente RBox) pri OC 101
Slika 78: Pomembnost preverjanja opisa lastnosti v naravnem jeziku, drugačnem od angleškega (opis komponente RBox) pri izračunu OC 102
Slika 79: Primer preverjanja opisa lastnosti v naravnem jeziku, drugačnem od angleškega (opis komponente RBox) pri izračunu OC 103

Slika 80: Pomembnost preverjanja opisa razreda (opis komponente RBox) pri izračunu OC 104
Slika 81: Primer preverjanja opisa razreda (opis komponente RBox) pri izračunu OC 104
Slika 82: Pomembnost preverjanja obstoja inverznih lastnosti (razdelitev komponente RBox) pri izračunu OC 105
Slika 83: Primer preverjanja obstoja inverznih lastnosti (razdelitev komponente RBox) pri izračunu OC 105
Slika 84: Pomembnost iskanja poti med razredi (razdelitev komponente RBox) pri izračunu OC 107
Slika 85: Primer iskanja poti med razredi (razdelitev komponente RBox) pri izračunu OC 107
Slika 86: Pomembnost preverjanja identične opredelitve lastnosti (odvečnost komponente RBox) pri izračunu OC 108
Slika 87: Primer preverjanja identične opredelitve lastnosti (odvečnost komponente RBox) pri izračunu OC 109
Slika 88: Pomembnost preverjanja odvečnosti podlastnosti (odvečnost komponente RBox) pri izračunu OC 110
Slika 89: Primer preverjanja odvečnosti podlastnosti (odvečnost komponente RBox) pri izračunu OC 110
Slika 90: Pomembnost iskanja ponavljajoče se skupine lastnosti (anomalija komponente RBox) pri izračunu OC 111
Slika 91: Primer iskanja ponavljajoče se skupine lastnosti (anomalija komponente RBox) pri izračunu OC v obliki popolnega dvodelnega grafa 112
Slika 92: Primer odprave ponavljajoče se skupine lastnosti (anomalija komponente RBox) pri izračunu OC v obliki popolnega dvodelnega grafa 112
Slika 93: Pomembnost preverjanja formalnega opisa primerkov (opis komponente ABox) pri izračunu OC 115
Slika 94: Primer preverjanja formalnega opisa primerkov (opis komponente ABox) pri izračunu OC 115
Slika 95: Pomembnost preverjanja obstoja primerkov razreda (opis komponente ABox) pri izračunu OC 116
Slika 96: Primer preverjanje obstoja primerkov razreda (opis komponente ABox) pri izračunu OC 116
Slika 97: Pomembnost preverjanja obstoja primerkov lastnosti (opis komponente ABox) pri izračunu OC 117
Slika 98: Primer preverjanja obstoja primerkov lastnosti (opis komponente ABox) pri izračunu OC 118

Slika 99: Pomembnost preverjanja skupnih primerkov (razdelitev komponente ABox) pri izračunu OC 119
Slika 100: Primer preverjanja skupnih primerkov (razdelitev komponente ABox) pri izračunu OC 120
Slika 101: Pomembnost preverjanja zunanjih primerkov (razdelitev komponente ABox) pri izračunu OC 121
Slika 102: Primer preverjanja zunanjih primerkov (razdelitev komponente ABox) pri izračunu OC 121
Slika 103: Primer odpravljenega problema pri preverjanju zunanjih primerkov (razdelitev komponente ABox) pri izračunu OC 121
Slika 104: Vsebinska vprašanja za identifikacijo relacij podrazred 122
Slika 105: Različni scenariji pri identifikaciji relacij tipa podrazred 123
Slika 106: Pomembnost predlaganja relacij podrazred (razdelitev komponente ABox) pri izračunu OC 124
Slika 107: Primer predlaganja relacij podrazred (razdelitev komponente ABox) pri izračunu OC 124
Slika 108: Scenarij pri identifikaciji relacije tipa ekvivalentni razred 125
Slika 109: Pomembnost predlaganja relacij ekvivalentni razred (razdelitev komponente ABox) pri izračunu OC 126
Slika 110: Primer predlaganja relacij ekvivalentni razred (razdelitev komponente ABox) pri izračunu OC 127
Slika 111: Scenarij pri identifikaciji relacij tipa razdružen razred 128
Slika 112: Pomembnost predlaganja relacij razdružen razred (razdelitev komponente ABox) pri izračunu OC 129
Slika 113: Primer predlaganja relacij razdružen razred (razdelitev komponente ABox) pri izračunu OC 129
Slika 114: Vsebinska vprašanja za identifikacijo relacij podlastnost 130
Slika 115: Različni scenariji pri identifikaciji relacij tipa podlastnost 130
Slika 116: Pomembnost predlaganja relacij podlastnost (razdelitev komponente ABox) pri izračunu OC 131
Slika 117: Primer predlaganja relacij podlastnost (razdelitev komponente ABox) pri izračunu OC 132
Slika 118: Scenarij pri identifikaciji relacije tipa ekvivalentna lastnost 133
Slika 119: Pomembnost predlaganja relacij ekvivalentna lastnost (razdelitev komponente ABox) pri izračunu OC 134
Slika 120: Primer predlaganja relacij ekvivalentna lastnost (razdelitev komponente ABox) pri izračunu OC 134

Slika 121: Pomembnost preverjanja identične opredelitve primerka (odvečnost komponente ABox) pri izračunu OC 136
Slika 122: Primer preverjanja identične opredelitve primerka (odvečnost komponente ABox) pri izračunu OC 136
Slika 123: Pomembnost preverjanja odvečnosti primerka (odvečnost komponente ABox) pri izračunu OC 138
Slika 124: Primer preverjanja pomembnosti odvečnosti primerka (odvečnost komponente ABox) pri izračunu OC 138
Slika 125: Pomembnost iskanja neuporabljenih razredov (anomalija komponente ABox) pri izračunu OC 139
Slika 126: Primer iskanja neuporabljenih razredov (anomalija komponente ABox) pri izračunu OC 140
Slika 127: Pomembnost iskanja neuporabljenih lastnosti (anomalija komponente ABox) pri izračunu OC 141
Slika 128: Primer iskanja neuporabljenih lastnosti (anomalija komponente ABox) pri izračunu OC 141
Slika 129: Preslikava problemske domene v ontologijo s pomočjo procesa ROD 142
Slika 130: Uporabniški vmesnik orodja IntelliOnto 143
Slika 131: Korak 1 dodajanja omejitev 145
Slika 132: Korak 2 dodajanja omejitev 145
Slika 133: Korak 3 dodajanja omejitev 145
Slika 134: Korak 1.1 dodajanja omejitev 146
Slika 135: Korak 1.2 dodajanja omejitev 146
Slika 136: Korak 1.2.1 dodajanja omejitev 147
Slika 137: Korak 1.2.1.1 dodajanja omejitev 147
Slika 138: Korak 1.2.1.2 dodajanje omejitev 148
Slika 139: Korak 1.2.2 dodajanja omejitev 148
Slika 140: Korak 1.2.2.1 dodajanja omejitev 149
Slika 141: Korak 1.2.2.2 dodajanja omejitev 149
Slika 142: Korak 1.2.3 dodajanja omejitev 150
Slika 143: Korak 1.2.3.1 dodajanja omejitev 150
Slika 144: Korak 1.2.3.2 dodajanja omejitev 151
Slika 145: Korak 1.3 dodajanja omejitev 152
Slika 146: Korak 1.3.1 dodajanja omejitev 153
Slika 147: Korak 1.3.2 dodajanja omejitev 153
Slika 148: Korak 2.1 dodajanja omejitev 154

Slika 149: Korak 2.2 dodajanja omejitev 154
Slika 150: Korak 2.2.1 dodajanja omejitev 155
Slika 151: Korak 2.2.2 dodajanja omejitev 155
Slika 152: Korak 2.2.2.1 dodajanja omejitev 156
Slika 153: Korak 2.2.2.2 dodajanja omejitev 156
Slika 154: Korak 2.2.2.3 dodajanja omejitev 157
Slika 155: Korak 3.2 dodajanja omejitev 157
Slika 156: Korak A vnosa omejitev 158
Slika 157: Korak A.A vnosa omejitev 158
Slika 158: Korak A.A.A vnosa omejitev 159
Slika 159: Korak A.A.A.A vnosa omejitev 159
Slika 160: Korak A.A.B vnosa omejitev 159
Slika 161: Korak A.A.B.A vnosa omejitev 160
Slika 162: Korak A.A.B.B vnosa omejitev 160
Slika 163: Korak A.A.B.C vnosa omejitev 160
Slika 164: Korak A.A.A.A.A vnosa omejitev 161
Slika 165: Korak A.C vnosa omejitev 161
Slika 166: Korak A.C.A vnosa omejitev 161
Slika 167: Korak A.C.B vnosa omejitev 162
Slika 168: Korak A.C.A.A vnosa omejitev 162
Slika 169: Korak A.C.A.A.A vnosa omejitev 162
Slika 170: Korak A.C.A.A.A.A vnosa omejitev 163
Slika 171: Korak A.C.A.A.A.A.A vnosa omejitev 163
Slika 172: Korak A.C.A.A.A.A.A.A vnosa omejitev 163
Slika 173: Korak A.C.A.A.A.A.B vnosa omejitev 164
Slika 174: Korak A.C.A.A.A.A.B vnosa omejitev 164
Slika 175: Korak A.C.B.A vnosa omejitev 165
Slika 176: Korak A.C.B.A.A vnosa omejitev 165
Slika 177: Korak A.C.B.A.A.A vnosa omejitev 165
Slika 178: Korak A.C.B.A.A.A.A vnosa omejitev 166
Slika 179: Korak A.C.B.A.A.A.A.A vnosa omejitev 166
Slika 180: Korak A.D vnosa omejitev 167
Slika 181: Korak A.D.A vnosa omejitev 167
Slika 182: Korak A.D.A.A vnosa omejitev 167

Slika 183: Korak A.D.A.A.A vnosa omejitev ... 168
Slika 184: Korak A.D.A.A.A.A vnosa omejitev ... 168
Slika 185: Korak A.D.A.A.A.A.A vnosa omejitev ... 168
Slika 186: Korak A.D.B vnosa omejitev ... 169
Slika 187: Korak A.D.B.A vnosa omejitev ... 169
Slika 188: Korak A.D.B.A.A vnosa omejitev ... 169
Slika 189: Korak A.D.B.A.A.A vnosa omejitev ... 170
Slika 190: Korak A.D.B.A.A.A.A vnosa omejitev ... 170
Slika 191: Korak A.D.C vnosa omejitev ... 170
Slika 192: Korak A.D.C.A vnosa omejitev ... 170
Slika 193: Korak A.D.C.A.A vnosa omejitev ... 171
Slika 194: Korak A.D.C.A.A.A vnosa omejitev ... 171
Slika 195: Zajem pravil za določitev barve telesa svečnika z modulom odločitvenega drevesa ... 172
Slika 196: Zajem pravil za proženje prodajnih in nakupnih signalov ... 174
Slika 197: Ocena popolnosti ontologije in število elementov v ontologiji glede na posamezne iteracije in faze v procesu ROD ... 181
Slika 198: Ocena popolnosti ontologije in število elementov v ontologiji glede na posamezne iteracije v ad hoc procesu razvoja ... 181
Slika 199: Ocena popolnosti ontologije v obliki radarskega grafa ... 183
Slika 200: Iteracije faze 2.1 ... 184
Slika 201: Iteracije faze 2.2 ... 184
Slika 202: Iteracije faze 2.3 ... 185
Slika 203: Iteracija faze 2.4 ... 186
Slika 204: Iteracije faze 2.5 ... 186
Slika 205: Iteracije faze 2.6 ... 187
Slika 206: Iteracije faze 3.1 ... 188
Slika 207: Vpliv socialnih omrežij ... 195
Slika 208: Povezanost ontologij pri trgovanju s finančnimi inštrumenti ... 202
Slika 209: Jedrna ontologija, v kateri so predstavljeni osnovni koncepti finančnega trgovanja ... 202
Slika 210: Ontologija za delnico Google Inc. (GOOG) iz vira Yahoo Finance v obliki datoteke CSV ... 203
Slika 211: Ontologija za delnico Apple Inc. (AAPL) iz vira Yahoo Finance v obliki datoteke CSV ... 203

Slika 212: Ontologija vira Yahoo Finance v obliki datoteke CSV 204
Slika 213: Ontologija za delnico Krka d. d. (KRKG) iz vira AmiBroker v obliki zapisa AQH 204
Slika 214: Ontologija vira AmiBroker v AQH zapisu 205
Slika 215: Enostavna trgovalna strategija 206
Slika 216: Trgovalna strategija enostavnih drsečih povprečij 206
Slika 217: Trgovalna strategija japonskih svečnikov 207

SEZNAM TABEL

Tabela 1: Življenjski cikel ontologij v metodologiji HCONE 46
Tabela 2: Zajem pravila v obliki odločitvene tabele 65
Tabela 3: Uteži pri izračunu indikatorja popolnosti (OC) ontologije za vsak posamezen korak v procesu ROD 82

SEZNAM ALGORITMOV

Algoritem 1: Predlagaj entitete in aksiome za vključitev v funkcionalno komponento 75
Algoritem 2: Izračun indikatorja popolnosti OC 80
Algoritem 3: Preveri obstoj razreda 85
Algoritem 4: Preveri opis razreda v angleškem naravnem jeziku 86
Algoritem 5: Preveri opis razreda v naravnem jeziku, drugačnem od angleškega 88
Algoritem 6: Preveri skupne razrede 90
Algoritem 7: Preveri identično opredelitev razreda 92
Algoritem 8: Preveri odvečnost podrazreda 93
Algoritem 9: Poišči ciklične napake 95
Algoritem 10: Poišči verigo dedovanja 97
Algoritem 11: Preveri obstoj lastnosti 99
Algoritem 12: Preveri opis lastnosti v angleškem naravnem jeziku 100
Algoritem 13: Preveri opis lastnosti v naravnem jeziku, drugačnem od angleškega 102
Algoritem 14: Preveri opis razreda 103
Algoritem 15: Preveri obstoj inverzne lastnosti 105
Algoritem 16: Poišči pot med razredi 106
Algoritem 17: Preveri identično opredelitev lastnosti 108
Algoritem 18: Preveri odvečnost podlastnosti 109
Algoritem 19: Poišči ponavljajočo se skupino lastnosti 111
Algoritem 20: Preveri formalen opis primerkov 114
Algoritem 21: Preveri obstoj primerkov razreda 116
Algoritem 22: Preveri obstoj primerkov lastnosti 117
Algoritem 23: Preveri skupne primerke 119
Algoritem 24: Preveri zunanje primerke 120
Algoritem 25: Predlagaj relacije podrazred 123
Algoritem 26: Predlagaj relacije ekvivalentni razred 126
Algoritem 27: Predlagaj relacije razdružen razred 128
Algoritem 28: Predlagaj relacije podlastnost 131
Algoritem 29: Predlagaj relacije ekvivalentna lastnost 133
Algoritem 30: Preveri identično opredelitev primerka 135
Algoritem 31: Preveri odvečnost primerka 137

Algoritem 32: Poišči neuporabljene razrede 139
Algoritem 33: Poišči neuporabljene lastnosti..... 140

1 UVOD

1.1 OPIS PODROČJA IN PROBLEMATIKE

V doktorski disertaciji bom pri opredelitvi modela za hiter razvoj ontologij s pristopom obvladovanja poslovnih pravil za uporabo v poslovnih aplikacijah združil raziskovalna področja **obvladovanja znanja**, **ontologij** in **pristopa obvladovanja poslovnih pravil pri razvoju informacijskih sistemov**. V nadaljevanju sledi kratek opis omenjenih področij, ki je motivacija za omenjen raziskovalni prispevek.

Današnja družba se spreminja iz industrijsko usmerjene v družbo znanja (Gupta and Sharma 2003). Ta sprememba je prisilila organizacije, da začnejo delovati bolj "inteligentno" ter se začnejo gibati izven ustaljenih okvirov delovanja. Znanje v tem kontekstu postaja ključni ekonomski vir organizacije, kjer je obvladovanje le-tega ravno tako pomembno kot npr. obvladovanje finančnega poslovanja ali kadrovanja.

Obvladovanje znanja *(Knowledge Management)* v okviru organizacije predstavlja aktivnosti, ki skrbijo za identifikacijo, ustvarjanje, predstavitev in porazdelitev znanja predvsem zaradi vzpodbujanja ponovne uporabe in učenja na ravni celotne organizacije. Ključna naloga današnjih sistemov za obvladovanje znanja je predvsem podpora ponovni uporabi znanja. Razloge za vzpostavitev takšnega sistema v organizacijah pa iščemo v:

- zmanjšanju stroškov v zvezi s podvajanjem podatkov,
- izgubi znanja, ko zaposleni zapustijo organizacijo in
- potrebnem času za iskanje pravih odgovorov.

Uspešne strategije obvladovanja znanja se ukvarjajo z vidiki same organizacije, ljudi, tehnologij in skupne kulture (Sure 2003). V tem kontekstu ponovna uporaba znanja ne zajema zgolj komunikacije med ljudmi, ampak tudi med ljudmi in tehnologijo (npr. inteligentni programski agenti, ki lahko komunicirajo z ljudmi). Če posplošimo, agenti (ljudje in programski agenti) za souporabo znanja potrebujejo nek skupen jezik.

Ontologije se na področju računalništva uporabljajo za povečanje souporabe in ponovne rabe znanja (Gruber 1993, Fensel 2001). V prvi vrsti predstavljajo skupno in dogovorjeno razumevanje znanja v okviru problemske domene. Poleg tega pa formalizirajo znanje v takšni obliki, da je vzpostavljena povezava človeškega razumevanja simbolov z zmožnostjo računalniškega procesiranja. V takšni obliki se uporabljajo kot skupni jezik pri komunikaciji med agenti.

Pojem ontologije je zgodovinsko gledano na številnih področjih dobil različne opredelitve. Na področju računalništva večina interpretacij izhaja iz razmišljanja filozofa Quina (Quine 1963), ki je trdil, da vse, kar lahko izmerimo, obstaja. Iz tega izhaja tudi Gruberjeva opredelitev (Gruber 1993), ki je ena najbolj sprejetih: "Ontologija je eksplicitna specifikacija konceptualizacije." Konceptualizacijo lahko podrobneje opredelimo kot abstrakten in poenostavljen pogled na svet, ki ga uporabljamo za predstavitvene namene. Razlogi za razvoj ontologij (Noy and McGuinness 2001, Abou-Zeid 2003) se gibljejo od souporabe skupnega razumevanja problemske domene med ljudmi in programskimi

agenti, omogočanja ponovne uporabe domenskega znanja do razjasnitve domnev v okviru določene domene.

Ontologija je tako slovar, ki se uporablja za opis in predstavitev določenega področja, kot tudi pomen tega slovarja. V strokovni in znanstveni literaturi ima izraz ontologija številne pomene, od **taksonomije** (Corcho, et al. 2003, Lavbič 2005, SanJuan and Ibekwe-SanJuan 2006, Veale 2006) (znanje z minimalno hierarhično strukturo), do **slovarja** (Miller 1995, Bechhofer and Goble 2001, Hartel, et al. 2005, Yu 2006) (besede in sinonimi), **tematskih področij** (Park and Hunting 2002, Schweiger, et al. 2003, Alvarado, et al. 2004, Dong and Li 2004, Olsevicova 2006) (podpora organizaciji in navigaciji po večji količini podatkov), **konceptualnega modela** (Mylopoulos 1998, Erdmann and Studer 1999, Jovanović and Gašević 2005, Valarakos, et al. 2006) (bolj kompleksno znanje) in **teorije logike** (Waterson and Preece 1999, Corcho, et al. 2003) (zelo bogato, kompleksno, konsistentno znanje).

Ontologije se uporabljajo v številne namene; na spodnjem seznamu jih je navedenih zgolj nekaj najbolj pogostih:

- procesiranje naravnega jezika in računalniško prevajanje (Dahlgen 1995, Miller 1995, Staab, et al. 1999),
- obvladovanje znanja (Staab, et al. 2001, Davies, et al. 2006),
- pridobivanje in integracija informacij (Wiederhold 1992),
- spletni katalogi (npr. Yahoo (Labrou and Finin 1999)),
- inteligentni iskalniki (Guarino 1998, Heflin and Hendler 2000),
- digitalne knjižnice (Kesseler 1996),
- inteligentni programski agenti (Uschold and Grueninger 1996, Lavbič, et al. 2006) in
- načrtovanje poslovnih procesov (Brambilla, et al. 2006, Rao, et al. 2006).

Obstoječe **metodologije za razvoj ontologij** v splošnem skušajo opredeliti aktivnosti za obvladovanje ontologije, aktivnosti za razvoj ontologije in podporne aktivnosti. Širši pregled metodologij na tem področju lahko najdemo v (Gomez-Perez, et al. 2003, Sure 2003, Davies, et al. 2006), medtem ko kratek opis najbolj pogosto uporabljanih podajamo v nadaljevanju v poglavju 2.4.

Seznam raziskovalnih prispevkov na temo ontologij in obvladovanja znanja s pomočjo tehnologij semantičnega spleta (Uschold and King 1995, Bernaras, et al. 1996, Uschold and Grueninger 1996, Swartout, et al. 1997, Erdmann and Studer 1999, Fernandez-Lopez, et al. 1999, Schreiber, et al. 1999, Cui, et al. 2001, Fensel 2001, Holsapple and Joshi 2002, Jarrar and Meersman 2002, Abou-Zeid 2003, Caplinskas, et al. 2003, Corcho, et al. 2003, Firat 2003, Kotis and Vouros 2003, Sure 2003, Nicola, et al. 2005, Brambilla, et al. 2006, Davies, et al. 2006) nakazuje na visoko dinamiko raziskav, saj gre za relativno novo področje, s katerim se ukvarjajo številni raziskovalci. Nobena od obstoječih metodologij razvoja pa ne pokriva vseh vidikov razvoja ontologij, tako da obstajajo še **številni nerešeni problemi** (Davies, et al. 2006). V znanstveni literaturi med drugim pogrešamo hitre pristope (Rapid Application Development – RAD) (Avison and Fitzgerald 2003) prototipiranja k razvoju in

uporabi ontologij v poslovnih aplikacijah ter pristope analogne agilnim metodologijam pri razvoju programske opreme. Kaže se tudi pomanjkanje pristopov, ki od poznavalcev problemske domene *(domain expert)* ne zahtevajo dobre tehnične podkovanosti v jezikih in tehnikah za zajem znanja. Večina pristopov pri procesu zapisa znanja zahteva dodatnega uporabnika z vlogo inženirja znanja *(knowledge engineer)*, ki domensko znanje prenese v formalno obliko v okviru baze znanja.

Ena od kritik obstoječih metodologij (Sure 2003, Davies, et al. 2006) je tudi ta, da ne posvečajo podpore procesu vzdrževanja razvite ontologije. Osredotočene so zgolj na razvoj ontologije za določeno aplikacijo, razvojni cikel pa se z zadnjo uspešno iteracijo konča. V primeru sprememb so tako domenski eksperti postavljeni pred dejstvo, kjer zopet potrebujejo inženirje znanja, če želijo ontologijo spreminjati, saj nimajo ustreznega znanja jezikov za formalni zapis ontologije. Za rešitev tega problema se kot zanimiv pristop kažejo poslovna pravila in *Model Driven Architecture (**MDA**)* (Frankel 2003), ki jo je opredelil Object Management Group (OMG) ter je podrobneje predstavljena v poglavju 2.3.3.

V zadnjih desetletjih so postala **poslovna pravila** priljubljena predvsem v okviru raziskav s področja informacijskih sistemov zaradi vpeljave fleksibilnosti in sposobnosti enostavnega spreminjanja poslovne logike (Bajec and Krisper 2005). Poslovna pravila lahko opredelimo kot "deklaracijo politike delovanja oz. pogoje, ki morajo biti izpolnjeni", njihova vloga pa je določitev načina izvedbe operativnih odločitev v organizaciji (Kardasis and Loucopoulos 2004). Poslovna pravila, tako kot ontologije, izhajajo s področja umetne inteligence, kjer se uporabljajo kot način predstavitve znanja. Najdemo lahko številne raziskovalne prispevke iz poslovnih pravil na področju podatkovnih baz (Morgenstern 1983, Widom and Ceri 1996), metod za predstavitev poslovnih pravil v podatkovnih modelih (Nijssen and Halpin 1989, Date 2000) in objektno usmerjenega razvoja (Greenspan, et al. 1982, Struck 1999).

Kot je že omenjeno, imata pristop poslovnih pravil in pristop h gradnji ontologij s pomočjo tehnologij semantičnega spleta skupne korenine, vendar naslavljata različno ciljno publiko. Namen pristopa poslovnih pravil je izboljšati komunikacijo med ljudmi, medtem ko z uporabo ontologij na semantičnem spletu želimo izboljšati komunikacijo med računalniki. Ravno zaradi tega se oba pristopa v okviru arhitekture MDA, uvrščata na različne ravni. Ontologija se nahaja na izvajalni ravni, kar je pri MDA analogno ravni PSM medtem ko se poslovni slovar in poslovna pravila nahajajo na najvišji ravni CIM.

Oba pristopa imata tudi isti cilj zajema znanja (semantike), neodvisno od določene aplikacije ali opravila. Tako se razlikujeta od pristopov konceptualnega modeliranja (npr. UML), ki domensko znanje opisujejo za uporabo v določenih aplikacijah. Predvsem ideja o neodvisnosti od določene aplikacije spodbuja ponovno uporabo poslovnih slovarjev in poslovnih pravil ter ontologij. Oba pristopa prav tako uporabljata podobno obliko. Poslovni slovar s poslovnimi pravili in ontologija sta sestavljena iz medsebojno povezanih konceptov in pravil (npr. identiteta, kardinalnost, taksonomija ipd.), ki omejujejo in določajo pomen teh konceptov. Pristop poslovnih pravil se bolj osredotoča na naravni

jezik in človeku lažje berljivo sintakso, medtem ko mora biti pri ontologijah vsak element modela skladen s formalnim jezikom, sicer ga ni mogoče uporabljati v izvajalnem okolju.

1.2 CILJ

Današnje aplikacije, ki temeljijo na ontologijah in tehnologijah semantičnega spleta, so na žalost pogosto omejene na akademska okolja, medtem ko širše uporabe v gospodarstvu ni. Iz besed W. Edwards Deminga "If you can't describe what you are doing as a process, you don't know what you're doing" lahko sklepamo, da je del problema prav gotovo v obstoječih metodologijah, saj je razpoložljivost preverjenih postopkov dober pokazatelj zrelosti tehnologije in postopka uporabe. Uporaba postopkov obvladovanja poslovnih pravil se kaže kot dober pristop k poenostavitvi razvoja in uporabe ontologij v poslovnih aplikacijah, zato bom v okviru doktorske disertacije poskušal opredeliti model za hiter razvoj ontologij s pristopom obvladovanja poslovnih pravil za uporabo v poslovnih aplikacijah in bom poskušal najti odgovore na naslednja raziskovalna vprašanja:

1. **Kateri so primerni pristopi za hiter razvoj delov poslovnih aplikacij, ki temeljijo na ontologijah?**
    a. Ali so obstoječi pristopi in metodologije zadostni pri celostni podpori uporabe ontologij v poslovnih aplikacijah?
    b. Ali je sprejetost ontologij pri razvoju poslovnih aplikacij zadostna?
    c. Kako oz. na kateri ravni ontologijo uporabiti v poslovnih aplikacijah?
2. **Ali lahko uporaba tehnologij semantičnega spleta zmanjša potrebno raven tehničnega znanja uporabnikov pri opredelitvi problemske domene in zajemu znanja?**
    a. Do kakšne mere lahko poslovni uporabnik brez ustreznih tehničnih znanj pripravi ontologijo, primerno za uporabo v poslovni aplikaciji?
    b. Kako doseči raven abstrakcije, kjer poslovni uporabnik vnaša domensko znanje v čim bolj neformalizirani, naravnem jeziku podobni obliki in hkrati doseči zadostno stopnjo formalizacije, da lahko smiselno uporabimo mehanizme sklepanja?
3. **Kako izboljšati pristop gradnje ontologij na semantičnem spletu s pristopom obvladovanja poslovnih pravil pri razvoju informacijskih sistemov?**
    a. Ali obstajajo skupne točke pristopa obvladovanja poslovnih pravil v informacijskih sistemih in uporabe ontologij na semantičnem spletu?
    b. Kakšne so razlike med poslovnimi pravili v informacijskemu sistemu in pravili na semantičnem spletu?
4. **Kako podpreti uporabo različnih virov podatkov pri grajenju poslovnega slovarja oz. ontologije?**
    a. Kakšne težave nastanejo pri združevanju in semantičnem združevanju podatkov?
    b. Kako se lotiti ponovne uporabe ontologij brez podrobnega poznavanja formalnega jezika za zapis?
    c. Kako pomembna je podpora orodja?

Cilj, ki ga poskušam doseči z omenjeno doktorsko disertacijo, je naslednji:

Opredeliti model za hiter razvoj ontologij, ki je primeren za uporabnike brez poglobljenega tehničnega znanja razvoja in načrtovanja ontologij. Pristop bo omogočal podporo celotnemu razvojnemu ciklu in tudi uporabi razvite ontologije v obliki funkcionalne komponente v drugih informacijskih sistemih. Uporabnik bo pri razvoju ontologije samodejno voden, saj bo s pomočjo nenehne evaluacije ontologije možno sledenje korakom procesa razvoja in obvladovanje primernosti ontologije. Namen omenjenega pristopa je olajšati razvoj aplikacij semantičnega spleta in vpeljati tehnologije semantičnega spleta tudi v poslovno okolje. Prispevek bo predvsem na ravni povečane prijaznosti do uporabnika, saj bo zahtevana raven tehničnega znanja, glede na obstoječe pristope, močno zmanjšana.

1.3 DELOVNE HIPOTEZE

Pri izdelavi doktorske disertacije je bil glede na zadane cilje uporabljen naslednji pristop:

1. V okviru **analize obstoječega stanja** sem za potrebe oblikovanja modela in procesa hitre izgradnje ontologij izhajal iz publikacij in raziskav s področja metodologij razvoja ontologij in širšega področja metodologij razvoja programske opreme ter pristopov razvoja poslovnih pravil v informacijskih sistemih.
2. Na podlagi analize in primerjave obstoječih metodologij sem identificiral potrebne korake in elemente, ki so uporabljeni pri pristopu. Namen je izboljšati obstoječe procese gradnje ontologij s smernicami pristopa obvladovanja poslovnih pravil. Rezultat so izdelane **smernice oz. model za hiter razvoj ontologije** s poudarkom na intuitivni uporabi, primerni za poslovnega uporabnika, s katero bo lahko zgradil bazo znanja kot osnovo za uporabo v poslovnih aplikacijah. Le ta bo zapisana v standardiziranih jezikih semantičnega spleta, kar bo omogočalo čim večjo stopnjo izmenljivosti in ponovne uporabljivosti. Pristop bo vključeval tudi čim večjo ponovno uporabljivost virov pri gradnji poslovnega slovarja in pripadajočih pravil:
    a. preučiti je bilo potrebno načine vključevanja obstoječih virov in semantičnega opisovanja teh virov zaradi lažje integracije,
    b. za pogosto uporabljane vire so bili razviti semantični opisi v obliki ontologij virov,
    c. opredeljeni načini ponovne uporabe obstoječih ontologij in
    d. opredeljeni načini uporabe razvite ontologije v poslovnih aplikacijah.
3. Za podporo opredeljenemu procesu je bil **razvit delujoči prototip orodja**, s pomočjo katerega lahko poslovni uporabnik sledi korakom, opredeljenim v modelu, pri razvoju ontologije in uporabe le-te v poslovni aplikaciji. Orodje je osredotočeno na enostavno in intuitivno uporabo, primerno za poslovnega uporabnika, z minimalno ravnjo tehničnih znanj. Podprta je gradnja iz obstoječih virov in opredelitev problemske domene na najvišji abstraktni ravni.

Model bo s pomočjo orodja **preverjen na realnem primeru uporabe** trgovanja s finančnimi inštrumenti.

V okviru doktorske disertacije bomo potrdili naslednje hipoteze, ki so podrobneje opisane v nadaljevanju.

1.3.1 *HIPOTEZA 1*

Na podlagi obstoječih metodologij in najboljših praks pri razvoju ontologij ter pristopov za obvladovanje poslovnih pravil je možno opredeliti model za hiter razvoj ontologij, ki je primeren za uporabnike brez znanja načrtovanja ontologij in poglobljenega tehničnega znanja.

1.3.2 *HIPOTEZA 2*

S pomočjo vpeljave nenehnega preverjanja ustreznosti razvite ontologije izboljšamo hkratno podporo procesu razvoja in izboljšamo končne izdelke.

1.3.3 *HIPOTEZA 3*

S pomočjo pristopa ROD je mogoče izdelati aplikacijo semantičnega spleta, ki je neodvisna od tehnologije in izvajalne platforme ter uporablja podatke iz različnih virov.

1.3.4 *HIPOTEZA 4*

Prototip orodja za podporo procesu ROD omogoča uporabniku sledenje korakom procesa za hiter razvoj in nenehno evaluacijo razvite ontologije.

1.4 NADALJNJA VSEBINA DISERTACIJE

V **poglavju 2** so predstavljena raziskovalna področja kot osnova in temelj za omenjeno doktorsko disertacijo. Najprej je prikazana ideja in motivacija za nastanek **semantičnega spleta** (poglavje 2.1, str. 9) kot nadgradnja obstoječega spleta. Poleg opredelitve in različnih interpretacij semantičnega spleta so v okviru sklada tehnologij predstavljeni najbolj pomembni gradniki in njihova medsebojna povezanost. Podrobno so raziskane tudi pogoste nejasnosti o različnih pristopih, tehnologijah za opisovanje, semantičnih vmesnikih, iskalnih in kontekstnih tehnologijah ter semantičnih podatkovnih bazah. V nadaljevanju je poudarek na **ontologijah** (poglavje 2.2, str. 16) kot osnovnih gradnikih semantičnega spleta. S pomočjo različnih komponent pomena (izraz, predstava in stvar v realnem svetu) je v trikotniku pomena predstavljen pomen ontologij pri predstavitvi znanja v računalništvu in kako se je skozi čas spreminjal pogled razvijalcev na podatke. Podana je tudi formalna opredelitev ontologije ter osnovni gradniki (koncepti, lastnosti, primerki in aksiomi). V nadaljevanju je poudarek na jezikih za predstavitev ontologij, in sicer v dveh dimenzijah: (1) načinu formalizacije in (2) kronološkem nastanku. Osnovni jeziki za predstavitev ontologij so zgolj omenjeni, medtem ko so podrobneje opisani označevalni jeziki, ki temeljijo na sintaktičnem jeziku XML. Ti so v okviru semantičnega spleta tudi največ v uporabi. Pomembno raziskovalno področje so tudi **poslovna pravila** in sistemi za obvladovanje poslovnih pravil (poglavje 2.3, str. 27), katerih razvoj in uporaba je predstavljena skozi čas. Predvsem zaradi poenostavitev razvoja ontologij so se

že pojavili nekateri pristopi za uporabo opredelitev ontologij na semantičnem spletu, saj je med obema pristopoma (semantični splet in poslovna pravila) veliko podobnosti. Predstavljen je tudi pristop MDA, ki je bil v okviru projekta SBVR uspešno uporabljen za semantično modeliranje poslovnih pravil. Sledi pregled obstoječih raziskav, ki je razdeljen na dva dela, in sicer pregled metodologij za razvoj ontologij v poglavju 2.4 na strani 33 in pregled pristopov za evaluacijo ontologij v poglavju 2.5 na strani 49. **Pregled pristopov za razvoj ontologij** je rezultat podrobne analize najbolj tipičnih predstavnikov posameznih pristopov in metodologij za razvoj ontologij. Vseh opisanih pristopov je 12, identificirane pa so tudi prednosti in slabosti posameznih pristopov. V okviru omenjenega podpoglavja je podana tudi primerjava predlaganega pristopa ROD z obstoječimi pristopi glede na atributa primernosti za poslovnega uporabnika in kompleksnosti razvite ontologije. Pregled pristopov za evaluacijo ontologij povzema vse pristope, najdene v literaturi, v 4 skupine, ki so kasneje podrobno opisane in predstavljene s posameznimi predstavniki.

V **poglavju 3** je predstavljen **model za hiter razvoj ontologij (ROD)** kot predlagan metodološki pristop k razvoju ontologij in semantično obogatenih aplikacij, primernih za semantični splet. V **uvodu** (poglavje 3.1, str. 51) sta predstavljena dva vidika razvoja ontologij po pristopu ROD: (1) vidik ponovne uporabe in (2) procesni vidik. V nadaljevanju je poudarek na procesnem vidiku, kjer je v poglavju 3.2 na strani 52 predstavljen **proces razvoja**. Proces ROD je razdeljen v faze pred razvoja, razvoja in po razvoja s podrobnim opisom posameznih aktivnosti, ki so vsebovane v omenjenih fazah. V okviru **predrazvoja** (poglavje 3.2.1, str. 53) je predstavljena študija izvedljivosti, medtem ko **razvoj** (poglavje 0, str. 54) sestavljajo številne aktivnosti. V okviru zajema poslovnega slovarja (poglavje 3.2.2.1, str. 54) so predstavljeni neformalizirani pristopi k opredelitvi problemske domene, semantično opisovanje virov in problemi pri združevanju ontologij, s katerimi se uporabnik sreča v začetnih korakih razvoja ontologij. Naslednji koraki so identifikacija seznama primerkov konceptov in lastnosti (poglavje 3.2.2.2, str. 61), identifikacije taksonomije (poglavje 3.2.2.3, str. 62), identifikacija ad hoc relacij (poglavje 3.2.2.4, str. 62), opis atributov in relacij konceptov (poglavje 3.2.2.5, str. 62). Zadnji korak v fazi razvoja pa je dodajanje kompleksnih omejitev in pravil (poglavje 3.2.2.6, str. 63), kjer so podrobno predstavljeni različni načini zajema pravil v obliki kaj-če pravil, odločitvene tabele in odločitvenega drevesa. V okviru **porazvoja** (poglavje 3.2.3, str. 67) so ravno tako opredeljene številne aktivnosti, ki pripomorejo k aplikabilnosti razvite ontologije v obliki funkcionalne komponente izbranega informacijskega sistema. Prva od aktivnosti je povezava slovarja s podatki (poglavje 3.2.3.1, str. 67), kjer razvito shemo ontologije povežemo z dejanskimi podatki. Predstavljene so različne možnosti uporabe in uvoza primerkov iz obstoječih ontologij, relacijskih oz. analitičnih podatkovnih baz, datotek CSV in delno nestrukturiranih virov. Sledi aktivnost kompozicije funkcionalne komponente (poglavje 3.2.3.2, str. 72), kjer je najprej potrebno opredeliti V/I, izbrati koncepte, relacije, omejitve in pravila ter na koncu identificirati cilj. Celoten proces razvoj ontologij je podprt z **indikatorjem popolnosti ontologij (OC)** (poglavje 3.3, str. 77), ki uporabnikom omogoča lažje sledenje procesu razvoja in ga z nenehnim nadzorovanjem kvalitete razvite ontologije usmerja pri odpravi napak in večanju aplikabilnosti razvite ontologije. V uvodu (poglavje 3.3.1, str. 77) je predstavljena formalna opredelitev indikatorja popolnosti in

način izračuna. Prikazano je tudi drevo pogojev indikatorjev popolnosti OC s semantičnimi testi v listih in pripadajočimi utežmi. V nadaljevanju so predstavljeni posamezni semantični testi, ki se izvajajo nad ontologijo po posameznih komponentah ontologije: TBox (poglavje 0, str. 85), RBox (poglavje 3.3.3, str. 98) in ABox (poglavje 3.3.4, str. 113). Klasifikacija na drugem nivoju pa je razdeljena na opis, razdelitev, odvečnost, doslednost in anomalijo.

Poglavje 4 opisuje podrobnosti **evaluacije** oz. preverjanja ustreznosti modela ROD, predstavljenega v 3. poglavju. Najprej so predstavljene podrobnosti **prototipa orodja IntelliOnto** (poglavje 4.1, str. 142), ki nudi uporabniku podporo pri razvoju ontologije od začetka, zajema poslovnega slovarja, do konca, izdelave funkcionalne komponente. Med podrobnostmi implementacije sta prikazana dva načina vnosa naprednih omejitev: (1) sprehod po semantični mreži in dodajanje omejitev (poglavje 4.1.1.1.1, str. 145) in (2) vnos omejitev v obliki atomov pravil (poglavje 4.1.1.1.2, str. 158). V nadaljevanju je tudi prikazan način zajema pravil v vseh treh oblikah v poglavju 4.1.1.2 na strani 171 ter podprti jeziki za zapis ontologije (poglavje 4.1.2, str. 179). Drugi del evaluacije je izveden eksperiment na **primeru trgovanja na finančnih trgih** (poglavje 4.2, str. 179). Pristop ROD je bil primerjan z ad hoc pristopom razvoja ontologij, ki temelji na dobrih praksah. Zahteve so bile razvoj ontologije, ki je jedro aplikacije semantičnega spleta za podporo finančnemu trgovanju z možnostjo opredelitve poljubnih trgovalnih strategij. Pri razvoju je bilo zrno opazovanja ena iteracija, kjer se je spremljala popolnost ontologije, rast števila aksiomov po posameznih iteracijah ter potrebno število iteracij za izpolnitev uporabniških zahtev.

V **poglavju 5** so najprej predstavljeni **povzetki dela** (poglavje 5.1, str. 190), opravljenega v doktorski disertaciji. V nadaljevanju sledijo **rezultati in strnjeni prispevki znanosti** (poglavje 5.2 na str. 191 in poglavje 5.3 na str. 194), na koncu pa še **vpliv in možnosti nadaljnjih raziskav** (poglavje 5.3, str. 194).

Priloga 1 na strani 202 vsebuje dodatno gradivo o primeru trgovanja s finančnimi inštrumenti in celotno ontologijo FITS, vključno z opisi, razvito v okviru doktorske disertacije.

2 RAZISKOVALNO PODROČJE

2.1 SEMANTIČNI SPLET

2.1.1 *SVETOVNI SPLET IN MOTIVACIJA ZA SEMANTIČNI SPLET*

Ob začetkih razvoja računalništva svetovni splet še ni bil prisoten. Splet je kot koncept začel razvijati leta 1989 Tim Berners-Lee, ki je bil takrat zaposlen pri CERN-u, evropski organizaciji za fizikalne raziskave. Tehnično osebje pri CERN-u je nujno potrebovalo storitev skupne rabe dokumentov med različnimi računalniki. Tim Berners-Lee je predhodno leta 1989 razvil že več takšnih sistemov in na podlagi tega se je razvil svetovni splet. Tehnična zasnova je bila relativno enostavna, kar je pripomoglo pri obstoju in pridobivanju kritične mase uporabnikov.

Osnovna ideja Berners-Leeja je bila omogočiti vsakemu uporabniku objavo podatkov na svojem računalniku tako, da so lahko drugi uporabniki dostopali do njih iz poljubne lokacije. Upal je, da bodo nekoč računalniki tudi sami sposobni uporabljati podatke na svetovnem spletu. V končni fazi si je želel, da bo na tej ideji nastalo učinkovito sodelovanje med ljudmi s pomočjo računalnikov:

> *I have always imagined the information space as something to which everyone has immediate and intuitive access, and not just to browse but to create …*
>
> *Machines become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers.*
>
> *… when [the Semantic Web] does [emerge], the day-to-day mechanisms of trade, bureaucracy, and our daily lives will be handled by machines talking to machines, leaving people to provide the inspiration and intuition.*
>
> *(Berners-Lee, 2000)*

2.1.2 *OPREDELITEV SEMANTIČNEGA SPLETA*

Izraz **semantika** se nanaša na pomen oz. ga različni viri opredeljujejo kot:

- "Nauk o pomenu besed, pomenoslovje oz. pomen, pomenskost" (Inštitut za slovenski jezik Frana Ramovša 2002) in
- "O ali v povezavi s študijo o pomenu in spremembah pomena" (Princeton University 2006).

Pri semantičnem spletu semantika označuje dejstvo, da lahko pomen podatkov na spletu poleg človeških uporabnikov odkrijejo tudi računalniki. V nasprotju s to ugotovitvijo dandanes večino pomena na svetovnem spletu interpretirajo uporabniki sami, ki pregledujejo spletne strani, oznake in povezave ter tehnični uporabniki, ki pripravljajo specializirane programe za delo s takšnimi podatki. Izraz **semantični splet** predstavlja vizijo, kjer bi računalniki (programska oprema) kot tudi človeški uporabniki lahko iskali,

prebirali, razumeli in uporabljali podatke s celotnega svetovnega spleta pri uresničevanju ciljev, ki si jih zastavijo uporabniki (Passin 2004).

Obstajajo številni pogledi na vizijo semantičnega spleta:

- Ideja **računalniku razumljivih podatkov** (W3C 2003) govori o podatkih na svetovnem spletu, ki so opredeljeni in povezani na takšen način, da jih računalniki ne uporabljajo zgolj za prikaz, ampak tudi za avtomatizacijo, integracijo in ponovno uporabo pri različnih aplikacijah.
- Naloga semantičnega spleta je pripraviti obstoječi splet v bolj računalniku razumljivo obliko, da bodo lahko **inteligentni agenti** (Seidenberg and Rector 2006) uporabljali in manipulirali s podatki.
- Ideja **porazdeljene podatkovne baze** (W3C 2000) govori o tem, da ima semantični splet pri podatkih podobno vlogo, kot jo je imel jezik HTML pri tekstovnih informacijskih sistemih – zagotoviti zadostno raven prilagodljivosti, ki nam omogoča predstavitev podatkovnih baz in logičnih pravil in medsebojno povezovanje z namenom pridobivanja dodatne vrednosti.
- Tim Berners-Lee trdi, da je semantični splet infrastruktura in ne aplikacija, zato je dejanski izziv pomanjkanje **avtomatizirane infrastrukture** v obstoječi različici svetovnega spleta.
- Ideja semantičnega spleta je **nadgradnja obstoječega spleta z opisi**, ki so zajeti v računalniško razumljivi obliki in medsebojno povezani.
- Vidik **izboljšanega iskanja**, kjer je dostop do spletnih virov mogoč po vsebini in ne zgolj ključnih besedah. Glavni cilj je izgradnja strukturiranega indeksa spletne strani.
- Semantični splet se ne uporablja zgolj za dostop do statičnih dokumentov, ki združujejo uporabne informacije, ampak tudi do storitev, ki omogočajo uporaben način procesiranja. S pomočjo semantičnega spleta lahko razširimo obstoječe **spletne storitve** in jih avtomatiziramo do te mere, da bodo programski agenti samodejno izvajali določena opravila.

Semantični splet lahko preprosto opišemo tudi z osnovnimi lastnostmi, ki jih pogosto povezujemo s semantičnim spletom (Allemang and Hendler 2008). **Slogan AAA**[1] je eden od osnovnih načel spleta na splošno, ki še posebej velja za semantični splet. Kot posledica slogana AAA lahko v vsakem trenutku v obstoječo bazo znanja dodamo novo dejstvo, kar posledično pomeni, da lahko v vsakem trenutku obstajajo informacije, ki nam niso znane. Zaradi tega semantični splet temelji na **predpostavki odprtega sveta**. Akterji pri dodajanju vsebine na semantičnem spletu niso vedno medsebojno usklajeni, zato se lahko zgodi, da isto entiteto najdemo opisano z različnimi imeni, tako da velja **predpostavka neenoličnega poimenovanja**. Pri semantičnem spletu je zelo pomemben tudi mrežni efekt, t. j. lastnost spleta, da se organsko širi naprej. Vrednost vključitve v omrežje se povečuje s številom sodelujočih, kar se posledično izkaže v povečanem krogu sodelovanja.

[1] Slogan **AAA** (ang. Anyone can say Anything about Any topic) pravi, da lahko vsak izrazi svoj pogled o poljubni temi.

2.1.3 *SKLAD TEHNOLOGIJ SEMANTIČNEGA SPLETA*

Trenutna različica svetovnega spleta (WWW) je ogromna množica med seboj povezanih dokumentov, ki jih prenašajo računalniki in predstavljajo ljudje. Nastali so iz povezanih sistemov z razliko, da lahko vsak prispeva svoj del. To pa tudi pomeni, da kvalitete informacij ali celo trajne hrambe dokumentov ne moremo vedno zagotoviti. Trenutni splet vsebuje številne podatke, informacije in znanja, vendar je vloga računalnikov v tem trenutku zgolj dostavljanje in predstavitev vsebine dokumentov, ki opisujejo to znanje. Uporabniki moramo povezati vse vire pomembnih informacij in jih tudi sami interpretirati.

Ideja semantičnega spleta je izboljšati trenutni splet, v tej meri, da bodo računalniki lahko procesirali podatke na spletu, jih interpretirali, povezali in na takšen način pomagali ljudem pri iskanju zahtevanega znanja. Na isti način kot je svetovni splet velik porazdeljen povezan sistem, je namen semantičnega spleta zagotoviti veliko porazdeljeno bazo znanja. Semantični splet se osredotoča na skupno rabo podatkov in ne dokumentov. Z drugimi besedami gre pri tem projektu za vzpostavitev skupnega ogrodja, ki bi omogočalo skupno rabo podatkov in ponovno uporabo med aplikacijami, podjetji in mejami skupnosti. Gre za skupno sodelovanje, ki ga vodi World Wide Web konzorcij (W3C).

Slika 1: Sklad semantičnega spleta

Slika 1 prikazuje arhitekturo semantičnega spleta v obliki sklada. Najnižja raven temelji na **URI**[2] in UNICODE, ki sta pomembna elementa obstoječega svetovnega spleta. **UNICODE** je standard za zapis množice mednarodnih znakov in omogoča, da lahko na spletu uporabljamo poljuben svetovni jezik (za pisanje in branje) v standardizirani obliki. URI je niz v standardizirani obliki, ki omogoča enolično poimenovanje virov (npr. dokumentov). Podmnožica URI-ja je URL[3], ki določa mehanizem dostopa in (omrežno) lokacijo dokumenta (npr. http://amor.fri.uni-lj.si/dejan/). Uporaba URI-ja je zelo pomembna v

[2] **URI** je enolični identifikator vira (ang. Uniform Resource Identifier).
[3] **URL** enolično opredeljuje lokacijo vira (ang. Uniform Resource Locator).

porazdeljenem internetnem okolju, saj je z njegovo pomočjo izvedena identifikacija vseh virov. Mednarodna različica URI-ja je IRI[4], ki pri identifikatorju omogoča uporabo UNICODE znakov.

Naslednja raven z jezikom **XML**[5] se ukvarja predvsem s sintakso podatkov in shema XML z opredelitvijo imenskega prostora. XML je označevalni jezik za splošno uporabo, ki se uporablja za opis dokumentov s strukturirano vsebino in je nastal leta 1998. Dokument XML vsebuje elemente, ki jih lahko gnezdimo in lahko imajo atribute ter vsebino. S pomočjo imenskega prostora XML pa lahko opredelimo različne označevalne slovarje v okviru istega dokumenta XML. **Shema XML** se uporablja za izražanje sheme določene množice dokumentov XML.

RDF se uporablja za opis grafa, ki je sestavljen iz trojic. Vsak lahko opredeli slovar izrazov, ki se uporablja za bolj podroben opis. **Shema RDF** (RDF/S) je naslednja raven, ki je nastala zato, da imamo na voljo standardiziran način za opis taksonomij. Formalna semantika sheme RDF je zapisana v jeziku RDF. RDF/S se uporablja za opis razrednih taksonomij in taksonomij lastnosti – na ta način lahko razvijemo "lahke ontologije".

Bolj podrobne ontologije lahko opredelimo v jeziku **OWL** (Web Ontology Language). Jezik OWL temelji na opisni logiki in ima na voljo več konstruktov kot RDF/S. Sintaktično je integriran v RDF, tako da kot RDF/S prinaša dodaten standardiziran slovar. OWL najdemo v treh oblikah – ***OWL Lite*** za opredelitev taksonomij in enostavnih omejitev, ***OWL DL*** za polno podporo opisni logiki in ***OWL Full*** za največjo izrazno moč.

RDF/S in OWL imata opredeljeno semantiko, ki je uporabna za sklepanje na ravni ontologij in baz znanj, opisanih s tema jezikoma. Za podporo pravilom poleg konstruktov že omenjenih jezikov, ki so že na voljo, na področju semantičnega spleta ravno tako obstajajo standardizirani jeziki za pravila. Trenutno sta najbolj uporabljana standarda RIF in **SWRL**.

Za poizvedovanje po podatkih v ontologijah RDF, RDF/S in OWL je na voljo **SPARQL** (Simple Protocol and RDF Query Language). Poizvedovalni jezik SPARQL je podoben SQL-u, vendar uporablja vire in trojice RDF za predstavitev poizvedbe in vrnjene rezultate. Ker RDF/S in OWL temeljita na RDF, je SPARQL neposredno uporaben tudi za poizvedovanje po ontologijah. SPARQL pa ni le poizvedovalni jezik, ampak je tudi protokol za dostop do podatkov RDF.

Vsa semantika in pravila se izvajajo na ravneh pod ravnjo zaupanja, rezultati mehanizma sklepanja pa se bodo uporabili za dokazovanje sklepov. Formalen dokaz, skupaj z zaupanjem vrednimi viri, bo pomenil, da lahko rezultatom zaupamo, kar je prikazano na zgornji ravni sklada arhitekture semantičnega spleta. Za zagotavljanje zanesljivih virov se bo uporabljala kriptografija, kot so digitalni podpisi za preverjanje izvora virov. Nad vsemi temi ravnmi lahko tako gradimo aplikacije z uporabniškimi vmesniki.

[4] **IRI** (ang. Internationalized Resource Identifier) je mednarodna in posplošena različica URI-ja.
[5] **Razširljiv označevalni jezik** oz. **XML** (ang. eXtensible Markup Language) predstavlja format za opisovanje strukturiranih podatkov in je še posebej primeren za medsebojno izmenjavo podatkov.

2.1.4 *RAZLIČNI POGLEDI IN PRIČAKOVANJA SEMANTIČNEGA SPLETA*

Semantični splet ima različne interpretacije, kot je bilo že predstavljeno v poglavju 2.1.2. Nekateri si ga predstavljajo kot splet podatkov, kjer so le-ti predstavljeni v obliki RDF in OWL. Nekateri RDF zamenjajo z mikroformati. Drugi razmišljajo o semantičnem spletu skozi oči spletnih storitev, za nekatere je semantični splet umetna inteligenca – računalniški programi, ki rešujejo kompleksne optimizacijske probleme, ki jih večina uporabnikov sploh ne razume. Poslovni uporabniki vedno opredelijo problem v pogledu uporabniške vrednosti in pravijo, karkoli že je, potrebuje enostavne in otipljive aplikacije za potrošnike in organizacije.

Do takšnega nestrinjanja nismo prišli po naključju, saj tehnologije in koncepti semantičnega spleta pokrivajo res zelo širok spekter. Veliko je res izvedljivega, vendar si lahko veliko tudi samo predstavljamo.

2.1.4.1 Pristop od spodaj navzgor in od zgoraj navzdol

Pristop od spodaj navzgor temelji na opisih podatkov na spletnih straneh in jeziku RDF tako, da so podatki zajeti v računalniku razumljivi obliki. Ta pristop uporablja že obstoječe podatke in poskuša samodejno izpeljati pomen.

Potrditev pristopa od spodaj navzgor je npr. odločitev Yahooja, da bo njihov iskalnik podpiral RDF in mikroformate. To je zelo dobro za vse vpletene strani – avtorji strani bodo lahko opisali strani na takšen način, da lahko Yahoo to upošteva in uporabniki bodo imeli na voljo boljše in bolj natančne rezultate.

Kljub vsemu je pristop od spodaj navzgor zelo težaven. Ključ Googlovega uspeha je v tem, da razume podatke v takšni obliki, kot so, brez zahteve po popolni skladnosti s standardi W3C in tehnikami za optimizacijo strani. Podobna orodja, ki temeljijo na pristopu od zgoraj navzdol, so npr. OpenCalais[6], Dapper[7], BlueOrganizer[8] ipd.

2.1.4.2 Tehnologije za opisovanje: RDF, mikroformati in meta oznake

Pri pristopu od spodaj navzgor imamo na voljo številne možnosti za opisovanje. Vse niso enako zmogljive, vendar gre pri praktično vseh za razmerje med enostavnostjo uporabe in popolnostjo. Najbolj obsežen pristop je RDF, ki je najbolj izrazen, točen, a vendar kompleksen.

Mikroformati so enostavnejši pristop, ki dodajajo pomen s pomočjo spreminjanja obstoječih strani HTML z uporabo določenih slogov CSS. Mikroformati pridobivajo na popularnosti zaradi svoje enostavnosti, vendar so zaradi tega tudi omejeni po funkcionalnosti. Trenutno se uporabljajo na straneh, kot je Flickr[9], LinkedIn[10] ipd.

---

[6] http://www.opencalais.com
[7] http://www.dapper.net
[8] http://getglue.com
[9] http://www.flickr.com
[10] http://www.linkedin.com

Še enostavnejši pristop je dodajanje meta oznak znotraj oznake glav. Trenutno še ni tako razširjen, a je zelo primeren za spletne strani, ki so osredotočene na določeno temo (npr. spletni časopisi).

2.1.4.3 Vidik potrošnika in organizacije

Naslednja dimenzija pogleda na semantični splet je z vidika potrošnikov in organizacij. Kar se tiče potrošnika, se že od nekdaj išče "killer application" – nekaj otipljivega, ki potrošniku enostavno prinaša dodano vrednost. Uporabnikov ne zanima, ali je aplikacija zgrajena na semantičnem spletu, vse, kar šteje, je koristnost in uporabnost. Zelo dolgo je bila ideja semantičnega spleta v akademski domeni, kjer so se vsi ukvarjali z opisovanjem informacij, z namenom, da so razumljive računalnikom. Ostaja pa obljuba, da bodo te informacije nekoč znale s pridom uporabljati aplikacije, zgrajene nad semantičnim spletom.

Pri organizacijah je slika nekoliko drugačna, saj so le-te bolj navajene tehničnega jezika. Uporaba semantičnih tehnologij se v njihovih glavah preslika v nekaj inteligentnega, kar posledično pomeni tudi dobro trženje – trditev "naši produkti so boljši in pametnejši, ker uporabljamo semantični splet" je zelo dobra reklama za organizacije. Poleg tržne vrednosti se moramo zavedati, da RDF rešuje težave izmenjave podatkov in to področje standardizira. Ta problem je prisoten že od nekdaj. Če pozabimo na semantiko in se osredotočimo zgolj na standarden protokol in standarden način pošiljanj podatkov med večimi programi, je že to za določene organizacije zelo veliko. RDF omogoča način komunikacije na temeljih že razširjenega jezika XML, poleg tega pa ima matematično osnovo za omogočanje semantike.

2.1.4.4 Semantični API-ji

Osnovna naloga semantičnih API-jev je na podlagi nestrukturiranega besedila na vhodu podati rezultat na izhodu, ki vsebuje entitete in njihove medsebojne povezave.

Prvi takšen primer je ***Open Calais API***, ki so ga razvili pri Thomson Reuters in sprejme tekst ter vrne podatke o ljudeh, krajih in podjetjih, ki jih je našel v dokumentu. Naslednji primer je ***SemanticHacker API***, ki klasificira podatke v dokumentu v kategorije, imenovane semantični opisi. Če zahtevamo analizo dokumenta, dobimo kot rezultat entitete in teme, ki se v njem pojavljajo. Podobno torej kot pri Open Calais, le da vrne še tematsko hierarhijo. Zelo zanimivo rešitev ponuja tudi ***Dapper***, ki olajša luščenje strukturiranih podatkov iz nestrukturiranih HTML strani. V praksi se izkaže za zelo uporabnega, saj omogoča zelo enostaven način pretvorbe spletne strani v spletno storitev.

2.1.4.5 Iskalne tehnologije

Mogoče je na začetku bilo zaslediti največji potencial semantičnega spleta ravno pri iskalnih tehnologijah, saj bi razumevanje spletnih strani močno izboljšalo rezultate iskanja.

Na tem področju lahko najdemo dva konkurenta ***Hakia*** in ***Powerset***, ki kažeta določen napredek. Vendar je težava v tem, da Googlov iskalni algoritem, ki temelji na statistični

analizi, še vedno dobro obvladuje entitete, kot so ljudje, mesta in organizacije. Ko ga vprašamo: "Kaj je glavno mesto Slovenije?", nam odgovori popolnoma točno (Slika 2).

Slika 2: Kaj je glavno mesto Slovenije?

Očitno je, da zgolj manjši napredek pri prikazovanju rezultatov iskanja ne bo dovolj, ampak bo zmagovalna kombinacija semantike, inovativne predstavitve in sledenja uporabniku.

2.1.4.6 Kontekstne tehnologije

Navigacija po kontekstnih informacijah neposredno ne izboljša iskanja, ampak ustvarja bližnjice do tega. Primera takšnih aplikacij sta ***Snap*** ali ***Yahoo Shortcuts***, ki razumeta objekte znotraj teksta in povežeta pomembne podatke znotraj uporabnikovega konteksta. Na voljo so tudi druge rešitve, ki temeljijo večinoma na pristopu od zgoraj navzdol – ***Interclue***, ***ThumbStrips***, ***Cooliris*** in ***BlueOrganizer***.

Skupna točka vseh omenjenih orodij pa je prepoznavanje podatkov in kreiranje določenega uporabniškega mikro konteksta, saj tako uporabniku omogočimo interakcijo in nadaljnje iskanje.

2.1.4.7 Semantične podatkovne baze

Primer takšne semantične podatkovne baze, ki je še vedno v razvojni fazi, je ***Twine***.

Slika 3: Ideja semantične podatkovne baze

Osredotočena je na izgradnjo osebne baze znanja in temelji na uvozu številnih nestrukturiranih podatkov ter gradi bazo ljudi, podjetij, stvari, lokacij ipd. Takšna baza znanja je npr. zelo uporabna pri pooseblenem iskanju. Drug poskus izgradnje semantične

podatkovne baze je ***Freebase***[11], ki je bolj strukturirana različica Wikipedije. Veliko takšnih virov pa najdemo tudi v okviru projekta LOD[12].

2.2 ONTOLOGIJE

2.2.1 *RAZLIČNI POGLEDI*

Ontologije so povezane s številnimi področji in tehnologijami, kar nam prikazuje Slika 4.

Slika 4: Različni pogledi na ontologije

Pojmovanje ontologije se v zalednem delu začne z razširjenimi ER modeli in predikatno logiko, s podporo odločanju, mehanizmi sklepanja in EAI, saj je na tej ravni semantika formalno določena. Ostali elementi pridejo v ospredje pri opravilih, kot so povpraševanje, pridobivanje informacij, navigacija in predvsem souporaba znanja.

Slika 5: Kompleksnost ontologije

Kompleksnost razvite ontologije številni avtorji označujejo kot lahka ali težka ontologija, ki je seveda pomensko bolj bogata od zapisa v podatkovni bazi, folksonomije ali

[11] http://www.freebase.com
[12] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

nestrukturiranega dokumenta. Slika 5 prikazuje omenjeno lestvico kompleksnosti razvite ontologije in indikator ravni ontologije, ki jo lahko razvijemo s pristopom ROD (podrobnosti so predstavljene v poglavju 3).

Pri opredelitvi ontologije je zelo koristno opredeliti razliko med izrazom in predstavo, saj nam takšen pogled pomaga pri razlikovanju slovarja in ontologije.

Slika 6: Homonim "jaguar" v trikotniku pomena

Kot primer si poglejmo izraz "jaguar", ki lahko v realnem svetu predstavlja številne stvari (Inštitut za slovenski jezik Frana Ramovša 2002) (Slika 6):

- leopardu podobna črna lisasta zver, ki živi v Ameriki,
- avtomobil angleške tovarne Jaguar,
- kodno ime operacijskega sistema Mac OS X 10.2,
- športno kolo angleške tovarne Jaguar.

Slika 7 poskuša v strnjeni obliki prikazati tri komponente pomena (Daconta, et al. 2003) (oglišča trikotnika), ki se pojavljajo v naravnem jeziku (npr. v slovenščini). Prva komponenta v spodnjem levem kotu je **izraz**, ki ga sestavljajo simboli (oznake za predstavo) ali besede v slovenskem jeziku in pravila za združevanje teh izrazov v stavke. Sami po sebi nimajo pomenov, dokler jih ne povežemo z ostalima komponentama oz. ogliščema trikotnika, to je **predstavo** in **stvarjo v realnem svetu**.

Če nas npr. nekdo vpraša, kaj pomeni izraz "D23978BMF1MB8H77E8X9TJB9D3", verjetno ne bomo poznali odgovora, saj za nas nima pomena. V nasprotnem primeru bomo skoraj zagotovo poznali pomen izraza "avtomobil", ker imamo predstavo o tem izrazu v realnem svetu (ima štiri gume, motor, porabi določeno količino goriva itd). Ravno zaradi tega je v trikotniku (Slika 7) med izrazom in stvarjo v realnem svetu šibkejša povezava, saj ni neposredne povezave, ker človek pri povezovanju pojmov in stvari v realnem svetu potrebuje predstavo.

Slika 7: Vloga slovarja in ontologije v trikotniku pomena

Slovar se v splošnem ukvarja z levo stranjo trikotnika (izrazi in predstave), medtem ko je ontologija večinoma osredotočena na desno stran (predstave in stvari v realnem svetu), kot to prikazuje Slika 7. Slovar je v prvi vrsti klasifikacijski prostor v določeni domeni ali množici domen ali celo v širšem svetu, predvsem za potrebe iskanja in pridobivanja informacij. Semantika klasifikacijskega prostora je tako relativno šibka, z enostavnimi semantičnimi relacijami med izrazi, ki so strukturirani s taksonomijo relacij širši in ožji. Vse, kar moramo poznati o izrazu v slovarju, je, da se semantično razlikuje od ostalih (s čimer izločimo dvoumnost) in da je širši oz. ožji pojem kot drugi.

Na drugi strani želi ontologija predstaviti zapleteno semantiko predstav in relacij med predstavami, njihovimi lastnostmi, vrednostmi, omejitvami in pravili. Namen ontologije se razlikuje od namena slovarja, saj ontologija skuša zajeti in predstaviti pomen določenih domen, množice domen ali celotnega sveta, ker skuša eksplicitno simulirati razumevanje, ki ga ima človek v svojih mentalnih modelih problemske domene ali sveta. V nasprotju s slovarjem želimo pri ontologijah konceptualno semantiko izraziti na čim bolj natančen, obsežen in konsistenten način.

Slika 8 prikazuje pogled razvijalcev na evolucijo podatkov, kjer je razviden vpliv podatkov na razvoj programske opreme (Daconta, et al. 2003, Lavbič 2005). Proceduralno programiranje je osredotočeno na funkcionalno dekompozicijo opravil. Podatke so spreminjale in obvladovale procedure, kjer je veljalo, da so podatki manj pomembni od kode.

Pri objektno usmerjenem razvoju se pojavijo enkapsulacija, dedovanje in polimorfizem. Ideja o varovanju podatkov s pomočjo rezerviranih besed in metod za dostop povzroči dvig ravni pomembnosti podatkov tako, da postanejo enakovredni metodam, ki s temi

podatki manipulirajo. S prihodom XML-a, MDA-ja (poglavje 2.3.3) in podpore meta podatkom v času izvajanja vstopamo v obdobje, kjer podatki postajajo bolj pomembni kot sama koda. V ospredje ne prihaja zgolj sintaktična pravilnost, ampak predvsem semantična.

Slika 8: Pogled razvijalcev na podatke

2.2.2 *OSNOVNI GRADNIKI IN FORMALNA OPREDELITEV*

Ključni element vseh aplikacij semantičnega spleta so ontologije (Davies, et al. 2006). Skupno sprejeto opredelitev ontologije je podal Gruber, ki pravi: "Ontologija je eksplicitna in formalna specializacija konceptualizacije problemske domene" (Gruber 1993). V tej opredelitvi zasledimo dva ključna podatka; konceptualizacija je formalna in omogoča računalniku, da na podlagi tega izvaja sklepanje in da je določena ontologija razvita za določeno problemsko domeno. Ontologijo sestavljajo **koncepti** (poznamo jih tudi pod imenom razredi), **relacije** (ali lastnosti oz. vloge), **primerki** (ali instance) in **aksiomi**, tako da lahko enostavno ontologijo bolj formalno zapišemo kot 4-terico $\langle C, R, I, A \rangle$, kjer je C množica razredov, R množica lastnosti, I množica primerkov in A množica aksiomov.

Ontologije se uporabljajo za organizacijo znanja v strukturirani obliki na številnih področjih od filozofije do obvladovanja znanja in semantičnega spleta. Ontologijo lahko opredelimo tudi kot graf oz. omrežno strukturo (Bloehdorn, et al. 2005, Davies, et al. 2006), ki je sestavljena iz:

1. množice konceptov (točk v grafu),
2. množice relacij med koncepti (usmerjene povezave v grafu),
3. množice primerkov posameznih konceptov.

Bolj podrobno lahko formalno ontologijo opredelimo kot strukturo naslednjih gradnikov $O = (C, T, R, A, I, V, \leq_C, \leq_T, \leq_R, \leq_A, \sigma_R, \sigma_A, \iota_C, \iota_T, \iota_R, \iota_A)$. Strukturo sestavljajo disjunktne množice konceptov (C), tipov (T), relacij (R), atributov (A), primerkov (I) in vrednosti (V). Delne urejenosti $\leq_C$ (na C), $\leq_T$ (na T), $\leq_R$ (na R) in $\leq_A$ (na A) opredeljujejo hierarhijo konceptov, tipov, relacij in atributov. Funkcija $\sigma_R: R \to C^2$ podrobneje določa relacije (za vsako relacijo funkcija določi, kateri koncepti se lahko s to relacijo povezujejo), medtem ko $\sigma_A: A \to C \times T$ podrobneje določa atribute (za vsak atribut funkcija določi, kateremu konceptu atribut pripada in kakšen je njen podatkovni tip). Poleg tega imamo še funkcije za delno določanje primerkov $\iota_C: C \to 2^I$ (določanje pripadnosti primerkov razredu), $\iota_T: T \to 2^V$ (določanje vrednosti tipom), $\iota_R: R \to 2^{I \times I}$ (določanje, kateri primerki so

povezani z določeno relacijo) in $\iota_A: A \to 2^{I \times V}$ (kakšna je vrednost določenega atributa pri določenem primerku).

Definicija 1 (Ontologija). Ontologija je struktura

$$\mathcal{O} := (C, \leq_C, R, \sigma, \leq_R, I, \iota_C, \iota_R),$$

ki jo sestavljajo:

- disjunktne množice C (identifikator koncepta oz. **koncept**), R (identifikator relacij oz. **relacija**) in I (identifikator primerka oz. **primerek**[13]),
- delna ureditev $\leq_C$ nad C se imenuje **hierarhija konceptov** oz. **taksonomija**,
- funkcija $\sigma: R \to C^+$, ki se imenuje **oznaka**,
- delna ureditev $\leq_R$ nad R se imenuje **hierarhija relacij**,
- funkcija $\iota_C: C \to \mathfrak{B}(I)$, ki se imenuje **pripadnost primerka konceptu**,
- funkcija $\iota_R: R \to \mathfrak{B}(I^+)$, ki se imenuje **pripadnost primerka relaciji** z $\iota_R(r) \subseteq \prod_{c \in \sigma(r)} \iota_C(c)$ za vse $r \in R$.

Nekateri avtorji trdijo, da bi se moral pojem ontologija nanašati zgolj na formalizacijo in namenski pogled na problemsko domeno. Da se izognemo zmešnjavi, bomo ta namenski pogled na problemsko domeno poimenovali jedrna ontologija. Ta ideja je popolnoma identična terminološkemu znanju ali komponenti TBox pri opisni logiki. Komplementaren del pa se uporablja za modeliranje razširjenega znanja in ga včasih imenujemo tudi baza znanja. Ta razširjen del ontologije vsebuje trditve o primerkih konceptov in relacij, kot je temu analogno izrazno znanje (ABox) pri opisni logiki.

Ontologijo, zapisano v jeziku OWL in SWRL, ki jo bomo uporabljali skozi celotno disertacijo, sestavljajo **razredi**, **lastnosti**, **primerki** in **aksiomi**.

Primerki v ontologiji predstavljajo objekte v naši problemski domeni. Pri zapisu primerkov v jeziku OWL je zelo pomembna predpostavka o enoličnih imenih (UNA[14]), ki v tem primeru ne drži. To praktično pomeni, da lahko dva različna primerka z istim imenom dejansko predstavljata isti primerek.

Lastnosti so binarne relacije med primerki, npr. povezava, ki med seboj povezuje dva primerka. Če smo natančni, dejansko govorimo o primerkih lastnosti, ki povezujejo primerke razredov, vendar zaradi enostavnosti govorimo o lastnostih.

Razrede se v jeziku OWL interpretira kot množice, ki vsebujejo primerke. Opisani so s formalnim (matematičnim) opisom, ki natančno opredeljuje pogoje za vključenost primerka v množico. Razred je tako predstavitev konceptualnega združevanja podobnih terminov. Razrede lahko tudi organiziramo v hierarhije tipa nadrazred in podrazred, kar imenujemo taksonomija. Podrazredi so specializacija (vsebovanost) svojih nadrazredov.

[13] **Primerek** (ang. Individual) je lahko primerek razreda oz. lastnost, ki medsebojno povezuje primerke.
[14] **Predpostavka enoličnega imena** (ang. Unique Name Assumption).

Ena od ključnih lastnosti OWL DL je, da so razmerja nadrazred oz. podrazred samodejno izvedena s pomočjo mehanizma sklepanja.

Aksiomi so deklarativne in rigorozne izjave, ki so vedno resnične, brez potrebnega dokazovanja. Z druge perspektive so aksiomi omejitve problemske domene, ki so resnični v vseh primerih uporabe. Enostavna ontologija je sestavljena iz hierarhije konceptov, ki so povezani z medsebojnimi relacijami. Pri kompleksnejših primerih ontologij so dodane omejitve glede dovoljenih vrednosti razredov in lastnosti, kot so kardinalnost, velikost ipd. Eden od tipov aksiomov so tudi pravila, ki so zapisana v jeziku SWRL.

2.2.3 *JEZIKI ZA PREDSTAVITEV ONTOLOGIJ*

Jeziki za predstavitev ontologij so formalni jeziki, ki jih na področju računalništva in umetne inteligence uporabljamo za konstruiranje ontologij. Omogočajo zajem znanja v določeni problemski domeni in pogosto vključujejo tudi pravila za sklepanje, ki podpirajo procesiranje tega znanja.

Slika 9 prikazuje različne načine formalizacije posameznih jezikov za predstavitev ontologij. Pri samem razvoju in implementaciji ontologije je vedno potreben razmislek, kaj bo aplikacija zahtevala v smislu izraznosti in mehanizmov sklepanja, saj vsi jeziki ne omogočajo predstavitev izbranih komponent na enak način. Predstavitev in sklepanje na osnovi enostavnih informacij, kot so koncepti, taksonomije in preproste binarne relacije, pogosto niso dovolj, ko razvijamo kompleksnejše ontologije.

Slika 9: Načini formalizacije jezikov za predstavitev ontologij

Od leta 1990 dalje je bilo razvitih kar nekaj jezikov za predstavitev znanja oziroma ontologij. Razdelimo jih lahko v dve skupini, in sicer:

- skupina osnovnih jezikov za predstavitev znanja in

- skupina označevalnih jezikov za predstavitev ontologij, ki večinoma temeljijo na sintaksi XML.

Slika 10 prikazuje kronološki pregled razvoja jezikov za predstavitev ontologij. Predstavljeni so tudi vplivi, ki so jih imeli določeni jeziki na razvoj novih jezikov.

Slika 10: Kronološki pregled razvoja jezikova za predstavitev ontologij

V nadaljevanju bosta podrobneje predstavljeni obe skupini z značilnimi predstavniki.

2.2.3.1 Osnovni jeziki za predstavitev ontologij

Cycl je formalni jezik s sintakso, ki izvira iz predikatne logike prvega reda. Uporablja se predvsem za opis splošnega znanja. Sestavljajo ga naslednji izrazni tipi: konstante, formule, funkcije, spremenljivke in količinski tipi. Prednosti omenjenega jezika so predvsem izraznost, natančnost in neodvisna predstavitev od uporabe.

KIF je računalniški jezik za izmenjavo znanja med različnimi programi in deklarativno semantiko[15]. S svojo splošnostjo[16] omogoča predstavitev znanja o znanju, opredelitev objektov, funkcij in relacij. Jezik KIF ne vključuje podpore odločanju.

Na osnovi jezika KIF je nastal tudi jezik **Ontolingua**, ki je jezik KIF razširil z dodatnimi sklopi aksiomov.

OCML je strukturni jezik in zagotavlja mehanizme za izražanje relacij, funkcij, pravil, razredov in primerkov. Za bolj učinkovito izvajalno podporo vsebuje tudi logične mehanizme sklepanja, nastal pa je z namenom podpore razvoja izvajalnih ontologij in modelov pri različnih metodah za reševanje problemov. Je soroden jeziku Ontolingua, s katerim je tudi združljiv.

Jezik **LOOM** je bil sprva namenjen zgolj predstavitvi znanja, vendar je nadalje omogočal tudi predstavitev naslednjih komponent ontologij: koncepti, taksonomije, n-tere relacije,

[15] Pri **deklarativni semantiki** je za pomen izrazov v predstavitvi razumljiv brez uporabe prevajalnika za omenjene izraze.
[16] **Splošnost jezika KIF** je v predstavitvi poljubnih stavkov v obliki predikatnega računa 1. reda.

funkcije, aksiome in pravila. Temelji na opisni logiki in produkcijskih pravilih ter omogoča samodejno klasifikacijo konceptov.

FLogic, jezik za predstavitev znanja in ontologij, je kombinacija strukturnega jezika in predikatne logike 1. reda. Omogoča nam predstavitev konceptov, taksonomij, binarnih relacij, funkcij, aksiomov in pravil. V povezavi z objektno usmerjenimi jeziki omogoča tudi predstavitev kompleksnejših objektov, dedovanje, polimorfizem, enkapsulacijo in poizvedovanje. FLogic lahko v povezavi z objektno usmerjenim programiranjem analogno postavimo ob bok klasičnemu predikatnemu računu in relacijskim podatkovnim bazam.

Protokol OKBC omogoča dostop do različnih sistemov za predstavitev znanja in s tem omogoča enoten model predstavitve znanja ter temelji na splošni konceptualizaciji razredov, primerkov, dedovanja ipd.

2.2.3.2 Označevalni jeziki za predstavitev ontologij

Z razširjenostjo interneta se je začel razvoj jezikov za predstavitev ontologij, ki so temeljili na lastnostih svetovnega spleta, njihova največja vrednost pa se kaže v kontekstu semantičnega spleta. Najbolj znani so jeziki, ki temeljijo na XML:

- **XOL** (Ontology Exchange Language),
- **SHOE** (Simple HTML Ontology Extension),
- **OML** (Ontology Markup Language),
- **RDF** (Resource Description Framework) in **RDF Schema**.

Kasneje so bili razviti še trije jeziki, ki razširjajo funkcionalnosti RDF(S) (združena RDF in shema RDF) (Slika 10):

- **OIL** (Ontology Inference Layer),
- **DAML+OIL** in
- **OWL** (Web Ontology Language).

Slika 11: Označevalni jeziki za predstavitev ontologij

V nadaljevanju bodo omenjeni jeziki tudi podrobneje predstavljeni.

2.2.3.2.1 XML (eXtended Markup Language)

XML primarno ni jezik za predstavitev ontologij, je pa osnova za več drugih jezikov. Uporabnikom omogoča kreiranje lastnih oznak in atributov, definiranje strukture podatkov (vključno z gnezdenjem), zbiranje podatkov iz dokumentov in razvoj aplikacij za testiranje strukturne pravilnosti dokumentov XML. XML je enostaven za analiziranje in

razčlenjevanje, njegova sintaksa je dobro definirana in tudi berljiva človeškemu uporabniku. Vendar pa jezik XML ne pove ničesar o semantiki podatkov. Na svetovnem spletu je postal standard za opis podatkov, pri jezikih za predstavitev ontologij pa se uporablja kot preprost način za predstavitev njihove sintakse.

2.2.3.2.2 XOL (Ontology Exchange Language)

Združenje ameriških bioinformatikov je pripravilo jezik XOL za izmenjavo opredelitev ontologij znotraj njihove domene med heterogenimi programskimi sistemi. Kot osnova za jezik XOL sta bila uporabljena jezika Ontolingua in OML, v katerih je bila združena visoka izraznost jezika OKBC ter sintakse OML-ja na osnovi jezika XML. XOL je funkcionalno zelo omejen in ne podpira mehanizmov sklepanja. Razvoj ontologij v jeziku XOL ne podpira nobeno orodje, vendar temelji na jeziku XML, tako da lahko za oblikovanje dokumentov XOL uporabljamo tudi urejevalnike XML.

2.2.3.2.3 SHOE (Simple HTML Ontology Extension)

SHOE je bil prvotno razvit kot razširitev jezika HTML, ki je vključeval dodane semantične podatke HTML drugim spletnim dokumentom, kar je omogočalo strojno berljivost dokumentov. Uporabljal je drugačne oznake kot HTML in omogočal vključitev ontologij v HTML dokumente. Z razvojem jezika XML kot standardnega jezika za izmenjavo informacij na spletu je bila osnovna sintaksa SHOE-a prilagojena jeziku XML. Programski agenti so lahko s pomočjo jezika SHOE zbirali informacije o spletnih straneh in dokumentih, izboljšali mehanizme iskanja in kopičili znanje. Ta proces je bil sestavljen iz treh faz: definiranja ontologije, označitve strani HTML s semantičnimi podatki iz ontologij o vsebini dokumenta in kreiranja programskega agenta za osveževanje informacij s pomočjo iskanja preko vseh obstoječih strani.

2.2.3.2.4 OML (Ontology Markup Language)

OML je nastal iz jezika SHOE, zato imata tudi veliko podobnih lastnosti. Obstajajo štiri ravni jezika OML. Jedro OML je povezano z logičnimi vidiki jezika in je vključeno v vse ostale ravni. Osnovni OML se uporablja za povezavo z jezikom RDF/S. Okrajšani OML vključuje funkcionalnosti za kreiranje konceptualnih diagramov. Standardni OML pa je najbolj izrazna različica jezika.

2.2.3.2.5 RDF (Resource Description Framework)

Jezik RDF so razvili pri organizaciji W3C za opredelitev meta podatkov, ki opisujejo spletne vire. RDF je tesno povezan z jezikom XML, saj je namen jezika RDF na standardiziran način opredeliti semantiko podatkov v jeziku XML. Z izjavami v jeziku RDF lahko opisujemo vire v obliki trojic $\langle osebek, predikat, predmet \rangle$ (Slika 12).

Slika 12: Primer izjave RDF

RDF je v prvi vrsti sistem za modeliranje podatkov. Njegova prednost je predvsem na področju prilagodljivosti, saj je vsako razmerje med poljubnima podatkovnima elementoma eksplicitno predstavljeno, kar predstavlja zelo enostaven model za integracijo podatkov iz različnih virov. Problem integracije podatkov je tako preveden na preprosto poenoteno obravnavo vseh takšnih izjav RDF iz različnih podatkovnih virov.

Edini izziv, ki še ostane pri integraciji, je problem identitete. Glede na to, da ta težava ni nova, se pri jeziku RDF uporablja že obstoječa rešitev svetovnega spleta – enolični identifikator vira (URI).

2.2.3.2.6 Shema RDF

V praksi obstajajo številne uspešne opredelitve shem v bolj pogosto uporabljanih jezikih. Vloga sheme XML je npr. ugotoviti, ali izbrani dokument XML ustreza tej shemi. Shema podatkovne baze zagotavlja podatke o glavi in ključne podatke o tabelah v relacijski podatkovni bazi. V sami relacijski tabeli ni nobenega podatka, ki bi nakazoval pomen informacij v določenem stolpcu, ali da je izbrani stolpec uporabljen kot indeks. Vsi ti podatki se nahajajo v shemi podatkovne baze, ki je neodvisna od relacije (se ne spreminja od zapisa do zapisa). Pri objektno usmerjenih sistemih razredni diagrami ravno tako opisujejo podatke, vendar nam omogočajo še nekaj več – omogočajo sistematičen način opisovanja podatkov in transformacij za te podatke. Vse sheme imajo nekaj skupnega, in sicer, da povedo nekaj o podatkih, ki so izraženi v sistemu. Shema predstavlja podatke o podatkih oz. meta podatke.

Ključna ideja pri shemi RDF je ta, da naj bi podatkom dodala pomen; način, s katerim to dosežemo, pa je mehanizem sklepanja. S sklepanjem lahko namreč pridemo do več podatkov, kot pa jih je dejansko neposredno izraženih. S tem, ko ti izpeljani podatki postanejo eksplicitni, dodamo izvirnim podatkom nov pomen. Kot pri vseh novejših jezikih za opredelitev shem, je tudi RDF/S zapisan v odvisnem jeziku RDF. Vsi podatki sheme RDF so zapisani kot trojice RDF. Shema RDF razširja jezik RDF s konstrukti, kot so razredi, dedovanje razredov, lastnosti, dedovanje lastnosti, definicijsko območje in zaloga vrednosti. V omenjenem jeziku ne moremo predstaviti aksiomov, edini mehanizem sklepanja, ki je na voljo, pa temelji le na obstoječih povezavah med razredi.

2.2.3.2.7 OIL (Ontology Interchange Language)

Jezik OIL je bil razvit v sklopu projekta On-To-Knowledge (Sure 2003) in omogoča medsebojno sodelovanje med spletnimi viri na podlagi semantike. Njegova sintaksa in semantika temeljita na obstoječih jezikih XOL, OKBC in RDF/S. Zagotavlja veliko prvin

modeliranja, ki se uporabljajo tako v ontologijah z opisno logiko, kot tudi v strukturiranih ontologijah. Njegova semantika je preprosta in dobro definirana, omogoča pa tudi avtomatično sklepanje. OIL ima štiri ravni, kjer je v vsaki naslednji ravni poleg prejšnje ravni dodanih nekaj funkcionalnosti. Na ta način je možno delno procesiranje ontologije, v primeru, da agent, ki je sposoben procesiranja manj izrazne ontologije, manipulira z ontologijo na višji ravni.

2.2.3.2.8 DAML + OIL

DAML+OIL je bil razvit s sodelovanjem ameriške agencije DARPA[17] in Evropske unije ter omogoča medsebojno sodelovanje med dokumenti XML na podlagi semantike. To je eden izmed semantično najbolj izraznih jezikov za opis dokumentov na svetovnem spletu. DAML+OIL je bil razvit na osnovi RDF(S) in tudi jezika OIL, s katerim sta si podobna in imata enake cilje.

2.2.3.2.9 OWL (Web Ontology Language)

OWL je podobno kot DAML+OIL eden najbolj izraznih jezikov za predstavitev znanja na semantičnem spletu.

Slika 13: Ravni jezika OWL

V nasprotju z DAML+OIL je nastal pod okriljem W3C, ki je prvo različico tega jezika predstavila leta 2003, trenutno pa je v pripravi različica 2.0. OWL je sestavljen iz naslednjih ravni (Slika 13):

- **OWL Lite** je primeren za problemske domene, kjer je potrebna le klasifikacija hierarhije in preproste omejitve. Pri omejitvi števnosti sta tako dovoljeni le vrednosti 0 in 1. V praksi se OWL Lite ne uporablja tako pogosto, kljub temu da ima večjo izrazno moč kot RDF/S.
- **OWL DL** je bil razvit za zagotovitev največje možne izraznosti, pri čemer se še ohranja celovitost izračunljivosti[18] in rešljivost[19] v sistemu mehanizmov sklepanja. OWL DL vključuje vse konstrukte jezika OWL, vendar so nekateri lahko

[17] **Defense Advanced Research Projects Agency** (DARPA) je ameriška organizacija ministrstva za obrambo, ki skrbi za razvoj novih tehnologij za uporabo v vojaške namene.
[18] **Celovitost izračunljivosti** (ang. computational completeness) je zagotovilo, da bodo vsi izračuni zanesljivo izvedeni.
[19] **Rešljivost** (ang. decidability) pomeni, da so vsi izračuni izvedljivi v končnem času.

uporabljeni samo pod določenimi pogoji. Oznako OWL DL je jezik dobil zaradi skladnosti z opisno logiko.

- **OWL Full** temelji na drugačni semantiki kot OWL Lite in OWL DL, saj je v tem primeru temeljna združljivost s shemo RDF. Namenjen je uporabnikom, ki želijo največjo možno raven izraznosti in sintaktično svobodo jezika RDF, vendar brez zagotavljanja celovitosti izračunljivosti. Jeziku OWL Full lahko na primer razred obravnava kot množico posameznih razredov ali kot primerek; to v jeziku OWL DL ni dovoljeno.

Opisane ravni OWL predstavljajo naraščajočo stopnjo izraznosti, kar pomeni, da višje ravni vsebujejo funkcionalnosti nižjih. Trditve, predstavljene na nižjem nivoju, so veljavne tudi na višjem, obratno pa to ne velja.

Podatki zajeti v jeziku OWL se interpretirajo kot sklopi primerkov in sklopi lastnosti, ki primerke povezujejo med seboj. Ontologije OWL sestavljajo sklopi aksiomov, ki postavljajo omejitve na primerke (razrede) in tipe relacij, ki so dovoljene med njimi. Ti aksiomi določajo semantiko in tako omogočajo sistemom povzemanje dodatnih informacij glede na podane semantične podatke. OWL si preprosto lahko predstavljamo kot množico trojic RDF, vendar le-ti uporabljajo slovar OWL, ki jim daje v okviru jezika OWL poseben pomen.

2.3 POSLOVNA PRAVILA

2.3.1 *OPREDELITEV*

Za poslovna pravila najdemo številne opredelitve, vendar je v okviru obstoječe doktorske disertacije uporabljena naslednja: **Poslovno pravilo je pravilo, ki je v pristojnosti poslovnega sistema** (OMG 2008).

Za pravilo, ki je v pristojnosti poslovnega sistema, velja, da ga lahko organizacija predpiše, pregleda, popravi in opusti glede na potrebe. Če pravilo ni v pristojnosti poslovnega sistema, potem ni poslovno pravilo. Primer je "zakon" težnosti, ki očitno ni poslovno pravilo; prav tako to niso matematična "pravila".

Bolj pomembno vprašanje pri opredelitvi poslovnega pravila je pomen "pravila". Zagotovo pravilo opredeljuje navodila pri izvajanju akcij tako v vsakdanjem življenju kot pri poslovanju. Takšno ali podobno opredelitev najdemo v večini slovarjev. Ko se vprašanja lotimo še bolj podrobno, ugotovimo, da v primeru, če pravila opredeljujejo navodila pri izvajanju akcij, potem morajo prav tako opredeljevati dejanske kriterije za odločanje in vodenje pri izvajanju akcij. Z drugimi besedami imajo pravila vlogo kriterija za sprejemanje odločitev v kontekstu poslovnih pravil.

Ta ugotovitev je zato zelo pomembna predvsem za uporabnike, ki skrbijo za izdelavo poslovnih modelov. Npr. pri obvladovanju poslovnih procesov je najbolj pogosta interpretacija poslovnega pravila kot kriterij v točkah odločitve procesnega modela. Takšne točke odločitve so pogosto relativno enostavne, npr. ali bomo stranko obravnavali kot komitenta z nizko, srednjo ali visoko boniteto. V ostalih primerih pa so takšne točke lahko zelo zapletene, npr. ali naj zahtevo po izplačilu zavarovalne premije obravnavamo

kot odobreno, zavrnjeno ali kot morebitno goljufijo. Za takšne bolj kompleksne primere potrebujemo posebne tehnike sklepanja, ki nam omogočajo zajem takšnega scenarija.

Področje poslovnih pravil izhaja z raziskovalnega področja umetne inteligence, kjer so jih uspešno uporabili kot način predstavitve znanja. V sistemih za obvladovanje znanja lahko znanje in sklepanje novih dejstev poslovnih uporabnikov zajamemo v obliki kompleksnih omrežij pravil. Pravila so ponavadi zajeta v deklarativnih jezikih, ki ne opredeljujejo urejenosti oz. vrstnega reda izvajanja. Pravila so shranjena v bazi pravil, medtem ko jih obdeluje posebna komponenta, ki se imenuje stroj za sklepanje. Stroj za sklepanje preverja pogoje pravil in se odloča glede na izpolnjenost pogojev, katero pravilo je potrebno izvršiti (Bajec and Krisper 2005).

Veliko raziskovalcev se je ukvarjalo z uporabo poslovnih pravil v okviru podatkovnih baz. Kot rezultat so se najprej pojavile aktivne podatkovne baze (Morgenstern 1983), ki v nasprotju s pasivnimi podatkovnimi bazami vsebujejo aktivne komponente, kot so prožilci in procedure, ki skrbijo za vzdrževanje podatkovne integritete. Naslednja raziskava se je osredotočila na deduktivne podatkovne baze (Gallaire, et al. 1984). Medtem ko se tradicionalne podatkovne baze ukvarjajo z znanjem v obliki dejanskih podatkov, so deduktivne podatkovne baze osredotočene tudi na namenske podatke. Takšno znanje lahko v celoti zapišemo v obliki pravil, ki so shranjena v bazi pravil, še preden je takšna podatkovna baza vzpostavljena.

Ko se je pojavila ideja o implementaciji poslovnih pravil v okviru sistemov za upravljanje s podatkovnimi bazami, se je veliko raziskav osredotočilo na odkrivanje metod za predstavitev poslovnih pravil v podatkovnih modelih (Tanaka 1992). V entitetnih modelih (ERM) lahko zajamemo zgolj statična poslovna pravila, medtem ko dinamičnih ne moremo, saj ERM ne dovoljuje eksplicitne predstavitve dogodkov, pogojev in akcij. Zato so najprej nastale številne razširitve (npr. ER-RM, BIER ipd.), kot tudi tehnike in metode (npr. ELH pri SSADM, Rossova metoda ipd.).

Z vedno večjo priljubljenostjo poslovnih pravil se je pojavila tudi potreba po neposrednem obvladovanju poslovnih pravil zaradi podpore celotnemu življenjskemu ciklu poslovnih pravil. Tako so nastali številni raziskovalni projekti (RUBRIC, TEMPORA ipd.), ki so se ukvarjali z odkrivanjem, analizo, modeliranjem, klasifikacijo, formalizacijo in dokumentacijo.

Poslovna pravila pa so se pojavila tudi v objektno usmerjeni skupnosti. Jezik OCL (Object Constraint Language) je zanimiva podpora pravilom v okviru objektno usmerjenega razvoja, vendar je primeren le na načrtovalski ravni, medtem ko za zajem zahtev in analizo ni primeren, saj je za poslovne uporabnike preveč tehnično usmerjen.

Organizacija OMG se ukvarja tudi s predstavitvijo pomena v okviru poslovnih pravil, kjer je nastal predlog Semantic of Business Vocabulary and Business Rules (SBVR). S pomočjo omenjenega standarda želijo poslovnim uporabnikom omogočiti opredelitev poslovnih pravil v jeziku, s katerim se srečujejo v okviru poslovanja ter zajem teh pravil v jasni in nedvoumni obliki, ki jo je mogoče preslikati tudi v druge predstavitve (OMG 2008).

Business Rules Group[20] je interesna skupina, ki se ukvarja s področjem poslovnih pravil že od leta 1980. V okviru svojega dolgoletnega delovanja so predstavili tudi **manifest poslovnih pravil**, kjer v sledečem seznamu predstavljamo zgolj nekaj ključnih elementov:

- **Zahteve so na prvem mestu.** Pravila se primarno uporabljajo kot del poslovnih in tehnoloških modelov.
- **Ločitev od procesov in ne delitev po aplikacijah.** Pravila temeljijo na dejstvih in dejstva na konceptih, opredeljeni z izrazi. Izrazi izražajo poslovne koncepte; dejstva uvajajo trditve o teh konceptih; pravila pa omejujejo in podpirajo ta dejstva. Pravila so osnova znanja organizacije, t. j. osnova poslovnega znanja. Za pravila je potrebno skrbeti in z njimi upravljati.
- **Deklarativna in ne proceduralna oblika.** Pravila naj bodo izražena v deklarativni obliki, podobno stavkom v naravnem jeziku, saj so namenjena poslovnim uporabnikom.
- **Dobro opredeljene in ne ad hoc omejitve.** Poslovna pravila morajo biti zajeta v takšni obliki, da lahko njihovo ustreznost preverijo poslovni uporabniki. Poslovna pravila morajo biti predstavljena tudi v takšni obliki, da jih lahko med seboj dosledno primerjamo.
- **Poudarek na poslovanju in ne na tehnologiji.** Pravila opredelujejo poslovno prakso in smernice vodenja, podpirajo pa jih poslovni cilji in preoblikujejo različni vplivi. Cena uveljavljanja pravil mora biti uravnovešena s poslovnimi tveganji in pričakovanji, do katerih lahko pride.
- **Poudarek na poslovnih in ne uporabnikih IT.** Pravila morajo izhajati iz znanja poslovnih uporabnikov.
- **Obvladovanje poslovne logike in ne strojne oz. programske platforme.** Za izboljšanje poslovne prilagodljivosti je ključnega pomena sposobnost učinkovitega spreminjanja pravil.

2.3.2 *PRIMERJAVA S SEMANTIČNIM SPLETOM*

Med področjem semantičnega spleta in poslovnih pravil obstajajo številne skupne točke in tudi razlike (Spreeuwenberg and Gerrits 2005). V nadaljevanju poglavja bodo le-te v strnjeni obliki tudi predstavljene.

Semantični splet in poslovna pravila imajo **skupne korenine**, saj imata obe področji začetke v raziskovalnem področju umetne inteligence. Kljub temu se akterji posameznih raziskovalnih skupnosti med seboj ločujejo. Eden od vzrokov je zagotovo zgodovinsko dejstvo, da umetna inteligenca ni nikoli uspela zadovoljiti visokih zahtev, ki so jih imeli prvotni uporabniki. Ena od osnovnih študij umetne inteligence je predstavitev znanja, kjer gre za formalizacijo domenskega znanja in sklepanje novih dejstev s pomočjo logike in logičnih izpeljav. Rezultat omenjenega raziskovalnega področja so predvsem ekspertni sistemi[21]. Akterji na področju semantičnega spleta pa se bolj zanimajo za področje

[20] http://www.businessrulesgroup.org/home-brg.shtml
[21] **Ekspertni sistem** (ang. expert system) je programska oprema, ki modelira delovanje človeškega eksperta, večinoma v točno določeni problemski domeni.

obvladovanja znanja, kjer je eden glavnih izzivov predstavitev pravih informacij ob pravem času in pravem mestu.

Področje poslovnih pravil in semantičnega spleta ima zelo **različne ciljne uporabnike**. Medtem ko je pri pristopu obvladovanja poslovnih pravil eden od glavnih ciljev izboljšana komunikacija med uporabniki, je pri semantičnem spletu glavni cilj izboljšava komunikacije med računalniki. Pri obeh pristopih pa je pričakovan stranski učinek tudi izboljšana komunikacija med uporabniki in računalniki. Glede na različne ciljne uporabnike se oba pristopa nahajata na različnih ravneh pri OMG-jevi arhitekturi MDA. Model ontologije se nahaja na izvajalni ravni, kar je pri MDA analogno ravni PSM, medtem ko se poslovni slovar in poslovna pravila nahajata na najvišji ravni CIM. Analiza obstoječih orodij za obvladovanje ontologij (Denny 2002, Gómez-Pérez 2002) pa prikazuje drugačen trend. Pri razvoju ontologij se veliko raziskovalcev ukvarja z zmanjšanjem kompleksnosti razvoja ontologij, ki bi jih razvijali domenski eksperti (poslovni uporabniki brez ustreznega tehničnega znanja formalne logike ali računalniškega programiranja) in ne strokovnjaki, ki obvladajo formalno logiko ter razvoj ontologij. Ravno s to težavo se ukvarjamo tudi v tej disertaciji, v kateri je podrobna rešitev predstavljena v obliki modela za hiter razvoj ontologij v poglavju 3.

Tehnike obeh področij, semantičnega spleta in poslovnih pravil imajo **isti cilj**, t. j. zajem pomena realnega sveta, neodvisno od izbrane aplikacije ali opravila. Na ta način se modeli ontologije in poslovni slovarji razlikujejo od pristopov konceptualnega načrtovanja (npr. UML ali ORM), ki so namenjeni predvsem opredelitvi domenskega znanja za izbrano aplikacijo. Pomembno je ostati čim bolj neodvisen od določene aplikacije ali opravila in na takšen način spodbujati ponovno uporabo ontologij in poslovnih slovarjev.

Poslovni slovar in ontologijo sestavljajo medsebojno povezani koncepti in pravila (npr. identiteta, kardinalnost, taksonomija ipd.), ki omejujejo pomen prvotno opredeljenih konceptov, tako da gre za **predstavitev v podobni obliki**. Pri ontologijah je pomen opredeljen s strukturiranimi relacijami med koncepti in dodatnimi omejitvami. Pri poslovnem slovarju pa lahko semantiko opredelimo tudi z opredelitvijo koncepta. Takšna opredelitev je lahko neformalna, vsak koncept pa takšno definicijo mora vsebovati. Pristop s poslovnimi pravili je tudi bolj osredotočen na naravni jezik oz. uporabniku berljiv način prikaza. To v praksi pomeni, da opredelitve in poslovna pravila niso omejena z izbranim formalnim jezikom za zajem. Pri semantičnem spletu pa mora biti vsak element, ki je del ontologije, skladen s formalnim jezikom, saj ga v nasprotnem primeru ni mogoče uporabiti na izvajalni ravni.

Oba pristopa se razlikujeta tudi v **izrazni moči**. Zahtevana raven izraznosti je pri poslovnih pravilih visoka, saj vključuje logike višjega reda, predikatno logiko itd. Pravila na semantičnem spletu pa v glavnem temeljijo na hornovi logiki ali kakšnem drugem mehanizmu predikatne logike, zato je izraznost omenjenih jezikov vprašljiva (Horrocks, et al. 2003). Zelo pomembna razlika pa je tudi v predpostavki zaprtega sveta, ki velja za sisteme za obvladovanje poslovnih pravil in predpostavki odprtega sveta, na kateri temeljijo jeziki za predstavitev znanja na semantičnem spletu.

2.3.3 *PRISTOP MDA*

Če pogledamo zgodovino razvoja programske opreme, je opaziti izrazito povečanje uporabe abstraktnih modelov. Načrtovanje postaja vedno bolj ločeno od izvajalne platforme, tako da so nastali modeli vedno bolj ponovno uporabljivi, ustvarjajo pa jih lahko tudi uporabniki, ki nimajo toliko tehničnega in računalniškega znanja. Tako je načrtovanje vedno bližje pridobivanju znanja, trenutno stanje na tem področju pa je zelo dobro predstavljeno v Model Driven Architecture (MDA) (Đurić, et al. 2005).

MDA je pristop k razvoju programske opreme, ki opredeljuje smernice za strukturiranje specifikacij v obliki modela. MDA opredeljuje tri poglede (različne ravni abstrakcije) na obravnavani sistem. Za vsako raven abstrakcije lahko opredelimo model, ki prikazuje izbran pogled na sistem. Ti modeli so naslednji:

- **računsko neodvisen model** (CIM)[22],
- **model neodvisen od platforme** (PIM)[23] in
- **model odvisen od platforme** (PSM)[24].

Pri poenostavitvi razvoja naprednih aplikacij je cilj vsekakor premakniti potrebno delo uporabnikov iz ravni PSM na raven CIM in delno PIM, medtem ko bi se določene podrobnosti implementacije iz ravni PIM na PSM generirale samodejno.

Pristop SBVR, ki smo ga že omenili v poglavju 2.3.1, je zelo zanimiv, saj se loteva procesa razvoja poslovnih pravil in pomena poslovnega slovarja loteva na način, ki je skladen z arhitekturo MDA (Slika 14).

Slika 14: Vloga SBVR pri pristopu MDA

Pristop SBVR je usmerjen k opredelitvi poslovnih pravil in poslovnih slovarjev, kjer razviti poslovni modeli opisujejo ter podpirajo poslovne in ne sisteme IT. V okviru MDA so

[22] **Računsko neodvisen model** (ang. Computation Independent Model (CIM)) je pogled na sistem, ki ne prikazuje podrobnosti sistemske strukture. Pri razvoju programske opreme ga imenujemo tudi domenski model, za katerega skrbijo domenski uporabniki. Zelo je podoben konceptu ontologije.

[23] **Model neodvisen od platforme** (ang. Platform Independent Model (PIM)) je odvisen od izvedbe, vendar brez podrobnosti posamezne računalniške platforme. Z drugimi besedami ga lahko opišemo kot model, namenjen tehnološko nevtralnim navideznim strojem.

[24] **Model odvisen od platforme** (ang. Platform Specific Model (PSM)) vsebuje podrobne opredelitve računalniškega sistema.

sistemi IT opredeljeni na ravni PIM in PSM, medtem ko je SBVR osredotočen predvsem na model CIM.

SBVR si lahko predstavljamo skozi naslednjih 5 vidikov (Slika 15).

Pri opredelitvi poslovnega slovarja sodeluje skupnost. Na poslovni ravni je najpomembnejša skupnost podjetje, ki poslovna pravila potrebuje in jih uporablja. Vendar je potrebno upoštevati tudi druge skupnosti, kjer podjetje obratuje – partnerska podjetja, interesne skupine, regulatorje ipd. Pomemben vidik skupnosti so podskupine znotraj podjetja, ki morda potrebujejo jedro skupnega razumevanja (začenši z osnovnimi koncepti) v različnih besednjakih, od specializiranega žargona do različnih svetovnih jezikov.

Slika 15: Meta model pristopa SBVR

K jedru skupnega razumevanja prispeva skupnost, in to jedro je sestavljeno iz konceptov in poslovnih pravil. Zavedati se je potrebno dejstva, da se souporablja pomen in ne oblika predstavitve.

Logična opredelitev skrbi za formalno, abstraktno in neodvisno sintakso za zajem pomena jedra skupnega razumevanja. Podpira številne oblike predstavitve, kot so izraz v obliki samostalnika in glagola, prikaz povezav v obeh smereh ipd. Logična opredelitev tako skrbi za (1) preslikavo jedra skupnega razumevanja v besednjake, ki jih uporabljajo skupnosti ter (2) preslikavo v obliko XMI, ki omogoča izmenjavo konceptov, dejstev in poslovnih pravil med posameznimi orodji.

Koncepti in poslovna pravila v jedru skupnega razumevanja morajo biti predstavljena v takšni obliki (slovarju), ki je sprejemljiv in uporaben s strani poslovnih uporabnikov, ki to predstavitev poslovanja uporabljajo. Ti besednjaki so lahko v različnih svetovnih jezikih, računalniških jezikih, kot je UML, ali v posebnih oblikah govornih jezikov, ki jih uporabljajo npr. inženirji ali pravniki.

Pristop SBVR je torej namenjen izmenjavi poslovnih slovarjev in pravil med organizacijami. Konceptualno je najbližje poslovnim uporabnikom in namenjen za

uporabo v poslovne namene, neodvisno od načrtovalcev informacijskih sistemov. Prav tako pa s pristopom SVBR omogočimo pretvorbo poslovnih slovarjev in pravil v predikatni logiki 1. reda v implementacijo informacijskega sistema, za kar skrbijo uporabniki IT.

2.4 PREGLED METODOLOGIJ ZA RAZVOJ ONTOLOGIJ

Že v uvodnem poglavju 1 smo ugotovili, da na področju metodologij za razvoj ontologij obstajajo številni pristopi, ki jih v okviru tega poglavja podrobneje predstavljamo. Ogrodje primerjave metodologij za razvoj ontologij povzemamo po (Corcho, et al. 2003, Senica 2008) in se pri primerjavi osredotočamo na številne elemente in poglede. Pri **načrtu strategije izgradnje** je pomemben načrt življenjskega cikla, strategija upoštevanja aplikacije, uporaba visokonivojskih ontologij in strategija identifikacije konceptov. Zelo pomemben atribut je **načrt procesa razvoja ontologije**, v katerem lahko aktivnosti pri metodologijah za razvoj ontologij razdelimo v naslednje skupine:

- **Aktivnosti upravljanja z ontologijo**
  Postopki pri aktivnostih upravljanja z ontologijo morajo vsebovati načrt razvrščanja nalog pri razvoju ontologije. Potrebno je tudi definirati kontrolne mehanizme in korake za zagotavljanje kakovosti.

- **Aktivnosti razvoja ontologije**
  Pri razvoju ontologije je pomembno, da so opredeljeni postopki za študije izvedljivosti in okolja, za katerega je ontologija razvita. Ob začetku gradnje mora razvijalec ontologije opredeliti postopke in ontologijo konceptualizirati, formalizirati in implementirati. Na koncu sledijo še napotki za vzdrževanje, uporabo in nadaljnji razvoj tako za uporabnike kot tudi razvijalce.

- **Spremljevalne aktivnosti**
  Pri podpori razvoja ontologij so prisotne številne aktivnosti od pridobivanja znanja do vrednotenja, integracije, združevanja, razvrščanja, upravljanja s konfiguracijami itd. Te aktivnosti so prisotne skozi celoten razvojni in upravljaljski cikel.

Pri pregledu obstoječih metodologij pa upoštevamo tudi **uporabo metodologije** (sprejetost s strani drugih razvojnih skupin) in **tehnološko podporo** (obstoj orodij).

Za razvoj ontologij obstajajo številne metodologije. Spodaj podajamo seznam najbolj pogosto uporabljanih in tipičnih predstavnikov določene smeri razvoja:

- **Cyc** (Lenart and Guha 1989),
- **Enterprise Ontology** (Uschold and King 1995),
- **TOVE** (Uschold and Grueninger 1996),
- **KACTUS** (Bernaras, et al. 1996),
- **METHONTOLOGY** (Fernandez-Lopez, et al. 1999),
- **SENSUS** (Swartout, et al. 1997),
- **CommonKADS** (Schreiber, et al. 1999),
- **DOGMA** (Jarrar and Meersman 2002),

- **OTK Methodology** (Sure 2003),
- **HCONE** (Kotis and Vouros 2003),
- **DILIGENT** (Davies, et al. 2006),
- **UPON** (Nicola, et al. 2005).

V nadaljevanju poglavja bodo podrobneje predstavljene obstoječe metodologije, medtem ko bo v poglavju 3 predstavljen predlagan proces hitrega razvoja ontologij. Slika 13 umešča proces ROD in obstoječe pristope glede na dva vidika: primernost za poslovnega uporabnika in kompleksnost razvite ontologije.

Slika 16: Proces razvoja ontologij ROD v primerjavi z obstoječimi pristopi

Iz primerjave vidimo, da je za tehnično neizobraženega uporabnika najbolj primeren pristop folksonomij[25], kjer uporabnik enostavno označuje koncepte z izbranimi oznakami. Na desni strani diagrama se nahajajo najbolj kompleksne metodologije, ki omogočajo tudi razvoj najbolj izraznih in popolnih ontologij. Proces ROD poskuša z zmanjšanjem potrebnega tehničnega znanja približati razvoj tehnično neizobraženim uporabnikom, vendar še vedno omogočiti razvoj kompleksnih ontologij, ki so primerne za uporabo v številnih sistemih.

2.4.1 *Cyc*

Metodologija Cyc je rezultat izkušenj pri razvoju baze znanja Cyc, kjer je zajeto zelo veliko splošnega znanja. Identificirane faze izgradnje ontologije so naslednje:

1. Ročni zapis eksplicitnega in implicitnega znanja, ki nastopa v različnih virih znanja, brez pomoči sistemov za procesiranje naravnega jezika ali sistemov učenja.
2. Kodiranje znanja s pomočjo orodij, ki uporabljajo že obstoječe znanje v bazi znanja Cyc.
3. Avtomatizirana obdelava z orodji, kjer ljudje pomagajo le pri priporočanju primernih virov znanja in razlagi najbolj zahtevnih delov besedila.

V vsaki izmed omenjenih faz se izvajata naslednji aktivnosti:

[25] **Folksonomija** (ang. folksonomy) je klasifikacijski sistem, ki izhaja iz prakse, kjer na podlagi metod sodelovanja obvladujemo oznake za označevanje in kategorizacijo vsebine.

- Razvoj predstavitve znanja in višjenivojske ontologije, ki vsebuje abstraktne koncepte. Termini, kot sta atribut in vrednost atributa, so primeri terminov za predstavitev znanja.
- Predstavitev ostalega znanja z uporabo prej omenjenih terminov.

Metodologija Cyc se je uporabljala v raziskovalnih programih za razvoj visoko zmogljivih baz znanj za izboljšanje tehnologije računalniškega pridobivanja, predstavitve in manipulacije z znanjem. Omenjena metodologija se uporablja samo pri gradnji baze znanja Cyc, kljub temu pa ima Cyc tudi nekaj primerov uporabe, ki prikazujejo znanje v različnih domenah in z različnih pogledov.

Za aplikacije, ki uporabljajo ontologijo Cyc, obstaja več modulov in mehanizmov sklepanja, ki so že vgrajeni v bazo znanja Cyc. Eden izmed njih je sistem za integracijo heterogenih podatkovnih baz. Ta modul povezuje slovar Cyc s podatkovnimi shemami ostalih podatkovnih baz. Na ta način se podatki v podatkovnih bazah interpretirajo na način, ki se uporablja v slovarju Cyc. Modul za iskanje znanja v zajetih informacijah omogoča izvajanja poizvedb v naravnem jeziku. Naslednji modul pa omogoča razširitev baze znanja Cyc z informacijami s svetovnega spleta. Poleg modulov je bilo razvitih nekaj agentov Cyc, ki uporabljajo osnovno znanje iz baze Cyc, ostalo pa iz specifičnih domen agentov.

Projekt Cyc je močno povečal prepoznavnost tehnologij na področju obvladovanja znanja, hkrati pa je njegova večja pomanjkljivost ravno dejstvo, da so želeli v okviru projekta zmodelirati "celoten svet". Med samim izvajanjem projekta se je ta kompleksna naloga razdelila na manjše dele, tudi z razdelitvijo višjenivojske ontologije.

2.4.2 *ENTERPRISE ONTOLOGY*

Metodologija Enterprise Ontology je nastala pri razvoju ontologije za modeliranje procesov in predpisuje navodila ter korake za razvoj ontologij, ki so naslednji:

1. **Identifikacija namena in področja**
   Pomembno je opredeliti, zakaj gradimo ontologijo, kakšna je njena načrtovana uporaba in kateri so pomembni termini, ki jih želimo zajeti.

2. **Gradnja ontologije**
   a. ***Zajem ontologije***
      Namen zajema ontologije je identifikacija ključnih konceptov in relacij v izbrani problemski domeni.
      Osredotočiti se moramo na koncepte in ne zgolj besede. Pri tem moramo uporabljati natančne in jasne opise konceptov in relacij.
      Na voljo imamo več strategij:
      - Pristop od *spodaj navzgor* je najbolj specifičen. Pri njem uporabljamo postopek generalizacije, da pridemo do abstraktnih konceptov. Rezultat je ontologija z visoko ravnjo podrobnosti.
      - Pri pristopu od *zgoraj navzdol* začnemo z abstraktnimi koncepti in pri njih uporabimo specializacijo. Rezultat je ontologija, v kateri lažje obvladujemo raven podrobnosti.

- Pri pristopu od *sredine navzven* začnemo z osnovnimi koncepti in uporabljamo generalizacijo ter specializacijo. Rezultat je ontologija, v kateri dobimo ravnovesje med ravnmi podrobnosti.

b. Kodiranje
Ekspliciten zajem pridobljenega znanja s formalnim jezikom.

c. Integracija obstoječih ontologij
Med procesom zajema in kodiranja ontologije je potrebno razmisliti o uporabi in vključevanju že obstoječih ontologij.

3. **Ocenjevanje**
Razvijalci tehnično presodijo ontologijo, povezano programsko okolje in dokumentacijo glede na specifikacijo zahtev, vsebinska vprašanja, dejansko stanje itd.

4. **Dokumentiranje**
Priporočena navodila za dokumentiranje ontologij, ki se razlikujejo glede na tip in namen uporabe.

Pomanjkljivosti metodologije Enterprise Ontology so predvsem v odsotnosti upravljanja projektov in procesov, ki se izvajajo pred in po razvoju ontologije. Prav tako so nekateri koraki, kot je npr. zajemanje znanja, premalo podrobno opredeljeni.

Orodji, ki podpirata omenjeno metodologijo za razvoj ontologij, sta Enterprise Toolset in Ontolingua Server. Enterprise Toolset temelji na agentih, ki omogočajo integracijo in združljivost z drugimi orodji. Komponente orodja so Procedure Builder za zajem procesnih modelov, Agent Toolkit za podporo razvoja agentov, Task Manager za integracijo in vizualizacijo procesov ter Enterprise Ontology za komunikacijo.

Omenjena metodologija je imela zelo velik vpliv na kasnejše pristope za razvoj ontologij. Prav tako večina orodij za obvladovanje ontologij temelji na omenjenem pristopu (Corcho, et al. 2003, Davies, et al. 2006).

2.4.3 *TOVE*

Metodologija je nastala na podlagi izkušenj iz projekta TOVE, kjer so razvili ontologijo v okviru problemske domene modeliranja poslovnih procesov in aktivnosti. Vključuje izdelavo logičnega modela znanja, ki je kasneje opisan v sklopu ontologije. Model sicer ni zgrajen neposredno, ampak se naprej opredeli neformalen opis na podlagi specifikacije ontologije, nato pa se ta opis formalizira.

Metodologija predlaga naslednje korake:

1. **Zajem motivacijskih scenarijev**
Avtorji metodologije trdijo, da mora biti razvoj ontologij podprt s scenariji, ki izhajajo iz aplikacij. Scenariji so problemi ali primeri, ki še niso vključeni v

obstoječih ontologijah. Tak scenarij hkrati ponuja možne intuitivne rešitve problema, ki tvorijo neformalno semantiko za objekte in relacije in bodo kasneje vključeni v ontologijo.
Vsak osnutek nove ontologije ali razširitve ontologije naj bi opisoval enega ali več motivacijskih scenarijev pripadajoče rešitve.

2. **Formulacija neformalnih vprašanj kompetence**
   Vprašanja temeljijo na zajetih scenarijih iz prejšnjega koraka in jih lahko obravnavamo kot zahteve, izražene v obliki vprašanj. S pomočjo ontologije moramo biti sposobni odgovoriti na omenjena vprašanja z uporabo opredeljenih konceptov, ki temeljijo na aksiomih. V tem koraku gre za neformalna vprašanja kompetence, ki še niso izražena v formalnem jeziku ontologije.
   Vprašanja kompetence so strukturirana tako, da se odgovor na določeno vprašanje lahko uporabi tudi kot odgovor pri bolj splošnem vprašanju iste ali druge ontologije, kar dosežemo s pomočjo operacij kompozicije in dekompozicije. Na takšen način je poenostavljeno tudi prepoznavanje že zajetega znanja pri ponovni uporabi oz. integraciji ontologij.
   Namen vprašanj je opredelitev omejitev, kaj vse ontologija vsebuje. Takšen pristop se razlikuje od drugih metodologij, ki temeljijo na modelu ontologije s pripadajočimi zahtevami. Vprašanja kompetence se lahko uporabijo tudi za oceno, ali ontologija vključuje vse zahteve.

3. **Določitev terminologije v jeziku opisne logike prvega reda**
   a. ***Zbiranje neformalne terminologije***
      Na podlagi vprašanj kompetence lahko opredelimo osnovne pojme, ki jih je potrebno zajeti v naši problemski domeni. Ti pojmi se kasneje uporabijo za specifikacijo terminologije v formalnem jeziku opisne logike prvega reda.

   b. ***Specifikacija formalne terminologije***
      Na podlagi neformalnih vprašanj kompetence se pri ontologiji določi terminologija ontologije s pomočjo izbranega jezika. Ti pojmi kasneje v 5. koraku omogočajo izražanje opredelitev in omejitev s pomočjo aksiomov.

4. **Oblikovanje formalnih vprašanj kompetence s pomočjo terminologije**
   Na podlagi neformalnih vprašanj kompetence in terminologije se formalno opredelijo vprašanja kompetence.

5. **Opredelitve aksiomov s pomočjo jezika opisne logike prvega reda**
   Aksiomi v ontologiji določajo opredelitev konceptov ontologije in omejitve pri njihovi interpretaciji. Preprosta opredelitev konceptov v logiki prvega reda nam ne zagotavlja uporabne ontologije, temveč potrebujemo za opredelitev semantike še dodatne omejitve v obliki aksiomov.
   Če predlagani aksiomi niso zadostni za predstavitev formalnih vprašanj kompetence, je potrebno v ontologijo dodati nove koncepte in aksiome. Razvoj aksiomov je tako očitno iterativen proces.

6. **Opredelitev pogojev za določanje popolnosti ontologije**
   Na osnovi formalno podanih vprašanj kompetence je potrebno opredeliti pogoje, pod katerimi so rešitve teh vprašanj popolne.

Omenjena metodologija ne vključuje procesov, ki potekajo pred in po razvoju ontologije ter procesov upravljanja projekta. Tehnike in aktivnosti ravno tako niso opisane natančno, vendar je omenjeni pristop z uporabo vprašanj kompetence, na katere mora biti ontologija sposobna dati odgovor, zelo praktičen, še posebej, če uporabniki niso tehnično izkušeni v modeliranju, imajo pa veliko domenskega znanja. Zaradi omenjene prednosti je bil ta pristop uporabljen tudi v nekaterih kasnejših metodologijah, kot bo predstavljeno v nadaljevanju.

Slika 17: Razvojne faze metodologije TOVE

2.4.4 *KACTUS*

Metodologija je nastala v okviru projekta Esprit KACTUS, po katerem nosi tudi ime. Eden izmed ciljev omenjenega projekta je bil raziskati možnosti ponovne uporabe znanja v kompleksnih tehničnih sistemih s pomočjo ontologij.

Pristop razvoja ontologij je v tem primeru pogojen z razvojem aplikacije, kjer se ob novi aplikaciji vedno zgradi tudi ontologija, ki vključuje potrebno znanje za aplikacijo. Ontologije je moč razviti s ponovno uporabo že obstoječih ontologij, lahko pa so vključene tudi v ontologije kasnejših aplikacij. Pri razvoju nove aplikacije se izvedejo naslednji koraki:

1. **Specifikacija aplikacije** predstavi kontekst aplikacije in pogled na komponente, ki jih z aplikacijo želimo razviti.
2. **Osnovni načrt**, ki temelji na povezanih kategorijah višjenivojskih ontologij. Uporabi se seznam pojmov in opravil iz prejšnjega koraka, da dosežemo različne poglede na globalni model v skladu s povezanimi kategorijami višjenivojskih ontologij. Proces načrtovanja vključuje tudi iskanje obstoječih ontologij za druge aplikacije.
3. **Preurejanje in strukturiranje ontologije**, da dosežemo končni model. Največjo homogenost znotraj modulov dosežemo z minimalnim povezovanjem, ki nam zagotavlja, da moduli med seboj niso preveč odvisni, a vendar čim bolj skladni.

Omenjena metodologija se je prvotno uporabljala pri problemski domeni električnih omrežij in ni opredeljena do podrobnosti. Življenjski cikel ontologij, razvitih s to metodologijo, je podoben življenjskemu ciklu aplikacije, na katero se nanaša, zato je tudi sam proces razvoja ontologij aplikacijsko odvisen.

2.4.5 *METHONTOLOGY*

Ogrodje METHONTOLOGY omogoča konstruiranje ontologij na nivoju znanja ter vpeljuje nekatere pristope iz bolj razvite discipline razvoja programske opreme. Razvojni proces namreč temelji na aktivnostih standarda IEEE za razvoj programske opreme, aktivnosti pa so razdeljene v življenjski cikel, skozi katerega potuje ontologija po določenih stopnjah, kjer se te aktivnosti izvedejo.

Pri gradnji ontologije se najprej opredeli konceptualni model, ki se kasneje preslika v formaliziran model. Na koncu pridemo do izvajalnega modela, kjer gre za implementacijo ontologije v izbranem jeziku (Slika 18).

Slika 18: Zapis ontologije v metodologiji METHONTOLOGY

2.4.5.1 Razvojni proces ontologij

Proces opredeljuje aktivnosti, ki se izvedejo pri gradnji ontologije in jih delimo v naslednje kategorije:

- **Aktivnosti upravljanja z ontologijo** vključujejo planiranje, kontrolo in zagotavljanje kakovosti.
  Pri planiranju se opravi identifikacija nalog, ki jih je potrebno opraviti, njihova priprava in izračun potrebnega časa in virov (ljudje, programska oprema, strojna oprema). Aktivnost kontrole skozi celoten življenjski cikel ontologije zagotavlja, da se aktivnosti izvedejo, kot je bilo načrtovano in da ni odstopanj. Aktivnost zagotavljanja kakovosti preverja kakovost rezultatov metodologije (ontologije, programska oprema, dokumentacija ipd.).
- **Razvojno usmerjene aktivnosti** delimo v aktivnosti pred razvojem, aktivnosti razvoja in aktivnosti po razvoju.
  Pred razvojem se preuči okolje, v katerem se bo ontologija uporabljala ter izvedljivost razvoja. Med razvojem se najprej v okviru aktivnosti specifikacije navedejo razlogi za gradnjo ontologije, predvidena uporaba ontologije in njeni končni uporabniki. V aktivnosti konceptualizacije se domensko znanje strukturira v razumljive modele na nivoju znanja. Formalizacija te konceptualne modele pretvori v formalne modele. Na koncu se formalni modeli implementirajo v aktivnosti implementacije. Po razvoju aktivnost vzdrževanja skrbi za posodabljanje in morebitne popravke ontologije.

- **Podporne aktivnosti** se izvajajo sočasno z razvojno usmerjenimi aktivnostmi. Prva izmed njih je aktivnost zajemanja znanja, ki se ukvarja s pridobivanjem znanja s strani strokovnjakov ali s pomočjo avtomatskega učenja ontologij. Sledi vrednotenje, ki ocenjuje razvito ontologijo, programsko opremo in dokumentacijo, ali so v skladu z zahtevami. Aktivnost integracije se izvede v primeru ponovne uporabe drugih ontologij, in sicer pri spajanju ali združevanju. Rezultat spajanja je nova ontologija, medtem ko združevanje vzpostavi le povezave med obstoječimi ontologijami in tako ohranja njihovo obliko. Aktivnost dokumentiranja zajema razlago vsake dokončane stopnje in izdelka, aktivnost upravljanja s konfiguracijami pa zajema verzije ontologij, programske opreme in dokumentacije z namenom kontrole nad spremembami.

Proces razvoja predstavlja aktivnosti, ki se morajo izvesti, vendar ne opisuje, kdaj katera aktivnost nastopi in kako si morajo le-te slediti. To določa drugi del metodologije, življenjski cikel ontologije, ki vzpostavlja stopnje in aktivnosti pri opredelitvi ontologije.

2.4.5.2 Življenjski cikel ontologije

Naloga življenjskega cikla ontologije je razporeditev aktivnosti razvoja. Je cikličen in temelji na prototipiranju, kar omogoča inkrementalni razvoj ontologij in dovoljuje tudi zgodnje preverjanje skladnosti in preurejanje ontologije v razvoju. Vsak cikel se začne s planiranjem aktivnosti, ki zajemajo načrtovanje opravil in virov. Sledijo aktivnosti razvoja. in sicer najprej specifikacija, vzporedno pa se pričnejo odvijati aktivnosti upravljanja, kontrole in zagotavljanja kakovosti ter podporne aktivnosti – zajemanje znanja, integracija, vrednotenje, dokumentiranje in upravljanje konfiguracij (Slika 19).

Slika 19: Razvojni proces in življenjski cikel metodologije METHONTOLOGY

V vsakem ciklu gre prototip ontologije skozi vse razvojne aktivnosti (specifikacija, konceptualizacija, formalizacija, implementacija in vzdrževanje). Vsakemu ciklu sledi nov, ki upošteva zaključke vrednotenja oziroma ocenjevanje, dokler ne postane prototip dovolj

zrel in zadošča zahtevam vrednotenja. V vsakem razvojnem ciklu potekajo naslednji koraki:

- **specifikacija prototipa**,
- **izgradnja konceptualnega modela** iz znanja, pridobljenega v sklopu aktivnosti zajemanja znanja, ki večinoma poteka med konceptualizacijo,
- **formalizacija konceptualnega modela**,
- **implementacija formaliziranega konceptualnega modela** (lahko se izvede samodejno, če je na voljo avtomatizirana preslikava iz formalizacije v jezik za implementacijo ontologije) in
- **vzdrževanje ontologije**, ki lahko vodi tudi do novega razvojnega cikla, če zahteve niso dosežene ali se pojavijo nove.

Kot je bilo že prej omenjeno, podporne aktivnosti in aktivnosti upravljanja potekajo vzporedno z razvojnimi aktivnostmi, medtem ko vključenost podpornih aktivnosti ni enakomerna skozi celoten življenjski cikel ontologije. Aktivnosti zajemanja znanja, integracije in vrednotenja se bolj intenzivno izvajajo med fazo konceptualizacije ontologije, ki je tudi glavna faza metodologije (Slika 19). Na začetku razvoja se zajame največ znanja, ontologije se na konceptualni ravni integrirajo pred implementacijo, priporočljivo pa je tudi izvajanje natančnega vrednotenja v fazi konceptualizacije, tako da se izognemo kasnejšim napakam.

Metodologija METHONTOLOGY se uporablja pri razvoju ontologij za različne problemske domene in podporo številnim aplikacijam. Osnovno orodje, ki nudi podporo metodologiji, je WebODE, medtem ko so primerna tudi orodja ODE, Protégé in OntoEdit.

2.4.6 *SENSUS*

Ontologija SENSUS se uporablja za procesiranje naravnih jezikov in obsega široko konceptualno strukturo za razvoj strojnih prevajalnikov. Vsebina ontologije je bila zbrana s povzemanjem in spajanjem informacij iz različnih elektronskih virov znanja. Baza ontologije je bila razvita s spajanjem dveh višjenivojskih lingvističnih ontologij in semantičnih kategorij slovarja, kasneje pa sta bila dodana še španski in japonski besednjak.

SENSUS vsebuje več kot 70.000 konceptov, ki so organizirani v hierarhijo glede na njihovo raven abstrakcije. Vključuje pojme na višji in srednji ravni abstrakcije, v splošnem pa ne vsebuje pojmov iz specifičnih domen. Za izgradnjo domenskih ontologij se domenski pojmi povezujejo z ontologijo SENSUS, nepotrebni pojmi v ontologiji pa se odstranijo.

2.4.6.1 Razvojni proces

Metodologija uporablja strategijo od sredine navzven za izpeljevanje domenskih ontologij iz večje ontologije. Koraki pri razvoju ontologije v določeni domeni so naslednji:

1. Zbirka pojmov je osnova,
2. pojmi se ročno povežejo z ontologijo SENSUS,
3. vključeni so vsi koncepti in poti od osnovnih pojmov do korena ontologije SENSUS,

4. vključijo se še dodatni pojmi, ki so v okviru domene pomembni,
5. vozlišča, skozi katera poteka veliko poti, se vključijo v ontologijo v obliki celotnega poddrevesa. Pri tem koraku predvidevamo, da je v primeru, ko je več vozlišč poddrevesa pomembnih, velika verjetnost, da so pomembna tudi ostala vozlišča poddrevesa.

Omenjen pristop spodbuja skupno rabo znanja, saj je osnovna ontologija vedno uporabljena za razvoj ontologij v določenih domenah. V primerjavi z drugimi pristopi SENSUS uporablja svojevrsten pristop. Metodologija ni opisana do podrobnosti, manjkajo pa tudi aktivnosti pred in po razvoju upravljanja. Strategija razvoja je tako delno odvisna od aplikacije, saj se osnovni koncepti pri razvoju nove ontologije določajo na osnovi aplikacije.

Slika 20: Pristop SENSUS k razvoju ontologij

2.4.7 *CommonKADS*

CommonKADS ni prava metodologija za razvoj ontologij. Razvita je bila v sklopu evropskega projekta ESPRIT. Pokriva vidike od upravljanja z znanjem, do analize, oblikovanja in implementacije informacijskih sistemov na osnovi znanja. CommonKADS se osredotoča bolj na začetne faze razvoja aplikacij za upravljanje z znanjem. CommonKADS ponuja metode za kreiranje podrobnih opisov nalog znotraj celotnega poslovnega procesa in tehnike za izčrpno analizo, razvoj in shranjevanje znanja. Uporabljena notacija je združljiva z jezikom UML, orodji za podporo metodologiji pa sta PC PACK in KADS22, ki omogočata sodelovanje inženirjev znanj in domenskih strokovnjakov. Metodologija je pomembna predvsem zaradi dejstva, da je uporabljena kot osnova številnim drugim pristopom.

2.4.8 *DOGMA*

Pri metodologiji DOGMA temelji pristop k razvoju ontologij na podatkovnih bazah. Ontologija je pri tem pristopu razdeljena na dva dela:

- **Jedro ontologije** je sestavljeno iz možnih konceptualizacij domene iz realnega sveta. Sestavljeno je iz atomarnih predikatov oz. t. i. leksonov, ki so izraženi s trojicami.
- **Izvršilno raven ontologije** sestavljajo pravila in omejitve, ki se uporabljajo za komunikacijo z aplikacijami, kot osnova pa je uporabljeno jedro ontologije. Plast se uporablja kot posrednik med aplikacijo in ontologijo, tako da lahko na različne aplikacije pripravimo različne poglede oz. interpretacije ontologije.

Za podporo pristopu sta bili razviti tudi orodji DOGMA Server in DOGMA Modeler. DOGMA Server je namenjen shranjevanju, DOGMA Modeler pa modeliranju in urejanju jedra ontologije in izvršilne plasti.

Slika 21: Pristop DOGMA k razvoju ontologij

2.4.9 *OTK METHODOLOGY*

Metodologija OTK je bila razvita v okviru evropskega projekta On-To-Knowledge, pri katerem so se ukvarjali s predstavitvijo in vzdrževanjem aplikacij, ki upravljajo z znanjem v podjetjih na podlagi ontologij.

Slika 22: Ortogonalna procesa razvoja metodologije OTK

Metodologija se deli na dva tipa procesov (Slika 22):

- **razvoj znanja** in
- **meta razvoj znanja**.

Proces razvoja znanja se ukvarja predvsem z uporabo ontologij, medtem ko se meta razvoj znanja ukvarja z začetnim razvojem in vzpostavitvijo ontologije.

2.4.9.1 Proces meta razvoja znanja

Slika 23 prikazuje proces meta razvoja znanja, ki ga sestavlja pet osnovnih korakov. Vsak korak vsebuje številne podkorake, na koncu pa zahteva odločitev o zadostitvi zahtevam ter specifične rezultate. Glavni tok predstavlja korake (faze), ki vodijo do aplikacije za upravljanje znanja na podlagi ontologije. Te faze so: študija izvedljivosti, začetek razvoja, plemenitenje, vrednotenje ter aplikacija in razvoj. Pri vsaki fazi so navedeni tudi najpomembnejši podkoraki. Pri vsaki fazi so označeni dokumenti, ki predstavljajo glavne rezultate oz. izdelke faze. Odločitvene točke pri fazah označujejo najpomembnejše odločitve, ki morajo biti sprejete pred napredovanjem procesa v naslednjo fazo. Rezultati

posamezne faze se ponavadi uporabijo za sprejemanje odločitev. Faze plemenitenje, vrednotenje ter aplikacija in razvoj se ponavadi izvajajo v iterativnih ciklih. Na razvoj takšnih aplikacij ima vpliv tudi proces razvoja programske opreme in človeške ugotovitve.

Slika 23: Koraki meta procesa razvoja metodologije OTK

2.4.9.2 Proces razvoja znanja

Po vzpostavitvi aplikacije za upravljanje z znanjem v organizaciji se proces razvoja znanja izvaja v naslednjih korakih (Slika 24):

- **Kreiranje znanja** in **uvažanje** dokumentov ter meta podatkov. Vsebine morajo biti prilagojene do te mere, da zadoščajo poslovnim pravilom podjetja oz. infrastrukturi upravljanja z znanjem.
- Opredelitev pomembnosti oz. **povezanosti zajetega znanja** z obstoječim znanjem, kjer se opredelijo tudi povezovalni meta podatki.
- Vzpostavitev in **dostop do znanja**, ki zadovolji potrebo po enostavnih povpraševanjih.
- **Uporaba znanja** je osredotočena na uporabo zajetega znanja v podjetjih za izvajanje analiz, povzemanje in njegovo uporabo v programskih agentih.

Slika 24: Proces razvoja znanja pri metodologiji OTK

Pri metodologiji OTK za razvoj ontologij je zelo pomembno, da so napotki domenskim strokovnjakom zelo strokovni. Sodelovanje pri razvoju ontologij zahteva fizično prisotnost in tudi napredno podporo orodij. Omenjena metodologija je zelo podrobno opisana in vsebuje tudi aktivnosti upravljanja projekta ter aktivnosti, ki se izvajajo pred in po razvoju. Določene smernice so povzete tudi po metodologijah CommonKADS, Enterprise Ontology in TOVE. Na voljo je tudi zelo napredno orodje OntoEdit, ki podpira razvoj ontologij po metodologiji OTK v vseh fazah.

2.4.10 *HCONE*

Pri metodologiji HCONE je prisoten dinamičen proces razvoja ontologij, ki se večinoma začne z osnovno različico ontologije, ki jo kasneje popravljamo, izboljšujemo, obogatimo in dopolnimo s podrobnostmi. Razvoj ontologije je podprt skozi celoten življenjski cikel, kar posledično pripelje do uporabne ontologije. Metodologija poskuša spodbujati strokovnjake na področju upravljanja znanja h kontinuiranemu urejanju formalnih konceptualizacij pri vsakodnevnih opravilih, kjer je poudarek na decentraliziranem razvoju in vrednotenju ontologij v kontekstu večjih skupin strokovnjakov za upravljanje znanja.

Metodološko podporo najdemo v orodjih HCONE in SharedHCONE, s pomočjo katerih lahko vzpostavimo okolje za decentraliziran razvoj ontologij, saj spodbujata sodelovanje med strokovnjaki in kontinuirano delo. Pri opredelitvi se srečamo s tremi prostori za shranjevanje ontologij:

- **osebnim prostorom**, v katerem uporabniki opredelijo in združujejo ontologije, obvladujejo verzije ontologij in povezujejo pojme s koncepti;
- **skupnim prostorom**, ki je namenjen shranjevanju razvijajoče se ontologije in je v skupni rabi, ki je na voljo vsem sodelujočim;
- **dogovorjenim prostorom**, ki je prostor, kamor se po razpravah in dogovarjanjih o ustreznosti ontologije prenese dogovorjena različica.

Slika 25: Decentraliziran razvoj ontologij HCONE

Končni cilj metodologije je razvoj usklajene in razumljive ontologije za učinkovito upravljanje z znanjem. Za dosego tega cilja pa morajo biti opravila za upravljanje ontologij vključena v cikle modeliranja in črpanja informacij. Tabela 1 predstavlja stopnje in opravila v življenjskem ciklu ontologije, kjer velja, da je razvoj ontologij v okviru omenjene metodologije iterativen in kontinuiran.

Faze življenjskega cikla ontologije	Procesi	Opravila
Specifikacija	Opredelitev ciljev in področja ter iskanje virov znanja.	• Razprava o potrebah, • izdelava dokumentov, • izbor sodelavcev, • specifikacija področja in ciljev ontologije.
Konceptualizacija	Pridobivanje znanja.	• Zajem in knjižnice ontologij, • pregled višjenivojskih ontologij, • posvetovanje z domenskimi strokovnjaki.
	Razvoj in vzdrževanje ontologije.	• Improviziranje, • upravljanje konceptualizacij, • združevanje različnih verzij, • primerjava lastnih verzij, • generalizacija/specializacija verzij, • dodajanje dokumentacije.
Izkoriščanje	Uporaba ontologije.	• Iskanje po ontologijah, • uporaba v aplikacijah.
	Vrednotenje ontologije.	• skupinsko obravnavanje kritik, • primerjava različnih verzij, • uporaba dogovorjenih ontologij, • vključevanje izvlečkov v ontologije.

Tabela 1: Življenjski cikel ontologij v metodologiji HCONE

2.4.11 *DILIGENT*

Pristop DILIGENT je bil prvotno namenjen obvladovanju pogostim spremembam uporabniških zahtev pri scenarijih dinamične souporabe znanja. Pri razvoju ontologij po metodologiji DILIGENT sodelujejo številni strokovnjaki z različnimi znanji, ki se med seboj dopolnjujejo, vsi skupaj pa sodelujejo pri izdelavi iste ontologije. V decentraliziranih okoljih semantičnega spleta lahko ti strokovnjaki pripadajo celo konkurenčnim organizacijam in so tudi geografsko porazdeljeni. Pogosto so domenski strokovnjaki, ki sodelujejo pri gradnji ontologij, tudi njeni uporabniki, vendar je kljub temu končna skupina uporabnikov veliko večja od razvijalcev.

Osnovna ontologija se zgradi v okviru manjše skupine razvijalcev in je nato na voljo vsem, uporabniki pa imajo možnost lokalne uporabe ter spreminjanja za lastne potrebe. Nadzorni svet razvijalcev vzdržuje in skrbi za kakovost osnovne ontologije. Odgovoren je tudi za nadgrajevanje, ki je ponavadi posledica lokalnih sprememb uporabnikov, ki se nato prenesejo na osnovno ontologijo. Ontologija se razvija preko predlaganih sprememb, celoten proces pa kontrolira nadzorni svet.

Metodologija DILIGENT pri razvoju ontologij opredeljuje pet osnovnih korakov:

1. **Gradnjo**. Proces se prične z gradnjo začetne ontologije, pri kateri sodeluje manjša skupina domenskih strokovnjakov, uporabnikov in razvijalcev ontologije, kjer skušajo najti majhno in sporazumno prvo verzijo skupne ontologije.
2. **Lokalno prilagajanje**. Ko je osnovna ontologija na voljo, jo lahko uporabniki uporabljajo in prilagajajo lastnim potrebam. Pri tem lahko prosto spreminjajo lokalno verzijo ontologije, ne pa tudi skupne ontologije, za katero nadzorni svet zbira predloge za spremembe in beleži lokalne prilagoditve.
3. **Analizo**. Nadzorni svet analizira uporabniške predloge za spremembe in skuša najti podobnosti med ontologijami uporabnikov. Pri tem je glavna aktivnost odločanje, katere spremembe bodo dodane naslednji verziji ontologije, v kateri se poskuša najti ravnotežje med potrebami in zahtevami uporabnikov.
4. **Popravek ontologije**. Nadzorni svet mora redno posodabljati skupno ontologijo, da se lokalne verzije ne oddaljijo preveč od skupine. Cilj pri nadgrajevanju oz. popravljanju ontologije je tako čim bolje poenotiti ontologijo s potrebami uporabnikov in s tem doseči večjo sprejemljivost skupne ontologije. Pri tem sodelujejo uporabniki s predlogi za spremembe, domenski strokovnjaki potrdijo skladnost sprememb s problemsko domeno, razvijalci ontologij pa skrbijo za tehnično izvedbo sprememb.
5. **Lokalno posodabljanje**. Ko je na voljo nova verzija skupne ontologije, lahko uporabniki posodobijo svojo lokalno verzijo in s tem uporabljajo novo dogovorjeno znanje.

Slika 26: Proces decentraliziranega razvoja ontologij metodologije DILIGENT

2.4.12 *UPON*

UPON je inkrementalna metodologija za razvoj ontologij, ki temelji na poenotenem procesu za razvoj programske opreme (Unified Software Development Process) in je podprta z jezikom za modeliranje UML. Od ostalih pristopov se razlikuje predvsem po tem, da gre za iterativen in inkrementalen razvoj, ki temelji na primerih uporabe. Namen razvoja, ki temelji na primerih uporabe, je izdelati ontologijo, ki čim bolj ustreza uporabnikom – ljudem in programskim agentom. Narava razvojnega procesa je iterativna,

kar razvijalcem omogoča osredotočenje na izbran del ontologije in inkrementalna, saj se ontologija skozi proces razširja in se vedno bolj osredotoča na podrobnosti.

Razvojni proces se ponavlja skozi serijo ciklov, ki vodijo do končne ontologije. Metodologijo UPON sestavljajo cikli, faze, iteracije in delovni tokovi. Vsak cikel sestavljajo štiri faze: zasnova, obdelava, konstruiranje in faza prehoda. Rezultat cikla je vedno nova verzija ontologije. Vsaka faza se naprej deli na iteracije, kjer pri vsaki iteraciji poteka pet delovnih tokov: zajem zahtev, analiza, načrtovanje, implementacija in testiranje. Začetne faze se bolj osredotočajo na zajem zahtev, poznejše pa ontologijo razširjajo in dopolnjujejo.

Slika 27: Razvojni proces metodologije UPON

2.4.12.1 Delovni tokovi metodologije UPON

Metodologija UPON opredeljuje naslednje delovne tokove:

1. **Zajem zahtev** je proces opredelitve znanja, ki bo v ontologiji zajeto. Glavni cilj je doseči sporazum med oblikovalci, razvijalci in končnimi uporabniki. Koraki so naslednji:
    a. opredelitev problemske domene,
    b. opredelitev razloga (motivacijski scenarij) in namen razvoja,
    c. opredelitev področja in obseg delovanja,
    d. identifikacija vprašanj kompetence in
    e. izdelava diagramov primerov uporabe.
2. **Analiza** vključuje pregled in strukturiranje zahtev, zajetih v prejšnjem delovnem toku. Koraki so naslednji:
    a. preučevanje morebitne ponovne uporabe,
    b. identifikacija pomembnih pojmov in
    c. opredelitev konceptov.

    Rezultat tega delovnega toka je model analize.
3. **Načrtovanje** sestavljajo naslednji koraki:

   a. strukturiranje konceptov in relacij (opredelitev razrednih diagramov) in
   b. realizacija primerov uporabe (opredelitev diagramov zaporedja in sodelovanja).
4. **Implementacija** se ukvarja s formalizacijo ontologije v izbranem jeziku in implementacijo posameznih komponent, ki so lahko formalizirane v različnih jezikih in notacijah. Rezultat tega delovnega toka je model implementacije.
5. **Testiranje** omogoča preverjanje pravilnosti zajetih specifikacij, kjer je razvitih več testnih primerov, ki testirajo skladnost z zahtevami.

V primerjavi s standardom IEEE razvoja programske opreme metodologija UPON vključuje podporo procesom razvoja, dokumentiranja, vrednotenja in zajemanja zahtev, ne vključuje pa procesa upravljanja projekta in procesov pred in po razvoju ontologij. Prednost omenjene metodologije je v podpori številnih orodij za načrtovanje v jeziku UML.

2.5 PREGLED PRISTOPOV ZA EVALUACIJO ONTOLOGIJ

V literaturi obstajajo številni pristopi za evaluacijo ontologij. Glede na pregled obstoječih pristopov v (Gómez-Pérez 1999, Hartmann, et al. 2004, Brank, et al. 2005, Gangemi, et al. 2006) lahko le-te klasificiramo v naslednje kategorije:

- primerjava ontologije s t. i. **zlatim standardom** (Maedche and Staab 2002),
- **uporaba** ontologije **v aplikaciji** in evaluacija rezultatov (Porzel and Malaka 2004),
- primerjava s **podatki o problemski domeni**, ki jo ontologija zajema (Brewster, et al. 2004),
- evaluacija **s strani uporabnikov** (Lozano-Tello and Gómez-Pérez 2004, Noy, et al. 2005).

Ponavadi je lažje in bolj praktično vrednotenje posameznih ravni ontologije posebej, kot pa neposredna evaluacija celotne ontologije. Tako lahko glede na ravni vrednotenja ontologije identificiramo naslednje evaluacije: ***leksikalna***, ***podatkovna***, ***strukturna evaluacija***, ***evaluacija ostalih semantičnih relacij***, ***evaluacija konteksta***, ***sintaktična***, ***strukturna***, ***arhitekturna*** in ***načrtovalska evaluacija***.

Pred uporabo ontologije v programu je potrebno zagotoviti, da je ontologija brez napak. V ta namen so avtorji v (Fahad and Quadir 2008) predstavili obširno klasifikacijo napak in njihov vpliv na kvaliteto ontologije. Napake so razdelili v naslednje skupine: ***napake nedoslednosti***, ***napake nepopolnosti***, ***napake odvečnosti*** in ***načrtovalske anomalije***.

V nadaljevanju bodo predstavljene nekatere metode in orodja iz omenjenih kategorizacij evaluacije ontologij.

Avtorji v (Porzel and Malaka 2004) predlagajo opravilno usmerjen pristop k evaluaciji ontologije. Omogoča uporabo različnih ontologij in merjenje njihove učinkovitosti na določenih opravilih. S preverjanjem doslednosti ontologije se ukvarjajo avtorji v (Baclawski, et al. 2002), kjer je predstavljeno orodje ***ConsVISor*** za preverjanje različnih aksiomov v UML razrednih diagramih in ontologijah RDF ter DAML+OIL. Podoben pristop je tudi ***evOWLution*** (Haase, et al. 2005, Haase and Stojanovic 2005), s katerim je mogoče

preverjati doslednost ontologij OWL. Implementacija evOWLution vsebuje strategije za evaluacijo različnih konstruktov jezika za zapis ontologij in strategije za evaluacijo logične doslednosti. S pristopom ***ONTOMETRIC*** (Lozano-Tello and Gómez-Pérez 2004) lahko načrtovalci ontologije merijo ustreznost obstoječih ontologij glede na zahteve njihovih sistemov. Načrtovalec mora primerjati pomembnost projektnih ciljev in podrobno preučiti značilnosti ontologije. Pristop, ki ga predlagajo avtorji v (Noy, et al. 2005), temelji na označevanju ontologij s strani uporabnikov na odprt način in temelji na uporabi spletnega brskalnika za **Protégé**[26] ontologije. Omogočanje označevanja in pregledovanja ontologij spletnim uporabnikom je zelo pomemben korak pri pospeševanju evaluacije ontologij in ponovni uporabi. Številne metrike za evaluacijo znanja, zajetega v ontologijah, lahko najdemo v (Baumeister and Seipel 2005), kjer avtorji dajejo poudarek predvsem inovativnim metrikam s področja anomalij, ki se pojavljajo pri načrtovanju ontologij. Za implementacijo omenjenih metrik se predlaga deklarativni pristop z uporabo logično usmerjenega jezika FNQuery. Avtorji v (Vrandečič and Gangemi 2006) predlagajo uporabo testiranja enot[27] pri razvoju ontologij, ki se pogosto uporablja pri razvoju programske opreme, kjer se je ta pristop tudi uveljavil. Pristop predvsem temelji na težavah pri načrtovanju in vzdrževanju ontologij pri delu z ontologijami v okviru SEKT[28] primerov uporabe. ***OntoClean*** pristop (Welty and Guarino 2001) temelji na filozofskih pojmih za formalno evaluacijo taksonomije ontologije. Vrednotenje temelji na štirih osnovnih ontoloških pojmih: togost, enotnost, istovetnost in odvisnost. Pristop OntoClean se v praksi ne uporablja tako pogosto predvsem zaradi visoke cene uvajanja, saj morajo biti pred evaluacijo entitete ustrezno označene. Zaradi tega razloga avtorji v (Völker, et al. 2005) predstavijo pristop **AEON** za samodejno vrednotenje ontologij. Bistvo pristopa je, da temelji na OntoClean pristopu in skrbi predvsem za predpripravo ontologije tako, da jo samodejno opiše z meta podatki in tako tudi zmanjša efekt subjektivnosti pri ocenjevanju.

[26] **Protégé** je odprtokodni urejevalnik ontologij in ogrodje za razvoj aplikacij, ki temeljijo na znanju (glej http://protege.stanford.edu).

[27] **Testiranje enot** (ang. unit testing) je metoda, ki se uporablja pri razvoju programske opreme z namenom pridobivanja zaupanja za posamezne enote programske kode tako, da so primerne za uporabo.

[28] Evropski projekt **SEKT** (ang. Semantically Enabled Knowledge Technologies) se je v osnovi ukvarjal z ustvarjanjem sinergij pri združevanju naslednjih raziskovalnih področij: obvladovanje ontologij, strojno učenje in procesiranje naravnega jezika.

3 MODEL ZA HITER RAZVOJ ONTOLOGIJ

3.1 UVOD

Na podlagi pregleda obstoječih pristopov in metodologij so bile identificirane pomanjkljivosti, ki so podrobneje predstavljene v poglavju 2.4. Namen modela za hiter razvoj ontologij ROD je predvsem poenostavitev razvoja ontologij ter zmanjšanje potrebnega tehničnega znanja pri razvoju semantično obogatenih aplikacij in tako odprava omenjenih pomanjkljivosti obstoječih pristopov. Na ta način želimo neposredno omogočiti razvoj ontologij tudi poslovnim uporabnikom in domenskim ekspertom. Slika 28 prikazuje različna vidika, ki jih proces razvoja ontologij ROD obravnava.

Slika 28: Različna vidika na proces razvoja ontologij ROD

Vidik ponovne uporabe temelji na treh enostavnih korakih, ki jih prikazuje Slika 29. Na tem mestu se pospešuje mantra semantičnega spleta ter govori o razvoju takšnih komponent, ki se lahko ponovno uporabijo.

Poslovni uporabnik najprej začne z zajemom konceptov, medsebojnih relacij in omejitev nad koncepti in relacijami. To opravilo lahko vključuje tudi ponovno uporabo gradnikov iz različnih virov, lahko pa začnemo z gradnjo ontologije od začetka. Ko je shematski model ontologije končan, ga je v naslednjem koraku potrebno povezati z dejanskimi primerki gradnikov tega poslovnega slovarja. To vključuje uporabo podatkov iz podatkovne baze, tekstovnih datotek, ostalih ontologij itd. Zadnji korak pri procesu razvoja ontologije je uporaba ontologije s kreiranjem funkcionalne komponente za uporabo v drugih sistemih.

Procesni vidik temelji na podrobnih korakih, ki jih mora uporabnik izvesti, pri tem pa mu pomaga indikator popolnosti ontologije OC (poglavje 2.5). OC temelji na obstoječih metodah za evaluacijo ontologij (poglavje 3.3), vendar te metode pri pristopu ROD ne apliciramo na koncu razvoja, ampak se posamezne komponente dinamično izvajajo v celotnem procesu razvoja. Na ta način omogočimo nenehno preverjanje popolnosti ontologije in vodimo uporabnika skozi korake razvojnega cikla.

Slika 29: Poenostavljen pogled na model za hiter razvoj aplikacij

3.2 PROCES RAZVOJA

Proces razvoja ontologij po principu modela hitrega razvoja ontologij ROD je sestavljen iz 3 faz: **predrazvoj**, **razvoj** in **porazvoj**, kot to prikazuje Slika 30.

Rezultat vsake faze je določen produkt, s katerim stremimo h končnemu cilju – funkcionalna komponenta, ki temelji na ontologiji in jo lahko uporabljamo v različnih sistemih in scenarijih. V fazi predrazvoja je rezultat študija izvedljivosti, ki se nato uporablja v poznejših fazah razvoja opredelitve osnovnega modela in modela implementacije. Model implementacije vsebuje shemo ontologije, ki je povezana z dejanskimi primerki. Do tega modela pridemo v zadnji fazi, v kateri kreiramo funkcionalne komponente za uporabo v ostalih sistemih.

Slika 30: Proces hitrega razvoja ontologij

3.2.1 *PREDRAZVOJ*

3.2.1.1 Študija izvedljivosti

Prvi korak v fazi predrazvoja se ukvarja s **študijo izvedljivosti** problemske domene (korak 1). Aktivnosti vključujejo izvedbo študije izvedljivosti, v kateri je eden najpomembnejših ciljev opredelitev obsega projekta z jasno določenimi mejami. Zahteva se popoln opis problema z zahtevami in cilji. Pri opisu projekta se uporablja naravni jezik, prav tako pa moramo pripraviti seznam vseh virov, ki vsebujejo pomembne informacije o projektu, da jih bomo lahko kasneje vključili v proces razvoja. Ti viri vključujejo relacijske in analitične podatkovne baze, tekstovne datoteke, dokument Office in PDF, spletne strani in reference na ostale ontologije.

3.2.2 *RAZVOJ*

Naslednja faza je razvoj, katerega osnovni cilj je **opredelitev osnovnega modela** (korak 2). Faza razvoja je razdeljena na številne aktivnosti: zajem poslovnega slovarja (korak 2.1), seznam primerov konceptov in lastnosti (korak 2.2), identifikacija taksonomije (korak 2.3), identifikacija ad hoc relacij (korak 2.4), opis atributov in relacij konceptov (korak 2.5) in dodajanje kompleksnih omejitev in pravil (korak 2.6).

Zelo pomemben element omenjene faze je nenehna evaluacija razvite ontologije s pomočjo **indikatorja popolnosti ontologije**, ki je podrobneje predstavljen v poglavju 3.3.

3.2.2.1 Zajem poslovnega slovarja

Prvi korak faze razvoja je zajem poslovnega slovarja, kjer začnemo z naštevanjem konceptov in ad hoc relacij v obliki miselnega vzorca. Uporabnik enostavno začne ustvarjati neformalno opisan miselni vzorec problemske domene. Ukvarjamo se s podrobnostmi, kot je ime koncepta, opis v naravnem jeziku in sinonimi[29].

Eden od pristopov je **kreiranje ontologije od začetka**, kjer uporabnik začne s praznim miselnim vzorcem. Glede na to, da vizija semantičnega spleta spodbuja ponovno uporabo podatkov in predvsem ontologij, gre pri drugem pristopu za **kreiranje ontologije iz obstoječih virov**. Na voljo imamo številne možnosti, od identifikacije konceptov in relacij iz opisa v naravnem jeziku, do uvoza iz podatkovne baze, sheme ontologije ipd. Kar se tiče ponovne uporabe gradnikov ontologije, imamo na voljo dve alternativi: **dinamični** in **statični uvoz**. Pri dinamičnem uvozu gradnik referenciramo z uporabo imenskega prostora[30], medtem ko pri statičnem uvozu zgolj izberemo želene gradnike, ki jih prekopiramo v razvijajočo se ontologijo.

V nadaljevanju bosta predstavljena dva pristopa, ki se uporabljata pri zajemu poslovnega slovarja, in sicer:

- pristop **od zgoraj navzdol**[31] v obliki neformaliziranih pristopov v opredelitvi problemske domene in
- pristop **od spodaj navzgor**[32] za semantično opisovanje virov.

V praksi se oba pogosto združita v pristop od sredine navzven[33], kar je podprto tudi pri hitrem razvoju ontologij ROD.

---

[29] **Sinonim**, sopomenka ali soznačnica je beseda, ki ima (skoraj) enak pomen kot kaka druga beseda.
[30] **Imenski prostor** (ang. namespace) se uporablja za enolično določanje elementov in atributov.
[31] Pristop od **zgoraj navzdol** (ang. top-down) je analiza oz. dekompozicija, v kateri obsežen problem razdelimo v obvladljive podprobleme. Vsak podproblem je v nadaljevanju podrobneje razdelan in obravnavan. Pristop pogosto prikazujemo analogno "črni škatli", saj na takšen način lažje obvladujemo probleme.
[32] Pristop od **spodaj navzgor** (ang. bottom-up) je sinteza, v kateri združujemo manjše probleme in tako gradimo večjo, kompleksnejšo rešitev. Najprej opredelimo vsako podrobnost, ob združevanju le-teh pa kasneje nastane končna rešitev.

3.2.2.1.1 Neformalizirani pristopi k opredelitvi problemske domene

Pri začetnih korakih razvoja ontologij se pogosto uporablja pristop od zgoraj navzdol, pri procesu ROD pa poleg tega predlagamo še neformaliziran pristop. Takšen način razvoja omenjajo v okviru izboljšav številni obstoječi pristopi (Davies, et al. 2006), saj se je izkazalo, da se v začetnih fazah ne obnesejo strogo formalizirani pristopi, ampak bolj neformalni. Predlagane tehnike za zajem ontologije so naslednje:

- **miselni vzorci**,
- **analiza besedil** in
- **vnos poslovnih pravil v uporabniku prijazni obliki**.

Obstoječe metodologije omenjajo kot možno izboljšavo pri gradnji uporabo miselnih vzorcev pri zgodnjih fazah načrtovanja, saj se izkaže, da stroga formalizacija na tem mestu ni primerna. Veliko raziskav s področja asociativnega razmišljanja (Chakrabarti, et al. 2000, Cole, et al. 2002, Ware 2004) in vizualizacije podatkov (Spence and Apperley 1982, Furnas 1986, Sarkar and Brown 1994) je prav tako potrdilo dejstvo, da različne tehnike, ki upoštevajo kontekst v veliki meri, poenostavijo obvladovanje podatkov.

Slika 31: Gradnja ontologije s pomočjo miselnih vzorcev

Pri zajemu podatkov v obliki miselnega vzorca dobimo strukturo podatkov v obliki grafa, ki je pojmovno zelo blizu ontologiji problemske domene. Na tem mestu gre za enostaven in neformalen opis, ki ga v kasnejših iteracijah izpopolnimo. Gradniki, ki jih pridobimo iz miselnega vzorca problemske domene, so razredi in lastnosti (podatkovne in objektne). Povezava med dvema točkama se vedno preslika v lastnost, medtem ko se točka po potrebi preslika v razred ali eno od lastnosti (podatkovna oz. objektna). Na tem mestu je tudi velik poudarek na opisovanju vseh točk grafa z opisi v naravnem jeziku.

Naslednja tehnika je identifikacija konceptov in relacij iz opisa v naravnem jeziku, ki se aplicira na opis projekta iz študije izvedljivosti in vse identificirane dodatne vire (tekstovne datoteke, dokumente Office in PDF, spletne strani ipd.). Temelje analize besedil najdemo na področjih analize naravnega jezika, umetne inteligence in strojnega učenja. Najenostavnejše metode temeljijo na analizi besed ter se nadgrajujejo s sinonimi, koncepti, hierarhijo konceptov, relacijami ter pravili.

---

[33] Pristop od **sredine navzven** (ang. middle-out) je hibridni pristop, ki združuje tako pristop od zgoraj navzdol kot od spodaj navzgor. Proces poteka tako, da najprej implementiramo del problema s pristopom od zgoraj navzdol in preostali del s pristopom od spodaj navzgor.

Slika 32: Gradnja ontologije s pomočjo analize besedila

Na semantičnem spletu so na voljo številne rešitve, storitve in pristopi, ki to olajšujejo. Takšne storitve preprosto imenujemo semantični API-ji, katerih osnovna naloga je na podlagi nestrukturiranega besedila, ki ga dobimo na vhodu, na izhodu vrniti entitete in njihove medsebojne povezave. Prvi takšen primer je ***Open Calais API***, ki so ga razvili pri Thomson Reuters in sprejme tekst ter vrne podatke o ljudeh, krajih in podjetjih, ki jih je našel v dokumentu. Naslednji primer je ***SemanticHacker API***[34], ki klasificira podatke v dokumentu v kategorije, imenovane semantični opisi. Če mu damo dokument, nam vrne entitete in teme, ki se v njem pojavljajo. Podobno torej kot pri Open Calais, le da vrne še tematsko hierarhijo. Zelo zanimivo rešitev ponuja tudi ***Dapper***, ki olajša luščenje strukturiranih podatkov iz nestrukturiranih strani HTML. V praksi se izkaže za zelo uporabnega, saj omogoča zelo enostaven način pretvorbe spletne strani v spletno storitev.

Slika 33: Gradnja ontologije s pomočjo enostavnih pravil

Eden od večjih problemov pri zajemu znanja v obliki ontologije so pravila in njihova manipulacija. Ta težava je še posebej prisotna pri večji količini pravil, saj postanejo težko obvladljiva, predvsem za tehnično neuke uporabnike.

[34] http://www.semantichacker.com/

Slika 34: Gradnja ontologije s pomočjo odločitvenega drevesa

Pri pristopu ROD zato uvajamo naslednje mehanizme za lažje obvladovanje pravil:

- **enostavna pravila** (Slika 33),
- **odločitvena drevesa** (Slika 34) in
- **odločitvene tabele** (Slika 35).

Dobavni rok		Tip ponudnika	Tip ponudnika
>	≤		
	7 dni	Ponudnik	Dober ponudnik
7 dni	21 dni	Ponudnik	Dober ponudnik
21 dni		Ponudnik	Slab ponudnik

Slika 35: Gradnja ontologije s pomočjo odločitvene tabele

Podrobnosti zajema znanja v omenjenih oblikah so predstavljene v razdelku 3.2.2.6, v katerem je poudarek na dodajanju kompleksnih omejitev in pravil. V tej fazi ni takšnega poudarka na formalnem opisu konceptov, ampak bolj na identifikaciji in opisu v naravnem jeziku.

3.2.2.1.2 Semantično opisovanje virov

Pri semantičnem opisovanju virov sledimo pristopu od spodaj navzgor, olajšati želimo načrtovanje z uporabo gradnikov, ki jih bomo kasneje tudi potrebovali.

V okviru konzorcija W3C deluje interesna skupina W3C SWEO Linking Open Data[35], ki želi razširiti obstoječi splet podatkov v odprti obliki s pomočjo jezika RDF in medsebojno povezati številne spletne vire (Slika 36 prikazuje vire, ki so medsebojno semantično povezani in vsebujejo kar 4,7 milijard trojic RDF ter 142 milijonov medsebojnih povezav). Te aktivnosti so zagotovo nujno potrebne pri večji sprejetosti semantičnega spleta. S pomočjo povezav RDF lahko zelo enostavno krmarimo med podatki v okviru enega podatkovnega vira ter povezanimi podatki v ostalih virih. Te povezave lahko izkoriščajo tudi spletni pajki, ki iščejo po semantičnem spletu. Prednost tako pridobljenih podatkov je predvsem v obliki rezultata, saj na takšen način dobimo strukturirane podatke, ki jih lahko neposredno uporabimo v drugih aplikacijah, in ne zgolj povezave na spletne strani HTML, kjer so podatki v nestrukturirani obliki.

[35] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/

Slika 36: Medsebojno povezani spletni viri, ki so nastali v okviru projekta W3C SWEO Linking Open Data (LOD)

Interne vire v organizaciji, ki bi jih prav tako radi uporabili pri razvoju semantično obogatenih informacij in še niso predstavljeni v strukturirani obliki, pa lahko s pomočjo vmesnikov ravno tako pripravimo v strukturirani obliki. Pri semantičnem opisovanju virov bi za vsakega od pogostih virov opredelili postopek, ki uporabniku omogoča enostavno vključitev v ontologijo. Rezultat procesa semantičnega opisovanja vira je ontologija vira, ki jo lahko poljubno uporabimo v semantično obogateni aplikaciji. Viri, ki so podprti v okviru razvoja ontologij ROD, so naslednji:

- shema podatkovne baze,
- podatki CSV,
- spletni viri (WS, RSS, HTML ipd.),
- strukturirana poročila in
- obstoječe ontologije.

Pri podatkih, shranjenih v podatkovnih bazah, lahko potrebne podatke črpamo kar neposredno iz logičnega modela, pogosteje pa kar iz fizičnega modela v obliki tabel, relacij in omejitev. Pri omenjenem postopku (Slika 37) najprej izberemo tabele, ki jih potrebujemo, in nadaljujemo z izbiro ustreznih atributov, medsebojnih relacij in kasneje po potrebi te elemente še dodatno opišemo (npr. omejitve (referenčne integritete) nad tabelami in nekatera imena atributov, ki nimajo uporabniku prijaznih imen, zato je na tem mestu potrebno dodati opise v naravnem jeziku).

Slika 37: Pretvorba sheme podatkovne baze v ontologijo

Zelo veliko podatkov je mogoče najti v obliki CSV[36], ki je najpreprostejši zapis podatkov. Podpirajo ga tudi številne aplikacije ter na ta način omogočajo uvoz in izvoz podatkov. Takšen način je v praksi izjemno priljubljen, saj se veliko podatkov nahaja v tabelarični obliki, ki jih enostavno izvozimo v obliko CSV. Podatki, ki jih na ta način dobimo, se nahajajo v 1. NO[37], za uporabo v praktičnih aplikacijah in pri integraciji z ostalimi viri pa se zahteva vsaj 3. NO. Pri tej pretvorbi izhajamo iz relacij, ki nimajo ponavljajočih se skupin podatkov. Doseči želimo, da je vsak atribut, ki funkcionalno ne določa koncepta, netranzitivno odvisen od vsakega atributa, ki funkcionalno določa koncept.

Slika 38: Pretvorba podatkov CSV v ontologijo

Proces pretvorbe podatkov CSV v ontologijo se najprej začne z identifikacijo imen atributov, sledi samodejno prepoznavanje podatkovnih tipov in nato samodejna normalizacija (t. j. iskanje funkcionalnih odvisnosti med atributi). Od uporabnika se pri samem procesu zahteva potrjevanje določenih hipotez (npr. ker nimamo na voljo podatkov za popolno pokritje določenega primera, lahko zgolj z neko verjetnostjo trdimo o obstoju določenih funkcionalnih odvisnosti) in dodatno opisovanje elementov.

[36] **Datoteka z zaporedno ločenimi podatki** (ang. Comma Seperated Values), ki ponavadi vsebuje zapise v 1. NO (normalni obliki).
[37] **Normalna oblika** (ang. normal form) določa kriterij za določanje stopnje ranljivosti podatkov v tabeli glede na logične nedoslednosti in anomalije. Višja kot je normalna oblika, manj so podatki izpostavljeni nedoslednostim.

Mantra semantičnega spleta temelji na ponovni uporabi podatkov, zato pri pristopu ROD močno podpiramo ponovno uporabo že razvitih ontologij in pripravo vmesnikov za dostop do ostalih virov, kot je bilo prikazano v začetnem delu tega podpoglavja.

Slika 39: Ponovna uporaba obstoječih ontologij

Kjer je le mogoče, se pri izgradnji semantično obogatenih aplikacij spodbuja ponovna uporaba (Slika 39) bodisi z dinamičnim ali statičnim uvozom. Podprto je iskanje po ontologijah semantičnega spleta (npr. iskalnik Swoogle[38]), uporaba podatkov iz virov, ki so povezani v okviru projekta LOD, ter uvoz podatkov iz splošnih ontologij, kot je Cyc[39], OpenGALEN[40], WordNet[41] idr.

3.2.2.1.3 Problemi pri združevanju ontologij

Pri združevanju različnih ontologij virov in globalnega pogleda na problemsko domeno se pojavijo številne težave in neskladja. Uporabnik ima za pogoste napake na voljo način razrešitve s pomočjo indikatorja popolnosti ontologije OC (podrobneje predstavljen v poglavju 3.3) in drugih pristopov za lažjo integracijo ontologij. Na tem mestu bomo zato zgolj omenili pogoste težave, ki nastanejo pri integraciji:

- **Strukturne težave**, kjer se pri istih entitetah uporabljajo različne strukture.
    - *Različna imena* pri istih entitetah (npr. za isti koncept se v eni ontologiji uporabi ime kategorija, v drugi pa skupina).
    - *Različni podatkovni tipi* lastnosti (npr. v enem od virov je lastnost tipa float, v drugem pa integer).
    - *Kršitev integritete* (npr. različne privzete vrednosti ali primarni ključi pri več različnih ontologijah).

[38] http://swoogle.umbc.edu
[39] **Cyc** je največja baza splošnega znanja ter stroj za sklepanje, tako da jo lahko neposredno uporabimo v aplikaciji. Več o ontologiji Cyc lahko najdete na http://www.cyc.com.
[40] **OpenGALEN** je zelo obsežna ontologija s področja medicine in je primerna za izobraževanje v omenjeni in povezanih problemskih domenah. Več o OpenGALEN lahko najdete na http://www.opengalen.org.
[41] **WordNet** je zelo obširna baza znanja v angleškem jeziku, ki je slovarskega tipa. Več o WordNet lahko najdete na http://wordnet.princeton.edu.

  - *Neskladnost atributov* (npr. pri konceptu v eni ontologiji se zapišejo podatki v okviru koncepta naslov, spet v drugi ontologiji pa v okviru posameznih lastnosti ulica, poštna številka in kraj).
  - *Neskladnost konceptov* (npr. v eni ontologiji so potrebni podatki zajeti v okviru enega koncepta, spet v drugi ontologiji pa so isti podatki razpršeni v obliki več konceptov).
  - *Manjkajoči podatki.*
  - *Neusklajenost na meta ravni lastnosti in koncepta* (npr. v eni ontologiji uporabimo lastnost naslov, v drugi pa koncept lastnost).
  - *Neusklajenost na meta ravni primerka in koncepta* (npr. ena ontologija določen podatek zajame kot primerek določenega koncepta, druga pa kot sam koncept).
  - *Neusklajenost na meta ravni podatka in lastnosti.*
- **Semantične težave** se pojavijo zaradi interpretacije različnih struktur.
  - **Semantična raznovrstnost podatkov** v obliki različnih formatov identičnih informacij.
    - *Konflikti glede skale in valute* (npr. $ 100 v enem primeru in 100 € v drugem).
    - *Konflikti predstavitve* (npr. šolske ocene v eni ontologiji so {1, 2, …, 5}, v drugi pa {A, B, …, E}).
    - *Konflikti preslikovanja* (npr. šolske ocene v eni ontologiji so na lestvici {1, 2, …, 10}, v drugi pa {1, 2, …, 5}).
  - **Semantična raznovrstnost domen** oz. različna konceptualizacija.
    - *Konflikti vključevanja* (npr. letna prodaja je del skupne prodaje)
    - *Konflikti prekrivanja* (npr. evropski poslovni partnerji v eni ontologiji imajo določen presek s ponudniki računalnikov v drugi ontologiji).
    - *Neskladnost* (npr. v eni ontologiji je zajet pojem hotel, v drugi pa apartma).
    - *Konflikti agregacij* (npr. konzorcij v eni ontologiji in podjetje v drugi).
- **Nekonsistentnost** in **podvojenost**
  - *Natančnost podatkov* (npr. vrednost 0,9999 v enem viru in 1 v drugem).
  - *Začasni konflikti* (npr. v določenem trenutku eden od virov vsebuje več podatkov kot drugi).
  - *Podvojenost* (npr. isti delni podatki pri različnih virih, ki se razlikujejo v posameznih atributih).
  - *Nepopolni podatki* (npr. manjkajoč podatek).

3.2.2.2 Seznam primerkov konceptov in lastnosti

Drugi korak faze razvoja je identifikacija seznama primerkov konceptov in lastnosti. V prejšnjem koraku smo opredelili enostavno shemo problemske domene, medtem ko mora uporabnik v tem koraku podati dejanske primere konceptov in relacij (primerke razredov in lastnosti). Vsak koncept in tudi relacija morata imeti opredeljene primere iz realnega sveta (npr. primerki delnic pri trgovanju s finančnimi inštrumenti bi lahko bili AAPL,

GOOG, MSFT itd.). Rezultati tega koraka so zelo pomembni v naslednjih korakih 2.3 in 2.4, kjer se iz dejanskih primerov gradi taksonomija konceptov in relacij.

3.2.2.3 Identifikacija taksonomije

Tretji korak faze razvoja je identifikacija taksonomije glede na dejanske primerke iz prejšnjega koraka. Algoritem za identifikacijo taksonomije (Algoritem 25, Algoritem 26 in Algoritem 27) poskuša najti podrazrede, ekvivalentne razrede in razdružene razrede. Pregledajo se skupni primerki in glede na ugotovitve se uporabniku zastavijo enostavna vsebinska vprašanja (Slika 143). Primer uspešne identifikacije je lahko naslednji: na podlagi dejanskih primerkov ugotovimo, da imajo koncepti *Finančni inštrument*, *Delnica* in *ETF* skupne primerke. Glede na ugotovitve se postavijo enostavna vsebinska vprašanja in se potrdijo podrazred relacije *Finančni inštrument* in *Delnica* ter *Finančni inštrument* in *ETF* (Slika 107).

3.2.2.4 Identifikacija ad hoc relacij

Korak faze razvoja identifikacija ad hoc relacij je analogen prejšnjemu, a s to razliko, da je v tem koraku pod drobnogledom identifikacija podlastnosti in ekvivalentnih lastnosti. Pregledajo se skupni primerki (Algoritem 28 in Algoritem 29) in glede na ugotovitve se uporabniku zastavijo enostavna vsebinska vprašanja.

3.2.2.5 Opis atributov in relacij konceptov

Korak opisa atributov in relacij konceptov je eden najpomembnejših. Do tega trenutku imamo opredeljeno enostavno ontologijo, zato mora uporabnik dodatno opisati koncepte z lastnostmi in omogočiti sprehod med posameznimi koncepti s pomočjo medsebojnih relacij. Pri opredelitvi vsake relacije se moramo vprašati, ali gre za **enostavno** ali **kompleksno relacijo**. Pri odločitvi za tip relacije moramo odgovoriti na vprašanje, ali relacija koncept povezuje z atomarnim podatkom (npr. leto, opis, število elementov ipd.) ali je sestavljena (npr. trgovalni dan), kar vodi do nadaljnje dekompozicije.

V primeru enostavne relacije (Slika 137) najprej določimo podatkovni tip z možnostjo vnosa nekaj primerov testnih podatkov in sprožimo samodejno identifikacijo ali pa sami izberemo podatkovni tip. V tem koraku je možno opredeliti tudi definicijsko območje.

Pri kompleksnih relacijah (Slika 138) imamo na voljo dve možnosti: povezovanje relacije z obstoječim konceptom ali kreiranje novega koncepta, ki bo del zaloge vrednosti kompleksne relacije. Za vsako kompleksno relacijo moramo opredeliti tudi inverzno relacijo. Privzeta kardinalnost vseh relacij je nastavljena na neobvezno v obe smeri. Uporabniku se priporoča nastavitev obveznosti relacije (ali je funkcionalna ali ne) in nastavitev kardinalnosti[42] v obliki (najmanjša kardinalnost, največja kardinalnost).

Podrobnosti posameznih korakov lahko najdete v razdelku 4.1.1.1.1 Sprehod po semantični mreži in dodajanje omejitev.

[42] **Kardinalnost** (ang. cardinality) omejuje število dejanskih primerkov povezav, ki so povezani s primerki razreda.

3.2.2.6 Dodajanje kompleksnih omejitev in pravil

Zadnji korak pri fazi razvoja je dodajanje kompleksnih omejitev in pravil. Dodajamo lahko dva tipa pravil: **strukturna pravila** oz. **omejitve** in **operativna pravila**.

Pri strukturnih pravilih lahko uporabnik opredeli stvari, ki veljajo (npr. z delnico trgujemo na številnih borzah) in so resnične po definiciji.

Pri operativnih pravilih pa opredeljujemo, kaj npr. organizacija mora narediti (npr. delnico kupimo, ko se pojavi japonski svečnik kladivo) in jih ne moremo kršiti. Operativna pravila največkrat vnašamo v KAJ-ČE obliki, v primeru večkratne odvisnosti med pogoji pa so pogosto bolj primerni pogledi odločitvene tabele in odločitvenega drevesa. Odločitveno drevo je še posebej uporabno kot grafična predstavitev toka odločitev. Odločitvena tabela pa je primerna predvsem takrat, ko moramo opisati več pogojev v strnjeni obliki.

Konstrukcija pravila se vedno začne z akcijo, ki predstavlja relacijo v ontologiji (npr. delnica kotira na borzi). Ko opredelimo kvalifikator potrebnosti ali zadostnosti (npr. obvezno velja, da delnica kotira na borzi), se dodajo še kolikostni podatki in omejitve (npr. obvezno velja, da vsaka delnica kotira na vsaj 1 in največ 4 borzah).

3.2.2.6.1 Načini zajema pravil

Vsako atomarno pravilo na semantičnem spletu je sestavljeno iz enega ali več **pogojev** in ene ali več **posledic**. **Atom pravila** je opredeljen kot atomarni pogoj ali atomarna posledica. Vsak atom pravila zajamemo s pomočjo obrazca, ki se razlikuje glede na tip atoma.

Slika 40: Primer obrazca za zajem atoma

Obrazec atoma sestavljajo argumenti in povezovalno besedilo, ki smiselno povezuje argumente izbranega atoma. Vezni tekst je lahko vnaprej opredeljen, lahko pa se tudi dinamično pridobi iz sheme ontologije (npr. pri entitetah se namesto imena prikaže opis v naravnem jeziku, ki ga morajo uporabniki opredeliti že pri koraku 3.2.2.1, v katerem se zajame poslovni slovar). V obrazcu lahko uporabnik spreminja samo vrednosti argumentov, ki so lahko primerki razredov, spremenljivke ali dejanske vrednosti. Primerke razredov in obstoječe spremenljivke izbiramo iz seznama, v katerem se upoštevajo omejitve ontologije (npr. domena, zaloga vrednosti, kardinalnost ipd.). S pomočjo omenjenih obrazcev uporabniku omogočimo vnašanje naprednih omejitev ontologije v obliki blizu naravnega jezika. Med posameznimi atomi v telesu in glavi pravila je privzet operator IN.

Zajem pravil v obliki $Telo \rightarrow Glava$ je najbolj prilagodljiv, vendar pri zajemu omejitev s strani poslovnih uporabnikov ne vedno najbolj primeren. Zato pristop poleg zajema pravil v kaj-če obliki predvideva tudi zajem v obliki odločitvene tabele in odločitvenih dreves. Na takšen način lahko v določenih primerih močno poenostavimo in pohitrimo zajem omejitev v ontologiji.

3.2.2.6.1.1 Kaj-če pravila

Osnovni način zajema pravil je v enostavni kaj-če obliki. Če del pravila imenujemo tudi telo pravila, medtem ko je kaj del glava pravila.

ČE velja
(Pogoj 1) IN **(Pogoj 2)** IN **…** **(Pogoj m)**
POTEM sledi
(Posledica 1) IN **(Posledica 2)** IN … **(Posledica n)**

Slika 41: Zajem pravila v kaj-če obliki

Slika 42 prikazuje primer pravila iz ontologije FITS, v kateri opredeljujemo prodajni signal na podlagi »engulfing« trgovalnega vzorca.

ČE velja
dan1 | ima naslednji trgovalni dan | dan2 IN
dan1 je belo telo svečnika IN
dan2 je črno telo svečnika IN
dan2 je veliko telo svečnika IN
dan2 je engulfing IN
dan2 je danes najvišja vrednost IN

POTEM sledi
dan2 je prodajni trgovalni dan
razlog za izvedbo akcije od dan2 je Prisotnost engulfing trgovalnega vzorca.

Slika 42: Primer pravila v kaj-če obliki

3.2.2.6.1.2 Odločitvena tabela

Odločitvena tabela ima v začetnih stolpcih najprej opredeljene pogoje, nadaljnji stolpci pa predstavljajo posledice. Vrstice v tabeli predstavljajo kombinacije pogojev, katerim sledi kombinacija posledic, ki velja pri izbrani kombinaciji pogojev.

Če imamo m pogojev, potem je največje število stolpcev v tabeli enako 2^m, ki predstavlja vse možne kombinacije m pogojev, kjer lahko opredelimo poljubne posledice.

Takšna tabela združuje več kaj-če pravil, kjer vsaka vrstica predstavlja novo kaj-če pravilo. Zajem pravil na takšen način je uporaben predvsem takrat, ko želimo zajeti več pravil hkrati, pri tem pa je unija pogojev v telesu pravila razmeroma majhna, obstaja pa večji presek med omenjenimi pogoji. Pomanjkljivost tega pristopa se pokaže pri zajemu pravil z večjim številom različnih pogojev v telesu, saj potem obvladovanje postane nepregledno, ker število permutacij in s tem stolpcev v tabel eksponentno narašča. Po drugi strani pa število pogojev v telesu na zajem pravil ne vpliva tako negativno, saj število vrstic raste linearno.

Pogoj 1	✓		✓	...
Pogoj 2	✓		✓	...
...				...
Pogoj m		✓	✓	...
Posledica 1		✓		...
Posledica 2	✓		✓	...
...				...
Posledica n		✓	✓	...

Tabela 2: Zajem pravila v obliki odločitvene tabele

Takšno tabelo pravil lahko zapišemo tudi v kaj-če obliki, kjer je vsak stolpec posamezno pravilo. Zaradi monotonosti jezikov semantičnega spleta moramo biti previdni pri vsebinski opredelitvi posameznih pravil. Če želimo zapisati npr. pravilo v 1. stolpcu (Tabela 2), potem dobimo naslednje pravilo

$$Pogoj\ 1\ \wedge Pogoj\ 2\ \rightarrow Posledica\ 2$$

in ne recimo

$$Pogoj\ 1\ \wedge Pogoj\ 2\ \wedge \ldots \wedge \neg\, Pogoj\ m \\ \rightarrow \neg\, Posledica\ 1\ \wedge\ Posledica\ 2\ \wedge \ldots \wedge\ \neg\, Posledica\ n$$

Slika 43 prikazuje primer japonske trgovalne strategije in opredelitev razlogov za izvedbo akcije ter nakupnih in prodajnih signalov.

dan1 ima naslednji trgovalni dan dan2	✓	✓	✓	✓	✓	✓
dan1 je belo telo svečnika	✓	✓				
dan1 je črno telo svečnika					✓	
dan1 je doji			✓	✓		✓
dan2 je belo telo svečnika					✓	✓
dan2 je črno telo svečnika	✓	✓	✓	✓		
dan2 je veliko telo svečnika	✓	✓	✓	✓	✓	✓
dan2 je engulfing	✓	✓	✓	✓	✓	✓
dan2 je danes najvišja vrednost	✓		✓			
dan2 je včeraj najvišja vrednost		✓		✓		
dan2 je danes najnižja vrednost					✓	✓
dan2 je nakupni trgovalni dan					✓	✓
dan2 je prodajni trgovalni dan	✓	✓	✓	✓		
razlog za izvedbo akcije od dan2 je Prisotnost engulfing trgovalnega vzorca.	✓	✓	✓	✓	✓	✓

Slika 43: Primer skupine pravil v obliki odločitvene tabele

Oznaka v stolpcu označuje prisotnost pogoja oz. posledice. Prvi stolpec lahko zapišemo kot

$$Pogoj\ 1\ \wedge Pogoj\ 2 \wedge Pogoj\ 6 \wedge Pogoj\ 7 \wedge Pogoj\ 8 \wedge Pogoj\ 9 \\ \rightarrow Posledica\ 2\ \wedge\ Posledica\ 3$$

in interpretiramo: *»Če ima dan1 naslednji trgovalni dan dan2, kjer ima dan1 belo telo svečnika, dan2 pa ima črno in veliko telo svečnika ter ima danes najvišjo vrednost, potem je dan2 prodajni trgovalni dan in razlog za izvedbo akcije na dan2 je prisotnost »engulfing« trgovalnega vzorca«.*

3.2.2.6.1.3 Zajem z odločitvenim drevesom

Zajem pravil v obliki odločitvenega drevesa je uporaben takrat, ko se telesi (če del) več pravil razlikujejo v zgolj nekaj pogojih, katerih pravilnost pa se odloča v glavi (kaj del). Zajem pravil tako poteka v obliki binarnega drevesa (Slika 44).

Slika 44: Zajem pravila v obliki odločitvenega drevesa

Pri gradnji drevesa vedno začnemo z dejstvi, ki so skupna vsem pravilom, nato pa na podlagi dodatnih pogojev vejimo drevo v poddrevesa. Končni listi v drevesih so posledice, ki nastopajo v glavi pravil. Pravila v odločitvenem drevesu lahko predstavimo tudi v enostavni kaj-če obliki, kjer za vsak list drevesa (skupino posledic) zapišemo pravilo.

Slika 45 prikazuje primer odločitvenega drevesa za opredelitev barve telesa svečnika, ki je lahko bele ali črne barve. Med dejstvi so opredeljeni osnovni podatki, s katerimi operiramo, kot je trgovalni dan ter cena ob odprtju in zaprtju. Pogoj je v tem primeru zgolj eden, in sicer razmerje večje ali enako med ceno zaprtja in ceno odprtja. Posledici omenjenega pravila pa sta klasifikacija trgovalnega dne v dan z belim ali črnim telesom svečnika.

Slika 45: Primer pravila v obliki odločitvenega drevesa

3.2.3 *PORAZVOJ*

Zadnja faza je porazvoj, katere je osnovni cilj je **opredeliti model implementacije** (korak 3). Faza porazvoja je razdeljena na dve pomembnejši aktivnosti: povezava slovarja s podatki (korak 3.1) in kompozicija funkcionalne komponente (korak 3.2). Nadalje je korak 3.2 sestavljen iz naslednjih podkorakov: opredelitev V/I (korak 3.2.1), izbira konceptov, relacij, omejitev in pravil (korak 3.2.2) in identifikacije cilja (korak 3.2.3).

S stališča evaluacije je faza porazvoja sorodna fazi razvoja, saj se ravno tako izvaja nenehna evaluacija s pomočjo indikatorja popolnosti (poglavje 3.3).

3.2.3.1 Povezava slovarja s podatki

Prvi korak porazvoja je povezava slovarja s podatki. Repozitorije, ki smo jih identificirali v študiji izvedljivosti v 1. koraku, smo najprej uporabili za generiranje sheme v 2. koraku, zdaj pa jih lahko zopet uporabimo za povezavo sheme ontologije z dejanskimi primerki podatkov. V jeziku tehnologij semantičnega spleta smo do tega trenutka opredelili terminološko komponento TBox (izjave o slovarju), medtem ko je potrebno opredeliti tudi izjave, ki temeljijo na slovarju, t. i. komponento ABox. Uporabnik lahko izbira med vrsto vgrajenih povezovalnikov za uvažanje primerkov iz:

- **obstoječih ontologij**,
- **relacijskih** in **analitičnih podatkovnih baz**,
- **datotek CSV** in
- **delno nestrukturiranih virov**.

Pri povezovanju s primerki obstoječih ontologij je pomembna dobra povezanost sheme, predvsem taksonomije razredov in lastnosti. Ena najpomembnejših načrtovalskih konstruktov jezikov semantičnega spleta je inverzna funkcijska lastnost[43], ki je zelo uporabna pri integraciji podatkov iz različnih virov. Omenjena lastnost ima podobno vlogo kot identifikator pri relacijskih podatkovnih bazah. Za razliko od relacijskih podatkovnih baz pri relacijah, ki imata isti identifikator, ne dobimo napake, ampak mehanizmi semantičnega spleta sklepajo, da gre za isti entiteti.

[43] owl:InverseFunctionalProperty

Slika 46: Integracija podatkov iz obstoječih ontologij

Slika 46 prikazuje integracijo primerkov iz dveh različnih ontologij, kjer s pomočjo inverzne funkcijske in ekvivalentne lastnosti zagotovimo samodejno integracijo. Če imamo več različnih virov, ki imajo skupni enolični identifikator (npr. simbol finančnega inštrumenta) in različne atribute ter imena, potem nam semantični splet omogoča izvedbo integracije na zelo enostaven način. Če na primer zahtevamo podatke o delnici GRVG, bomo kot rezultat dobili integrirane podatke obeh virov (iz enega vira npr. črpamo trgovalne podatke, medtem ko iz drugega podatke o poslovanju). Pri analogni integraciji v podatkovni bazi bi na podoben način prišli do napake, saj sistem za upravljanje s podatkovno bazo ne dovoljuje več relacij z istim identifikatorjem.

Slika 47: Integracija podatkov iz relacijske podatkovne baze

Pri integraciji ontologij po pristopu ROD se za opredelitev taksonomije priporočajo aksiomi podrazred, ekvivalentni razred, podlastnost in ekvivalentna lastnost, medtem ko se pri primerih uporabe, kjer je poudarek na vsebinski integraciji, priporoča uporaba inverznih funkcijskih lastnosti, kar vodi do ekvivalence primerkov.

Slika 48: Integracija primerkov iz datoteke CSV

Zelo veliko podatkov je na voljo v relacijskih in analitičnih podatkovnih bazah, ki jih lahko pri procesu ROD neposredno uporabimo pri gradnji semantično obogatenih aplikacij. Uporabnik izhaja iz dela sheme obstoječe ontologije, ki jo želimo povezati z dejanskimi podatki iz realnega sveta (t. i. primerki razredov in lastnosti). Na drugi strani pa imamo dejanske podatke iz obstoječih aplikacij shranjene v relacijski obliki, tako da je potrebno zgolj določiti, katere tabele in podrobneje, katere atribute želimo, da se povežejo z razvijajočo se ontologijo. Integracijo tako izvedemo na ravni sheme ontologije in relacijske sheme, relacije iz relacijske baze pa se prenašajo v primerke v ontologiji. Slika 47 prikazuje takšno integracijo na področju finančnega trgovanja, kjer imamo prometne podatke o finančnih inštrumentih in tudi trgovalne podatke. Integracijo izpeljemo tako, da lastnosti (podatkovne in objektne) povežemo z atributi in relacijske tabele z razredi. V primeru naprednejših preslikav imamo za opredelitev na voljo tudi booleanovo algebro.

Za povezovanje z delno strukturiranimi viri je na voljo povezovalni čarovnik. S pomočjo čarovnika in uporabe regularnih izrazov[44] ima uporabnik skoraj neomejene možnosti pri ponovni uporabi nestrukturiranih podatkov iz tekstovnih datotek, dokumentov Office in PDF, spletnih strani itd. Ker je poznavanje vzorcev regularnih izrazov tehnično zahtevno, ROD na tem mestu temelji na čarovniku, kjer uporabnik v grafičnem okolju opredeli omejitev in jo kasneje preslika v ontologijo.

Slika 49: Integracija primerkov iz delno strukturirane spletne strani

Zelo veliko podatkov je na voljo v enostavni tekstovni obliki CSV, saj takšen način izvoza podpira večina programov. S pomočjo regularnega izraza uporabnik opredeli vzorec podatkov, ki jih vir podatkov CSV vsebuje in nato kasneje te podatke v realnem času zapiše

[44] **Regularni izraz** (ang. regular expression) v računalništvu omogoča jedrnat in prilagodljiv način za identifikacijo nizov, kot so določeni znaki, besede ali vzorci znakov.

v ontologijo v obliki primerkov. Slika 48 prikazuje uvoz zgodovinskih prometnih podatkov za izbrane finančne inštrumente iz spletnega vira Yahoo Finance.

Zadnja od podprtih možnosti uvoza primerkov v ontologijo so delno strukturirani podatki na spletnih straneh in v datotekah. Slika 49 prikazuje uvoz podatkov iz spletnega portala Google Finance, in sicer podrobnosti o finančnih inštrumentih, kot sta opis in borza, kjer kotirajo. To je najbolj generičen dostop do poljubnega delno strukturiranega vira podatkov, saj v celoti temeljimo na regularnih izrazih, s pomočjo katerih opišemo vzorec, kjer se zahtevani podatek nahaja.

Ko je ta korak končan, dobimo delujočo različico ontologije, ki je pripravljena na povezovanje s strojem za sklepanje. S pomočjo mehanizma za sklepanje lahko izpeljemo nova dejstva o konceptih ali primerkih, ki se nahajajo v ontologiji. Vendar ontologija na tem mestu še ni popolnoma pripravljena za uporabo v inteligentnih ali ostalih aplikacijah, ki so močno odvisne od baze znanja.

3.2.3.2 Kompozicija funkcionalne komponente

Zaradi pomislekov, ki se nam porajajo v prejšnjem koraku, se v tem koraku ukvarjamo s kompozicijo funkcionalne komponente. Namen je priprava delujoče funkcionalne enote, ki temelji na izbranih gradnikih ontologije in je primerna za izbran primer uporabe.

3.2.3.2.1 Opredelitev V/I

Prvi korak pri kompoziciji funkcionalne komponente je opredelitev vhodnih in izhodnih podatkov. Uporabnik mora identificirati primer uporabe in določiti vhod in izhod. Vhod oz. izhod lahko vsebuje:

- množico primerkov konceptov in relacij,
- napredno (SparQL) poizvedbo ali
- prazno poizvedbo.

Uporabnik v grafičnem načinu izbere želen vhod in izhod, ki se kasneje preslika v poizvedbo SparQL in postane del meta podatkov funkcionalne komponente. Za napredne poizvedbe je naprednim uporabnikom na voljo tudi neposreden vnos poizvedbe.

V problemski domeni finančnega trgovanja (Slika 50) bi lahko vhodni podatek bila izbrana delnica, medtem ko na izhodu pričakujemo glede na trgovalno strategijo klasifikacijo teh trgovalnih dni, ki bi predstavljali nakupne ali prodajne signale.

Izbira **vhodno/izhodnih** parametrov

```
PREFIX FI: <http://amor.fri.uni-lj.si/dejan/SW/FI.owl>
INSERT DATA {
   ?FI FI:symbol ?simbol.
   ?FI rdf:type FI:FinancialInstrument
}
```

Vhodna poizvedba komponente

```
PREFIX FI: <http://amor.fri.uni-lj.si/dejan/SW/FI.owl>
SELECT ?simbol ?datum ?razlogZaAkcijo {
   { ?day rdf:type FI:BuyTradingDay } UNION { ?day rdf:type FI:SellTradingDay } .
   { ?day FI:date ?datum } .
   { ?day FI:tradeReason ?razlogZaAkcijo } .
   { ?day FI:validFor ?instrument } .
   { ?instrument FI:symbol ?simbol }
}
```

Izhodna poizvedba komponente

Slika 50: Primer opredelitve vhoda in izhoda funkcionalne komponente

3.2.3.2.2 Izbira konceptov, relacij, omejitev in pravil

Naslednji korak je izbira konceptov, relacij, omejitev in pravil, kjer uporabniki podrobneje opredelijo vhod in izhod. To opravilo vključuje opredelitev vseh dinamičnih uvozov, ki se izvedejo v času izvajanja, in vseh omejitev pogledov na izbranih gradnikih ontologije iz prejšnjega koraka (Slika 51).

Uporabniku pri izbiri potrebnih entitet in aksiomov pomaga algoritem (Algoritem 1), na voljo pa ima tudi možnost dodatnih omejitev po meri.

Ontologijo predstavimo kot neusmerjen graf $G = (V, E)$, kjer je V množica razredov, E pa množica relacij podrazred in objektne lastnosti. Pri iskanju povezanih entitet in aksiomov iščemo maksimalno povezan podgraf, ki vsebuje izbrane vhodne in izhodne parametre (poglavje 3.2.3.2.1).

Slika 51: Primer izbire konceptov, relacij, omejitev, pravil in dinamičnih uvozov

V problemski domeni finančnega trgovanja bi ta korak vključeval opredelitev regularnih izrazov za luščenje podatkov o delnicah iz izbranih spletnih strani in datotek CSV ter preslikave v gradnike ontologije. V primeru naprednih presejalnih pogledov se uporabijo tudi poizvedbe po ontologiji, s pomočjo katerih pridemo do točno izbranega dela ontologije.

predlagajEntiteteInAksiomeZaVkljucitev
V ← Točke grafa so entitete (razredi, podatkovne in objektne lastnosti)
E ← Povezave v grafu so relacije podrazred in povezava razredov ter lastnosti
A ← Aksiomi, ki jih sestavljajo atomi entitet in povezav
$V' = V_{vhod} \cup V_{izhod} \cup V(A)$ ← Točke grafa izbranih vhodnih in izhodnih entitet ter aksiomov
$E' = E_{vhod} \cup E_{izhod} \cup E(A)$ ← Izbrane vhodne in izhodne povezave v grafu ter aksiomov
$G' = (V', E')$
za vsako $v \in V'$
  $G' = G' \cup BFS'(G, v)$
konec sprehoda po vseh točkah grafa
vrni $\boldsymbol{G'}$
konec

BFS' (G, v)
izvedi iskanje v širino na grafu G, z začetkom v točki v
vrni obiskan pod-graf G''
konec

Algoritem 1: Predlagaj entitete in aksiome za vključitev v funkcionalno komponento

3.2.3.2.3 Identifikacija cilja

Zadnji korak je identifikacija cilja, kjer je omejitev okolje uporabe ontologije in se kreira funkcionalna komponenta za uporabo ob izbranih nastavitvah. Obstajajo številne možnosti za uporabo funkcionalne komponente (Slika 29) in jih lahko razdelimo v dve kategoriji:

- uporaba **sheme** (TBox, RBox in pravila)
  - podatkovna baza,
  - semantično preverjanje podatkov,
  - semantika komunikacijskega protokola inteligentnega agenta,
  - omejitev uporabniškega vmesnika ipd.
- uporaba **sheme skupaj s primerki** (TBox, RBox, ABox in pravila),
  - gradnik BPEL v obliki točke odločitve,
  - način delovanja inteligentnega agenta (JADE),
  - paket poslovnih pravil,
  - samostojna aplikacija (J2SE, J2EE, WS).

Pri gradnji ontologije izhajamo iz ponovne uporabe virov (npr. tekstovnih datotek, svetovnega spleta, podatkovnih baz ipd.). Pri procesu ROD je podrt tudi proces v nasprotno smer in sicer izvoz ontologije v eno od zgoraj omenjenih ciljnih okolij.

Pri izvozu ontologije v **shemo podatkovne baze**, se razredi izvozijo v relacijske tabele, lastnosti v relacije med tabelami ter aksiomi v integritete omejitve in prožilce v podatkovni bazi.

Komponenta **semantičnega preverjanja podatkov** se aplicira na strukturirane vire podatkov, ki zadoščajo sintaktičnim zahtevam in jih želimo preveriti tudi semantično. Primer takšnega vira je podatkovno skladišče, kjer želimo preveriti smiselnost združenih podatkov iz različnih virov. Pri tem se preverjajo omejitve in aksiomi v ontologiji z dejanskimi podatki v zvezdni shemi.

Inteligentni agent, kot je računalniški sistem, ki je sposoben prilagodljivega in avtonomnega delovanja v dinamičnem in nepredvidljivem okolju (Wooldridge 2002), postaja vedno bolj pomemben gradnik pri razvoju naprednih informacijskih rešitev. Za potrebe samodejnega in avtonomnega obvladovanja podatkov so tehnologije semantičnega spleta več kot primerne in tudi uporaba ontologij pri opredelitvi **semantike komunikacijskega protokola inteligentnega agenta**. Proces ROD zato podpira izvoz ontologije v obliki semantike komunikacijskega protokola platforme JADE (Bellifemine, et al. 2006) za podporo izvajanju inteligentnih agentov.

Uporaba ontologije pri **omejitvi uporabniškega vmesnika** se predlaga v primeru preverjanja uporabniškega vnosa v sistem. Semantična pravilnost vnosa se lahko preverja še pred persistenco in za ta namen se lahko uporabi shema ontologije z vsemi omejitvami in aksiomi.

Področje avtomatizacije poslovnih procesov postaja vedno pomembnejše predvsem zaradi vse večjega povezovanja (B2B, B2C, B2G ipd.), zato je uporaba ontologij na tem področju, predvsem za naprednejše obvladovanje takšne avtomatizacije, pristop, ki prinaša veliko prednosti. Za izvajanje poslovnih procesov je standardizacija na tem področju poskrbela, da je najbolj uporabljan jezik za izvajanje poslovnih procesov BPEL (Margolis and Sharpe 2007). Proces ROD zato omogoča izvoz funkcionalne komponente v obliki **gradnika BPEL**, ki ga lahko vključimo v poslovni proces in je primeren predvsem za vlogo točke odločitve (Slika 52). Številne odločitve v poslovnih procesih, ki so v domeni poslovnih uporabnikov, lahko formaliziramo v obliki ontologije in jih tako objavimo v obliki spletne storitve in uporabimo v procesu BPEL. Primerki lahko v funkcionalno komponento ontologije prihajajo neposredno iz procesa in ostalih virov in se aplicirajo na obstoječo shemo ter aksiome, ki na podlagi omenjenih primerkov privedejo do rezultata, ki odloči, po kateri poti se proces nadaljuje.

Slika 52: Primer izvoza funkcionalne komponente v obliki BPEL

Za namen uporabe sheme ontologije smo že v začetku poglavja omenili opredelitev semantike komunikacijskega protokola inteligentnega agenta. V primeru naprednejše

uporabe lahko v izvoz vključimo tudi primerke in na ta način podpremo **delovanje inteligentnega agenta**, kjer je podprta agentna platforma JADE.

Proces ROD temelji na pristopu obvladovanja poslovnih pravil, zato je posledično podprt tudi izvoz ontologije v obliki **paketa poslovnih pravil**. Glede na to, da ROD podpira poenostavljene načine vnosa poslovnih pravil (KAJ-ČE pravila, odločitveno drevo in odločitvena tabela), je sam proces neposredno primeren za zajem poslovnih pravil.

Eden od načinov za neposredno preverjanje funkcionalnosti je v obliki **samostojne aplikacije**, kjer nam sistem omogoča izvoz aplikacije v Java okolju (J2SE in J2EE), za širšo podporo pa je na voljo tudi objava v obliki spletne storitve.

3.3 INDIKATOR POPOLNOSTI ONTOLOGIJE

3.3.1 *UVOD*

Indikator popolnosti[45] OC je vpeljan kot pomoč uporabnikom in poenostavitev prehajanja med posameznimi koraki procesa ROD. Namen indikatorja popolnosti je dvojen:

1. **Uporaba neodvisno od procesa ROD** z ostalimi metodologijami za razvoj ontologij ali brez uporabe kakršnekoli metodologije.
    - Glede na opravljen semantičen pregled ontologije, se uporabniku predlagajo popravki, ki so predstavljeni v obliki množice akcij za izboljšanje ontologije in so urejeni glede na učinek[46], ki ga imajo. Poleg samih akcij in njihovih učinkov je na voljo tudi podroben opis akcij.
    - Primer akcije: Če odstranite anomalijo v komponenti RBox, potem boste pridobili 8 % pri popolnosti ontologije.
    - Primer opisa akcije: Koncepta 'Poslovalnica' in 'Oseba' sta povezana s ponavljajočo se skupino lastnosti 'ulica', 'hišna številka', 'poštna številka' in 'pošta'. Združite prej omenjene lastnosti v okvir novega koncepta in odstranite koncepta 'Poslovalnica' in 'Oseba' iz definicijskega območja teh lastnosti ter povežite novokreiran koncept s konceptoma 'Poslovalnica' in 'Oseba'.
2. Uporaba v obliki **orodja za pomoč** in **hitrejše ter lažje napredovanje med posameznimi koraki procesa ROD**.
    - Medtem ko se uporabnik nahaja pri določenem koraku procesa ROD, se indikator popolnosti OC prilagaja trenutnemu koraku s ponastavitvijo uteži (Tabela 3), ki se uporabljajo pri izračunu (npr. v koraku 2.1, v katerem poteka zajem poslovnega slovarja, ni potrebe po preverjanju odvečnih primerkov, obstoju neuporabljenih konceptov ali inverznih lastnosti,

[45] V literaturi najdemo več vrst **popolnosti** (ang. completeness) od logične do matematične ipd. Popolnost v okviru omenjene doktorske disertacije se nanaša na semantično popolnost, i. e. popolnost v pomenu pripravljenosti za vključitev v aplikacijo semantičnega spleta in da ne vsebuje napak ter načrtovalskih anomalij.

[46] **Učinek** je opredeljen kot delež izboljšave ontologije v primeru, da izvedemo korake za predlagano izboljšavo.

ampak je poudarek na opisu komponent TBox in RBox v naravnem jeziku in obstoju povezave med vsemi koncepti).

- Ko indikator popolnosti OC doseže določeno raven (npr. 80 %), lahko uporabnik nadaljuje z aktivnostmi naslednjega koraka v procesu. Posodobljena vrednost OC se izračuna sproti za vsako fazo posebej in se vedno, ko le-ta preseže določeno raven, sproži priporočilo za nadaljevanje v naslednji korak. Na ta način se uporabniku pomaga pri izvajanju aktivnosti procesa ROD, od zajema poslovnega slovarja pa do kreiranja funkcionalnih komponent za uporabo v ostalih sistemih.
- V primeru da ontologija že obstaja, lahko s pomočjo ocene OC umestimo razvito ontologijo v pripadajoč korak pri procesu ROD in začnemo z izpopolnjevanjem ontologije od predlaganega koraka naprej. Npr. ontologija že ima opredeljeno taksonomijo, zato lahko nadaljujemo kar s korakom 2.4, v katerem poteka identifikacija binarnih relacij).

Indikator popolnosti (OC) je opredeljen kot

$$OC = f(C, P, R, I) \in [0,1],$$

kjer je C množica razredov, P množica lastnosti, R množica pravil in I množica primerkov. Glede na te vhodne podatke se izračuna ocena OC, katere zaloga vrednosti je v intervalu [0,1]. Višja je vrednost, bolj popolna je ontologija. OC je obtežena vsota semantičnih testov, medtem ko se uteži dinamično spreminjajo pri prehajanju iz ene faze v drugo fazo procesa ROD (Tabela 3). OC lahko tako opredelimo kot

$$OC = \sum_{i=1}^{n} w_i' \cdot leafCondition_i,$$

kjer je n število vseh listov v drevesu kriterijev (Slika 53), *leafCondition* pa list v drevesu kriterijev, kjer se izvede semantičen test. Za relativne uteži in izračun semantičnih testov veljajo naslednje omejitve $\sum_i w_i' = 1$, $\forall w_i' \in [0,1]$ in $\forall leafCondition_i \in [0,1]$. Relativna utež w_i' označuje globalno pomembnost semantičnega testa $leafCondition_i$ in je odvisna od vseh uteži na poti od lista do korena v drevesu kriterijev. V primeru $leafCondition_i = ekvivlanten\ razred$ (Slika 53), potem je

$$w_i' = \prod_{j \in \left\{ \begin{matrix} ABoxPartHierClassEquiv, \\ ABoxPartHierClass, \\ ABoxPartHier, \\ ABoxPart \end{matrix} \right\}} k_j$$

ki predstavlja verigo vseh uteži od lista do korena.

Slika 53: Drevo pogojev indikatorja popolnosti OC s semantičnimi testi v listih in pripadajočimi utežmi

Drevo pogojev (Slika 53), ki se upošteva pri izračunu popolnosti ontologije, vsebuje semantične teste, ki se izvedejo pri preverjanju ontologije. Zgornja raven je razdeljena na komponento **TBox**, **RBox** in **ABox**. Naslednje ravni so naprej razdeljene glede na klasifikacijo napak v (Fahad and Quadir 2008) in so **opis**, **razdelitev**, **odvečnost**, **doslednost** in **anomalija**. Predlagano strukturo lahko enostavno dopolnimo in spremenimo za želeno uporabo. Listi drevesa, ki se upoštevajo pri izračunu ocene OC so implementirani kot semantični testi, medtem ko vsi ostali elementi predstavljajo

agregacijo z ustreznimi utežmi (Slika 53). Pri algoritmu (Algoritem 2) za izračun indikatorja popolnosti OC je X pogoj in $w = w(X, Y)$ utež med pogojema X in Y.

```
' Izračun se izvede na korenskem pogoju "komponente OC" z utežjo 1 (Slika 53)
izracunaj (X, w)
  ocena_OC = 0
  označi pogoj X kot obiskan
  če ne obstaja podpogoj od X, potem
    ' Izvedi semantični test na listu
    vrni w · izvedi(X)
  sicer za vse pogoje Y, ki so podpogoji od X, tako da Y še ni obiskan
    ' Agregiraj ocene popolnosti
    če w(X, Y) ≠ 0 potem
      ocena_OC = ocena_OC + izracunaj (Y, w(X, Y))
  vrni w · ocena_OC
konec
```

Algoritem 2: Izračun indikatorja popolnosti OC

Tabela 3 predstavlja porazdelitev uteži pogojev (Slika 53). V naslovnih celicah vrstic najdemo imena pogojev, ki so urejena v obliki drevesne strukture, medtem ko naslovne celice stolpcev predstavljajo posamezne korake v procesu ROD. Celice, ki predstavljajo presek vrstice in stolpca, vsebujejo dve vrednosti v obliki w_i (w_i'), kjer je w_i absolutna utež med pogojem očeta in pogojem sina, medtem ko je w_i' relativna utež med izbranim pogojem in korenskim pogojem. Iz tabele jasno vidimo, da je v nekaterih fazah razvoja poudarek na izbranih komponentah. V začetnih korakih se uporabnik ukvarja z opisovanjem in samo strukturo razredov, medtem ko proti koncu opredelitve osnovnega modela postanejo pomembne tudi omejitve in pravila. Kar pa se tiče porazvojnih aktivnosti in opredelitve modela implementacije, mora biti uporabnik pozoren na primerke shematskega dela ontologije, tako da je večji poudarek na komponenti ABox.

	(2.1) Zajem **poslovnega slovarja**	**(2.2)** Seznam **primerov konceptov** in **lastnosti**	**(2.3)** Identifikacija **taksonomije**	**(2.4)** Identifikacija **ad hoc relacij**	**(2.5)** Opis **atributov** in **relacij konceptov**	**(2.6)** Dodajanje **kompleksnih omejitev** in **pravil**	**(3.1)** **Povezava slovarja s podatki**
TBox	**0,70 (0,70)**	**0,40 (0,40)**	**0,40 (0,40)**	**0,30 (0,30)**	**0,50 (0,50)**	**0,50 (0,50)**	**0,40 (0,40)**
Opis	**0,70 (1,00)**	**0,40 (1,00)**	**0,16 (0,40)**	**0,12 (0,40)**	**0,10 (0,20)**	**0,10 (0,20)**	**0,08 (0,20)**
Obstoj razreda	0,14 (0,20)	0,08 (0,20)	0,03 (0,20)	0,02 (0,20)	0,02 (0,20)	0,02 (0,20)	0,02 (0,20)
Angleško	0,28 (0,40)	0,16 (0,40)	0,06 (0,40)	0,05 (0,40)	0,04 (0,40)	0,04 (0,40)	0,03 (0,40)
Ostalo	0,28 (0,40)	0,16 (0,40)	0,07 (0,40)	0,05 (0,40)	0,04 (0,40)	0,04 (0,40)	0,03 (0,40)
Razdelitev	**-**	**-**	**0,06 (0,15)**	**0,05 (0,15)**	**0,05 (0,10)**	**0,05 (0,10)**	**0,04 (0,10)**
Skupni razred	-	-	0,06 (1,00)	0,05 (1,00)	0,05 (1,00)	0,05 (1,00)	0,04 (1,00)
Odvečnost	**-**	**-**	**0,06 (0,15)**	**0,04 (0,15)**	**0,15 (0,30)**	**0,15 (0,30)**	**0,12 (0,30)**
Identična opredelitev	-	-	-	-	0,07 (0,50)	0,07 (0,50)	0,06 (0,50)
Odvečnost podrazreda	-	-	0,06 (1,00)	0,05 (1,00)	0,08 (0,50)	0,08 (0,50)	0,06 (0,50)
Doslednost	**-**	**-**	**0,06 (0,15)**	**0,05 (0,15)**	**0,10 (0,20)**	**0,10 (0,20)**	**0,08 (0,20)**
Ciklična napaka	-	-	0,06 (1,00)	0,05 (1,00)	0,10 (1,00)	0,10 (1,00)	0,08 (1,00)

	(2.1) Zajem **poslovnega slovarja**	(2.2) Seznam **primerov konceptov** in **lastnosti**	(2.3) Identifikacija **taksonomije**	(2.4) Identifikacija **ad hoc relacij**	(2.5) Opis **atributov** in **relacij konceptov**	(2.6) Dodajanje **kompleksnih omejitev** in **pravil**	(3.1) **Povezava slovarja s podatki**
Anomalija	**-**	**-**	**0,06 (0,15)**	**0,04 (0,15)**	**0,10 (0,20)**	**0,10 (0,20)**	**0,08 (0,20)**
Veriga dedovanja	-	-	0,06 (1,00)	0,04 (1,00)	0,10 (1,00)	0,10 (1,00)	0,08 (1,00)
RBox	**0,30 (0,30)**	**0,20 (0,20)**	**0,20 (0,20)**	**0,30 (0,30)**	**0,50 (0,50)**	**0,50 (0,50)**	**0,40 (0,40)**
Opis	**0,24 (0,80)**	**0,16 (0,80)**	**0,16 (0,80)**	**0,18 (0,60)**	**0,10 (0,20)**	**0,10 (0,20)**	**0,08 (0,20)**
Obstoj lastnosti	0,07 (0,30)	0,05 (0,30)	0,05 (0,30)	0,05 (0,30)	0,03 (0,30)	0,03 (0,30)	0,02 (0,30)
Opis lastnosti	0,12 (0,50)	0,08 (0,50)	0,08 (0,50)	0,09 (0,50)	0,05 (0,50)	0,05 (0,50)	0,04 (0,50)
Angleško	0,06 (0,50)	0,04 (0,50)	0,04 (0,50)	0,04 (0,50)	0,02 (0,50)	0,02 (0,50)	0,02 (0,50)
Ostalo	0,06 (0,50)	0,04 (0,50)	0,04 (0,50)	0,05 (0,50)	0,03 (0,50)	0,03 (0,50)	0,02 (0,50)
Opis razreda	0,05 (0,20)	0,03 (0,20)	0,03 (0,20)	0,04 (0,20)	0,02 (0,20)	0,02 (0,20)	0,02 (0,20)
Razdelitev	**0,06 (0,20)**	**0,04 (0,20)**	**0,04 (0,20)**	**0,06 (0,20)**	**0,10 (0,20)**	**0,10 (0,20)**	**0,08 (0,20)**
Obstoj inverzne lastnosti	-	-	-	-	0,05 (0,50)	0,05 (0,50)	0,04 (0,50)
Pot med razredi	0,06 (1,00)	0,04 (1,00)	0,04 (1,00)	0,06 (1,00)	0,05 (0,50)	0,05 (0,50)	0,04 (0,50)
Odvečnost	**-**	**-**	**-**	**0,06 (0,20)**	**0,15 (0,30)**	**0,15 (0,30)**	**0,12 (0,30)**
Identična opredelitev	-	-	-	-	0,08 (0,50)	0,08 (0,50)	0,06 (0,50)
Odvečnost podlastnosti	-	-	-	0,06 (1,00)	0,07 (0,50)	0,07 (0,50)	0,06 (0,50)
Anomalija	**-**	**-**	**-**	**-**	**0,15 (0,30)**	**0,15 (0,30)**	**0,12 (0,30)**
Ponavljajoča se skupina lastnosti	-	-	-	-	0,15 (1,00)	0,15 (1,00)	0,12 (1,00)
ABox	**-**	**0,40 (0,40)**	**0,40 (0,40)**	**0,40 (0,40)**	**-**	**-**	**0,20 (0,20)**
Opis	**-**	**0,40 (1,00)**	**0,12 (0,30)**	**0,12 (0,30)**	**-**	**-**	**0,08 (0,40)**
Formalen opis	-	-	-	-	-	-	0,06 (0,80)
Obstoj primerka	-	0,40 (1,00)	0,12 (1,00)	0,12 (1,00)	-	-	0,02 (0,20)
Razred	-	0,24 (0,60)	0,07 (0,60)	0,07 (0,60)	-	-	0,01 (0,60)
Lastnost	-	0,16 (0,40)	0,05 (0,40)	0,05 (0,40)	-	-	0,01 (0,40)
Razdelitev	**-**	**-**	**0,28 (0,70)**	**0,28 (0,70)**	**-**	**-**	**0,04 (0,20)**
Skupni primerek	-	-	0,06 (0,20)	0,06 (0,20)	-	-	0,02 (0,50)
Zunanji primerek	-	-	0,05 (0,20)	0,06 (0,20)	-	-	0,02 (0,50)
Hierarhija	-	-	0,17 (0,60)	0,16 (0,60)	-	-	-
Razred	-	-	0,17 (1,00)	0,08 (0,50)	-	-	-
Podrazred	-	-	0,08 (0,50)	0,04 (0,50)	-	-	-
Ekvivalenten razred	-	-	0,05 (0,30)	0,02 (0,30)	-	-	-
Razdružen razred	-	-	0,04 (0,20)	0,02 (0,20)	-	-	-
Lastnost	-	-	-	0,08 (0,50)	-	-	-
Podlastnost	-	-	-	0,05 (0,60)	-	-	-
Ekvivalentna lastnost	-	-	-	0,03 (0,40)	-	-	-
Odvečnost	**-**	**-**	**-**	**-**	**-**	**-**	**0,04 (0,20)**
Identična opredelitev	-	-	-	-	-	-	0,02 (0,50)
Odvečnost primerka	-	-	-	-	-	-	0,02 (0,50)

	(2.1) Zajem **poslovnega slovarja**	(2.2) Seznam **primerov konceptov** in **lastnosti**	(2.3) Identifikacija **taksonomije**	(2.4) Identifikacija **ad hoc relacij**	(2.5) Opis **atributov** in **relacij konceptov**	(2.6) Dodajanje **kompleksnih omejitev** in **pravil**	(3.1) **Povezava slovarja s podatki**
Anomalija	-	-	-	-	-	-	**0,04 (0,20)**
Neuporabljen razred	-	-	-	-	-	-	0,02 (0,50)
Neuporabljena lastnost	-	-	-	-	-	-	0,02 (0,50)

Tabela 3: Uteži pri izračunu indikatorja popolnosti (OC) ontologije za vsak posamezen korak v procesu ROD

Kot je že bilo omenjeno, imamo pri taksonomiji semantičnih testov za izračun indikatorja popolnosti dve dimenziji klasifikacij. Prva razdeljuje semantične teste **glede na komponento ontologije**, ki se preverja. Delimo jih na komponento, s katero opredelimo shemo ontologije (komponente TBox in RBox) ter komponento, ki vsebuje primerke sheme (komponente ABox). Slika 54 prikazuje pomembnost posameznih komponent glede na korake v procesu ROD. Iz diagrama lahko opazimo, da je v začetnem koraku 2.1, t. j. zajem poslovnega slovarja, pomembna zgolj terminološka komponenta (TBox in RBox). V nasprotju s tem pa pri naslednjih korakih 2.2, 2.3 in 2.4 s pomočjo vnašanja primerkov razredov in lastnosti poskušamo pomagati uporabniku pri opredelitvi taksonomije, zato na pomembnosti pridobi tudi komponenta ABox. V korakih 2.5 in 2.6 sledi podrobnejša opredelitev sheme, zato je pozornost zopet na komponentah TBox in RBox. Na koncu želimo razvito ontologijo uporabiti kot komponento v informacijskem sistemu. Pomen preverjanja pravilnosti terminološke komponente se nekoliko zmanjša na račun semantičnih testov komponente, ki vsebuje primerke.

Slika 54: Vpliv uteži komponent na 1. nivoju (TBox, RBox in ABox) korakov v procesu ROD

Druga dimenzija delitve je **glede kategorije napake**, ki se pojavi v ontologiji. To so napake opisa, razdelitve, odvečnosti, doslednosti in anomalije. Slika 55 prikazuje pomembnost preverjanja posameznih napak glede na korake v procesu ROD. Iz diagrama lahko opazimo, da je v začetnih korakih 2.1 in 2.2 zelo velik poudarek na opisu razredov in lastnosti ontologije. V naslednjih korakih 2.3 in 2.4 poleg opisa postane pomembna tudi razdelitev razredov lastnosti, kjer se preverja predvsem pravilnost opredeljene

taksonomije gradnikov. V naslednjih korakih, ko ontologijo še dodatno opredeljujemo, pa postane pomembno tudi preverjanje odvečnosti, anomalije in doslednosti.

Slika 55: Vpliv uteži komponent na 2. nivoju (opis, razdelitev, odvečnost, doslednost in anomalija) korakov v procesu ROD

Pri izračunu indikatorja popolnosti OC dobimo dva rezultata (Slika 56):

- **oceno OC** in
- **priporočila** za izboljšavo ocene OC.

Slika 56: Uporabniški vmesnik prikaza rezultata izračuna OC

Slika 56 prikazuje procentualno vrednost ocene OC (npr. 68 %) in je prikazana v obliki grafičnega indikatorja stanja. Poleg tega osnovnega prikaza je na voljo tudi pogled radarskega grafa. Takšen prikaz daje poudarek na področja, ki potrebujejo izboljšavo, in

na najvišji ravni prikazuje komponente opis, razdelitev, odvečnost, doslednost in anomalija, saj so za uporabnika bolj primerni od komponent TBox, RBox in ABox. Vsa priporočila, ki so rezultat izračuna indikatorja popolnosti, so prikazana v tabelarični obliki in urejena v padajočem zaporedju po vplivu, ki ga imajo na popolnost ontologije. Ko uporabnik izbere določeno priporočilo, se pričakovan vpliv prikaže tudi na radarskem grafu, tako da si uporabnik lažje predstavlja kakšna bo dejanska posledica v primeru izvedbe spremembe v ontologiji.

Ko uporabnik sledi procesu ROD in se nahaja v določenem koraku razvoja, se indikator OC prilagodi omenjenem koraku z utežmi semantičnih testov (Tabela 3 prikazuje porazdelitev uteži v procesu ROD) pri izračunu (npr. v koraku 2.1 procesa ROD, kjer se zajame poslovni slovar, ni potrebe po semantičnih testih, kot so odvečnost primerkov, neuporabljen razred ali obstoj inverzne lastnosti, ampak je poudarek na opisu komponent TBox in RBox ter obstoju poti med razredi).

Slika 57: Izračun popolnosti ontologije

Ko ocena indikatorja OC doseže določeno mejo (npr. 80 %), lahko uporabnik nadaljuje z naslednjim korakom (Slika 57). Prilagojena vrednost popolnosti ontologije se izračuna sproti in se vedno, ko je dosežena mejna vrednost, generira priporočilo nadaljevanja v naslednji korak. Na takšen način proces ROD uporabniku pomaga pri procesu razvoja ontologije od opredelitve poslovnega slovarja do izdelave funkcionalne komponente.

V nadaljevanju bodo predstavljene posamezne komponente in semantični testi znotraj njih (Slika 53). Vsak semantičen test je prikazan z različnih vidikov.

3.3.2 *KOMPONENTA TBOX*

3.3.2.1 Opis

3.3.2.1.1 Obstoj razreda

Problem: Razredi so osnovni gradniki ontologije, zato so nujno potrebni. V primeru, da ne obstajajo, ne moremo uporabljati naprednih mehanizmov za predstavitev znanja.

Cilj: Preveriti, ali obstajajo razredi v ontologiji.

Postopek:

preveriObstojRazreda
  n_C ← Število razredov v ontologiji
  če $n_C < 1$ potem
    $ocena = 0$
  sicer
    $ocena = 1$
  konec pogoja če
  vrni $\boldsymbol{ocena}$
konec

Algoritem 3: Preveri obstoj razreda

Rezultat: Ocena indikatorja popolnosti ontologije, ki upošteva obstoj razredov.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxDesc} \cdot k_{TBoxDescClassExist}$

Pomembnost v procesu ROD:

Slika 58: Pomembnost preverjanja obstoja razreda (opis komponente TBox) pri izračunu OC

Primer: Slika 59 prikazuje enostavno ontologijo, v kateri so opredeljeni 3 razredi.

Slika 59: Primer preverjanja obstoja razreda (opis komponente TBox) pri izračunu OC

Po izvedbi algoritma dobimo oceno 1, ki je rezultat dejstva $n_c = 3 \geq 1$.

3.3.2.1.2 Angleško

Problem: Ko razmišljamo o ponovni uporabi ontologije in uporabi ontologije v aplikacijah, je opis v naravnem jeziku zelo pomemben. Še posebej pomemben je opis v angleškem jeziku, saj s tem dosežemo večjo dostopnost razvite ontologije. Če takšni opisi obstajajo, lahko načrtovalci ontologijo, ki jo je razvil nekdo drug in dal v skupno rabo, hitreje najdejo in uporabijo. Opis komponente TBox je predvsem v obliki oznak in opisov.

Cilj: Identificirati razrede, pri katerih manjkajo oznake ali opisi v angleškem jeziku ter omogočiti uporabniku, da doda manjkajoče opise.

Postopek: Za vsak razred C_i pogledamo, ali obstajata oznaka in opis v angleškem jeziku.

preveriOpisRazredaVAngleskemJeziku
$ocena = 1$
n_C ← Število razredov v ontologiji
$desc_{ENG}^{label}(C_i)$ ← oznaka razreda C_i v angleškem jeziku
$desc_{ENG}^{comment}(C_i)$ ← opis razreda C_i v angleškem jeziku
$manjkajociOpisi = \emptyset$
za vsak $C_i \in TBox$
  če $\nexists desc_{ENG}^{label}(C_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_C}$
    $manjkajociOpisi = manjkajociOpis \cup \{C_i, "oznaka"\}$
  konec pogoja če
  če $\nexists desc_{ENG}^{comment}(C_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_C}$
    $manjkajociOpisi = manjkajociOpisi \cup \{C_i, "opis"\}$
  konec pogoja če
konec sprehoda po vseh razredih
vrni $\boldsymbol{manjkajociOpisi}$ in $\boldsymbol{ocena}$
konec

Algoritem 4: Preveri opis razreda v angleškem naravnem jeziku

Rezultat: Množica parov $\{razred, tip\ opisa\}$, ki označujejo manjkajočo oznako ali opis razreda, ter ocena indikatorja popolnosti ontologije, ki upošteva opis razreda v angleškem jeziku.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxDesc} \cdot k_{TBoxDescEng}$

Pomembnost v procesu ROD:

Slika 60: Pomembnost preverjanja opisa razreda v angleškem naravnem jeziku (opis komponente TBox) pri izračunu OC

Primer: Slika 61 prikazuje ontologijo z oznakami in opisi razredov v angleškem jeziku.

Slika 61: Primer preverjanja opisa razreda v angleškem naravnem jeziku (opis komponente TBox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednje razrede z manjkajočimi opisi v angleškem jeziku: $\{Delnica, "opis"\}$ in $\{K.O.certifikat, "oznaka"\}$. Ocena testne ontologije z manjkajočimi opisi v angleškem naravnem jeziku je $1 - \frac{1}{2 \cdot 3} - \frac{1}{2 \cdot 3} = \frac{4}{6} = 0{,}667$.

3.3.2.1.3 Ostalo

Problem: Ko razmišljamo o ponovni uporabi ontologije in uporabi ontologije v aplikacijah, je opis v naravnem jeziku zelo pomemben. Poleg opisa v angleškem jeziku, je pomemben tudi opis v drugem jeziku, saj tako podpremo večjezičnost ontologije. Če takšni opisi obstajajo, lahko načrtovalci ontologijo, ki jo je razvil nekdo drug in dal v skupno rabo, hitreje najdejo in uporabijo. Opis komponente TBox je predvsem obliki v oznak in opisov.

Cilj: Identificirati razrede, pri katerih manjkajo oznake ali opisi v jeziku, drugačnem od angleškega, ter omogočiti uporabniku, da doda manjkajoče opise.

Postopek: Za vsak razred C_i pogledamo, ali obstajata oznaka in opis v jeziku, drugačnem od angleškega.

preveriOpisRazredaVDrugemJeziku
$ocena = 1$
n_C ← Število razredov v ontologiji
$languages$ ← Množica jezikov, uporabljenih za opis v naravnem jeziku
n_{lang}^{desc} ← Število opisov razredov (desc) v izbranem jeziku (lang)
$desc_{lang}^{label}(C_i)$ ← oznaka razreda C_i v jeziku lang
$desc_{lang}^{comment}(C_i)$ ← opis razreda C_i v jeziku lang
$manjkajociOpisi = \emptyset$
izberi najboljši $lang$ iz $languages$, kjer velja $max(n_{lang}^{desc})$
za vsak $C_i \in TBox$
  če $\nexists desc_{lang}^{label}(C_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_C}$
    $manjkajociOpisi = manjkajociOpisi \cup \{C_i, "oznaka"\}$
  konec pogoja če
  če $\nexists desc_{lang}^{comment}(C_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_C}$
    $manjkajociOpisi = manjkajociOpisi \cup \{C_i, "opis"\}$
  konec pogoja če
konec sprehoda po vseh razredih
vrni ***manjkajociOpisi*** in ***ocena***
konec

Algoritem 5: Preveri opis razreda v naravnem jeziku, drugačnem od angleškega

Rezultat: Množica parov $\{razred, tip\ opisa\}$, ki označuje manjkajočo oznako ali opis razreda ter ocena indikatorja popolnosti ontologije, ki upošteva naravni opis razreda v jeziku, drugačnem od angleškega.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxDesc} \cdot k_{TBoxDescOther}$

Pomembnost v procesu ROD:

Slika 62: Pomembnost preverjanja opisa razreda v naravnem jeziku, drugačnem od angleškega (opis komponente TBox) pri izračunu OC

Primer: Slika 63 prikazuje ontologijo z oznakami in opisi razredov v slovenskem in španskem jeziku.

Slika 63: Primer preverjanja opisa razreda v naravnem jeziku, drugačnem od angleškega (opis komponente TBox) pri izračunu OC

Po izvedbi algoritma je preferenčni jezik za ocenjevanje slovenski, v nasprotju s španskim, ker obstajajo trije opisi v slovenskem jeziku in le eden v španskem. Odkriti so naslednji razredi z manjkajočimi opisi v slovenskem jeziku: $\{Delnica, "opis"\}$, $\{K.O.certifikat, "oznaka"\}$ in $\{K.O.certifikat, "opis"\}$. Ocena testne ontologije z manjkajočimi opisi v naravnem jeziku, drugačnem od angleškega, je $1 - \frac{1}{2 \cdot 3} - \frac{1}{2 \cdot 3} - \frac{1}{2 \cdot 3} = \frac{3}{6} = 0{,}500$.

3.3.2.2 Razdelitev

3.3.2.2.1 Skupni razred

Problem: Pri razdelitvi razredov v taksonomijo se pogosto zgodi, da je določen razred podrazred razredov, ki so razdruženi[47]. V tem primeru pride do napake skupnih razredov.

Cilj: Identificirati skupine razredov, pri katerih se pojavlja napaka skupnih razredov, in omogočiti uporabniku spremembo ontologije v smeri odprave relacij podrazredov do razdruženih razredov.

Postopek: Za vsak razred C_i preverimo, ali obstajata nadrazreda C_j in C_k, ki sta razdružena.

[47] Za **razdružene razrede** (eng. disjoint class) velja, da je presek med njimi prazna množica, kar lahko zapišemo kot $\bigcap C_i = \emptyset$.

preveriSkupneRazrede
$ocena = 1$
$n_{\subset}^{direkt}$ ← Število neposredni podrazred relacij v ontologiji
$C_1 \subset C_2$ ← C_1 je podrazred od C_2
$C_1 \cap C_2 = \emptyset$ ← C_1je razdružen z C_2
$razdruzeniRazredi = \emptyset$
za vsak $C_i \in TBox$
  če $\exists C_j, C_k \in TBox : C_i \neq C_j \neq C_k \wedge C_i \subset C_j \wedge C_i \subset C_k \wedge C_j \cap C_k = \emptyset$ potem
    $ocena = ocena - \frac{1}{n_{\subset}^{direkt}}$
    $razdruzeniRazredi = razdruzeniRazredi \cup \{C_i, C_j, C_k\}$
  konec pogoja če
konec sprehoda po vseh razredih
vrni $\boldsymbol{manjkajociOpisi}$ in $\boldsymbol{ocena}$
konec

Algoritem 6: Preveri skupne razrede

Rezultat: Množica trojic $\{podrazred, 1.razdružen\ nadrazred, 2.razdružen\ nadrazred\}$, s pomočjo katerih identificiramo problem skupnih razredov, in ocena indikatorja popolnosti ontologije, ki upošteva skupne razrede.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxPart} \cdot k_{TBoxPartComClass}$

Pomembnost v procesu ROD:

Slika 64: Pomembnost preverjanja skupnih razredov (razdelitev komponente TBox) pri izračunu OC

Primer: Slika 65 prikazuje ontologijo, ki vsebuje razrede in njihovo medsebojno povezanost z relacijami podrazred ter številne napake skupnih razredov.

Slika 65: Primer preverjanja skupnih razredov (razdelitev komponente TBox) pri izračunu OC

Po izvedbi algoritma dobimo seznam naslednjih skupnih razredov: {*Dolgoročno posojilo, Depozit, Posojilo*}, {*Dolg, Obrestni inštrument, Izvedeni inštrument na blago*} in {*Dolžniški inštrument, Delnica, Lastniško izvedeni inštrument*}. Ocena testne ontologije z napakami skupnih razredov je $1 - \frac{1}{19} - \frac{1}{19} - \frac{1}{19} = \frac{16}{19} = 0{,}842$.

3.3.2.3 Odvečnost

3.3.2.3.1 Identična opredelitev

Problem: Pri modeliranju ontologij uporabniki pogosto kreirajo več razredov z različnimi imeni, vendar z identično formalno opredelitvijo. Dejstvo je, da gre za isti pojem, vendar drugače poimenovan (t. i. sinonim).

Cilj: Preveriti, ali v ontologiji obstajajo razredi z identično formalno opredelitvijo in omogočiti uporabniku rešitev problema – uvedba ekvivalentnih razredov, združevanja razredov ali odstranitve odvečnih razredov.

Postopek: Za vsak razred C_i preverimo vse pripadajoče aksiome. Če je množica aksiomov razreda C_i identična množici aksiomov razreda C_j, in velja $C_i \neq C_i$, potem smo identificirali dva razreda, ki imata različni imeni in identični formalni opredelitvi.

preveriIdenticnoOpredelitevRazreda
  $ocena = 1$
  n_C ← Število razredov v ontologiji
  $axiom(C) = [tip,\ entiteta,\ vrednost]$ ← Aksiom razreda C, kjer velja $entiteta = C$
  $identicneOpredelitve = \emptyset$
  za vsak $C_i \in TBox$, kjer C_i še ni obiskan
    označi C_i kot obiskan
    za vsak $axiom(C_i)$
      če $\exists axiom(C_j)$, kjer velja $C_i \neq C_j$, $axiom^{tip}(C_i) = axiom^{tip}(C_j)$ in
        $axiom^{vrednost}(C_i) = axiom^{vrednost}(C_j)$ potem
        $ocena = ocena - \frac{2}{(n_C - 1) \cdot n_C}$
        $identicneOpredelitve = identicneOpredelitve \cup \{C_i, C_j\}$
        označi C_j kot obiskan
      konec pogoja če
    konec sprehoda po aksiomih razreda C_i
  konec sprehoda po razredih
  vrni $\boldsymbol{identicneOpredelitve}$ in $\boldsymbol{ocena}$
konec

Algoritem 7: Preveri identično opredelitev razreda

Rezultat: Množica parov razredov (C, D) z identično opredelitvijo in različnimi imeni ter ocena indikatorja popolnosti ontologije, ki upošteva identično opredelitev razreda.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxRed} \cdot k_{TBoxRedIdent}$

Pomembnost v procesu ROD:

Slika 66: Pomembnost preverjanja identične opredelitve razreda (odvečnost komponente TBox) pri izračunu OC

Primer: Slika 67 prikazuje ontologijo, ki vsebuje razreda z identično formalno opredelitvijo in različnima imenoma.

Slika 67: Primer preverjanja identične opredelitve razreda (odvečnost komponente TBox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednjo identično opredelitev razreda: {*ETF, Na borzi trgovan sklad*}. Ocena testne ontologije, ki upošteva identično opredelitev razreda, je $1 - \frac{2}{5 \cdot 4} = \frac{18}{20} = 0{,}900$.

3.3.2.3.2 Odvečnost podrazreda

Problem: Do odvečnosti podrazreda pride takrat, ko načrtovalec ontologije neposredno ali posredno opredeli, da je razred C_i podrazred razredov C_j in C_k, medtem ko velja, da je C_j podrazred razreda C_k. Do problema pride takrat, ko neposredno ali posredno obstaja ena ali več omenjenih relacij podrazred.

Cilj: Preveriti, ali v ontologiji obstajajo odvečne relacije podrazred, in omogočiti uporabniku rešitev problema – odstranitev odvečnih relacij.

Postopek: Za vsak razred C_i preverimo, ali v množici nadrazredov $superClass_{direct}(C)$ obstajajo relacije podrazred.

preveriOdvecnostPodRazreda
  $ocena = 1$
  $n_{\subset}^{direct}$ ← Število neposrednih relacij podrazred v ontologiji
  $C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
  $superClass_{direct}(C)$ ← Neposredni nadrazredi razreda C
  $odvecnostPodRazredov = \emptyset$
  za vsak $C_i \in TBox$
    če $\exists C_j, C_k \in superClass_{direct}(C_i): C_j \neq C_k \wedge C_j \subseteq C_k$ potem
      $ocena = ocena - \frac{1}{n_{\subset}^{direct}}$
      $odvecnostPodRazredov = odvecnostPodRazredov \cup \{C_i, C_k\}$
    konec pogoja če
  konec sprehoda po vseh razredih
  vrni $\boldsymbol{odvecnostPodRazredov}$ in $\boldsymbol{ocena}$
konec

Algoritem 8: Preveri odvečnost podrazreda

Rezultat: Množica parov razredov (C, D) z odvečno relacijo podrazred in ocena indikatorja popolnosti ontologije, ki upošteva odvečnost podrazreda.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxRed} \cdot k_{TBoxRedSubclass}$

Pomembnost v procesu ROD:

Slika 68: Pomembnost preverjanja odvečnosti podrazreda (odvečnost komponente TBox) pri izračunu OC

Primer: Slika 69 prikazuje ontologijo s številnimi podrazred relacijami, tako neposrednimi kot tudi posrednimi. Nekatere izmed njih kršijo pravilo odvečnosti podrazreda in so ustrezno označene.

Slika 69: Primer preverjanja odvečnosti podrazreda (odvečnost komponente TBox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednje odvečne podrazred relacije: $\{Nekotirajoča\ delnica,\ Finančni\ inštrument\}$ in $\{Opcijsko\ izveden\ inštrument,\ Inštrument\ lastniškega\ kapitala\}$. Ocena testne ontologije, ki upošteva odvečnost podrazreda, je $1 - \frac{1}{11} - \frac{1}{11} = \frac{9}{11} = 0{,}818$.

3.3.2.4 Doslednost

3.3.2.4.1 Ciklična napaka

Problem: Do ciklične napake pride takrat, ko je razred opredeljen kot podrazred ali nadrazred samega sebe na poljubni ravni hierarhije v ontologiji. Pojavljajo se lahko na razdalji 2, 3 ali n, in sicer odvisno od števila razredov oz. števila relacij podrazred, ki so udeležene pri sprehodu po hierarhiji konceptov od izbranega koncepta pa zopet nazaj do njega.

Cilj: Preveriti, ali v ontologiji obstajajo cikli v razredni taksonomiji, in omogočiti uporabniku rešitev problema – odstranitev odvečnih relacij.

Postopek: Ontologija je predstavljena v obliki usmerjenega grafa $G = (V, E)$, kjer je V množica razredov in E množica relacij podrazred. Pri iskanju cikličnih napak v grafu iščemo cikle C_n, kjer je n število povezanih točk v zaprti verigi, kar hkrati predstavlja tudi razdaljo ciklične napake.

poisciCiklicne Napake
  $ocena = 1$
  n_C ← Število razredov v ontologiji
  V ← Točke grafa so razredi
  E ← Povezave v grafu so relacije podrazred
  $cikli = \emptyset$
  medtem ko v $G(V, E)$ obstajajo cikli C_n
    izberi cikel C_n iz množice vseh ciklov, kjer velja $max(n)$
    $cikli = cikli \cup C_n$
    odstrani vse točke cikla C_n iz $G(V, E)$
    $ocena = ocena - \frac{n}{n_C}$
  konec pogoja
  vrni ***cikli*** in ***ocena***
konec

Algoritem 9: Poišči ciklične napake

Rezultat: Množica ciklov s pripadajočimi razredi, ki predstavljajo ciklične napake in ocena indikatorja popolnosti ontologije, ki upošteva ciklične napake.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxConsist} \cdot k_{TBoxConsistCirc}$

Pomembnost v procesu ROD:

Slika 70: Pomembnost preverjanja cikličnih napak (doslednost komponente TBox) pri izračunu OC

Primer: Slika 71 prikazuje ontologijo s številnimi cikličnimi napakami (C_2, C_3 in C_4).

Slika 71: Primer preverjanja cikličnih napak (doslednost komponente TBox) pri izračunu OC

Po izvedbi algoritma so odkrite naslednje ciklične napake: {*Trgovalni dan, Nakup, Doji nakup, Dolg Doji nakup*}, {*Delnica, Vroča delnica, Zelo vroča delnica*} in {*Finančni inštrument, ETF*}. Ocena testne ontologije z upoštevanjem cikličnih napak je $1 - \frac{4}{12} - \frac{3}{12} - \frac{2}{12} = \frac{3}{12} = 0{,}250$.

3.3.2.5 Anomalija

3.3.2.5.1 Veriga dedovanja

Problem: Pri opredelitvi taksonomije razredov lahko načrtovalec ontologije uporablja relacijo podrazred na poljubnih ravneh med posameznimi razredi. Ko je takšna veriga dedovanja (zaporedna uporaba relacij podrazred) dovolj dolga in obstajajo razredi v tej verigi, ki nimajo ustreznih opisov, razen zadnjega člena v verigi, gre za napako verige dedovanja.

Cilj: Poiskati verige dedovanja in omogočiti uporabniku rešitev problema – razdelitev verige dedovanja v podhierarhije ali odstraniti nepotrebne relacije podrazred.

Postopek: Iščemo taksonomijo razredov $[C_j, C_n, \dots, C_2, C_1, C_i]$ v smeri od zadnjega elementa taksonomije do korenskega razreda in temelji na relacijah podrazred. Začne se z razredom C_j in konča z razredom C_i, ki sta dodatno opisana, medtem ko razredi $C_1, C_2, \dots, C_n$ zgolj podedujejo opise in ne vsebujejo nobenih dodatnih opisov.

poisciVerigoDedovanja
$ocena = 1$
$n_{\subseteq}^{direct}$ ← Število neposrednih relacij podrazred v ontologiji
$C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
$axiom(C) = [tip, entiteta, vrednost]$ ← Aksiom razreda C, kjer velja $entiteta = C$
$A(C) = \forall axiom(C) : entiteta = C$ ← Množica vnaprej opredeljenih[48] aksiomov razreda C
$A_{\subseteq}^{-}(C)$ ← Množica vnaprej opredeljenih aksiomov razreda C brez aksiomov tipa podrazred
$superClass(C)$ ← Množica razred, katerih razred C je podrazred
$verigeDedovanja = \emptyset$
dokler $\exists C_i, C_j \in TBox \wedge \exists C_1, C_2, \dots, C_n \in TBox : \left(C_j \subseteq C_n \subseteq C_{n-1} \subseteq \cdots \subseteq C_2 \subseteq C_1 \subseteq C_i\right) \wedge$
$(\forall C_1, C_2, \dots, C_n : |superClass(C_n)| = 1 \wedge A_{\subseteq}^{-}(C_n) = \emptyset) \wedge |A_{\subseteq}^{-}(C_i)| > 0 \wedge \left|A_{\subseteq}^{-}(C_j)\right| > 0$ potem
$ocena = ocena - \frac{n}{n_{\subseteq}^{direct}}$
$verigeDedovanja = verigeDedovanja \cup \left\{C_i, C_j, \{C_1, C_2, \dots, C_n\}\right\}$
konec pogoja
vrni $\boldsymbol{verigeDedovanja}$ in $\boldsymbol{ocena}$
konec

Algoritem 10: Poišči verigo dedovanja

Rezultat: Množica verig dedovanja v obliki {*podroben razred, splošen razred,* {*razredi, ki sestavljajo verigo dedovanja*}} in ocena indikatorja popolnosti ontologije, ki upošteva verigo dedovanja.

Globalna utež v procesu ROD: $w' = k_{TBox} \cdot k_{TBoxAnom} \cdot k_{TBoxAnomInherChain}$

Pomembnost v procesu ROD:

Slika 72: Pomembnost iskanja verige dedovanja (anomalija komponente TBox) pri izračunu OC

Primer: Slika 73 prikazuje ontologijo z neposrednimi relacijami podrazred in dodatnimi opisi z lastnostmi. Nekateri razredi kršijo pravilo verige dedovanja in so zato tudi ustrezno barvno označeni.

[48] **Vnaprej opredeljeni aksiomi** (ang. asserted axioms) so tiste izjave v bazi znanja, ki so eksplicitno že prisotne pred izvajanjem mehanizma sklepanja.

Slika 73: Primer iskanja verige dedovanja (anomalija komponente TBox) pri izračunu OC

Po izvedbi algoritma dobimo naslednje verige dedovanja: {*Opcijsko izveden inštrument, Inštrument lastniškega kapitala,* {*Lastniško izveden inštrument*}}, {*Večinoma obvezniški sklad, Sklad denarnega trga,* {*Mešani sklad, Uravnoteženi sklad, Obvezniško usmerjen sklad*}} in {*Zadolžnica, Investicije in skladi,* {*Izveden inštrument, Pogodbeni izvedeni inštrument*}}. Ocena testne ontologije z upoštevanjem verige dedovanja je $1 - \frac{1}{17} - \frac{3}{17} - \frac{2}{17} = \frac{11}{17} = 0{,}647$.

3.3.3 *Komponenta RBox*

3.3.3.1 Opis

3.3.3.1.1 Obstoj lastnosti

Problem: Lastnosti so poleg razredov osnovni gradniki ontologije, zato so nujno potrebni. V primeru, da ne obstajajo, ne moremo uporabljati naprednih mehanizmov za predstavitev znanja.

Cilj: Preveriti, ali obstajajo lastnosti v ontologiji.

Postopek:

preveriObstojLastnosti
n_P ← Število lastnosti v ontologiji
če $n_P < 1$ potem
$ocena = 0$
sicer
$ocena = 1$
konec pogoja če
vrni $\boldsymbol{ocena}$
konec

Algoritem 11: Preveri obstoj lastnosti

Rezultat: Ocena indikatorja popolnosti ontologije, ki upošteva obstoj lastnosti.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxDesc} \cdot k_{RBoxDescPropExist}$

Pomembnost v procesu ROD:

Slika 74: Pomembnost preverjanja obstoja lastnosti (opis komponente RBox) pri izračunu OC

Primer: Slika 75 prikazuje enostavno ontologijo, v kateri ni opredeljenih nobenih lastnosti

Slika 75: Primer preverjanja obstoja lastnosti (opis komponente RBox) pri izračunu OC

Po izvedbi algoritma dobimo oceno 1, ki je rezultat dejstva $n_P = 0 < 1$.

3.3.3.1.2 Opis lastnosti

3.3.3.1.2.1 Angleško

Problem: Ko razmišljamo o ponovni uporabi ontologije in uporabi ontologije v aplikacijah, je opis v naravnem jeziku zelo pomemben. Še posebej pomemben je opis v angleškem jeziku, saj s tem dosežemo večjo dostopnost razvite ontologije. Če takšni opisi obstajajo, lahko načrtovalci ontologijo, ki jo je razvil nekdo drug in dal v skupno rabo, hitreje najdejo in uporabijo. Opis komponente RBox je predvsem v obliki oznak in opisov.

Cilj: Identificirati lastnosti, pri katerih manjkajo oznake ali opisi v angleškem jeziku ter omogočiti uporabniku, da doda manjkajoče opise.

Postopek: Za vsako lastnost P_i pogledamo, ali obstajata oznaka in opis v angleškem jeziku.

preveriOpisLastnostiVAngleskemJeziku
$ocena = 1$
n_P ← Število lastnosti v ontologiji
$desc_{ENG}^{label}(P_i)$ ← oznaka lastnosti P_i v angleškem jeziku
$desc_{ENG}^{comment}(P_i)$ ← opis lastnosti P_i v angleškem jeziku
$manjkajociOpisi = \emptyset$
za vsako $P_i \in RBox$
  če $\nexists desc_{ENG}^{label}(P_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_P}$
    $manjkajociOpisi = manjkajociOpisi \cup \{P_i, "oznaka"\}$
  konec pogoja če
  če $\nexists desc_{ENG}^{comment}(P_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_P}$
    $manjkajociOpisi = manjkajociOpisi \cup \{P_i, "opis"\}$
  konec pogoja če
konec sprehoda po vseh lastnostih
vrni ***manjkajociOpisi*** in ***ocena***
konec

Algoritem 12: Preveri opis lastnosti v angleškem naravnem jeziku

Rezultat: Množica parov $\{lastnost, tip\ opisa\}$, ki označuje manjkajočo oznako ali opis lastnosti, ter ocena indikatorja popolnosti ontologije, ki upošteva naravni opis lastnosti v angleškem jeziku.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxDesc} \cdot k_{RBoxDescProp} \cdot k_{RBoxDescPropEng}$

Pomembnost v procesu ROD:

Slika 76: Pomembnost preverjanja opisa lastnosti v angleškem naravnem jeziku (opis komponente RBox) pri izračunu OC

Primer: Slika 77 prikazuje ontologijo z oznakami in opisi lastnosti v angleškem jeziku.

Slika 77: Primer preverjanja opisa lastnosti v angleškem naravnem jeziku (opis komponente RBox) pri OC

Po izvedbi algoritma odkrijemo naslednjo lastnost z manjkajočim opisom v angleškem jeziku: $\{trguje\ na, "opis"\}$. Ocena testne ontologije z manjkajočimi opisi v angleškem naravnem jeziku je $1 - \frac{1}{2 \cdot 2} = \frac{3}{4} = 0{,}750$.

3.3.3.1.2.2 Ostalo

Problem: Ko razmišljamo o ponovni uporabi ontologije in uporabi ontologije v aplikacijah, je opis v naravnem jeziku zelo pomemben. Poleg opisa v angleškem jeziku je pomemben tudi opis v drugem jeziku, saj tako podpremo večjezičnost ontologije. Če takšni opisi obstajajo, lahko načrtovalci ontologijo, ki jo je razvil nekdo drug in dal v skupno rabo, hitreje najdejo in uporabijo. Opis komponente RBox je predvsem v obliki oznak in opisov.

Cilj: Identificirati lastnosti, pri katerih manjkajo oznake ali opisi v jeziku, drugačnem od angleškega, ter omogočiti uporabniku, da doda manjkajoče opise.

Postopek: Za vsako lastnost P_i pogledamo, ali obstajata oznaka in opis v jeziku, drugačnem od angleškega.

preveriOpisLastnostiVDrugemJeziku
$ocena = 1$
n_P ← Število lastnosti v ontologiji
$languages$ ← Množica jezikov, uporabljenih za opis v naravnem jeziku
n_{lang}^{desc} ← Število opisov lastnosti v izbranem jeziku
$desc_{lang}^{label}(P_i)$ ← oznaka lastnosti P_i v jeziku lang
$desc_{lang}^{comment}(P_i)$ ← opis lastnosti P_i v jeziku lang
$manjkajociOpisi = \emptyset$
izberi najboljši $lang$ iz $languages$, kjer velja $max(n_{lang}^{desc})$
za vsako $P_i \in RBox$
  če $\nexists desc_{lang}^{label}(P_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_P}$
    $manjkajociOpisi = manjkajociOpisi \cup \{P_i, "oznaka"\}$
  konec pogoja če
  če $\nexists desc_{lang}^{comment}(P_i)$ potem
    $ocena = ocena - \frac{1}{2 \cdot n_P}$
    $manjkajociOpisi = manjkajociOpisi \cup \{P_i, "opis"\}$
  konec pogoja če
konec sprehoda po vseh lastnostih
vrni ***manjkajociOpisi*** in ***ocena***
konec

Algoritem 13: Preveri opis lastnosti v naravnem jeziku, drugačnem od angleškega

Rezultat: Množica parov $\{lastnost, tip\ opisa\}$, ki označuje manjkajočo oznako ali opis razreda ter ocena indikatorja popolnosti ontologije, ki upošteva naravni opis razreda v jeziku, drugačnem od angleškega.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxDesc} \cdot k_{RBoxDescProp} \cdot k_{RBoxDescPropOther}$

Pomembnost v procesu ROD:

Slika 78: Pomembnost preverjanja opisa lastnosti v naravnem jeziku, drugačnem od angleškega (opis komponente RBox) pri izračunu OC

Primer: Slika 79 prikazuje ontologijo z oznakami in opisi lastnosti v slovenskem in španskem jeziku.

Slika 79: Primer preverjanja opisa lastnosti v naravnem jeziku, drugačnem od angleškega (opis komponente RBox) pri izračunu OC

Po izvedbi algoritma je preferenčni jezik za ocenjevanje slovenski, v nasprotju s španskim, ker obstajata dva opisa v slovenskem jeziku in le eden v španskem. Odkrita je naslednja lastnost z manjkajočimi opisi v slovenskem jeziku: $\{kotira\ na, "opis"\}$ in $\{kotira\ na, "oznaka"\}$. Ocena testne ontologije z manjkajočimi opisi v naravnem jeziku, drugačnem od angleškega, je $1 - \frac{1}{2 \cdot 2} - \frac{1}{2 \cdot 2} = \frac{2}{4} = 0{,}500$.

3.3.3.1.3 Opis razreda

Problem: Pri načrtovanju ontologije ob opredelitvi lastnosti načrtovalec ontologije včasih pozabi definirati definicijsko območje in zalogo vrednosti. Za popolno opredelitev lastnosti mora le-ta opisovati vsaj en razred z najmanj enim elementom v zalogo vrednosti, ki je lahko bodisi enostaven podatkovni tip bodisi kompleksen (razred).

Cilj: Identificirati pomanjkljivo opredeljene lastnosti, ki ne prispevajo k povezanosti elementov ontologije, in omogočiti uporabniku, da doda manjkajoče opredelitve.

Postopek: Vsaka lastnost P_i mora imeti ustrezno opredeljeno definicijsko območje in zalogo vrednosti.

preveriOpisRazreda
  $ocena = 1$
  n_P ← Število lastnosti v ontologiji
  $domain(P)$ ← definicijsko območje lastnosti P
  $range(P)$ ← zaloga vrednosti lastnosti P
  $manjkajociOpisi = \emptyset$
  za vsako $P_i \in RBox$
    če $|domain(P)| < 1$ potem
      $ocena = ocena - \frac{1}{2 \cdot n_P}$
      $manjkajociOpisi = manjkajociOpisi \cup \{P_i, "definicijsko\ območje"\}$
    konec pogoja če
    če $|range(P_i)| < 1$ potem
      $ocena = ocena - \frac{1}{2 \cdot n_P}$
      $manjkajociOpisi = manjkajociOpisi \cup \{P_i, "zaloga\ vrednosti"\}$
    konec pogoja če
  konec sprehoda po vseh lastnostih
  vrni ***manjkajociOpisi*** in ***ocena***
konec

Algoritem 14: Preveri opis razreda

Rezultat: Množica parov $\{lastnost, pomanjkljivost\}$, ki označuje manjkajočo definicijsko območje ali zalogo vrednosti določene aktivnosti ter ocena indikatorja popolnosti ontologije, ki upošteva opise razredov.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxDesc} \cdot k_{RBoxDescConc}$

Pomembnost v procesu ROD:

Slika 80: Pomembnost preverjanja opisa razreda (opis komponente RBox) pri izračunu OC

Primer: Slika 81 prikazuje enostavno ontologijo z 2 razredoma in 2 lastnostmi, v kateri lastnosti niso pravilno opredeljene.

Slika 81: Primer preverjanja opisa razreda (opis komponente RBox) pri izračunu OC

Po izvedbi algoritma je odkrita naslednja lastnost, ki je pomanjkljivo opredeljena: $\{ISIN, "zaloga\ vrednosti"\}$. Ocena testne ontologije z nepopolnim opisom razreda je $1 - \frac{1}{2 \cdot 2} = \frac{3}{4} = 0{,}750$.

3.3.3.2 Razdelitev

3.3.3.2.1 Obstoj inverzne lastnosti

Problem: Če želimo omogočiti popolno sprehajanje med razredi v ontologiji, k temu zelo veliko prispeva dejstvo, da ima vsaka objektna lastnost opredeljeno tudi inverzno lastnost. S to zahtevo dosežemo, da lahko uporabnik prosto preiskuje informacijski prostor v poljubni smeri in tako nikoli ne pride v "mrtvo ulico".

Cilj: Poiskati objektne lastnosti brez opredeljenih inverznih lastnosti in omogočiti uporabniku dodajanje manjkajočih opredelitev.

Postopek: Za vsako objektno lastnost P_i preverimo, če obstaja tudi njej inverzna.

preveriObstojInverzneLastnosti
  $ocena = 1$
  n_P^o ← Število objektnih lastnosti v ontologiji
  P' ← Inverzna lastnost lastnosti P
  $manjkajoceInverzneLastnosti = \emptyset$
  za vsako objektno lastnost $P_i \in RBox$, kjer P_i še ni obiskana
    če $\exists P_i'$ potem
      označi P_i' kot obiskano
    sicer
      $ocena = ocena - \frac{1}{n_P^o}$
      $manjkajoceInverzneLastnosti = manjkajoceInverzneLastnosti \cup P_i$
    konec pogoja če
    označi P_i kot obiskano
  konec sprehoda po objektnih lastnostih
  vrni ***manjkajoceInverzneLastnosti*** in ***ocena***
konec

Algoritem 15: Preveri obstoj inverzne lastnosti

Rezultat: Množica lastnosti, ki nima opredeljenih inverznih lastnosti, in ocena indikatorja popolnosti ontologije, ki upošteva obstoj inverznih lastnosti.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxPart} \cdot k_{RBoxPartInv}$

Pomembnost v procesu ROD:

Slika 82: Pomembnost preverjanja obstoja inverznih lastnosti (razdelitev komponente RBox) pri izračunu OC

Primer: Slika 83 prikazuje enostavno ontologijo, ki vsebuje različne objektne lastnosti, kjer nekatere nimajo opredeljenih inverznih lastnosti.

Slika 83: Primer preverjanja obstoja inverznih lastnosti (razdelitev komponente RBox) pri izračunu OC

Po izvedbi algoritma dobimo naslednjo lastnost, ki nima opredeljene inverzne lastnosti: $\{je\ del\}$. Ocena testne ontologije, ki upošteva obstoj inverzne lastnosti je $1 - \frac{1}{3} = \frac{2}{3} =$ 0,667.

3.3.3.2.2 Pot med razredi

Problem: Pri gradnji ontologije želimo opredeliti takšen podatkovni slovar, v katerem so koncepti medsebojno povezani. Nepovezane množice konceptov (t.i. otoki) so nezaželene. V primeru, da do njih pride, se priporoča izločitev le-teh v drugo ontologijo, ker niso povezane z obstoječo problemsko domeno, ali pa uvedemo povezavo med nepovezanimi množicami konceptov.

Cilj: Omogočiti sprehod med vsemi koncepti v ontologiji, s tem da poiščemo nepovezane množice konceptov. Uporabnik lahko opredeli dodatne povezave ali odstrani množice nepovezanih konceptov.

Postopek: Ontologijo predstavimo kot neusmerjen graf $G = (V, E)$, kjer je V množica razredov, E pa množica relacij podrazred in objektne lastnosti. Pri iskanju nepovezanih množic konceptov iščemo maksimalno povezane podgrafe.

poisciPotMedRazredi
  $ocena = 0$
  n_C ← Število razredov v ontologiji
  V ← Točke grafa so razredi
  E ← Povezave v grafu so relacije podrazred in objektne lastnosti
  $nepovezaneMnozice = \emptyset$
  za vsako $v \in V$
    $povezanaMnozica = BFS'(G, v)$
    $nepovezaneMnozice = nepovezaneMnozice \cup povezanaMnozica$
    $ocena = ocena + \frac{|povezanaMnozica| \cdot (|povezanaMnozica| - 1)}{n_C \cdot (n_C - 1)}$
  konec sprehoda po vseh točkah grafa
  vrni $\boldsymbol{nepovezaneMnozice}$ in $\boldsymbol{ocena}$
konec

BFS' (G, v)
  izvedi iskanje v širino na grafu G, z začetkom v točki v
  odstrani vse obiskane točke iz grafa G
  vrni množico vseh obiskanih točk
konec

Algoritem 16: Poišči pot med razredi

Rezultat: Nepovezane množice razredov in ocena indikatorja popolnosti ontologije, ki upošteva pot med razredi.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxPart} \cdot k_{RBoxPartPath}$

Pomembnost v procesu ROD:

Slika 84: Pomembnost iskanja poti med razredi (razdelitev komponente RBox) pri izračunu OC

Primer: Slika 85 prikazuje ontologijo z dvema množicama nepovezanih razredov, tako da ne obstaja pot med vsemi razredi ontologije.

Slika 85: Primer iskanja poti med razredi (razdelitev komponente RBox) pri izračunu OC

Po izvedbi algoritma so identificirane naslednje nepovezane množice konceptov: $\{Finančni\ inštrument, Borza, ETF, Delnica, K.O.certifikat\}$ in $\{Trgovalni\ nakup, Prodaja,\ Nakup\}$. Ocena testne ontologije, ki upošteva poti med razredi, je $\frac{5\cdot4+3\cdot2}{8\cdot7} = \frac{26}{56} = 0{,}464$.

3.3.3.3 Odvečnost

3.3.3.3.1 Identična opredelitev

Problem: Pri modeliranju ontologij uporabniki pogosto kreirajo več lastnosti z različnimi imeni, vendar z identično formalno opredelitvijo. Dejstvo je, da gre za isti pojem, vendar drugače poimenovan (t. i. sinonim).

Cilj: Preveriti, ali v ontologiji obstajajo lastnosti z identično formalno opredelitvijo in omogočiti uporabniku rešitev problema – uvedba ekvivalentnih lastnosti, združevanja lastnosti ali odstranitve razredov.

Postopek: Za vsako lastnost P_i preverimo vse pripadajoče aksiome. Če je množica aksiomov lastnosti P_i identična množici aksiomov lastnosti P_j, in velja $P_i \neq P_i$, potem smo identificirali dve lastnosti, ki imata različni imeni in identični formalni opredelitvi.

preveriIdenticnoOpredelitevLastnosti
  $ocena = 1$
  n_P ← Število lastnosti v ontologiji
  $axiom(P) = [tip,\ entiteta,\ vrednost]$ ← Aksiom lastnosti P, kjer velja $entiteta = P$
  $identicneOpredelitve = \emptyset$
  za vsako $P_i \in TBox$, kjer P_i še ni obiskana
    označi P_i kot obiskano
    za vsak $axiom(P_i)$
      če $\exists axiom(P_j)$, kjer velja $P_i \neq P_j$, $axiom^{tip}(P_i) = axiom^{tip}(P_j)$ in
        $axiom^{vrednost}(P_i) = axiom^{vrednost}(P_j)$ potem
        $ocena = ocena - \frac{2}{(n_P - 1) \cdot n_P}$
        $identicneOpredelitve = identicneOpredelitve \cup \{P_i, P_j\}$
        označi P_j kot obiskano
      konec pogoja če
    konec sprehoda po aksiomih lastnosti P_i
  konec sprehoda po lastnostih
  vrni $\boldsymbol{identicneOpredelitve}$ in $\boldsymbol{ocena}$
konec

Algoritem 17: Preveri identično opredelitev lastnosti

Rezultat: Množica parov lastnosti $\{P, Q\}$ z identično opredelitvijo in različnimi imeni ter ocena indikatorja popolnosti ontologije, ki upošteva identično opredelitev lastnosti.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxRed} \cdot k_{RBoxRedIdent}$

Pomembnost v procesu ROD:

Slika 86: Pomembnost preverjanja identične opredelitve lastnosti (odvečnost komponente RBox) pri izračunu OC

Primer: Slika 87 prikazuje ontologijo, ki vsebuje lastnosti z identično formalno opredelitvijo in različnimi imeni.

Slika 87: Primer preverjanja identične opredelitve lastnosti (odvečnost komponente RBox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednji identični opredelitvi lastnosti: $\{temelji\ na, odvisen\ od\}$ in $\{je\ del,\ je\ udeležen\ pri\}$. Ocena testne ontologije, ki upošteva identično opredelitev lastnosti, je $1 - \frac{2}{5 \cdot 4} - \frac{2}{5 \cdot 4} = \frac{16}{20} = 0{,}800$.

3.3.3.3.2 Odvečnost podlastnosti

Problem: Do odvečnosti podlastnosti pride takrat, ko načrtovalec ontologije neposredno ali posredno opredeli, da je lastnost P_i podlastnost lastnosti P_j in P_k, medtem ko velja, da je P_j podlastnost lastnosti P_k. Do problema pride takrat, ko neposredno ali posredno obstaja ena ali več omenjenih relacij podlastnost.

Cilj: Preveriti, ali v ontologiji obstajajo odvečne relacije podlastnost, in omogočiti uporabniku rešitev problema – odstranitev odvečnih lastnosti.

Postopek: Za vsako lastnost P_i preverimo, ali v množici nadlastnosti $superProperty_{direct}(P)$ obstajajo relacije podlastnost.

preveriOdvecnostPodLastnosti
  $ocena = 1$
  $n_{\subset}^{direct}$ ← Število neposrednih relacij podlastnost v ontologiji
  $P_1 \subseteq P_2$ ← P_1 je podlastnost od P_2
  $superProperty_{direct}(P)$ ← Neposredne nadlastnosti lastnosti P
  $odvecnostPodLastnosti = \emptyset$
  za vsako $P_i \in RBox$
    če $\exists P_j, P_k \in superProperty_{direct}(P_i) : P_j \neq P_k \wedge P_j \subseteq P_k$ potem
      $ocena = ocena - \frac{1}{n_{\subset}^{direct}}$
      $odvecnostPodLastnosti = odvecnostPodLastnosti \cup \{P_i, P_k\}$
    konec pogoja če
  konec sprehoda po vseh lastnostih
  vrni ***odvecnostPodLastnosti*** in ***ocena***
konec

Algoritem 18: Preveri odvečnost podlastnosti

Rezultat: Množica parov lastnosti (P, Q) z odvečno relacijo podlastnost in ocena indikatorja popolnosti ontologije, ki upošteva odvečnost podlastnosti.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxRed} \cdot k_{RBoxRedSubprop}$

Pomembnost v procesu ROD:

Slika 88: Pomembnost preverjanja odvečnosti podlastnosti (odvečnost komponente RBox) pri izračunu OC

Primer: Slika 89 prikazuje ontologijo s številnimi podlastnost relacijami, neposrednimi kot tudi posrednimi. Nekatere izmed njih kršijo pravilo odvečnosti podlastnosti in so ustrezno označene.

Slika 89: Primer preverjanja odvečnosti podlastnosti (odvečnost komponente RBox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednjo odvečno podlastnost relacijo: $\{se\ dnevno\ trguje\ na,\ se\ trguje\ ne\}$. Ocena testne ontologije, ki upošteva odvečnost podrelacije, je $1 - \frac{1}{5} = \frac{4}{5} = 0{,}800$.

3.3.3.4 Anomalija

3.3.3.4.1 Ponavljajoča se skupina lastnosti

Problem: Včasih načrtovalci ontologije slabo opredelijo ontologijo z uporabo iste skupine lastnosti pri opredelitvi različnih razredov. Ta skupina lastnosti se imenuje ponavljajoča se skupina lastnosti in jo je potrebno zamenjati z abstraktnim razredom, ki vsebuje vse omenjene lastnosti v vseh opredelitvah razredov, kjer se ta ponavljajoča skupina lastnosti uporablja.

Cilj: Preveriti, ali v ontologiji obstajajo ponavljajoče se skupine lastnosti, in omogočiti uporabniku rešitev problema – kreiranje novega razreda z omenjenimi lastnostmi in povezava le-tega na tiste razrede, ki uporabljajo ponavljajoče se skupine lastnosti. S tem se zmanjša število povezav (podatkovne in objektne lastnosti) med razredi. Posledica pa je večje število razredov, vendar preglednejša ontologija.

Postopek: Ontologija je predstavljena v obliki grafa $G = (V, E)$, kjer je V množica razredov in podatkovnih lastnosti ter E množica vseh povezav. Ločimo dve vrsti povezav: objektne

lastnosti, ki povezujejo dva razreda ali podatkovne lastnosti, ki povezujejo razred z enostavnim podatkovnim tipom. Pri identifikaciji ponavljajočih se skupin lastnosti iščemo popolne dvodelne podgrafe $G' = (V_1 + V_2, E)$, tako da za poljubni točki v grafu $v_1 \in V_1$ in $v_2 \in V_2$ velja, da je $v_1 v_2$ povezava v podgrafu G'. Popoln dvodelen graf z razdelkoma velikosti $|V_1| = m$ in $|V_2| = n$ označimo kot $K_{m,n}$. Rešitve, ki jih iščemo mi, so podgrafi $K'_{m,n}$, kjer velja $m \geq 2$ in $n \geq 2$.

poisciPonavljajocoSeSkupinoLastnosti
  $ocena = 1$
  n_R ← Število povezav (podatkovnih in objektnih lastnosti) v ontologiji
  V ← Oznake razredov ∪ oznake lastnosti
  E ← Podatkovne lastnosti ∪ objektne lastnosti
  $ponavljajoceSeSkupine = \emptyset$
  dokler obstaja popoln dvodelni podgraf $K'_{m,n}$ v grafu $G(V, E)$
    izberi najboljšega kandidate $K''_{m,n}$ iz množice $K'_{m,n}$, kjer velja $\left(\frac{m'' \cdot n''}{m'' + n''}\right)$
    $ponavljajoceSeSkupine = ponavljajoceSeSkupine \cup K''_{m,n}$
    odstrani vse povezave, ki se pojavljajo v $K''_{m,n}$ iz $G(V, E)$
    $ocena = ocena - \frac{m'' \cdot n'' - (m'' + n'')}{n_R}$
  konec pogoja
  vrni ***ponavljajoceSeSkupine*** in ***ocena***
konec

Algoritem 19: Poišči ponavljajočo se skupino lastnosti

Rezultat: Množica ponavljajočih skupin lastnosti v obliki (*razdelek z razredi, razdelek s povezavami*) in ocena indikatorja popolnosti ontologije, ki upošteva ponavljajoče se skupine lastnosti.

Globalna utež v procesu ROD: $w' = k_{RBox} \cdot k_{RBoxAnom} \cdot k_{RBoxAnomPropClump}$

Pomembnost v procesu ROD:

Slika 90: Pomembnost iskanja ponavljajoče se skupine lastnosti (anomalija komponente RBox) pri izračunu OC

Primer: Slika 91 prikazuje ontologijo z dvema ponavljajočima se skupinama lastnosti in je predstavljena v obliki dvodelnega grafa.

Slika 91: Primer iskanja ponavljajoče se skupine lastnosti (anomalija komponente RBox) pri izračunu OC v obliki popolnega dvodelnega grafa

Po izvedbi algoritma sta odkriti naslednji ponavljajoči se skupini lastnosti: {{*Oseba, Dolg*}, {*ulica, hišna številka, poštna številka*}} in {{*Dolg, Račun*}, {*vrednost, valuta*}}. Če upoštevamo predloge za odpravo ponavljajočih se skupin lastnosti, dobimo novo ontologijo (Slika 92). Ocena testne ontologije z upoštevanjem ponavljajočih se skupin lastnosti je $1 - \frac{2 \cdot 4 - (2+4)}{29} - \frac{3 \cdot 2 - (3+2)}{29} = \frac{26}{29} = 0{,}897$.

Slika 92: Primer odprave ponavljajoče se skupine lastnosti (anomalija komponente RBox) pri izračunu OC v obliki popolnega dvodelnega grafa

3.3.4 *KOMPONENTA ABOX*

3.3.4.1 Opis

3.3.4.1.1 Formalen opis

Problem: Primerki razredov in lastnosti morajo biti v skladu z aksiomi, ki so opredeljeni v shematskem delu ontologije. Načrtovalci ontologij pogosto izpustijo podrobnosti primerkov razredov, ki so drugače zahtevani, npr. kardinalnost ipd. Isto velja za primerke lastnosti, kjer pride do izpuščanja definicijskega območja in zaloge vrednosti.

Cilj: Poiskati vse primerke, ki kršijo pravila, zapisana z aksiomi, v shematskem delu ontologije. Uporabniku so na podlagi seznama kršitev na voljo akcije, ki jih lahko izvede za odpravo nepravilnosti.

Postopek: Za vsak primerek razreda in lastnosti preverimo, ali kršijo aksiome. V primeru primerka razreda preverimo vse lastnosti (podatkovne in lastnosti) za ustreznost kardinalnosti in zaloge vrednosti. Pri primerkih lastnosti pa preverjamo definicijsko območje in zalogo vrednosti.

preveriFormalenOpis
$ocena = 0$
n_i ← Število primerkov (razredov in primerkov) v ontologiji
$card(p)$ ← Kardinalnost lastnosti p (npr. [2,∞])
$range(p)$ ← Zaloga vrednosti lastnosti p
$domain(p)$ ← Definicijsko območje lastnosti p
$nepopolniPrimerki = \emptyset$
za vsak $i \in ABox$
  če je i primerek razreda potem
    C_i' je množica razredov, katere i je primerek
    P_i' je množica lastnosti, povezana z unijo razredov iz C_i'
    za vsako $p_{C_i'}(i) \in P_i'$
      $card\left(p_{C_i'}(i)\right)$ ← Kardinalnost lastnosti $p_{C_i'}(i)$
      $n\left(p_{C_i'}(i)\right)$ ← Število primerkov lastnosti $p_{C_i'}(i)$
      $i\left(p_{C_i'}(i)\right)$ ← Primerek lastnosti $p_{C_i'}(i)$
      če $\left(n\left(p_{C_i'}(i)\right) \in card\left(p_{C_i'}(i)\right)\right) \wedge \left(range\left(p_{C_i'}(i)\right) \cap range\left(i\left(p_{C_i'}(i)\right)\right) \neq \emptyset\right)$ potem
        $ocena = ocena + \frac{1}{|P_i'| \cdot n_i}$
      sicer
        $nepopolniPrimerki = nepopolniPrimerki \cup \left\{i, p_{C_i'}(i)\right\}$
      konec pogoja če
    konec sprehoda po lastnostih
  sicer če je i primerek lastnosti potem
    P_i' je množica lastnosti, katere i je primerek
    če $domain(P_i') \cap domain(i) = \emptyset$ potem
      $nepopolniPrimerki = nepopolniPrimerki \cup \{i, "definicijsko\ območje"\}$
    sicer če $range(P_i') \cap range(i) = \emptyset$ potem
      $nepopolniPrimerki = nepopolniPrimerki \cup \{i, "zaloga\ vrednosti"\}$
    sicer
      $ocena = ocena + \frac{1}{n_i}$
    konec pogoja če
  konec pogoja če
konec sprehoda po primerkih
vrni ***nepopolniPrimerki*** in ***ocena***
konec

Algoritem 20: Preveri formalen opis primerkov

Rezultat: Množica parov $\{primerek, vzrok\}$, kjer je za vsak primerek prikazan tudi vzrok nepopolnosti formalnega opisa, in ocena indikatorja popolnosti ontologije, ki upošteva formalen opis primerkov.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxDesc} \cdot k_{ABoxDescForm}$

Pomembnost v procesu ROD:

Slika 93: Pomembnost preverjanja formalnega opisa primerkov (opis komponente ABox) pri izračunu OC

Primer: Slika 94 prikazuje ontologijo z osmimi primerki (štirje primerki razredov in štirje primerki lastnosti). Testna ontologija vsebuje številne kršitve sheme, in sicer se nanašajo na kršitev kardinalnosti lastnosti.

Slika 94: Primer preverjanja formalnega opisa primerkov (opis komponente ABox) pri izračunu OC

Po izvedbi algoritma so odkrite naslednje kršitve formalnega opisa primerkov: $\{FXI, kotira\ na\}$, $\{PTR, ISIN\}$, $\{PTR, kotira\ na\}$, $\{CHA, ISIN\}$, $\{CHA, kotira\ na\}$, $\{LFX, ISIN\}$ in $\{LFX, kotira\ na\}$. Ocena testne ontologije z nepopolnim formalnim opisom primerkov je $\frac{4\cdot 1+\frac{4}{5}+3\cdot 0}{8} = \frac{3}{5} = 0{,}600$.

3.3.4.1.2 Obstoj primerka

3.3.4.1.2.1 Razred

Problem: Za vsak razred v ontologiji, ki ima tudi primerek, velja, da je takšen gradnik dejansko uporabljiv v aplikaciji. Če razred nima primerkov, potem ga lahko iz obstoječe ontologije odstranimo ali premaknemo v kakšno drugo.

Cilj: Poiskati razrede, ki nimajo primerkov, in omogočiti uporabniku, da razred brez primerkov odstrani ali ga premakne v drugo ontologijo.

Postopek: Za vsak razred preverimo, ali ima tudi primerke.

preveriObstojPrimerkovRazreda
  $ocena = 1$
  n_C ← Število razredov v ontologiji
  $manjkajociPrimerki = \emptyset$
  za vsak $C_i \in TBox$
    če $\nexists instance(C_i)$ potem
      $ocena = ocena - \frac{1}{n_C}$
      $manjkajociPrimerki = manjkajociPrimerki \cup C_i$
    konec pogoja če
  konec sprehoda po vseh razredih
  vrni ***manjkajociPrimerki*** in ***ocena***
konec

Algoritem 21: Preveri obstoj primerkov razreda

Rezultat: Množica razredov brez primerkov in ocena indikatorja popolnosti ontologije, ki upošteva obstoj primerkov razreda.

Globalna utež v procesu ROD:
$w' = k_{ABox} \cdot k_{ABoxDesc} \cdot k_{ABoxDescInstExist} \cdot k_{ABoxDescInstExistClass}$

Pomembnost v procesu ROD:

Slika 95: Pomembnost preverjanja obstoja primerkov razreda (opis komponente ABox) pri izračunu OC

Primer: Slika 96 prikazuje ontologijo s štirimi razredi in štirimi primerki razredov.

Slika 96: Primer preverjanje obstoja primerkov razreda (opis komponente ABox) pri izračunu OC

Po izvedbi algoritma identificiramo naslednje razrede z manjkajočimi primerki: $\{Trgovalni\ dan,\ Borza\}$. Ocena testne ontologije z manjkajočimi primerki razreda je $1 - \frac{1}{4} - \frac{1}{4} = \frac{2}{4} = 0{,}500$.

3.3.4.1.2.2 Lastnost

Problem: Za vsako lastnost v ontologiji, ki ima tudi primerek, velja, da je takšen gradnik dejansko uporabljiv v aplikaciji. Če lastnost nima primerkov, potem jo lahko iz obstoječe ontologije odstranimo ali premaknemo v kakšno drugo.

Cilj: Poiskati lastnosti, ki nimajo primerkov in omogočiti uporabniku, da lastnost brez primerkov odstrani ali jo premakne v drugo ontologijo.

Postopek: Za vsako lastnost preverimo, ali ima tudi primerke.

preveriObstojPrimerkovLastnosti
  $ocena = 1$
  n_P ← Število lastnosti v ontologiji
  $manjkajociPrimerki = \emptyset$
  za vsako $P_i \in RBox$
    če $\nexists instance(P_i)$ potem
      $ocena = ocena - \frac{1}{n_P}$
      $manjkajociPrimerki = manjkajociPrimerki \cup P_i$
    konec pogoja če
  konec sprehoda po vseh lastnostih
  vrni $\boldsymbol{manjkajociPrimerki}$ in $\boldsymbol{ocena}$
konec

Algoritem 22: Preveri obstoj primerkov lastnosti

Rezultat: Množica lastnosti brez primerkov in ocena indikatorja popolnosti ontologije, ki upošteva obstoj primerkov lastnosti.

Globalna utež v procesu ROD:
$w' = k_{ABox} \cdot k_{ABoxDesc} \cdot k_{ABoxDescInstExist} \cdot k_{ABoxDescInstExistProp}$

Pomembnost v procesu ROD:

Slika 97: Pomembnost preverjanja obstoja primerkov lastnosti (opis komponente ABox) pri izračunu OC

Primer: Slika 98 prikazuje ontologijo s štirimi lastnostmi in dvema primerkoma lastnosti.

Slika 98: Primer preverjanja obstoja primerkov lastnosti (opis komponente ABox) pri izračunu OC

Po izvedbi algoritma identificiramo naslednje lastnosti z manjkajočimi primerki: $\{trguje\ na,\ je\ del\}$. Ocena testne ontologije z manjkajočimi primerki lastnosti je $1 - \frac{1}{4} - \frac{1}{4} = \frac{2}{4} = 0{,}500$.

3.3.4.2 Razdelitev

3.3.4.2.1 Skupni primerek

Problem: Pri kreiranju primerkov razredov se pogosto zgodi, da določen primerek pripada razredom, ki so razdruženi. V tem primeru pride do napake skupnih primerkov.

Cilj: Identificirati skupne primerke in pripadajoče razrede ter omogočiti uporabniku spremembo ontologije v smeri odprave pripadnosti razredom, ki so razdruženi.

Postopek: Za vsak primerek I_i se preveri, ali v množici razredov $C(I_i)$, kjer so razredi, katerih I_i je neposredni primerek, obstajata razreda C_m in C_n, ki sta razdružena.

preveriSkupnePrimerke
$ocena = 1$
n_{inst}^{direkt} ← Število neposrednih primerkov v ontologiji
$C(I)$ ← Množica razredov, katerih I je neposredni primerek
$C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
$C_1 \cap C_2 = \emptyset$ ← C_1 je razdružen s C_2
$skupniPrimerki = \emptyset$
za vsak $I_i \in ABox$
  če $\exists C_j, C_k \in C(I_i) \wedge C_m, C_n \in TBox : C_j \neq C_k \wedge C_m \neq C_n \wedge C_j \subseteq C_m \wedge C_k \subseteq C_n \wedge C_m \cap C_n = \emptyset$ potem
    $ocena = ocena - \frac{1}{n_{inst}^{direkt}}$
    $skupniPrimerki = skupniPrimerki \cup \{I_i, C_m, C_n\}$
  konec pogoja če
konec sprehoda po vseh primerkih
vrni ***skupniPrimerki*** in ***ocena***
konec

Algoritem 23: Preveri skupne primerke

Rezultat: Množica trojic $\{primerek, 1.razdružen\ razred, 2.razdružen\ razred\}$, ki predstavlja napako skupnega primerka in ocena indikatorja popolnosti ontologije, ki upošteva skupne primerke.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartComInst}$

Pomembnost v procesu ROD:

Slika 99: Pomembnost preverjanja skupnih primerkov (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 100 prikazuje ontologijo, ki vsebuje razrede, njihovo medsebojno povezanost z relacijami podrazred, primerke razredov in številne napake skupnih primerkov.

Po izvedbi algoritma dobimo seznam naslednjih skupnih primerkov: {*DEP301, Depozit, Posojilo*}, {*BRENT, Obrestni inštrument, Izvedeni inštrument na blago*} in {*FXI, Delnica, Lastniški izveden instrument*}. Ocena testne ontologije z napakami skupnih primerkov je $1 - \frac{1}{9} - \frac{1}{9} - \frac{1}{9} = \frac{6}{9} = 0{,}667$.

Slika 100: Primer preverjanja skupnih primerkov (razdelitev komponente ABox) pri izračunu OC

3.3.4.2.2 Zunanji primerek

Problem: Pri dekompoziciji razredov v podrazrede se pogosto zgodi, da primerek razreda ne pripada nobenemu od podrazredov. V tem primeru pride do napake zunanjih primerkov.

Cilj: Identificirati razrede, kjer prihaja do zunanjih primerkov in omogočiti uporabniku spremembo ontologije v smeri drugačne taksonomije razredov ali odstranitve primerkov.

Postopek: Za vsak razred C_i preverimo, če obstaja kakšen primerek, ki ne obstaja v množici primerkov podrazredov.

preveriZunanjePrimerke
n_c ← Število razredov v ontologiji
n_{inst}^{direkt} ← Število neposrednih primerkov v ontologiji
$subClass(C)$ ← Množica podrazredov razreda C
$zunanjiPrimerki = \emptyset$
za vsak $C_i \in TBox$
  če $\exists i(C_i) \in ABox : \left(\nexists i(C_j) \in ABox : C_j \in subClass(C_i)\right)$ potem
    $zunanjiPrimerki = zunanjiPrimerki \cup \{C_i\}$
  konec pogoja če
konec sprehoda po vseh razredih
$ocena = ocena - \frac{1}{n_{inst}^{direkt}}$
vrni $\boldsymbol{zunanjiPrimerki}$ in $\boldsymbol{ocena}$
konec

Algoritem 24: Preveri zunanje primerke

Rezultat: Množica razredov, kjer je prišlo do kršitve zunanjih primerkov, in ocena indikatorja popolnosti ontologije, ki upošteva zunanje primerke.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartExtInst}$

Pomembnost v procesu ROD:

Slika 101: Pomembnost preverjanja zunanjih primerkov (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 102 prikazuje enostavno ontologijo, v kateri so prisotne napake zunanjih primerkov.

Slika 102: Primer preverjanja zunanjih primerkov (razdelitev komponente ABox) pri izračunu OC

Po izvedbi algoritma dobimo naslednji seznam zunanjih primerkov: $\{Finančni\ inštrument\}$. Ocena testne ontologije z napakami zunanjih primerkov je $1 - \frac{1}{3} = \frac{2}{3} = 0{,}667$. Če na podlagi omenjenih rezultatov poskušamo odpraviti to napako, je ena od možnih rešitev prikazana na naslednjem primeru (Slika 103).

Slika 103: Primer odpravljenega problema pri preverjanju zunanjih primerkov (razdelitev komponente ABox) pri izračunu OC

Ocena testne ontologije z odpravljeno napako zunanjih primerkov je tako $1 - \frac{0}{4} = 1$.

3.3.4.2.3 Hierarhija

3.3.4.2.3.1 Razred

3.3.4.2.3.1.1 Podrazred

Problem: Tehnično neizobraženi uporabniki imajo pogosto težave pri opredelitvi taksonomije razredov brez dodatne pomoči. Če uporabniku pri tem pomagamo s klasifikacijo vnesenih primerkov razredov, se takšno opravilo močno poenostavi, saj mora uporabnik le vnesti primerke posameznih razredov in odgovoriti na nekaj enostavnih vsebinskih vprašanj.

Cilj: S podatkovno usmerjenim pristopom na podlagi primerkov razredov predlagati relacije podrazred med posameznimi razredi. Uporabnik lahko spremeni razredno taksonomijo z vnašanjem primerkov razredov.

Postopek: Pari razredov (C, D) so identificirani kot potencialni kandidati za podrazrede, če vsebujejo skupne primerke.

Slika 104: Vsebinska vprašanja za identifikacijo relacij podrazred

Po identifikaciji omenjenih parov razredov se uporabniku postavijo vsebinska vprašanja Q_1, Q_2 in Q_3 (Slika 104) in glede na odgovore so možni naslednji scenariji (Slika 105):

$$Q_1(C,D) \wedge \neg Q_2(C,D) \wedge \neg Q_3(C,D) \rightarrow C \subseteq D$$

$$\neg Q_1(C,D) \wedge \neg Q_2(C,D) \wedge Q_3(C,D) \rightarrow D \subseteq C$$

Slika 105: Različni scenariji pri identifikaciji relacij tipa podrazred

Ocena indikatorja popolnosti ontologije temelji na številu relacij podrazred, ki so predlagane, a v ontologiji ne obstajajo.

predlagajRelacijePodRazred
  $equal(i_1, i_2) = \{i_1, i_2 \in ABox, \exists lang : label_{lang}(i_1) = label_{lang}(i_2)\}$ ← Enakost primerkov i_1 in i_2
  $C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
  $n_m = 0$ ← Število manjkajočih relacij podrazred v ontologiji
  $podRazredPriporocila = \emptyset$
  za vsak $C_i \in TBox$
    za vsak $C_j \in TBox$, kjer velja $C_i \neq C_j$
      če $\exists i(C_i), i(C_j) \in ABox : equal\left(i(C_i), i(C_j)\right)$ potem
        če $Q_1(C_i, C_j) \wedge \neg Q_2(C_i, C_j) \wedge \neg Q_3(C_i, C_j)$ potem
          $podRazredPriporocila = podRazredPriporocila \cup (C_i, C_j)$
          če $C_i \nsubseteq C_j$ potem
            $n_m = n_m + 1$
          konec pogoja če
        sicer če $\neg Q_1(C_i, C_j) \wedge \neg Q_2(C_i, C_j) \wedge Q_3(C_i, C_j)$ potem
          $podRazredPriporocila = podRazredPriporocila \cup (C_j, C_i)$
          če $C_j \nsubseteq C_i$ potem
            $n_m = n_m + 1$
          konec pogoja če
        konec pogoja če
      konec pogoja če
    konec sprehoda po vseh razredih
  konec sprehoda po vseh razredih
  $ocena = 1 - \frac{n_m}{|podRazredPriporocila|}$
  vrni $\boldsymbol{podRazredPriporocila}$ in $\boldsymbol{ocena}$
konec

Algoritem 25: Predlagaj relacije podrazred

Rezultat: Množica parov razredov (C, D), kjer se predlaga relacija podrazred $C \subseteq D$ in ocena indikatorja popolnosti ontologije, ki upošteva predlagane relacije podrazred.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartHier} \cdot k_{ABoxPartHierClass} \cdot k_{ABoxPartHierClassSub}$

Pomembnost v procesu ROD:

Slika 106: Pomembnost predlaganja relacij podrazred (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 107 prikazuje proces predlaganja relacij podrazred na podlagi primerkov razredov. Začnemo z razredi Finančni inštrument, Delnica in ETF ter opredeljeno relacijo podrazred med Delnica in Finančni inštrument. Glede na odgovore vsebinskih vprašanj je prikazan tudi končni rezultat.

Slika 107: Primer predlaganja relacij podrazred (razdelitev komponente ABox) pri izračunu OC

Po izvedbi algoritma in posredovanih odgovorih na vsebinska vprašanja dobimo naslednji relaciji podrazred: $\{Delnica, Finančni\ inštrument\}$ in $\{Finančni\ inštrument, ETF\}$. Ocena testne ontologije s predlaganimi relacijami podrazred je $1 - \frac{1}{2} = \frac{1}{2} = 0{,}500$.

3.3.4.2.3.1.2 Ekvivalenten razred

Problem: Tehnično neizobraženi uporabniki imajo pogosto težave pri opredelitvi razredne taksonomije brez dodatne pomoči, še posebej pri združevanju več ontologij skupaj. Če uporabniku pri tem pomagamo s klasifikacijo vnesenih primerkov razredov, se takšno opravilo močno poenostavi, saj mora uporabnik le vnesti primerke posameznih razredov in odgovoriti na nekaj enostavnih vsebinskih vprašanj.

Cilj: S podatkovno usmerjenim pristopom na podlagi primerkov razredov predlagati ekvivalentne relacije med posameznimi razredi.

Postopek: Pari razredov (C, D) so identificirani kot potencialni kandidati za ekvivalentne razrede, če vsebujejo skupne primerke. Po identifikacij omenjenih parov razredov se uporabniku postavijo vsebinska vprašanja Q_1, Q_2 in Q_3 (Slika 104) in glede na odgovore je možen naslednji scenarij (Slika 108):

$$Q_1(C, D) \wedge Q_2(C, D) \wedge Q_3(C, D) \rightarrow C \equiv D$$

Slika 108: Scenarij pri identifikaciji relacije tipa ekvivalentni razred

Ocena indikatorja popolnosti ontologije temelji na številu relacij ekvivalentni razred, ki so predlagane, a v ontologiji ne obstajajo.

predlagajRelacijeEkvivalentenRazred
$equal(i_1, i_2) = \{i_1, i_2 \in ABox, \exists lang : label_{lang}(i_1) = label_{lang}(i_2)\}$ ← Enakost primerkov i_1 in i_2
$C_1 \equiv C_2$ ← C_1 je ekvivalenten C_2
$n_m = 0$ ← Število manjkajočih relacij ekvivalenten razred v ontologiji
$ekvivalentenRazredPriporocila = \emptyset$
za vsak $C_i \in TBox$
  za vsak $C_j \in TBox$, kjer velja $C_i \neq C_j$
    če $\exists i(C_i), i(C_j) \in ABox : equal\left(i(C_i), i(C_j)\right)$ potem
      če $Q_1\left(C_i, C_j\right) \wedge Q_2\left(C_i, C_j\right) \wedge Q_3\left(C_i, C_j\right)$ potem
        $ekvivalentenRazredPriporocila = ekvivalentenRazredPriporocila \cup \left(C_i, C_j\right)$
        če $C_i \not\equiv C_j$ potem
          $n_m = n_m + 1$
        konec pogoja če
      konec pogoja če
    konec pogoja če
  konec sprehoda po vseh razredih
konec sprehoda po vseh razredih
$ocena = 1 - \frac{n_m}{|ekvivalentenRazredPriporocila|}$
vrni $\boldsymbol{ekvivalentenRazredPriporocila}$ in $\boldsymbol{ocena}$
konec

Algoritem 26: Predlagaj relacije ekvivalentni razred

Rezultat: Množica parov razredov (C, D), kjer se predlaga relacija ekvivalentni razred $C \equiv D$ in ocena indikatorja popolnosti ontologije, ki upošteva predlagane relacije ekvivalenten razred.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartHier} \cdot k_{ABoxPartHierClass} \cdot k_{ABoxPartHierClassEquiv}$

Pomembnost v procesu ROD:

Slika 109: Pomembnost predlaganja relacij ekvivalentni razred (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 110 prikazuje proces predlaganja relacij ekvivalentni razred na podlagi primerkov razredov. Začnemo z razredi Finančni inštrument, Delnica in ETF ter opredeljeno relacijo podrazred med ETF in Finančni inštrument. Glede na odgovore vsebinskih vprašanj je prikazan tudi končni rezultat.

Slika 110: Primer predlaganja relacij ekvivalentni razred (razdelitev komponente ABox) pri izračunu OC

Po izvedbi algoritma in posredovanih odgovorih na vsebinska vprašanja dobimo naslednjo relacijo ekvivalenten razred: $\{Delnica, Finančni\ inštrument\}$. Ocena testne ontologije s predlaganimi relacijami ekvivalenten razred je $1 - \frac{1}{1} = 0$.

3.3.4.2.3.1.3 Razdružen razred

Problem: Tehnično neizobraženi uporabniki imajo pogosto težave pri opredelitvi razdruženih razredov brez dodatne pomoči, še posebej pri večkratnih podrazredih določenega razreda. Če uporabniku pri tem pomagamo s klasifikacijo vnesenih primerkov razredov, se takšno opravilo močno poenostavi, saj mora uporabnik le vnesti primerke posameznih razredov in odgovoriti na nekaj enostavnih vsebinskih vprašanj.

Cilj: S podatkovno usmerjenim pristopom na podlagi primerkov razredov predlagati relacije razdruženi razred med posameznimi razredi. Uporabnik lahko spremeni razredno taksonomijo z vnašanjem primerkov razredov.

Postopek: Pari razredov (C, D), ki imajo skupni nadrazred so identificirani kot potencialni kandidati za razdružene razrede, če vsebujejo skupne primerke. Po identifikaciji omenjenih parov razredov se uporabniku postavijo vsebinska vprašanja Q_1, Q_2 in Q_3 (Slika 104) in glede na odgovore je možen naslednji scenarij (Slika 111):

$$\neg Q_1(C,D) \wedge \neg Q_2(C,D) \wedge \neg Q_3(C,D) \rightarrow C \cap D = \emptyset$$

Slika 111: Scenarij pri identifikaciji relacij tipa razdružen razred

Ocena indikatorja popolnosti ontologije temelji na številu vseh razdruženih parov podrazredov, ki so predlagani, a v ontologiji ne obstajajo.

predlagajRelacijeRazdruženRazred
$C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
$\tau_{\subseteq}^{sibling} = \emptyset$ ← Množica vseh parov podrazredov (C, D) v ontologiji
$razdruzenRazredPriporocila = \emptyset$
za vsak $C_i \in TBox$
  dodaj vse pare podrazredov razreda C_i v $\tau_{\subseteq}^{sibling}$
  za vsak par podrazredov $(C_j, C_k) \in TBox$, kjer velja $C_j \subseteq C_i \wedge C_k \subseteq C_i \wedge C_j \neq C_k$
    če $\exists i(C_j), i(C_k) \in ABox : \left(\neg Q_1(C_j, C_k) \wedge \neg Q_2(C_j, C_k) \wedge \neg Q_3(C_j, C_k)\right)$ potem
      če $C_j \cap C_k \neq \emptyset$ potem
        $razdruzenRazredPriporocila = razdruzenRazredPriporocila \cup (C_j, C_k)$
      konec pogoja če
    konec pogoja če
  konec sprehoda po parih podrazredov
konec sprehoda po vseh razredih
$ocena = 1 - \frac{|razdruzenRazredPriporocila|}{|\tau_{\subseteq}^{sibling}|}$
vrni $\boldsymbol{razdruzenRazredPriporocila}$ in $\boldsymbol{ocena}$
konec

Algoritem 27: Predlagaj relacije razdružen razred

Rezultat: Množica parov razredov (C, D), kjer se predlaga relacija razdružen razred $C \cap D \neq \emptyset$ in ocena indikatorja popolnosti ontologije, ki upošteva predlagane relacije razdružen razred.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartHier} \cdot k_{ABoxPartHierClass} \cdot k_{ABoxPartHierClassDis}$

Pomembnost v procesu ROD:

Slika 112: Pomembnost predlaganja relacij razdružen razred (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 113 prikazuje proces predlaganja relacij razdružen razred na podlagi primerkov razredov. Začnemo z razredi Finančni inštrument, Delnica in ETF. Glede na odgovore vsebinskih vprašanj je prikazan tudi končni rezultat.

Slika 113: Primer predlaganja relacij razdružen razred (razdelitev komponente ABox) pri izračunu OC

Po izvedbi algoritma in posredovanih odgovorih na vsebinska vprašanja dobimo naslednjo relacijo razdružen razred: $\{Delnica, ETF\}$. Ocena testne ontologije s predlaganimi relacijami razdružen razred je $1 - \frac{1}{1} = 0$.

3.3.4.2.3.2 Lastnost

3.3.4.2.3.2.1 Podlastnost

Problem: Tehnično neizobraženi uporabniki imajo pogosto težave pri opredelitvi taksonomije lastnosti brez dodatne pomoči. Če uporabniku pri tem pomagamo s klasifikacijo vnesenih primerkov lastnosti, se takšno opravilo močno poenostavi, saj mora uporabnik le vnesti primerke posameznih lastnosti in odgovoriti na nekaj enostavnih vsebinskih vprašanj.

Cilj: S podatkovno usmerjenim pristopom na podlagi primerkov lastnosti predlagati relacije podlastnost med posameznimi lastnostmi. Uporabnik lahko spremeni taksonomijo lastnosti z vnašanjem primerkov lastnosti.

Postopek: Pari lastnosti (P, Q) so identificirani kot potencialni kandidati za podlastnosti, če vsebujejo skupne primerke.

Slika 114: Vsebinska vprašanja za identifikacijo relacij podlastnost

Po identifikaciji omenjenih parov lastnosti se uporabniku postavijo vsebinska vprašanja Q_1, Q_2 in Q_3 (Slika 114) in glede na odgovore so možni naslednji scenariji (Slika 115):

$$Q_1(P,Q) \wedge \neg Q_2(P,Q) \wedge \neg Q_3(P,Q) \rightarrow P \subseteq Q$$

$$\neg Q_1(P,Q) \wedge \neg Q_2(P,Q) \wedge Q_3(P,Q) \rightarrow Q \subseteq P$$

Slika 115: Različni scenariji pri identifikaciji relacij tipa podlastnost

Ocena indikatorja popolnosti ontologije temelji na številu relacij podlastnost, ki so predlagane, a v ontologiji ne obstajajo.

predlagajRelacijePodLastnost
$equal(i_1, i_2) = \{i_1, i_2 \in ABox, \exists lang : label_{lang}(i_1) = label_{lang}(i_2)\}$ ← Enakost primerkov i_1 in i_2
$P_1 \subseteq P_2$ ← P_1 je podlastnost od P_2
$n_m = 0$ ← Število manjkajočih relacij podlastnost v ontologiji
$podLastnostPriporocila = \emptyset$
za vsako $P_i \in RBox$
  za vsako $P_j \in RBox$, kjer velja $P_i \neq P_j$
    če $\exists i(P_i), i(P_j) \in ABox : equal\left(i(P_i), i(P_j)\right)$ potem
      če $Q_1(P_i, P_j) \wedge \neg Q_2(P_i, P_j) \wedge \neg Q_3(P_i, P_j)$ potem
        $podLastnostPriporocila = podLastnostPriporocila \cup (P_i, P_j)$
        če $P_i \nsubseteq P_j$ potem
          $n_m = n_m + 1$
        konec pogoja če
      sicer če $\neg Q_1(P_i, P_j) \wedge \neg Q_2(P_i, P_j) \wedge Q_3(P_i, P_j)$ potem
        $podLastnostPriporocila = podLastnostPriporocila \cup (P_j, P_i)$
        če $P_j \nsubseteq P_i$ potem
          $n_m = n_m + 1$
        konec pogoja če
      konec pogoja če
    konec pogoja če
  konec sprehoda po vseh lastnostih
konec sprehoda po vseh lastnostih
$ocena = 1 - \frac{n_m}{|podLastnostPriporocila|}$
vrni $\boldsymbol{podLastnostPriporocila}$ in $\boldsymbol{ocena}$
konec

Algoritem 28: Predlagaj relacije podlastnost

Rezultat: Množica parov lastnosti (P, Q), kjer se predlaga relacija podlastnost $P \subseteq Q$ in ocena indikatorja popolnosti ontologije, ki upošteva predlagane relacije podlastnost.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartHier} \cdot k_{ABoxPartHierProp} \cdot k_{ABoxPartHierPropSub}$

Pomembnost v procesu ROD:

Slika 116: Pomembnost predlaganja relacij podlastnost (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 117 prikazuje proces predlaganja relacij podlastnost na podlagi primerkov lastnosti. Začnemo z lastnostmi se trguje na, se redko trguje na in se pogosto trguje na. Glede na odgovore vsebinskih vprašanj je prikazan tudi končni rezultat.

Slika 117: Primer predlaganja relacij podlastnost (razdelitev komponente ABox) pri izračunu OC

Po izvedbi algoritma in posredovanih odgovorih na vsebinska vprašanja dobimo naslednji relaciji podlastnost: {*se redko trguje na, se trguje na*} in {*se pogosto trguje na, se trguje na*}. Ocena testne ontologije s predlaganimi relacijami podlastnost je $1 - \frac{1}{2} = \frac{1}{2} = 0{,}500$.

3.3.4.2.3.2.2 Ekvivalentna lastnost

Problem: Tehnično neizobraženi uporabniki imajo pogosto težave pri opredelitvi taksonomije lastnosti brez dodatne pomoči, še posebej pri združevanju več ontologij skupaj. Če uporabniku pri tem pomagamo s klasifikacijo vnesenih primerkov lastnosti, se takšno opravilo močno poenostavi, saj mora uporabnik le vnesti primerke posameznih lastnosti in odgovoriti na nekaj enostavnih vsebinskih vprašanj.

Cilj: S podatkovno usmerjenim pristopom na podlagi primerkov lastnosti predlagati ekvivalentne relacije med posameznimi razredi.

Postopek: Pari lastnosti (P, Q) so identificirani kot potencialni kandidati za ekvivalentne lastnosti, če vsebujejo skupne primerke. Po identifikaciji omenjenih parov lastnosti se uporabniku postavijo vsebinska vprašanja Q_1, Q_2 in Q_3 (Slika 114) in glede na odgovore je možen naslednji scenarij (Slika 118):

$$Q_1(P,Q) \wedge Q_2(P,Q) \wedge Q_3(P,Q) \rightarrow P \equiv Q$$

Slika 118: Scenarij pri identifikaciji relacije tipa ekvivalentna lastnost

Ocena indikatorja popolnosti ontologije temelji na številu relacij ekvivalentna lastnost, ki so predlagane, a v ontologiji ne obstajajo.

predlagajRelacijeEkvivalentnaLastnost
$equal(i_1, i_2) = \{i_1, i_2 \in ABox, \exists lang : label_{lang}(i_1) = label_{lang}(i_2)\}$ ← Enakost primerkov i_1 in i_2
$P_1 \equiv P_2$ ← P_1 je ekvivalentna P_2
$n_m = 0$ ← Število manjkajočih relacij ekvivalentna lastnost v ontologiji
$ekvivalentnaLastnostPriporocila = \emptyset$
za vsako $P_i \in RBox$
  za vsako $P_j \in RBox$, kjer velja $P_i \neq P_j$
    če $\exists i(P_i), i(P_j) \in ABox : equal\left(i(P_i), i(P_j)\right)$ potem
      če $Q_1\left(P_i, P_j\right) \wedge Q_2\left(P_i, P_j\right) \wedge Q_3\left(P_i, P_j\right)$ potem
        $ekvivalentnaLastnostPriporocila = ekvivalentnaLastnostPriporocila \cup \left(P_i, P_j\right)$
        če $P_i \not\equiv P_j$ potem
          $n_m = n_m + 1$
        konec pogoja če
      konec pogoja če
    konec pogoja če
  konec sprehoda po vseh lastnostih
konec sprehoda po vseh lastnostih
$ocena = 1 - \frac{n_m}{|ekvivalentnaLastnostPriporocila|}$
vrni $\boldsymbol{ekvivalentnaLastnostPriporocila}$ in $\boldsymbol{ocena}$
konec

Algoritem 29: Predlagaj relacije ekvivalentna lastnost

Rezultat: Množica lastnosti (P, Q), kjer se predlaga relacija ekvivalentna lastnost $P \equiv Q$ in ocena indikatorja popolnosti ontologije, ki upošteva predlagane relacije ekvivalentna lastnost.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxPart} \cdot k_{ABoxPartHier} \cdot k_{ABoxPartHierProp} \cdot k_{ABoxPartHierPropEquiv}$

Pomembnost v procesu ROD:

Slika 119: Pomembnost predlaganja relacij ekvivalentna lastnost (razdelitev komponente ABox) pri izračunu OC

Primer: Slika 120 prikazuje proces predlaganja relacij ekvivalentna lastnost na podlagi primerkov lastnosti. Začnemo z lastnostmi se trguje na, se redko trguje na in se pogosto trguje na ter opredeljeno relacijo podlastnost med se trguje na in se pogosto trguje na. Glede na odgovore vsebinskih vprašanj je prikazan tudi končni rezultat.

Slika 120: Primer predlaganja relacij ekvivalentna lastnost (razdelitev komponente ABox) pri izračunu OC

Po izvedbi algoritma in posredovanih odgovorih na vsebinska vprašanja dobimo naslednjo relacijo ekvivalentna lastnost: $\{se\ redko\ trguje\ na,\ se\ trguje\ na\}$. Ocena testne ontologije s predlaganimi relacijami ekvivalentna lastnost je $1 - \frac{1}{1} = 0$.

3.3.4.3 Odvečnost

3.3.4.3.1 Identična opredelitev

Problem: Pri kreiranju primerkov pogosto pride do težave različnih primerkov, ki imajo identično formalno opredelitev, vendar različna imena. Dejstvo je, da gre za isti pojem, vendar drugače poimenovan (t. i. sinonim).

Cilj: Preveriti, ali v ontologiji obstajajo primerki z identično formalno opredelitvijo, in omogočiti uporabniku rešitev problema – uvedba identičnih primerkov, združevanje primerkov ali odstranitev odvečnih primerkov.

Postopek: Potrebno je preveriti vsak primerek razreda $i(C)$ in vsak primerek lastnosti $i(P)$. Pri primerkih razredov se preverja enakost množice vseh primerkov lastnosti, povezanih s primerkom razreda in enakost množice razredov, katerim primerek pripada. Pri primerkih lastnosti pa se primerja množica definicijskih območij, množica zalog vrednosti in množica lastnosti, katerim primerek pripada.

preveriIdenticnoOpredelitevPrimerka
$ocena = 1$
$n_{i(C)}$ ← Število razredov v ontologiji
$n_{i(P)}$ ← Število lastnosti v ontologiji
$i(C)$ ← Primerek razreda
$i(P)$ ← Primerek lastnosti
$P(i(C))$ ← Množica vseh primerkov lastnosti (opisne, podatkovne in objektne), povezanih s primerkov razreda $i(C)$
$superClass(i(C))$ ← Množica razredov, katerih $i(C)$ je primerek
$superProperty(i(P))$ ← Množica lastnosti, katerih $i(P)$ je primerek
$domain(i(P))$ ← Definicijsko območje primerka lastnosti $i(P)$
$range(i(P))$ ← Zaloga vrednosti primerka lastnosti $i(P)$
$identicneOpredelitve = \emptyset$
za vsak $i_x(C) \in ABox$
  če $\exists i_y(C)$, kjer velja $i_x(C) \neq i_y(C)$, $P(i_x(C)) = P\left(i_y(C)\right)$ in $superClass(i_x(C)) = superClass\left(i_y(C)\right)$
potem
    $ocena = ocena - \frac{1}{2\cdot(n_{i(C)}-1)\cdot n_{i(C)}}$
    $identicneOpredelitve = identicneOpredelitve \cup \{"primerek\ razreda", i_x(C), i_y(C)\}$
  konec pogoja če
konec sprehoda po vseh primerkih razredov
za vsak $i_z(P) \in ABox$
  če $\exists i_w(P)$, potem velja $i_z(P) \neq i_w(P)$, $domain(i_z(P)) = domain(i_z(P))$, $range(i_z(P)) = range(i_z(P))$ in
    $superClass(i_x(C)) = superClass\left(i_y(C)\right)$ potem
    $ocena = ocena - \frac{1}{2\cdot(n_{i(P)}-1)\cdot n_{i(P)}}$
    $identicneOpredelitve = identicneOpredelitve \cup \{"primerek\ lastnosti", i_z(C), i_w(C)\}$
  konec pogoja če
konec sprehoda po vseh primerkih lastnosti
vrni $\boldsymbol{identicneOpredelitve}$ in $\boldsymbol{ocena}$
konec

Algoritem 30: Preveri identično opredelitev primerka

Rezultat: Množica trojic opisa in primerkov (tip, I, J) z identično opredelitvijo in različnimi imeni ter ocena indikatorja popolnosti ontologije, ki upošteva identično opredelitev primerka.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxRed} \cdot k_{ABoxRedIdent}$

Pomembnost v procesu ROD:

Slika 121: Pomembnost preverjanja identične opredelitve primerka (odvečnost komponente ABox) pri izračunu OC

Primer: Slika 122 prikazuje ontologijo, ki vsebuje primerek z identično formalno opredelitvijo in različnima imenoma.

Slika 122: Primer preverjanja identične opredelitve primerka (odvečnost komponente ABox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednjo identično opredelitev primerka: {primerek razreda, $AAPL, Apple\ Inc.$}. Ocena testne ontologije, ki upošteva identično opredelitev primerka, je $1 - \frac{1}{2 \cdot 3 \cdot 2} = \frac{11}{12} = 0{,}917$.

3.3.4.3.2 Odvečnost primerka

Problem: Do odvečnosti relacije pripadnosti razredu pride takrat, ko načrtovalec ontologije opredeli primerek I kot primerek razredov C_1 in C_2, kjer velja, da je C_1 podrazred razredu C_2. Neposredna pripadnost primerka I razredu C_2 je odvečna, saj velja posredna pripadnost razredu C_2 preko relacije podrazred med razredoma C_1 in C_2. Isto velja za primerke lastnosti.

Cilj: Preveriti, ali v ontologiji obstajajo odvečne relacije pripadnosti razredu ali lastnosti in omogočiti uporabniku rešitev problema – odstranitev odvečnih relacij.

Postopek: Za vsak primerek (razreda ali lastnosti) preverimo, ali vsebuje večkratne pripadnosti razredu ali lastnosti, ki so odvečne. Če v primeru obstoja posredne povezave kljub temu obstaja še neposredna, se predlaga odstranitev neposredne.

preveriOdvecnostPrimerka
  $ocena = 1$
  n_i ← Število vseh primerkov (razredov in lastnosti) v ontologiji
  $C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
  $P_1 \subseteq P_2$ ← P_1 je podlastnost od P_2
  $odvecnostPrimerkov = \emptyset$
  za vsak $i \in ABox$
    če je i primerek razreda
      C_i' ← Množica razredov, katerih i je primerek
      za vsak $c_x \in C_i'$
        če $\exists c_y \in C_i: c_x \neq c_y \wedge c_y \subset c_x$ potem
          $ocena = ocena - \frac{1}{|C_i| \cdot n_i}$
          $odvecnostPrimerkov = odvecnostPrimerkov \cup \{i, c_x\}$
        konec pogoja če
      konec sprehoda po razredih
    sicer če je i primerek lastnosti
      P_i' ← Množica lastnosti, katerih i je primerek
      za vsako $p_x \in P_i'$
        če $\exists p_y \in P_i: p_x \neq p_y \wedge p_y \subset p_x$ potem
          $ocena = ocena - \frac{1}{|P_i| \cdot n_i}$
          $odvecnostPrimerkov = odvecnostPrimerkov \cup \{i, p_x\}$
        konec pogoja če
      konec sprehoda po lastnostih
    konec pogoja če
  konec sprehoda po vseh primerkih
  vrni $\boldsymbol{odvecnostPrimerkov}$ in $\boldsymbol{ocena}$
konec

Algoritem 31: Preveri odvečnost primerka

Rezultat: Množica parov (i, C) in (i, P), kjer je odkrita odvečnost primerka in ocena indikatorja popolnosti ontologije, ki upošteva odvečnost primerka.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxRed} \cdot k_{ABoxRedInst}$

Globalna pomembnost v procesu ROD:

Slika 123: Pomembnost preverjanja odvečnosti primerka (odvečnost komponente ABox) pri izračunu OC

Primer: Slika 124 prikazuje ontologijo z neposrednimi primerki, v kateri nekateri primerki kršijo pravilo odvečnosti primerka in so obarvani v rdeči barvi, medtem ko so pravilni označeni z modro barvo.

Slika 124: Primer preverjanja pomembnosti odvečnosti primerka (odvečnost komponente ABox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednje odvečne primerke: {*EB5EP4, K. O. certifikat*}, {*EB5EP4, Finančni inštrument*}, {*GOOG, Finančni inštrument*} in {*AAPL, Finančni inštrument*}. Ocena testne ontologije, ki upošteva odvečnost primerkov, je $1 - \frac{\frac{2}{3}+\frac{1}{2}+\frac{1}{2}}{12} = \frac{31}{36} =$ 0,861.

3.3.4.4 Anomalija

3.3.4.4.1 Neuporabljen razred

Problem: Neuporabljen razred je zadnji razred v razredni hierarhiji, ki se nikoli ne uporabi v aplikaciji in nima primerkov. Največkrat se pojavlja v večjih ontologijah, in sicer v situacijah, ko je bil v fazi razvoja določen razred zamenjan s specializiranim ali generaliziranim razredom. Neuporabljen razred ni vedno kandidat za brisanje, saj ga je včasih smiselno obdržati zaradi ponovne uporabljivosti ali standardizacije ontologije. Načrtovalec ontologije se mora tako odločiti med ponovno uporabljivostjo prvotne ontologije in poenostavljenega pristopa k razvoju ontologij, kjer obdržimo le razrede, ki jih potrebujemo.

Cilj: Preveriti, ali v ontologiji obstajajo neuporabljeni razredi, in omogočiti uporabniku rešitev problema – odstranitev razreda ali dodajanje primerkov.

Postopek: Za vsak razred C_i preveri, ali je zadnji razred v razredni hierarhiji in ali nima primerkov $i(C_i)$.

poisciNeuporabljenRazred
$ocena = 1$
n_C^{leaf} ← Število zadnjih razredov v razredni hierarhiji v ontologiji
$C_1 \subseteq C_2$ ← C_1 je podrazred od C_2
$i(C)$ ← Primerek i pripada razredu C
$neuporabljeniRazredi = \emptyset$
za vsak $C_i \in TBox$
  če $\nexists C_j \in TBox : C_j \subseteq C_i \wedge \nexists i(C_i) \in ABox$ potem
    $ocena = ocena - \frac{1}{n_C^{leaf}}$
    $neuporabljeniRazredi = neuporabljeniRazredi \cup C_i$
  konec pogoja če
konec sprehoda po vseh razredih
vrni $\boldsymbol{neuporabljeniRazredi}$ in $\boldsymbol{ocena}$
konec

Algoritem 32: Poišči neuporabljene razrede

Rezultat: Množica neuporabljenih razredov in ocena indikatorja popolnosti ontologije, ki upošteva odvečnost primerka.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxAnom} \cdot k_{ABoxAnomLazyConc}$

Globalna pomembnost v procesu ROD:

Slika 125: Pomembnost iskanja neuporabljenih razredov (anomalija komponente ABox) pri izračunu OC

Primer: Slika 126 prikazuje ontologijo, ki vsebuje neuporabljene koncepte.

Slika 126: Primer iskanja neuporabljenih razredov (anomalija komponente ABox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednja neuporabljena razreda: $\{ETF, Turbo\ nakupni\}$. Ocena testne ontologije, ki upošteva neuporabljene razrede, je $1 - \frac{1}{3} - \frac{1}{3} = \frac{1}{3} = 0{,}333$.

3.3.4.4.2 Neuporabljena lastnost

Problem: Neuporabljena lastnost je zadnja lastnost v hierarhiji, ki se nikoli ne uporabi v aplikaciji in nima primerkov. Največkrat se pojavlja v večjih ontologijah, in sicer v situacijah, ko je bila v fazi razvoja določena lastnost zamenjana s specializirano ali generalizirano lastnostjo. Neuporabljena lastnost ni vedno kandidat za brisanje, saj jo je včasih smiselno obdržati zaradi ponovne uporabljivosti ali standardizacije ontologije. Načrtovalec ontologije se mora tako odločiti med ponovno uporabljivostjo prvotne ontologije in poenostavljenim pristopom k razvoju ontologij, kjer obdržimo le lastnosti, ki jih potrebujemo.

Cilj: Preveriti, ali v ontologiji obstajajo neuporabljene lastnosti, in omogočiti uporabniku rešitev problema – odstranitev lastnosti ali dodajanje primerkov.

Postopek: Za vsako lastnost P_i preveri, ali je zadnja lastnost v hierarhiji in ali nima primerkov $i(P_i)$.

poisciNeuporabljaneLastnosti
  $ocena = 1$
  n_P^{leaf} ← Število zadnjih lastnosti v hierarhiji v ontologiji
  $P_1 \subseteq P_2$ ← P_1 je podlastnost od P_2
  $i(P)$ ← Primerek i pripada lastnosti P
  $neuporabljeneLastnosti = \emptyset$
  za vsako $P_i \in RBox$
    če $\nexists P_j \in RBox : P_j \subseteq P_i \wedge \nexists i(P_i) \in ABox$ potem
      $ocena = ocena - \frac{1}{n_P^{leaf}}$
      $neuporabljeneLastnosti = neuporabljeneLastnosti \cup P_i$
    konec pogoja če
  konec sprehoda po vseh lastnostih
  vrni ***neuporabljeneLastnosti*** in ***ocena***
konec

Algoritem 33: Poišči neuporabljene lastnosti

Rezultat: Množica neuporabljenih lastnosti in ocena indikatorja popolnosti ontologije, ki upošteva odvečnost primerka.

Globalna utež v procesu ROD: $w' = k_{ABox} \cdot k_{ABoxAnom} \cdot k_{ABoxAnomLazyProp}$

Globalna pomembnost v procesu ROD:

Slika 127: Pomembnost iskanja neuporabljenih lastnosti (anomalija komponente ABox) pri izračunu OC

Primer: Slika 128 prikazuje ontologijo, ki vsebuje neuporabljene lastnosti.

Slika 128: Primer iskanja neuporabljenih lastnosti (anomalija komponente ABox) pri izračunu OC

Po izvedbi algoritma odkrijemo naslednjo neuporabljeno lastnost: $\{se\ trguje\ vsako\ uro\}$. Ocena testne ontologije, ki upošteva neuporabljene lastnosti, je $1 - \frac{1}{4} = \frac{3}{4} = 0{,}750$.

4 EVALUACIJA

Za podporo hitremu razvoju ontologij po procesu ROD je bil razvit **prototip orodja IntelliOnto** (podrobnosti v poglavju 4.1). Omenjeno orodje nudi uporabniku podporo pri razvoju ontologije od začetka, t. j. zajema poslovnega slovarja, do konca, t. j. izdelave funkcionalne komponente. Pri celotnem procesu je na voljo tudi indikator popolnosti ontologije (OC), ki uporabniku pomaga pri napredovanju po korakih procesa ROD ter nenehno preverja kvaliteto ontologije.

Aplikabilnost in primernost procesa ROD pri hitrem razvoju ontologij sta bili izmerjeni na **primeru ontologije finančnega trgovanja na finančnih trgih** (podrobnosti v poglavju 4.2). Pri procesu razvoja se je merila vsaka posamezna iteracija razvoja ki jo je uporabnik izvedel. Namen evaluacije je pokazati, da se s procesom pristopa ROD k hitremu razvoju ontologij, zmanjša število iteracij in da se kvaliteta razvite ontologije konstantno vzdržuje in ni večjih nihanj, saj uporabniku v primeru nepopolne ontologije ni dovoljen prehod v naslednji korak procesa.

4.1 ORODJE INTELLIONTO

Za lažjo in hitrejšo uporabo hitrega pristopa za razvoj ontologij (ROD) je bil razvit prototip orodja **IntelliOnto**. Gre za spletno aplikacijo, ki uporablja številne odprtokodne tehnologije za manipulacijo z ontologijami (Protege, JENA[49], KAON2[50] ipd.) in Web 2.0 tehnologije za prikaz uporabniškega vmesnika (npr. AJAX). Trinivojska arhitektura je bila izbrana zaradi enostavnejše integracije v obstoječa razvojna okolja in vključitve v obstoječe aplikacije socialnih mrež (npr. Facebook, MySpace ipd.).

Slika 129: Preslikava problemske domene v ontologijo s pomočjo procesa ROD

Slika 130 prikazuje uporabniški vmesnik orodja IntelliOnto, ki je razdeljen v več funkcionalnih delov.

[49] **JENA** je semantično ogrodje za manipulacijo ontologij v programskem jeziku Java. Več najdete na spletni strani http://jena.sourceforge.net.
[50] **KAON2** je infrastruktura za obvladovanje OWL-DL, SWRL in F-Logic ontologij. Več najdete na spletni strani http://kaon2.semanticweb.org.

Slika 130: Uporabniški vmesnik orodja IntelliOnto

Prikaz sosednosti konceptov omogoča obvladovanje kompleksnosti ontologije z uporabo filtra sosednosti. S pomočjo tega pristopa uvedemo lokalnost, tako da se uporabnik v določenem trenutku ukvarja zgolj s sosednjimi koncepti, ki so z opazovanim povezani. Prikazujejo se samo tisti gradniki, ki so semantično povezani z opazovanim. Uporabnik lahko s pomočjo drsnika enostavno nastavlja število korakov, ki se pri izračunu sosednosti upoštevajo.

Seznam konceptov enostavno prikazuje množico vseh konceptov, ki smo jih opredelili in jih lahko uporabljamo pri modeliranju problemske domene. Seznam je urejen glede na relacije podrazred s prikazom razmerja nadrejen/podrejen.

Indikator popolnosti ontologije (podrobnosti so predstavljene v poglavju 3.3) opredeljuje raven ontologije, ki uporabniku prikazuje, do katere mere je ontologija dokončana in kakšne so možnosti vključitve obstoječe ontologije v obliki funkcionalne komponente v ostale sisteme. Rezultati evaluacije (Slika 56 in Slika 130) so predstavljeni v obliki indikatorja od 0 % do 100 % in seznama akcij, ki jih je potrebno izvesti, da dosežemo boljšo stopnjo popolnosti ontologije.

Koraki procesa razvoja ontologije pomagajo uporabniku pri sledenju zaporednih korakov procesa ROD. To je doseženo s prikazovanjem trenutnega stanja in koraka, v katerem se nahajamo, ter navodila, kaj vse je potrebno storiti za prehod v naslednji korak. Vedno imamo tudi možnost vrnitve v prejšnji korak ali izpustiti določen korak. Z vsakim korakom se prilagaja tudi uporabniški vmesnik IntelliOnto orodja, kar pomeni, da je v

začetnih fazah procesa na voljo zgolj enostavno urejanje konceptov, medtem ko pri napredovanju skozi proces dobite na voljo še ostale napredne možnosti.

Status je del uporabniškega vmesnika, kjer so prikazane podrobnosti o izvedenih akcijah in nekateri odzivi sistema.

Osrednji del za manipulacijo s koncepti in pravili je grafično usmerjen in omogoča uporabnikom enostavno kreiranje konceptov, medsebojnih povezav in omejitev na način, ki kar najbolj spominja na izražanje v naravnem jeziku. Uporabnik lahko vnaša njihove trditve in izjave na številne načine – enostavne izjave v polstrukturiranem jeziku, omejitve v obliki odločitvenih tabel in odločitvenih dreves. Zelo pomembna je tudi podpora sinonimom in homonimom[51]. Pri delu npr. uporabljamo pogovorne izraze, medtem ko poslovni svet zahteva uporabo uradno sprejete terminologije, kar lahko zelo elegantno rešimo z uporabo sinonimov.

4.1.1 *PODROBNOSTI IMPLEMENTACIJE*

Orodje IntelliOnto podpira vnos naprednih omejitev ontologije na dva načina:

- sprehod po semantični mreži in dodajanje omejitev (poglavje 4.1.1.1.1) ter
- vnos omejitev v obliki atomov pravil (poglavje 4.1.1.1.2).

Pri sprehodu po semantični mreži uporabniku omogočimo pregled obstoječe baze znanja in neposredne manipulacije s tem obstoječim znanjem ter možnost dodajanja novih aksiomov v uporabniku prijazni obliki. Vnos omejitev v obliki atomov pravil je vmesnik, ki je povezan z dodajanjem poslovnih pravil problemske domene. Kot je že bilo omenjeno v poglavju 3.2.2.6.1, so na voljo številni pristopi, ki uporabniku olajšajo vnos: kaj-če, odločitveno drevo in odločitvene tabele. Vsi so tudi podprti v okviru IntelliOnto orodja.

Ontologija trgovanja na finančnih trgih (Priloga 1: Primer trgovanja s finančnimi inštrumenti) je uporabljena kot primer skozi celotno disertacijo. Jedro omenjene ontologije so predvsem strategije trgovanja, ki so podrobneje predstavljene v poglavju 4.1.1.2, skupaj z različnimi načini vnosa pravil (kaj-če, odločitveno drevo in odločitvene tabele).

4.1.1.1 Vnos naprednih omejitev ontologije

Pri vnosu naprednih omejitev ontologije ima uporabnik na voljo več načinov vnosa:

- sprehod po semantični mreži in dodajanje omejitev in
- vnos omejitev v obliki atomov pravil
    - vnos omejitev v kaj-če obliki,
    - vnos omejitev v obliki odločitvenega drevesa in
    - vnos omejitev v obliki odločitvene tabele.

V nadaljevanju bodo podrobneje predstavljeni vsi omenjeni načini.

[51] **Homonim** ali enakozvočnica je beseda, ki ima enako zvočno ali pisno podobo kot druga beseda, a drugačen pomen.

4.1.1.1.1 Sprehod po semantični mreži in dodajanje omejitev

Semantična mreža je pri ontologijah, zapisanih v jezikih semantičnega spleta, sestavljena iz entitet in aksiomov. Osnovni gradniki, ki nastopajo v točkah grafa, so:

- razredi,
- lastnosti in
- primerki.

Nadgradnjo omenjenih gradnikov pa predstavljajo pravila in napredne omejitve, katerih zajem bo podrobneje predstavljen v naslednjem razdelku 4.1.1.1.2.

Pri vsakem od gradnikov (točk v grafu semantične mreže) je mogoče nastaviti določene opise. Slika 131 prikazuje podrobnejšo opredelitev razreda, kjer lahko dodamo: opis v naravnem jeziku, dodaten opis in primerke koncepta.

Slika 131: Korak 1 dodajanja omejitev

Pri lastnostih lahko dodajamo opis v naravnem jeziku ter dodaten opis (Slika 132).

Slika 132: Korak 2 dodajanja omejitev

Pri dodatnem opisovanju primerkov imamo ravno tako na voljo več možnosti, in sicer opis primerka ter opredelitev enakosti oz. različnosti, kot prikazuje Slika 133.

Slika 133: Korak 3 dodajanja omejitev

V začetnih fazah razvoja ontologije je zelo pomemben opis entitet v naravnem jeziku. Pri vseh opisih je podprta večjezičnost, kar pomeni, da lahko opise zapišemo v poljubnem jeziku, medtem ko sistem uporabnika spodbuja k uporabi vsaj dveh jezikov (angleškega jezika kot splošno sprejetega, kar pozitivno vpliva na ponovno uporabnost in lokalni jezik, v katerem je ontologija razvita, npr. slovenski jezik). Pri dodajanju opisov konceptu sta na voljo dva različna, in sicer naziv ter opis (Slika 134). Naziv koncepta se vedno uporablja

pri vizualizaciji, kar posledično pomeni, da uporabnik nikoli ne operira z enoličnimi identifikatorji razredov, ampak neposredno z njihovimi nazivi v naravnem jeziku. Tak pristop se pokaže kot zelo uporaben tudi pri opredelitvi naprednejših omejitev in pravil, saj se pri takšni opredelitvi samodejno in dinamično generira približek opisa v naravnem jeziku, tako da je semantično modeliranje uporabniku zelo poenostavljeno.

Slika 134: Korak 1.1 dodajanja omejitev

Rezultat omenjene akcije sta naslednja aksioma v ontologiji:

```
<owl:Class rdf:ID="FinancniInstrument">
  <rdfs:label xml:lang="si">Finančni inštrument</rdfs:label>
  <rdfs:comment xml:lang="si">Poljubna oblika
financiranja.</rdfs:comment>
</owl:Class>
```

Razrede je možno tudi dodatno opisati, in sicer na naslednje načine (Slika 135):

- nov opis z lastnostjo,
- dopolnjen opis z lastnostjo in
- dopolnjena opredelitev koncepta.

Slika 135: Korak 1.2 dodajanja omejitev

Pri dodajanju novega opisa v obliki lastnosti uporabnik najprej izbere za kakšen tip opisa gre (Slika 136):

- enostaven opis v obliki podatkovne lastnosti ali
- kompleksen opis v obliki objektne lastnosti.

Slika 136: Korak 1.2.1 dodajanja omejitev

Pri enostavnem opisu mora uporabnik opredeliti naziv in podatkovni tip opisa, kot to prikazuje Slika 137. Zaradi enostavnejšega vnosa ima pri tipu lastnosti na voljo več načinov. Če pozna enostavni podatkovni tip (npr. niz, datum, število ipd.), ga preprosto izbere s seznama. V nasprotnem primeru lahko našteje testne podatke, iz katerih se nato sklepa o podatkovnem tipu.

Slika 137: Korak 1.2.1.1 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:DatatypeProperty rdf:ID="simbol">
  <rdfs:label xml:lang="si">simbol</rdfs:label>
  <rdfs:domain rdf:resource="#FinancniInstrument"/>
  <rdfs:range rdf:resource="xsd:string"/>
</owl:Class>
```

Pri kompleksnem opisu je potrebno opredeliti nov razred, ki dodatno opisuje obstoječega (Slika 138). Če razred, ki dodatno opisuje izbranega, že obstaja, ga preprosto izberemo s seznama, sicer kreiramo novega.

Slika 138: Korak 1.2.1.2 dodajanje omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:ObjectProperty rdf:ID="seTrgujeNa">
  <rdfs:label xml:lang="si">se trguje na</rdfs:label>
  <rdfs:domain rdf:resource="#FinancniInstrument"/>
  <rdfs:range rdf:resource="#TrgovalniDan"/>
</owl:Class>
```

Uporabnik lahko tudi dodatno opredeli že obstoječe lastnosti (Slika 139), in sicer najprej izbere lastnost s seznama, kjer so prikazane zgolj lastnosti z definicijskim območjem izbranega razreda. Ko je lastnost izbrana, se lahko nastavi:

- vrednost in
- števnost lastnosti.

Slika 139: Korak 1.2.2 dodajanja omejitev

Pri vrednosti lastnosti lahko uporabnik nastavlja omejitve ontologije *owl:someValuesFrom*, *owl:allValuesFrom* in *owl:hasValue*. Ker omenjenih aksiomov uporabnik ne pozna, so te omejitve predstavljene v opisni obliki, in sicer, da mora za vse primerke lastnosti izbranega razreda veljati, da so vse ali vsaj ena vrednost iz omejitve.

Slika 140: Korak 1.2.2.1 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:Class rdf:ID="FinancniInstrument">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectPropertyrdf:ID="kotiraNaBorzi">
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:ID="Borza"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </owl:equivalentClass>
  ...
</owl:Class>
```

Pri števnosti lastnosti uporabnik nastavlja najmanjše in največje število primerkov, ki je lahko povezano z določenim primerkom izbranega razreda (Slika 141).

Slika 141: Korak 1.2.2.2 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:Class rdf:ID="FinancniInstrument">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectPropertyrdf:ID="kotiraNaBorzi">
        </owl:onProperty>
        <owl:minCardinality
         rdf:datatype=xsd:int>1</owl:minCardinality>
      </owl:Restriction>
      <owl:Restriction>
```

```
        <owl:onProperty>
          <owl:ObjectPropertyrdf:ID="kotiraNaBorzi">
        </owl:onProperty>
        <owl:maxCardinality
         rdf:datatype=xsd:int>4</owl:maxCardinality>
      </owl:Restriction>
    </owl:Class>
  </owl:equivalentClass>
  ...
</owl:Class>
```

Tretja od možnosti dodatnega opisa razreda je dopolnjena opredelitev razreda (Slika 142). Uporabnik ima na tem mestu možnost dodatne opredelitve razreda v smislu vsebovanosti primerkov oz. omejitve opredelitve.

Slika 142: Korak 1.2.3 dodajanja omejitev

Pri prvi od možnosti (vsebovanost primerkov) se opredeljujejo aksiomi *owl:equivalentClass, owl:disjointWith in rdfs:subClassOf*. Za vsakega od omenjenih aksiomov so za uporabnika pripravljena vsebinska vprašanja:

- *owl:equivalentClass*
  - Obstaja koncept, ki je identičen (sinonim) *<izbranemu konceptu>*, saj velja, da so vsi primerki enega tudi primerki drugega in obratno.
- *owl:disjointWith*
  - Obstaja koncept, katerega primerki ne morejo biti tudi primerki koncepta *<izbrani koncept>* in obratno.
- *rdfs:subClassOf*
  - Obstaja koncept, kjer vsi primerki pripadajo tudi *<izbranemu konceptu>*, vendar obratno ne velja.

Slika 143: Korak 1.2.3.1 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:Class rdf:ID="FinancniInstrument">
  <owl:equivalentClass><owl:Class rdf:ID="FI"/></owl:equivalentClass>
  <owl:disjointWith>
    <owl:Class rdf:ID="TrgovalniDan"/>
  </owl:disjointWith>
  ...
</owl:Class>

<owl:Class rdf:ID="Delnica">
  <rdfs:subClassOf rdf:resource="#FinancniInstrument"/>
</owl:Class>
```

Naslednja možnost dopolnitve opisa opredelitve koncepta je omejitev opredelitve (Slika 144), kjer se razrede opredeli s pomočjo teorije množic. V tem primeru se v ontologijo zapišejo naslednji aksiomi:

- *owl:unionOf*
  - Za primerke *<izbranega koncepta>* velja, da so, ali primerki koncepta *<koncept 1>* ali primerki koncepta *<koncept 2>*.
- *owl:intersectionOf*
  - Za primerke *<izbranega koncepta>* velja, da so, in primerki koncepta *<koncept 1>* in primerki koncepta *<koncept 2>*.
- *owl:complementOf*
  - Za primerke *<izbranega koncepta>* velja, da so vsi ostali primerki, ki niso *<koncept>*.

Slika 144: Korak 1.2.3.2 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:Class rdf:ID="Posojilo">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
```

```
        <owl:Class>
          <owl:complementOf>
            <owl:Class rdf:ID="Depozit"/>
          </owl:complementOf>
        </owl:Class>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:ID="KratkorocnoPosojilo"/>
            <owl:Class rdf:ID="DolgorocnoPosojilo"/>
          </owl:unionOf>
        </owl:Class>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:ID="DenarnoSredstvo"/>
            <owl:Class rdf:ID="FinancniInstrument"/>
          </owl:intersectionOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Poleg opisa razreda v naravnem jeziku in dodatnega opisa razreda lahko uporabnik dodaja tudi primerke koncepta (Slika 145). Na voljo sta dve možnosti:

- dodajanje novega primerka in
- izbira obstoječega primerka in nastavljanja pripadnosti izbranemu razredu.

Slika 145: Korak 1.3 dodajanja omejitev

Pri dodajanju novega primerka mora uporabnik zgolj izpolniti obrazec, ki se samodejno generira iz obstoječe ontologije. Za izbran razred se preberejo vse omejitve in aksiomi, ki so predstavljeni v obliki zahtevanih opisov in podatkov koncepta. Najprej je potrebno opredeliti ime, sledijo obvezni opisi primerka (podatkovne in objektne lastnosti, ki so obligacijske) in nato opcijski opisi primerka, ki jih omejitve v ontologiji izrecno ne zahtevajo, jih pa dopuščajo.

Slika 146: Korak 1.3.1 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<FinancniInstrument rdf:ID="GoogleInc">
  <rdfs:label rdf:datatype="xsd:string">Google Inc.</rdfs:label>
  <ISIN rdf:datatype="xsd:string">US38259P5089</ISIN>
  <ime rdf:datatype="xsd:string">Google Inc.</ime>
  <simbol rdf:datatype="xsd:string">GOOG</simbol>
  <kotiraNaBorzi><Borza rdf:ID="NASDAQ"/></kotiraNaBorzi>
</FinancniInstrument>
```

Pri izbiri obstoječega primerka in nastavljanja pripadnosti izbranemu razredu je proces še enostavnejši, saj uporabnik zgolj izbere želen razred, kateremu naj primerek pripada (Slika 147).

Slika 147: Korak 1.3.2 dodajanja omejitev

Rezultat omenjene akcije je naslednji aksiom v ontologiji:

```
<FinancniInstrument rdf:ID="Krkadd">
  ...
</FinancniInstrument>
```

Pri podrobni opredelitvi lastnosti imamo podobno kot pri razredih najprej na voljo dodajanje opisa v naravnem jeziku. To opravilo se izkaže za izjemno pomembno, saj se

opis v naravnem jeziku kasneje uporabi za prikaz entitet uporabniku in tudi za opredelitev naprednejših omejitev. Pri vnosu opisa je podprta tudi večjezičnost, kar pomeni, da lahko opise zapišemo v poljubnem jeziku. Pri dodajanju opisov lastnosti sta na volja dva različna, in sicer naziv in opis (Slika 148). Naziv lastnosti se vedno uporablja pri vizualizaciji, kar posledično pomeni, da uporabnik nikoli ne operira z enoličnimi identifikatorji razredov, ampak neposredno z njihovimi nazivi v naravnem jeziku.

Slika 148: Korak 2.1 dodajanja omejitev

Rezultat omenjene akcije sta naslednja aksioma v ontologiji:

```
<owl:ObjectProperty rdf:ID="seTrgujeNa">
  <rdfs:label xml:lang="si">se trguje na</rdfs:label>
  <rdfs:comment xml:lang="si">
    Finančni inštrument se trguje na trgovalni dan.
  </rdfs:comment>
</owl:ObjectProperty>
```

Pri opredelitvi dodatnega opisa lastnosti lahko uporabnik nastavlja smer povezave in omejitve (Slika 149).

Slika 149: Korak 2.2 dodajanja omejitev

Pri smeri lastnosti se upošteva načelo sprehajanja po semantični mreži, kjer naj ne bi obstajali izolirani otoki razredov, ki med seboj niso povezani. Eden od načinov za doseganje boljše povezljivosti konceptov je opredelitev dvosmernih povezav ter s tem uporabniku omogočitev enostavnega sprehajanja med koncepti (Slika 150).

Slika 150: Korak 2.2.1 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:ObjectProperty rdf:ID="seTrgujeNa">
  <rdfs:domain rdf:resource="#FinancniInstrument"/>
  <rdfs:range rdf:resource="#TrgovalniDan">
  <owl:inverseOf><owl:ObjectProperty
rdf:ID="veljaZa"/></owl:inverseOf>
  ...
</owl:ObjectProperty>
```

Pri omejitvah lastnosti ima uporabnik na voljo naslednje možnosti (Slika 151):

- omejitev opredelitve,
- funkcionalnost in
- prehajanje.

Slika 151: Korak 2.2.2 dodajanja omejitev

Pri omejitvi opredelitve se podobno kot pri razredih omejuje hierarhija lastnosti s pomočjo aksiomov *owl:equivalentProperty* in *rdfs:subPropertyOf*. Za vsakega od omenjenih aksiomov so za uporabnika pripravljena vsebinska vprašanja:

- *owl:equivalentProperty*
  - Za lastnost *<izbrana lastnost>* velja, da obstaja identična lastnost (sinonim), saj velja, da so vsi primerki lastnosti tudi primerki te lastnosti.
- *rdfs:subPropertyOf*
  - Za lastnost *<izbrana lastnost>* velja, da obstaja lastnost, katere vsi primerki pripadajo tudi lastnosti *<lastnost>*, vendar obratno to ne velja.

Slika 152: Korak 2.2.2.1 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:ObjectProperty rdf:ID="sePogostoTrgujeNa">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:ID="seTrgujeNa">
  </rdfs:subPropertyOf>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="seZamenjujeLastnistvo">
  <owl:equivalentProperty>
    <owl:ObjectProperty rdf:ID="seTrgujeNa">
  </owl:equivalentProperty>
</owl:ObjectProperty>
```

Naslednja od omejitev je funkcionalnost povezav, kjer je potrebno določiti, ali je lastnost funkcionalna ali inverzno funkcionalna. Za lažje sprejemanje odločitev ima uporabnik na voljo obrazec, kjer s pomočjo vsebinskih vprašanj določi formalne omejitve:

- *owl:FunctionalProperty*
  - Za lastnost *<izbrana lastnost>* velja, da je primerek *<definicijskega območja izbrane lastnosti>* lahko opisan samo z enim primerkom *<zaloge vrednosti izbrane lastnosti>*.
- *owl:InverseFunctionalProperty*
  - Za lastnost *<izbrana lastnost>* velja, da primerek *<definicijskega območja izbrane lastnosti>* lahko opisuje samo en primerek *<zaloge vrednosti izbrane lastnosti>*.

Slika 153: Korak 2.2.2.2 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<owl:InverseFunctionalProperty rdf:about="seTrgujeNa">
  ...
</owl:InverseFunctionalProperty>
```

Prehajanje povezav je tudi ena od podrobnosti, ki jih lahko uporabnik opredeli pri povezavah. Pri prehodnosti povezav upoštevamo lastnosti tranzitivnosti in simetričnosti, kar uporabnik ugotovi s pomočjo vsebinskega vprašalnika (Slika 154):

- *owl:SymmetricProperty*
- *owl:TransitiveProperty*

Slika 154: Korak 2.2.2.3 dodajanja omejitev

Pri dodatnem opisovanju primerkov lahko primerek opišemo oz. opredelimo kot medsebojno enakost ali različnost med primerki. Če želimo dodatno opisati primerek, potem gre za identično opravilo, ki smo ga že predstavili v koraku 1.3.1 pri dodajanju primerkov razredu.

V primeru opredelitve enakosti oz. različnosti med primerki pa ima uporabnik na voljo obrazec, na katerem določi, ali je izbrani primerek identičen obstoječemu primerku oz. ali obstaja kakšen primerek, ki je eksplicitno različen od izbranega primerka (Slika 155).

Slika 155: Korak 3.2 dodajanja omejitev

Rezultat omenjene akcije so naslednji aksiomi v ontologiji:

```
<FinancniInstrument rdf:ID="GoogleInc">
  <owl:sameAs rdf:resource="#Google">
  <owl:differentFrom rdf:resource="#AppleInc"/>
</FinancniInstrument>
```

4.1.1.1.2 Vnos omejitev v obliki atomov pravil

Ko uporabnik izbere način vnosa pravil (**kaj-če**, **odločitveno drevo** ali **odločitvena tabela**) je vnos omejitev identičen za vse tri načine. Pravila vnašamo s pomočjo obrazca pravil, kamor dodajamo atome, ki sestavljajo pravilo.

Slika 156: Korak A vnosa omejitev

Na začetku uporabnik izbere tip atoma, ki ga želi vnesti. Pri tem ima na voljo (Slika 156):

- **primerek razreda**,
- **zalogo vrednosti**,
- **omejitev relacije** in
- **vnaprej opredeljeno omejitev**.

4.1.1.1.2.1 Primerek razreda

Omejitev primerka razreda se uporablja za izražanje pripadnosti primerka izbranemu razredu. Slika 157 prikazuje obrazec za vnos primerka razreda. Kot je razvidno, obrazec sestavljata dva argumenta in vezni tekst.

Slika 157: Korak A.A vnosa omejitev

Uporabnik lahko spreminja zgolj argumente in sicer najprej razred, kateremu naj primerek pripada. Slika 158 prikazuje omenjeno akcijo, kjer se uporabniku prikaže seznam vseh razredov ontologije.

Slika 158: Korak A.A.A vnosa omejitev

Ko je izbira potrjena, se obrazec le-temu prilagodi, kot prikazuje Slika 159.

Slika 159: Korak A.A.A.A vnosa omejitev

Vnos omejitve se lahko začne tudi z vnosom primerka, kjer so na voljo naslednje možnosti (Slika 160):

- nova spremenljivka,
- obstoječa spremenljivka in
- obstoječi primerek.

Slika 160: Korak A.A.B vnosa omejitev

Če se uporabnik odloči uporabiti novo spremenljivko, ga vnese s pomočjo obrazca (Slika 161), kjer se preverja tudi potencialen obstoj spremenljivke.

Slika 161: Korak A.A.B.A vnosa omejitev

Izbira obstoječe spremenljivke je izvedena tako, da se uporabniku prikaže seznam vseh spremenljivk, ki so na voljo (Slika 162). V primeru, da je drugi argument (razred) že določen, pa se prikažejo zgolj tiste spremenljivke, ki so vezane na izbrani razred.

Slika 162: Korak A.A.B.B vnosa omejitev

Zadnja od možnosti izbire primerka je obstoječi primerek (Slika 163). Podobno kot v prejšnjem primeru se upošteva morebitno izbrani razred pri prikazu primerkov.

Slika 163: Korak A.A.B.C vnosa omejitev

Po opredelitvi obeh potrebnih omejitev se obrazec prilagodi (Slika 164) in vnos omejitve primerka razreda je končan.

Slika 164: Korak A.A.A.A.A vnosa omejitev

Omenjeno omejitev lahko v abstraktni sintaksi zapišemo kot

```
TrgovalniDan(?dan1)
```

medtem ko je izvorna koda v jeziku semantičnega spleta SWRL naslednja

```
<swrl:ClassAtom>
  <swrl:classPredicate rdf:resource="#TrgovalniDan"/>
  <swrl:argument1 rdf:resource="#dan1"/>
</swrl:ClassAtom>
```

4.1.1.1.2.2 Omejitev relacije

Pri omejitvi relacije uporabnik opredeli omejitev povezave med dvema entitetama oz. podatkovnima tipoma. Najprej je potrebno izbrati za kakšno relacijo gre – ali je **enostavna**, ki povezuje primerek razreda s podatkovno lastnostjo ali **kompleksna**, ki povezuje med seboj primerke razredov (Slika 165).

Slika 165: Korak A.C vnosa omejitev

Obrazec za opredelitev omejitve enostavne relacije (Slika 166) vsebuje tri argumente (opis, primerek in vrednost) ter vezni tekst.

Slika 166: Korak A.C.A vnosa omejitev

Pri obrazcu za opredelitev omejitve kompleksne relacije (Slika 167) imamo dva argumenta (primerka) in vezni tekst.

Slika 167: Korak A.C.B vnosa omejitev

Vezni tekst se v obeh primerih kreira dinamično in je odvisen od izbrane relacije ter se pridobi iz sheme ontologije (opis entitete v naravnem jeziku).

Pri omejitvi enostavne relacije se od uporabnika najprej zahteva opredelitev relacije, ki jo želimo omejiti (Slika 168). Na tem mestu se prikaže seznam vseh podatkovnih lastnosti ter pripadajoča zaloga vrednosti, da lahko uporabnik lažje izlušči kontekst uporabe (npr. razlog za izvedbo akcije od Trgovalni dan).

Slika 168: Korak A.C.A.A vnosa omejitev

Ob izbiri podatkovne lastnosti se spremeni tudi vnosni obrazec, in sicer se upošteva definicijsko območje, tako da se spremeni naziv argumenta *Primerek* v *Primerek <zaloga vrednosti podatkovne lastnosti>*, v našem primeru je to *Primerek Trgovalni dan* (Slika 169).

Slika 169: Korak A.C.A.A.A vnosa omejitev

Uporabnik mora v nadaljevanju nastaviti še primerek podatkovne lastnosti in vrednost omenjene lastnosti (Slika 170). Pri nastavljanju primerka so na voljo naslednje možnosti:

- nova spremenljivka,
- obstoječa spremenljivka in
- obstoječi primerek.

Slika 170: Korak A.C.A.A.A.A vnosa omejitev

Podobno je tudi pri nastavljanju vrednosti, kjer so na voljo naslednje možnosti:

- nova spremenljivka,
- obstoječa spremenljivka in
- konstantna vrednost.

Če uporabnik izbere novo spremenljivko pri primerku in konstantno vrednost podatkovne lastnosti, se pojavi vnosno okno, kjer uporabnik vnese potrebne podatke (Slika 171).

Slika 171: Korak A.C.A.A.A.A.A vnosa omejitev

Omejitev je tako opredeljena in prikazan v približku naravnega jezika (Slika 172).

Slika 172: Korak A.C.A.A.A.A.A.A vnosa omejitev

Omenjeno omejitev lahko v abstraktni sintaksi zapišemo kot

```
razlogZaIzvedboAkcije(?dan2,
"Prisotnost »engulfing« trgovalnega vzorca.")
```

medtem ko je izvorna koda v jeziku semantičnega spleta SWRL naslednja

```
<swrl:DatavaluedPropertyAtom>
  <swrl:propertyPredicate rdf:resource="#razlogZaIzvedboAkcije"/>
  <swrl:argument1 rdf:resource="#dan2"/>
  <swrl:argument2 rdf:datatype="xsd:string">
    Prisotnost »engulfing« trgovalnega vzorca
  </swrl:argument2>
</swrl:DatavaluedPropertyAtom>
```

V primeru izbire obstoječih spremenljivk imamo na voljo izbiro že opredeljenih spremenljivk, kjer se upoštevajo tudi omejitve ontologije. Pri spremenljivkah primerka se upoštevajo le tiste, ki so povezane z izbranim razredom, ki je v definicijskem območju podatkovne relacije. Pri spremenljivkah vrednosti pa se upošteva zaloga vrednosti podatkovne relacije, tako da je dovoljen vnos le izbranega enostavnega podatkovnega tipa.

Slika 173: Korak A.C.A.A.A.A.B vnosa omejitev

Zadnja možnost je izbira obstoječih primerkov in nove spremenljivke pri vrednosti podatkovne relacije (Slika 174). Pri prikazu obstoječih primerkov je seznam omejen na primerke razredov definicijskega območja podatkovne funkcije.

Slika 174: Korak A.C.A.A.A.A.B vnosa omejitev

Pri omejitvi kompleksne relacije se od uporabnika najprej zahteva opredelitev relacije, ki jo želimo omejiti (Slika 175). Na tem mestu se prikaže seznam vseh podatkovnih lastnosti ter pripadajoča definicijska območja in zaloge vrednosti, da lahko uporabnik lažje izlušči kontekst (npr. Finančni inštrument kotira na Borza).

Slika 175: Korak A.C.B.A vnosa omejitev

Ob izbiri objektne lastnosti se spremeni tudi vnosni obrazec, in sicer se upošteva definicijsko območje in zaloga vrednosti, tako da se spremeni naziv argumenta *Primerek* v *Primerek <definicijsko območje objektne lastnosti>* in *Primerek <zaloga vrednosti objektne lastnosti>* (Slika 176).

Slika 176: Korak A.C.B.A.A vnosa omejitev

Uporabnik mora v nadaljevanju nastaviti še primerek podatkovne lastnosti in vrednost omenjene lastnosti (Slika 177). Pri nastavljanju primerka so na voljo naslednje možnosti:

- nova spremenljivka,
- obstoječa spremenljivka in
- obstoječi primerek.

Slika 177: Korak A.C.B.A.A.A vnosa omejitev

Če uporabnik izbere novo spremenljivko pri definicijskem območju objektne lastnosti in obstoječi primerek pri zalogi vrednosti, se pojavi vnosno okno, kjer uporabnik vnese potrebne podatke in izbere obstoječe primerke s seznama (Slika 178).

Slika 178: Korak A.C.B.A.A.A.A vnosa omejitev

Omejitev je tako opredeljena in prikazana v približku naravnega jezika (Slika 179).

Slika 179: Korak A.C.B.A.A.A.A.A vnosa omejitev

Omenjeno omejitev lahko v abstraktni sintaksi zapišemo kot

```
kotiraNa(?izbranFI, :LJSE)
```

medtem ko je izvorna koda v jeziku semantičnega spleta SWRL naslednja

```
<swrl:IndividualPropertyAtom>
  <swrl:propertyPredicate rdf:resource="#kotiraNa"/>
  <swrl:argument1 rdf:resource="#izbranFI"/>
  <swrl:argument2 rdf:resource="#LJSE"/>
</swrl:IndividualPropertyAtom>
```

4.1.1.1.2.3 Vnaprej opredeljena omejitev

Pri jezikih semantičnega spleta imamo na voljo veliko vnaprej opredeljenih omejitev, zelo enostavno pa lahko dodajamo tudi svoje, tako da ni težav glede razširljivosti. Uporabnik lahko pri gradnji ontologije izbira med naslednjimi možnostmi (Slika 180):

- **primerjava podatkov**,
- **primerjava primerkov**,
- **matematične funkcije** in
- **operacije z nizi**.

Slika 180: Korak A.D vnosa omejitev

Obrazec za vnos omejitve **primerjave podatkov** je sestavljen iz treh atributov, in sicer dveh podatkov ter operacije, ki se izvede (Slika 181).

Slika 181: Korak A.D.A vnosa omejitev

Uporabnik lahko izbira med najbolj pogosto uporabljanimi operacijami za primerjavo podatkov, in sicer (Slika 182):

- enakost (=),
- neenakost (≠),
- manjše (<),
- manjše ali enako (≤),
- večje (>) in
- večje ali enako (≥).

Slika 182: Korak A.D.A.A vnosa omejitev

Ko je operacija primerjave določena, je potrebno izbrati podatek, kjer imamo na voljo naslednje možnosti (Slika 183):

- nova spremenljivka,
- obstoječa spremenljivka in
- konstantna vrednost.

Slika 183: Korak A.D.A.A.A vnosa omejitev

Če uporabnik izbere novo spremenljivko in konstantno vrednost pri podatkih, potem ima na voljo obrazec za vnos vrednosti (Slika 184).

Slika 184: Korak A.D.A.A.A.A vnosa omejitev

Omejitev je tako opredeljena in prikazana v približku naravnega jezika (Slika 185).

Slika 185: Korak A.D.A.A.A.A.A vnosa omejitev

Omenjeno omejitev lahko v abstraktni sintaksi zapišemo kot

```
swrlb:greaterThanOrEqual(?danasnjaCenaObOdprtju, 112.36)
```

medtem ko je izvorna koda v jeziku semantičnega spleta SWRL naslednja

```
<swrl:BuiltinAtom>
  <swrl:builtin rdf:resource="swrlb:greaterThanOrEqual"/>
  <swrl:argument1 rdf:resource="#danasnjaCenaObOdprtju"/>
  <swrl:argument2 rdf:datatype="xsd:float">112.36</swrl:argument2>
</swrl:BuiltinAtom>
```

Obrazec za vnos omejitve **primerjave primerkov** je sestavljen iz treh atributov, in sicer dveh primerkov ter operacije (ne)enakosti (Slika 186).

Slika 186: Korak A.D.B vnosa omejitev

Uporabnik lahko izbira med operacijama enakosti in neenakosti (Slika 187). Pri enakosti gre za opredelitev identitete primerkov, t. j. dve poimenovanji, ki sta sinonima za isti primerek. Analogno temu je različnost, kjer eksplicitno zahtevamo, da se dva primerka razlikujeta.

Slika 187: Korak A.D.B.A vnosa omejitev

Ko izberemo ustrezno operacijo primerjave med primerki, se prilagodi tudi obrazec za opredelitev omejitve (Slika 188).

Slika 188: Korak A.D.B.A.A vnosa omejitev

Uporabnik mora v nadaljevanju določiti še vrednosti obeh primerkov, ki jih izbere iz padajočega seznama vrednosti (Slika 189).

Slika 189: Korak A.D.B.A.A.A vnosa omejitev

Po izbiri ustreznih primerkov je omejitev opredeljena in prikazana v približku naravnega jezika (Slika 190).

Slika 190: Korak A.D.B.A.A.A.A vnosa omejitev

Pri **matematičnih omejitvah** imamo na voljo vnaprej opredeljene enostavne operacije seštevanja (+), odštevanja (-), množenja (*), deljenja (/) in absolutne vrednosti.

Slika 191: Korak A.D.C vnosa omejitev

Obrazec za vnos omejitve matematične operacije je sestavljen iz štirih atributov, dveh števil, na katerih se operacija izvede in rezultata.

Slika 192: Korak A.D.C.A vnosa omejitev

Pri izbiri števila imamo podobno kot pri ostalih vnosih omejitev na voljo naslednje možnosti (Slika 193):

- nova spremenljivka,
- obstoječa spremenljivka in

- konstantna vrednost.

Slika 193: Korak A.D.C.A.A vnosa omejitev

Po ustreznem nastavljanju vrednosti se obrazec prilagodi in izjava je v približku naravnega jezika prikazana na Slika 194.

Slika 194: Korak A.D.C.A.A.A vnosa omejitev

Omenjeno omejitev lahko v abstraktni sintaksi zapišemo kot

```
swrlb:subtract(?velikostSvecnikaTelesa,
?danasnjaNajvisjaCena, ?danasnjaNajnizjaCena)
```

medtem ko je izvorna koda v jeziku semantičnega spleta SWRL naslednja

```
<swrl:BuiltinAtom>
  <swrl:builtin rdf:resource="swrlb:subtract"/>
  <swrl:argument1 rdf:resource="#velikostSvecnikaTelesa"/>
  <swrl:argument2 rdf:resource="#danasnjaNajvisjaCena"/>
  <swrl:argument3 rdf:resource="#danasnjaNajnizjaCena"/>
</swrl:BuiltinAtom>
```

4.1.1.2 Implementacija strategije trgovanja

Poslovna pravila za prepoznavanje vzorcev svečnikov so bila zajeta s pomočjo razvitega pristopa ROD. V nadaljevanju bodo prikazane vse oblike dodajanja pravil in kompleksnih omejitev v ontologijo.

4.1.1.2.1 Zajem pravil z odločitvenim drevesom

Pravila za določanje barve telesa svečnika so naslednja:

- Če je cena ob zaprtju trgovalnega dne višja ali enaka od cene ob odprtju, je telo svečnika bele barve.

- Če je cena ob zaprtju trgovalnega dne nižja od cene ob odprtju, je telo svečnika črne barve.

Omenjeni pravili sta primerni za vnos s pomočjo modula odločitvenega drevesa, saj se pogoji, ki se pojavljajo v telesu obeh pravil, prekrivajo; razlikujejo se zgolj v eni omejitvi. Slika 195 prikazuje vnos omenjenih pravil.

Slika 195: Zajem pravil za določitev barve telesa svečnika z modulom odločitvenega drevesa

Izvorna koda 1 prikazuje abstraktno sintakso jezika SWRL za omenjeni pravili, zajeti s pomočjo odločitvenega drevesa. Razvidno je, da je vsaka veja odločitvenega drevesa implementirana kot enostavno pravilo, kar nam omogoča gradnjo kompleksnejših dreves, ki niso nujno uravnotežena.

```
TradingDay(?dan) ∧ close(?dan, ?cenaZaprtja) ∧ open(?dan,
?cenaOdprtja) ∧ swrlb:lessThan(?cenaZaprtja, ?cenaOdprtja) →
BlackBody(?dan)

TradingDay(?dan) ∧ close(?dan, ?cenaZaprtja) ∧ open(?dan,
?cenaOdprtja) ∧ swrlb:greaterThanOrEqual(?cenaZaprtja, ?cenaOdprtja) →
WhiteBody(?dan)
```

Izvorna koda 1: Pravili za določitev barve telesa svečnika v abstraktni sintaksi SWRL

Izvorna koda 2 prikazuje sintakso SWRL omenjenih pravil, ki se uporablja na izvajalni ravni v stroju za sklepanje.

```
<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#WhiteBody"/>
      <swrl:argument1 rdf:resource="#dan"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#TradingDay"/>
      <swrl:argument1 rdf:resource="#dan"/>
    </swrl:ClassAtom>
    <swrl:DatavaluedPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#cenaZaprtja"/>
```

```
      <swrl:argument1 rdf:resource="#dan"/>
      <swrl:argument2 rdf:resource="#close"/>
    </swrl:DatavaluedPropertyAtom>
    <swrl:DatavaluedPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#cenaOdprtja"/>
      <swrl:argument1 rdf:resource="#dan"/>
      <swrl:argument2 rdf:resource="#open"/>
    </swrl:DatavaluedPropertyAtom>
    <swrl:BuiltinAtom>
      <swrl:builtin rdf:resource="swrlb:greaterThanOrEqual"/>
      <swrl:argument1 rdf:resource="#cenaZaprtja"/>
      <swrl:argument2 rdf:resource="#cenaOdprtja"/>
    </swrl:BuiltinAtom>
  </swrl:body>
</swrl:Imp>

<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BlackBody"/>
      <swrl:argument1 rdf:resource="#dan"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#TradingDay"/>
      <swrl:argument1 rdf:resource="#dan"/>
    </swrl:ClassAtom>
    <swrl:DatavaluedPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#cenaZaprtja"/>
      <swrl:argument1 rdf:resource="#dan"/>
      <swrl:argument2 rdf:resource="#close"/>
    </swrl:DatavaluedPropertyAtom>
    <swrl:DatavaluedPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#cenaOdprtja"/>
      <swrl:argument1 rdf:resource="#dan"/>
      <swrl:argument2 rdf:resource="#open"/>
    </swrl:DatavaluedPropertyAtom>
    <swrl:BuiltinAtom>
      <swrl:builtin rdf:resource="swrlb:lessThan"/>
      <swrl:argument1 rdf:resource="#cenaZaprtja"/>
      <swrl:argument2 rdf:resource="#cenaOdprtja"/>
    </swrl:BuiltinAtom>
  </swrl:body>
</swrl:Imp>
```

Izvorna koda 2: Pravili za določitev barve telesa svečnika v sintaksi SWRL

4.1.1.2.2 Zajem pravil z odločitveno tabelo

Pravila za določanje prodajnih in nakupnih signalov so naslednja:

- Če ima predhodni trgovalni dan belo telo svečnika in je trenutni dan »engulfing« vzorec svečnika z velikim črnim telesom, ki je danes dosegel najvišjo vrednost, potem gre za prodajni trgovalni signal, zaradi prisotnosti »engulfing« trgovalnega vzorca.

- Če ima predhodni trgovalni dan belo telo svečnika, ki je dosegel najvišjo vrednost in je trenutni dan »engulfing« vzorec svečnika z velikim črnim telesom, potem gre za prodajni trgovalni signal, zaradi prisotnosti »engulfing« trgovalnega vzorca.
- Če je predhodni trgovalni dan »doji« vzorec svečnika in je trenutni dan »engulfing« vzorec svečnika z velikim črnim telesom, ki je danes dosegel najvišjo vrednost, potem gre za prodajni trgovalni signal, zaradi prisotnosti »engulfing« trgovalnega vzorca.
- Če je predhodni trgovalni dan »doji« vzorec svečnika, ki je dosegel najvišjo vrednost in je trenutni dan »engulfing« vzorec svečnika z velikim črnim telesom, potem gre za prodajni trgovalni signal, zaradi prisotnosti »engulfing« trgovalnega vzorca.
- Če ima predhodni trgovalni dan črno telo svečnika in je trenutni dan »engulfing« vzorec svečnika z velikim belim telesom, ki je danes dosegel najnižjo vrednost, potem gre za nakupni trgovalni signal, zaradi prisotnosti »engulfing« trgovalnega vzorca.
- Če je predhodni trgovalni dan »doji« vzorec svečnika, ki je dosegel najnižjo vrednost in je trenutni dan »engulfing« vzorec svečnika z velikim belim telesom, potem gre za nakupni trgovalni signal, zaradi prisotnosti »engulfing« trgovalnega vzorca.

Omenjen paket pravil je primeren za vnos s pomočjo modula odločitvene tabele, saj se pogoji, ki se pojavljajo v telesu in glavi vseh pravil, izmenično pojavljajo, zato je za uporabnika najbolj enostavno obvladovanje te kompleksnosti v obliki odločitvene tabele (Slika 196). Vsak stolpec prikazuje določeno trditev v okviru problemske domene, kot kombinacijo izbranih pogojev, kjer se zgodijo izbrane posledice.

dan1 ima naslednji trgovalni dan dan2	✔	✔	✔	✔	✔	✔
dan1 je belo telo svečnika	✔	✔				
dan1 je črno telo svečnika					✔	
dan1 je doji			✔	✔		✔
dan2 je belo telo svečnika					✔	✔
dan2 je črno telo svečnika	✔	✔	✔	✔		
dan2 je veliko telo svečnika	✔	✔	✔	✔	✔	✔
dan2 je engulfing	✔	✔	✔	✔	✔	✔
dan2 je danes najvišja vrednost	✔		✔			
dan2 je včeraj najvišja vrednost		✔		✔		
dan2 je danes najnižja vrednost					✔	✔
dan2 je nakupni trgovalni dan					✔	✔
dan2 je prodajni trgovalni dan	✔	✔	✔	✔		
razlog za izvedbo akcije od dan2 je Prisotnost engulfing trgovalnega vzorca.	✔	✔	✔	✔	✔	✔

Slika 196: Zajem pravil za proženje prodajnih in nakupnih signalov

Izvorna koda 3 prikazuje abstraktno sintakso jezika SWRL za omenjena pravila, zajeta s pomočjo odločitvene tabele. Razvidno je, da je vsak stolpec v odločitveni tabeli implementiran kot enostavno pravilo, kar nam omogoča gradnjo kompleksnih odločitvenih tabel.

```
nextTradingDay(?dan1, ?dan2) ∧ WhiteBody(?dan1) ∧ Engulfing(?dan2) ∧
LargeBody(?dan2) ∧ BlackBody(?dan2) ∧ MHT(?dan2) →
SellTradingDay(?dan2)

nextTradingDay(?dan1, ?dan2) ∧ WhiteBody(?dan1) ∧ Engulfing(?dan2) ∧
LargeBody(?dan2) ∧ BlackBody(?dan2) ∧ MHY(?dan2) →
SellTradingDay(?dan2)

nextTradingDay(?dan1, ?dan2) ∧ Doji(?dan1) ∧ Engulfing(?dan2) ∧
LargeBody(?dan2) ∧ BlackBody(?dan2) ∧ MHT(?dan2) →
SellTradingDay(?dan2)

nextTradingDay(?dan1, ?dan2) ∧ Doji(?dan1) ∧ Engulfing(?dan2) ∧
LargeBody(?dan2) ∧ BlackBody(?dan2) ∧ MHY(?dan2) →
SellTradingDay(?dan2)

nextTradingDay(?dan1, ?dan2) ∧ BlackBody(?dan1) ∧ Engulfing(?dan2) ∧
LargeBody(?dan2) ∧ WhiteBody(?dan2) ∧ MLT(?dan2) →
BuyTradingDay(?dan2)

nextTradingDay(?dan1, ?dan2) ∧ Doji(?dan1) ∧ Engulfing(?dan2) ∧
LargeBody(?dan2) ∧ WhiteBody(?dan2) ∧ MLT(?dan2) →
BuyTradingDay(?dan2)
```

Izvorna koda 3: Pravila za proženje prodajnih in nakupnih signalov v abstraktni sintaksi SWRL

Izvorna koda 4 prikazuje sintakso SWRL omenjenih pravil, ki se uporablja na izvajalni ravni v stroju za sklepanje.

```
<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#SellTradingDay"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#nextTradingDay"/>
      <swrl:argument1 rdf:resource="#dan1"/>
      <swrl:argument2 rdf:resource="#dan2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Doji"/>
      <swrl:argument1 rdf:resource="#dan1"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Engulfing"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
```

```
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#LargeBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BlackBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#MHT"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:body>
</swrl:Imp>

<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#SellTradingDay"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#nextTradingDay"/>
      <swrl:argument1 rdf:resource="#dan1"/>
      <swrl:argument2 rdf:resource="#dan2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Doji"/>
      <swrl:argument1 rdf:resource="#dan1"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Engulfing"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#LargeBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BlackBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#MHY"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:body>
</swrl:Imp>

<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#SellTradingDay"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:head>
```

```
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#nextTradingDay"/>
      <swrl:argument1 rdf:resource="#dan1"/>
      <swrl:argument2 rdf:resource="#dan2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#WhiteBody"/>
      <swrl:argument1 rdf:resource="#dan1"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Engulfing"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#LargeBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BlackBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#MHT"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:body>
</swrl:Imp>

<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#SellTradingDay"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#nextTradingDay"/>
      <swrl:argument1 rdf:resource="#dan1"/>
      <swrl:argument2 rdf:resource="#dan2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#WhiteBody"/>
      <swrl:argument1 rdf:resource="#dan1"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Engulfing"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#LargeBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BlackBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
```

```
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#MHY"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:body>
</swrl:Imp>

<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BuyTradingDay"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="#nextTradingDay"/>
      <swrl:argument1 rdf:resource="#dan1"/>
      <swrl:argument2 rdf:resource="#dan2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#BlackBody"/>
      <swrl:argument1 rdf:resource="#dan1"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Engulfing"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#LargeBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#WhiteBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#MLT"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:body>
</swrl:Imp>

<swrl:Imp>
  <swrl:head rdf:parseType="Collection">
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="&b;BuyTradingDay"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:head>
  <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&b;nextTradingDay"/>
      <swrl:argument1 rdf:resource="#dan1"/>
      <swrl:argument2 rdf:resource="#dan2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Doji"/>
```

```
      <swrl:argument1 rdf:resource="#dan1"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#Engulfing"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#LargeBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#WhiteBody"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
    <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#MLT"/>
      <swrl:argument1 rdf:resource="#dan2"/>
    </swrl:ClassAtom>
  </swrl:body>
</swrl:Imp>
```

Izvorna koda 4: Pravila za proženje prodajnih in nakupnih signalov v sintaksi SWRL

4.1.2 *PODPRTI JEZIKI ZA ZAPIS ONTOLOGIJE*

Podprta jezika pri razvoju ontologije sta OWL in SWRL. Za zajem konceptov, relacij in aksiomov opisne logike (DL) se uporablja zapis OWL, medtem ko se pravila shranjujejo v obliki SWRL.

Uporabniku, ki uporablja IntelliOnto orodje in sledi procesu razvoja ontologije ROD, ni potrebno poznati podrobnosti obeh jezikov za zajem ontologij. Orodje je oblikovano do te mere, da v največji meri uporabniku poskuša pomagati pri razvoju in tako zmanjšati potrebo po tehničnem znanju, ki bi bilo drugače potrebno za načrtovanje ontologij. Uporabniku npr. ni potrebno poznati pojmov, kot so podrazred, ekvivalenten razred, razdružen razred, ekvivalentna lastnost, enostaven podatkovni tip ipd., zato ker obstajajo mehanizmi (poglavje 0), ki omogočajo identifikacijo teh omejitev glede na vnesene podatke in odgovore na vsebinska vprašanja (npr. identifikacija taksonomije v korakih 2.3 in 2.4 pristopa ROD).

Razvito ontologijo lahko po zaključku porazvojne faze izvozimo v različne funkcionalne komponente, kot je bilo predstavljeno v poglavju 3.2.3.2.3.

4.2 PRIMER TRGOVANJA NA FINANČNIH TRGIH

4.2.1 *METODA*

Model hitrega razvoja ontologij (ROD) je bil preverjen na problemski domeni trgovanja s finančnimi inštrumenti.

Zahteva je bila razvoj ontologije, ki vsebuje osnovne podatke o finančnih inštrumentih in trgovalnih strategijah s pomočjo procesa ROD. Rešitev naj bi uporabniku omogočala testiranje različnih trgovalnih strategij (vsaj treh) na dejanskih trgovalnih podatkih, ki so za izbrane finančne inštrumente na voljo v pripravljenih datotekah CSV, na voljo pa je tudi

spletni vir Yahoo Finance[52] s trenutnimi in zgodovinskimi podatki o trgovanju poljubnih finančnih inštrumentov na ameriških borzah. Razvita rešitev je morala biti predstavljena kot funkcionalna komponenta v obliki samostojne aplikacije J2SE z vhodnim parametrom simbola finančnega inštrumenta in izhodnimi parametri trgovalnih dni, kjer so se generirali nakupni ali prodajni signali skupaj z razlago za akcijo.

Uporabnik (dober poznavalec problemske domene z osnovnim znanjem računalništva) je opredelil ontologijo z zahtevanimi podatki o finančnih inštrumentih in vsaj tri trgovalne strategije, ki jih možno enostavno združevati in testirati samostojno ali v poljubni kombinaciji. Pri procesu gradnje so se nadzorovale akcije uporabnika. Zrno zbiranja podatkov je bil en korak iteracije pri razvoju ontologije, kjer nas je zanimala ocena popolnosti ontologije in število gradnikov ontologije (razredi, lastnosti in aksiomi, skupaj s pravili).

Pristop hitrega razvoja ontologije po procesu ROD (**pristop ROD**) je bil primerjan z **ad hoc pristopom** razvoja ontologij, ki je temeljil predvsem na obstoječih metodologijah CommonKADS, OTK in METHONTOLOGY. Pri razvoju ontologije po pristopu ROD je bil uporabljen predlagan model razvoja in orodji IntelliOnto ter Protégé.

Na koncu testiranja je bil rezultat v obliki razvite ontologije in funkcionalne komponente ter podatki o poteku samega procesa po posameznih iteracijah. Končno ontologijo je pregledal domenski ekspert ter potrdil vsebinsko ustreznost. Na tehnični ravni se je v končni različici pričakovala ontologija, ki je vsebovala od 250 do 350 aksiomov shematskega dela in številne primerke iz različnih virov.

4.2.2 *REZULTATI IN DISKUSIJA*

Ontologija FITS, ki je nastala v okviru razvoja po modelu ROD, je podrobneje predstavljena v prilogi (Priloga 1: Primer trgovanja s finančnimi inštrumenti). Pri gradnji ontologije smo pri obeh pristopih izhajali iz naslednjih enostavnih dejstev o trgovanju s finančnimi inštrumenti:

- s finančnim inštrumentom trgujemo na borzi in
- finančni inštrument se trguje na trgovalni dan.

Specializacija koncepta finančni inštrument je predstavljena z uvajanjem konceptov delnica, ETF in certifikat K.O. Podobno je podrobneje opredeljen tudi trgovalni dan. Uvedena sta nakup in prodaja, ki predstavljata trgovalne dneve, ko se pojavijo nakupni oz. prodajni signali. Vsi koncepti so v skladu s pravili indikatorja popolnosti ontologije opisani s formalnimi lastnostmi kot tudi opisi v naravnem jeziku.

Trgovalne strategije so opredeljene v ločenih ontologijah in so v obstoječo ontologijo vključene s pomočjo dinamičnega uvoza. Ta pristop je bil izbran zaradi zahteve po modularnosti in možnosti uporabe posameznih ali združenih strategij. Trgovalna strategija bi sicer vsebovala omejitve in pravila, medtem ko je pri bolj kompleksnih

[52] **Yahoo Finance** je spletni finančni portal in je na voljo na spletnem naslovu http://finance.yahoo.com.

opredelitvah (npr. trgovalna strategija japonskih svečnikov) potrebno dodati tudi dodatne koncepte. V okviru eksperimenta so nastale tri različne trgovalne strategije:

- naivna trgovalna strategija,
- trgovalna strategija na podlagi drsečega povprečja in
- trgovalna strategija japonskih svečnikov.

Vhodni podatki v ontologijo so bili trgovalni podatki delnic Apple Inc. (AAPL), Google Inc. (GOOG) in UltraShort S&P ProShares (SDS) v obdobju enega meseca. Primerki trgovalnih podatkov so bili v času izvajanja pridobljeni iz spletne strani Yahoo Finance in datotečnega sistema iz datotek CSV.

Proces gradnje ontologije in izvoza le-te v obliki funkcionalne komponente je prikazan na spodnjih grafih (Slika 197 in Slika 198). Grafa prikazujeta gibanje popolnosti ontologije in števila elementov v ontologiji glede na iteracije pri razvoju ontologije.

Slika 197: Ocena popolnosti ontologije in število elementov v ontologiji glede na posamezne iteracije in faze v procesu ROD

Slika 198: Ocena popolnosti ontologije in število elementov v ontologiji glede na posamezne iteracije v ad hoc procesu razvoja

Pri primerjavi obeh pristopov lahko pridemo do naslednjih izsledkov:

- za dosego zadanega cilja razvoja ontologije za trgovanje s finančnimi inštrumenti je po procesu ROD potrebno manj iteracij (30) kot pri ad hoc pristopu (37),
- ontologija je skozi celoten proces in tudi na koncu bolj popolna ter primerna za uporabo pri procesu ROD kot pri ad hoc pristopu, saj se napake preverjajo in opozarjajo uporabnika sproti,
- potrebna raven tehničnega znanja za zajem ontologije pri procesu ROD je manjša, saj so na voljo mehanizmi preslikave v obliki orodja IntelliOnto, ki to izboljšajo.

Pri procesu razvoja ontologije ROD je imel uporabnik stalno podporo indikatorja popolnosti ontologije (OC) in priporočil za napredovanje v naslednje faze razvoja. Ko je

uporabnik začel z določeno fazo razvoja ontologije in izvajal aktivnosti (poglavje 3.2) v okviru te faze, se je sproti izračunavala popolnost ontologije. Rezultati evaluacije so predstavljeni v grafični obliki procentualne vrednost (Slika 56), kar je podrobneje predstavljeno v poglavju 3.3. Medtem ko je ocena OC manjša od 100 %, uporabnik sledi navodilom in priporočilom za izboljšanje (Slika 56). Uporabnik ima na osnovi evaluacije na voljo številne podatke – osnovna statistika ontologije (število razredov, lastnosti, pravil, primerkov itd.), grafični pogled na indikator popolnosti ontologije in podrobnosti evaluacije, priporočila za izboljšavo ter zgodovina sprememb. Osnovni element je indikator popolnosti ontologije, ki je predstavljen v grafični obliki in prikazuje procentualno vrednost popolnosti ontologije od 0 % do 100 %. Sledijo priporočila za izboljšavo ontologije in njihovi vplivi (npr. odstrani ciklične napake (+10 %), opiši koncepte v naravnem jeziku (+8 %), poveži koncepte med seboj (+7 %) itd.). Ko izberemo določeno izboljšavo (npr. odstrani ciklične napake), se prikažejo podrobnosti akcije (vpliv, opravilo in podrobnosti). Slika 56 prikazuje, da lahko ciklične napake odstranimo tako, da odstranimo dvosmerno povezavo med konceptom Delnica in Finančni inštrument. S to akcijo pa pridobimo 10 % pri popolnosti ontologije.

Izboljšave in načrtovane akcije so jasno prikazane tudi na radarskem grafu (Slika 56). Zasenčeno območje z močnejšimi robovi predstavlja trenutno stanje, medtem ko rdeča točka predstavlja stanje, ki ga bomo dosegli, če izberemo predlagano izboljšavo.

Ko ocena popolnosti ontologije pade pod mejno vrednost (v našem eksperimentu je to 80 %, kot prikazuje Slika 197), sistem predlaga prehod v naslednjo fazo. V našem primeru lahko vidimo, da se je takšno priporočilo generiralo v 21. iteraciji, kjer se predlaga prehod iz faze 2.5 v fazo 2.6. Vrednost OC v 20. iteraciji je 76,5 %, medtem ko v naslednji, 21. iteraciji, znaša 91,3 %, kar je nad mejno vrednostjo 80 %.

Slika 197, kot smo že omenili, prikazuje gibanje popolnosti ontologije in števila elementov v ontologiji. Pri prehajanju med posameznimi fazami in koraki vidimo, da število elementov v ontologiji konstantno raste. Po drugi strani pa ocena popolnosti ontologije niha – raste dokler ne dosežemo mejne vrednosti za prehod v naslednjo fazo in strmo pade, ko dejansko začnemo z naslednjo fazo. Glede na priporočila uporabnik začne izvajati spremembe na ontologiji in popolnost ontologije začne ponovno rasti.

4.2.2.1 Podroben proces razvoja ontologije

Uporabnik je dobil zahteve, na podlagi katerih je moral zgraditi osnovno različico trgovalne strategije. Sledenje uporabniku je bilo izvedeno tako, da se je ob vsaki akciji shranjevanja ontologije končala trenutna iteracija razvoja, kjer se je izvedla tudi evaluacija razvite ontologije. Uporabnik je imel na voljo dva načina razvoja:

- ad hoc, brez upoštevanja priporočil in
- sledenje pristopu ROD ter upoštevanje priporočil OC (rezultat v prilogi (Priloga 1: Primer trgovanja s finančnimi inštrumenti)).

Pri procesu ROD je imel uporabnik ves čas na voljo pomoč indikatorja popolnosti (OC) ontologije, ki je podrobneje razdeljen na komponente, predstavljene v poglavju 3.3 (Slika 199).

- **D**escription → opis,
- **P**artition → razdelitev,
- **R**edundancy → odvečnost,
- **C**onsistency → doslednost in
- **A**nomaly → anomalija.

Slika 199: Ocena popolnosti ontologije v obliki radarskega grafa

V nadaljevanju bodo podrobneje predstavljene posamezne akcije v izbranih korakih procesa ROD in kako je OC vplival na sam razvoj z generiranjem priporočil za izboljšavo ontologije.

4.2.2.1.1 Zajem poslovnega slovarja (faza 2.1)

Prvi korak faze razvoja je zajem poslovnega slovarja, v katerem začnemo z naštevanjem konceptov in ad hoc relacij v obliki miselnega vzorca. Uporabnik enostavno začne ustvarjati neformalno opisan miselni vzorec o problemski domeni. Ukvarjamo se s podrobnostmi, kot so ime koncepta, opis v naravnem jeziku in sinonimi.

Začetno stanje pri razvoju je prazna ontologija, ki jo začne uporabnik razširjati (1. iteracija) z dodajanjem konceptov **finančni inštrument**, **borza** in **trgovalni dan**. Doda se tudi povezava med konceptoma finančni inštrument in borza, ki predstavlja **kotiranje finančnega inštrumenta na borzi**. Ko se omenjena iteracija konča, je uporabniku predstavljena evaluacija trenutno razvite ontologije *OC = 29 %*. Po pregledu priporočil in grafičnega prikaza popolnosti ontologije v obliki radarskega grafa ugotovimo, da sta pomanjkljivi komponenti opis (D) in razdelitev (P). Pri opisu smo pozabili na opis entitet v naravnem jeziku, medtem ko pri razdelitvi manjka povezava med vsemi koncepti v modelu. Slika 55 prikazuje, da se v fazi 2.1 v največji meri ukvarjamo z opisom ter tudi razdelitvijo.

V nadaljevanju uporabnik še naprej dodaja koncepte (**delnica**, **K.O. certifikat**, **ETF**, **nakupni trgovalni dan** in **prodajni trgovalni dan**) in njihove medsebojne povezave (finančni inštrument **se trguje na** borzi) ter skrbi za opis omenjenih entitet v naravnem jeziku, saj bodo ti podatki kasneje potrebni pri opredelitvi naprednejših omejitev ontologije.

V zadnjih dveh iteracijah (4. in 5.) je ocena popolnosti ontologije že presegla mejno vrednost 80 %. V tem primeru so bili doseženi potrebni pogoji za napredovanje v naslednjo razvojno fazo.

Slika 200: Iteracije faze 2.1

4.2.2.1.2 Seznam primerov konceptov in lastnosti (faza 2.2)

Ko uporabnik potrdi napredovanje v naslednjo fazo, se uteži pri vrednotenju popolnosti ontologije prilagodijo tej fazi. 1. iteracija v omenjeni fazi je vrednotenje ontologije iz zadnje iteracije prejšnje faze. Kot lahko vidimo, se je pri nespremenjeni ontologiji njena ocena popolnosti močno spremenila – OC v 5. iteraciji je bila *91,4 %*, medtem ko je v 6. iteraciji le *54,3 %*. Razlog je v dejstvu, da se v tej fazi ontologija vrednoti z dodatnimi semantičnimi testi, ki v prejšnji fazi niso bili potrebni, saj ni bilo na tem poudarka.

Slika 201: Iteracije faze 2.2

V omenjeni fazi razvoja je potrebno identificirati seznam primerkov konceptov in lastnosti. V prejšnjem koraku smo opredelili enostavno shemo problemske domene, medtem ko mora uporabnik v tem koraku podati dejanske primere konceptov in relacij (primerke razredov in lastnosti). Vsak koncept in tudi relacija morajo imeti opredeljenih

nekaj primerov iz realnega sveta. Rezultati tega koraka so zelo pomembni v naslednjih korakih 2.3 in 2.4, kjer se iz dejanskih primerov gradi taksonomija konceptov in relacij.

Analiza ocene popolnosti v iteracijah od 6 do 10 pokaže, da je poudarek na opisu ontologije, in sicer predvsem komponente ABox. To vključuje dodajanje primerkov razredov in njihovih medsebojnih povezav. Dodani so bili primerki **finančnega inštrumenta** (EEB, CF, FXI, AAPL, DIG, AA0UX4), **delnice** (CMP, KO, GOOG, ISRG, CF, AAPL), **ETF** (EEB, FXI, EWZ, ILF, DIG), **K.O. certifikata** (AA0UX4, ABN0CB, EB5DJF, CG88DJ), **borze** (LJSE, NYSE, NASDAQ, Stuttgart), **trgovalnega dne, trgovalnega dne s prodajnim signalom** in **trgovalnega dne z nakupnim signalom** (CMP @ 12.5.2009, DIG @ 23.5.2009, DIG @ 3.8.2009, AAPL @ 14.5.2009) itd. S tem zaporedoma izboljšujemo kakovost razvite ontologije vse do 10. iteracije, kjer presežemo mejno vrednost in to nam omogoča napredovanje v naslednjo fazo razvoja ontologije.

4.2.2.1.3 Identifikacija taksonomije (faza 2.3)

Tretji korak faze razvoja je identifikacija taksonomije glede na dejanske primerke iz prejšnjega koraka. Algoritem za identifikacijo taksonomije poskuša najti podrazrede, ekvivalentne razrede in razdružene razrede. Pregledajo se skupni primerki in glede na ugotovitve se uporabniku zastavijo enostavna vsebinska vprašanja.

S prehodom iz prejšnje faze je a priori popolnost ontologije v tej fazi 74,3 %. Na tem mestu s pomočjo indikatorja OC poskušamo izboljšati predvsem razdelitev (P). Ker imajo tehnično neizobraženi uporabniki pogosto težave pri opredelitvi taksonomije brez dodatne pomoči, jim pomagamo s klasifikacijo vnesenih primerkov. To opravilo se posledično močno poenostavi, saj se zgolj uporabijo primerki razredov iz prejšnje faze ter se odgovori na nekaj enostavnih vsebinskih vprašanj (Slika 104).

Slika 202: Iteracije faze 2.3

S tem podatkovno usmerjenim pristopom, na podlagi primerkov razredov, se predlagajo relacije podrazred, ekvivalenten razred in razdružen razred med posameznimi razredi. V omenjenem primeru so identificirani razredi **delnica**, **ETF** in **K.O. certifikat** kot podrazredi **finančnega inštrumenta** ter tudi **trgovalni dan s prodajnim signalom** in **trgovalni dan z nakupnim signalom** kot podrazreda **trgovalnega dne**. Na podlagi odgovorov na vsebinska vprašanja so razredi **delnica**, **ETF** in **K.O. certifikat** med seboj razdruženi.

4.2.2.1.4 Identifikacija ad hoc relacij (faza 2.4)

Korak faze razvoja identifikacija ad hoc relacij je zelo podoben prejšnjemu s to razliko, da je v tem koraku pod drobnogledom identifikacija podlastnosti in ekvivalentnih lastnosti. Pregledajo se skupni primerki in glede na ugotovitve se uporabniku zastavijo enostavna vsebinska vprašanja.

15. iteracija

99,3 %

Slika 203: Iteracija faze 2.4

Ker v omenjenem modeliranju problemske domene ne obstajajo podlastnosti ali ekvivalentne lastnosti, se popolnost ontologije iz prejšnjega koraka celo poviša na 99,3 % in nadaljevanje v naslednjo fazo razvoja je neposredno omogočeno.

4.2.2.1.5 Opis atributov in relacij konceptov (faza 2.5)

Korak opisa atributov in relacij konceptov je eden najpomembnejših. Do tega trenutku imamo opredeljeno enostavno ontologijo, zato mora uporabnik dodatno opisati koncepte z lastnostmi in omogočiti sprehod med posameznimi koncepti s pomočjo medsebojnih relacij. Pri opredelitvi vsake relacije se moramo vprašati, ali gre za **enostavno** ali **kompleksno relacijo**. Pri odločitvi za tip relacije moramo odgovoriti na vprašanje, ali relacija koncept povezuje z atomarnim podatkom (npr. leto, opis, število elementov ipd.) ali je sestavljena (npr. trgovalni dan), kar vodi do nadaljnje dekompozicije.

16. iteracija

95 %

→

17. iteracija

92 %

→

18. iteracija

96 %

→

19. iteracija

96,5 %

20. iteracija

100 %

→

21. iteracija

100 %

Slika 204: Iteracije faze 2.5

Kot pokaže analiza popolnosti ontologije v iteraciji od 16 do 21, se pomanjkljivosti pojavljajo predvsem v komponenti opisa (D) in razdelitve (P).

Uporabnik opredeli povezave med razredi in s podatkovnimi lastnostmi dodatno opiše razrede. Najprej uporabnik doda objektne lastnosti (kompleksne relacije) **finančni inštrument temelji na finančnem inštrumentu** in **trgovalni dan ima naslednji trgovalni dan** ter pripadajoče inverzne lastnosti, ki izboljšajo sprehajanje po ontologiji v obliki semantične mreže. Enostavne relacije ISIN, K.O. meja, lokacija, datum, cena odprtja, cena zaprtja, najnižja cena, najvišja cena, število poslov, razlog za akcijo se uporabijo za dodatno opisovanje razredov. Za popolnost 100 %, ki je dosežena v 21. Iteraciji, pa je potrebno relacije opremiti tudi z opisi v naravnem jeziku in njihovimi zalogami vrednosti.

4.2.2.1.6 Dodajanje kompleksnih omejitev in pravil (faza 2.6)

Zadnji korak pri fazi razvoja je dodajanje kompleksnih omejitev in pravil. Dodajamo lahko dva tipa pravil: **strukturna pravila** oz. **omejitve** in **operativna pravila**. V začetnih korakih (22. iteracija) uporabnik opredeli **enostavno trgovalno strategijo** z dodajanjem dveh pravil za kreiranje prodajnih in nakupnih signalov, in sicer na podlagi preverjanja tridnevne zaporedne rasti ali padca zaključnih cen. Namen 23. iteracije je predpriprava na opredelitev preostalih dveh trgovalnih strategij. Za **strategijo enostavnih drsečih povprečij** je uporabnik opredelil podatkovni lastnosti **kratka drseča sredina** in **dolga drseča sredina** ter ustrezni pravili za izračun. Pri bolj kompleksni **japonski trgovalni strategiji** je uporabnik dodal naslednje podporne razrede za opredelitev japonskih trgovalnih pravil: **japonski trgovalni vzorci**, **črno telo svečnika**, **belo telo svečnika**, »**doji**«, »**engulfing**«, **veliko telo svečnika**, **danes najvišja vrednost**, **včeraj najvišja vrednost**, **danes najnižja vrednost**.

22. iteracija *100 %* → **23. iteracija** *96 %* → **24. iteracija** *100 %* → **25. iteracija** *100 %*

Slika 205: Iteracije faze 2.6

V zadnjih iteracijah (24. in 25.) uporabnik doda vsa potrebna pravila v okviru strategije enostavnih drsečih povprečij (Slika 216) in japonske trgovalne strategije (Slika 217). Ocena popolnosti je skozi vse iteracije obravnavane faze zelo visoka, saj je uporabnik zgolj v nekaterih vmesnih korakih pozabil na opise omejitev v naravnem jeziku (pri dodajanju podpornih entitet za opredelitev trgovalnih pravil).

4.2.2.1.7 Povezava slovarja s podatki (faza 3.1)

Namen obravnavanega koraka je priprava smiselno povezanih ontologij, ki tvorijo funkcionalno celoto. Končni rezultat celotnega procesa prikazuje Slika 208, na kateri so prikazane vse vsebovane ontologije in njihova medsebojna povezanost. V začetnih korakih je uporabnik gradil ontologijo FI, z dodajanjem različnih trgovalnih strategij pa so nastale

še naslednje ontologije **FI-Strategy-Simple**, **FI-Strategy-SMA** in **FI-Strategy-Japanese**. Manjka še povezava razvitega slovarja s podatki iz različnih virov; uporabnik je pri omenjeni implementaciji uporabil tri različne vire:

- zgodovinske trgovalne podatke v obliki datotek CSV iz spletnega vira Yahoo Finance,
- trenutne trgovalne podatke neposredno iz HTML spletnih strani spletnega vira Yahoo Finance in
- trgovalne podatke iz programske paketa AmiBroker za finančno trgovanje.

Tako so nastale tri ontologije virov **FI-Yahoo-Finance-CSV**, **FI-Yahoo-Finance-HTML** in **FI-AmiBroker-AQH**.

Slika 206: Iteracije faze 3.1

V nadaljevanju si je uporabnik izbral naslednje finančne inštrumente: Google Inc., Apple Inc., PCX Corporation in Krka d. d., na podlagi katerih so ravno tako nastale ontologije posameznih finančnih inštrumentov, ki črpajo podatke iz izbranih virov; **FI-Yahoo-Finance-CSV-GOOG**, **FI-Yahoo-Finance-CSV-AAPL**, **FI-Yahoo-Finance-HTML-PCX** in **FI-AmiBroker-KRKG**.

Popolnost ontologije skozi celotno fazo ostane nad zahtevano spodnjo mejo (80 %), manjša nihanja se pojavijo predvsem zaradi dodajanja novih ontologij, vendar so le-te enostavne in obvladljive, proces ROD pa uporabniku pri integraciji tudi pomaga s čarovniki.

4.2.2.1.8 Kompozicija funkcionalne komponente (faza 3.2)

Zadnji korak porazvojnih aktivnosti je kompozicija funkcionalne komponente, ki jo nadalje sestavljajo podaktivnost opredelitve V/I, izbira konceptov, relacij, omejitev in pravil ter identifikacija cilja.

Na začetku je uporabnik preprosto izbral vhodne in izhodne parametre, ki so del funkcionalne komponente. Na vhodu je potrebno vnesti simbol finančnega inštrumenta, medtem ko na izhodu pričakujemo trgovalni dan s prodajnim ali nakupnim signalom, datum pojava in opis trgovalnega signala (Slika 50). Izbiro konceptov, relacij, omejitev in pravil je uporabnik prepustil algoritmu za predlaganje entitet in aksiomov za vključitev (Algoritem 1). Na koncu je cilj za funkcionalno komponento J2SE aplikacija, kot je zahtevano v metodi evaluacije.

5 ZAKLJUČEK

5.1 POVZETEK

Semantični splet je tistim uporabnikom, ki že poznajo različne načine modeliranja znanja, zelo blizu. Pojmi, kot so razred, podrazred, lastnost in primerek, so bili temeljni gradniki objektno usmerjenih in tudi ekspertnih sistemov. Pogosto se zgodi, da napredni uporabniki tehnologije semantičnega spleta vidijo kot pristope iz preteklosti in tako za njih ne prinašajo nobene dodane vrednosti. Prav gotovo so omenjene kritike na mestu, saj so načrtovalske strukture semantičnega spleta tesno povezane s predhodnimi pristopi in jeziki za opredelitev znanja. Vendar se je v tem času kljub temu nekaj spremenilo in je od zgodnjih časov objektno usmerjenega razvoja in ekspertnih sistemov imelo zelo velik vpliv na kulturo, poslovanje, trgovanje in izobraževanje. Pogosto se ta revolucionarna sprememba, **pojav svetovnega spleta**, po pomembnosti primerja celo z izumom tiska.

Splet se je razlikoval od informacijskih sistemov tako, da je združeval različne vire, dejansko iz celotnega sveta, in tako omogočal enostaven dostop. Zelo pomembna pri celotni zasnovi je bila njegova odprtost, saj je lahko vsebino prispeval kdorkoli in prav tako je vsak od tega imel tudi koristi. Prisotnost vseh potrebnih podatkov je bila bolj pomembna, kot kako dobro je bila organizirana neka majhna količina podatkov.

Semantični splet se od ekspertnih sistemov razlikuje po podobnem principu. V primerjavi s sistemi za predstavitev znanja, ki so bili razviti v kontekstu ekspertnih sistemov, je OWL zelo omejen; vendar je to za spletni jezik čisto primerno. Prednost semantičnega spleta je predvsem z vidika spleta. Celo jezik z zelo omejeno izraznostjo lahko prinese veliko dodano vrednost, ko ga uporabljamo s podatki virov s celotnega sveta. V okviru ekspertnih sistemov so tudi cilji semantičnega spleta zelo skromni. Ideja ekspertnega sistema je bila simulirati reševanje problemov z učinkovitostjo, podobno človeškemu strokovnjaku. Svetovni splet nas je zagotovo naučil, da uporabniki ne želijo računalnikov, ki bi delovali kot strokovnjaki, ampak želijo imeti dostop do podatkov, na katerih uporabijo svoje strokovno znanje, ko je seveda to potrebno.

Semantični splet to idejo razvije še korak dlje. Splet je učinkovit pri prikazovanju poljubnega vira spletnemu uporabniku, vendar če potrebnih podatkov ni na enem mestu, mora za integracijo poskrbeti uporabnik sam. Semantični splet ne uporablja tehnologije ekspertnih sistemov in ne posnema delovanje strokovnjaka, temveč uporablja to tehnologijo za zbiranje informacij, tako da ima posamezen uporabnik integriran dostop do spleta informacij. Pomembna lastnost, ki je prispevala k uspehu svetovnega spleta, je njegova odprtost. Vsakdo lahko prispeva podatke, tudi uporabniki, ki objavijo podatke z zavajajočo ali celo napačno vsebino. Na prvi pogled takšen pristop deluje zelo kaotično. Kako je lahko sploh uspešen? Uspeh spleta (in tudi specializiranih spletnih strani, kot je Wikipedia) je dokazal, da obstaja dovolj velik motiv za dodajanje kvalitetnih podatkov, kar posledično pomeni, da je splet v splošnem koristna in celo nepogrešljiva struktura.

Eden od pomembnih načrtovalskih načel na semantičnem spletu je način predstavitve podatkov, da so le-ti odzivni na spletno okolje. Temeljna načela, na katerih temelji

semantični splet so: **AAA slogan**, **predpostavka neenoličnega poimenovanja** in **predpostavka odprtega sveta**. Te omejitve so vodile tudi do glavnih načrtovalskih odločitev pri jezikih RDF, RDF/S in OWL.

Svetovni splet, s katerim imamo opravka vsakodnevno, sestavljajo predvsem dokumenti, ki jih prebirajo in obdelujejo uporabniki pri brskanju po spletu. Vendar za mnoge spletne strani velja, da v podatkovnih bazah skrivajo veliko več podatkov, kot pa jih je prikazanih na strani. Da te podatke predstavimo v globalni in integrirani obliki, moramo na nek način opredeliti, kako so podatki na enem mestu povezani s podatki na drugem. Modeli na semantičnem spletu imajo ravno to povezovalno vlogo, ki opisujejo povezave med podatki iz različnih virov.

V okviru doktorske disertacije smo se osredotočili na vidike opredelitve ontologij, ki so osnovni gradniki semantičnega spleta. Po pregledu obstoječih pristopov smo ugotovili, da so le-ti preveč zapleteni in povsem neprimerni za uporabnike, ki nimajo ustreznega tehničnega znanja in znanja načrtovanja ontologij. Namen raziskave, predstavljene v omenjeni doktorski disertaciji, je bil v opredelitvi modela za hiter razvoj ontologij, ki bo primeren ravno za takšen profil uporabnikov.

5.2 Rezultati

Na podlagi raziskovalnih vprašanj, podanih v uvodnem poglavju 1.2, smo prišli do izsledkov in zaključkov ter konkretnih prispevkov znanosti, ki so povzeti v tekočem poglavju.

Pri pregledu primernih pristopov za hiter razvoj delov poslovnih aplikacij, ki temeljijo na ontologijah, je bilo najprej najpomembnejše raziskovalno vprašanje, ali so obstoječi pristopi in metodologije zadostni pri celostni podpori uporabe ontologij v poslovnih aplikacijah. Kot smo ugotovili s podrobnim pregledom obstoječih pristopov in metodologij za razvoj ontologij v poglavju 2.4 na strani 33, so obstoječi pristopi glede podpore uporabnikom brez tehnološkega znanja in znanja načrtovanja ontologij zelo omejeni, saj večina pristopov potrebuje vmesni člen v obliki uporabnika, ki znanje poslovnega uporabnika zapiše v formalni obliki z jeziki semantičnega spleta. Ugotovitve so tudi pokazale, da obstoječi pristopi niso zadostni pri celostni podpori uporabe ontologij v poslovnih aplikacijah, saj ne podpirajo celotnega življenjskega cikla razvoja ontologij, proces razvoja pa se ponavadi konča z zadnjo uspešno iteracijo. Prav tako ni podpore porazvojnim aktivnostim. Sprejetost ontologij pri razvoju poslovnih aplikacij je prav tako nezadostna, saj je področje semantičnega spleta in uporabe ontologij še nezrelo, zato je v gospodarstvu malo uspešnih implementacij, vendar se stanje izboljšuje. Pristop ROD naslavlja raziskovalno vprašanje tudi o načinu in ravni, kjer lahko ontologijo uporabimo v poslovnih aplikacijah. Pri pristopu ROD se predlaga uporaba razvite ontologije v obliki funkcionalne komponente, kot je podrobneje predstavljeno v poglavju 3.2.3.2.3 na strani 75.

Eden od raziskovalnih izzivov je bila tudi raziskava, ali lahko uporaba tehnologij semantičnega spleta zmanjša potrebno raven tehničnega znanja uporabnikov pri opredelitvi problemske domene in zajemu znanja. Mera, do katere lahko poslovni

uporabnik brez ustreznih tehničnih znanj pripravi ontologijo, primerno za uporabo v poslovni aplikaciji, je prikazana na praktičnem primeru ontologije FITS za trgovanje s finančnimi inštrumenti, katere razvoj je podrobno predstavljen v poglavju 4.2 na strani 179, končni rezultat pa v prilogi (Priloga 1: Primer trgovanja s finančnimi inštrumenti). V okviru omenjenega raziskovalnega vprašanja smo se spraševali tudi, kako doseči raven abstrakcije, kjer poslovni uporabnik vnaša domensko znanje v čim bolj neformalizirani, naravnemu jeziku podobni obliki, hkrati pa želimo doseči zadostno stopnjo formalizacije, da lahko na izvajalni ravni smiselno uporabimo mehanizme sklepanja. S pomočjo pristopa ROD, predstavljenem v poglavju 3.2 na strani 52, je uporabniku omogočena opredelitev problemske domene od zajema poslovnega slovarja do kompozicije funkcionalne komponente. Podrobnosti neposredne podpore uporabniku pa so zajete v okviru prototipa orodja IntelliOnto, ki je predstavljen v poglavju 4.1 na strani 142.

Ukvarjali smo se tudi s problemom izboljšave gradnje ontologij na semantičnem spletu s pristopom obvladovanja poslovnih pravil pri razvoju informacijskih sistemov. Ugotovili smo, da skupne točke in podobnosti ter tudi razlike med področjem semantičnega spleta in poslovnimi pravili obstajajo in so podrobneje predstavljene v poglavju 2.3.2 na strani 29. Vpeljava pristopa obvladovanja poslovnih pravil pri razvoju informacijskih sistemov ter aplikacija na hiter proces razvoja ontologij je predstavljena v poglavju 3.2.2.1.1 na strani 55 in poglavju 3.2.2.6 na strani 63.

Z odgovori na omenjena raziskovalna vprašanja, ki so povezana z delovnimi hipotezami, predstavljenimi v nadaljevanju, smo posredno potrdili tudi omenjene hipoteze. V naslednjih podpoglavjih bodo zato te hipoteze povzete in podrobneje predstavljene.

5.2.1 *HIPOTEZA 1*

Na podlagi obstoječih metodologij in najboljših praks pri razvoju ontologij ter pristopov za obvladovanje poslovnih pravil je možno opredeliti model za hiter razvoj ontologij, ki je primeren za uporabnike brez znanja načrtovanja ontologij in poglobljenega tehničnega znanja.

Osnovni cilj doktorske disertacije, predstavljen v uvodnem poglavju 1.2, je opredeliti model za hiter razvoj ontologij, ki je primeren za uporabnike brez poglobljenega tehničnega znanja razvoja in načrtovanja ontologij. Opredelitev omenjenega modela je zajeta v poglavju 3, kjer je poudarek na procesnem vidiku, za lažjo sledljivost procesu razvoja pa je v poglavju 3.3 predstavljen indikator popolnosti ontologije, ki ima vlogo pomoči uporabnikom in poenostavitve prehajanja med posameznimi koraki procesa ROD. Za podporo hitremu razvoju ontologij je v poglavju 4.1 predstavljen tudi prototip orodja IntelliOnto.

5.2.2 *HIPOTEZA 2*

S pomočjo vpeljave nenehnega preverjanja ustreznosti razvite ontologije izboljšamo hkratno podporo procesu razvoja in izboljšamo končne izdelke.

Ideja nenehnega preverjanja ustreznosti ontologije je temeljila na obstoječih pristopih za evaluacijo ontologij, vendar se omenjena aktivnost pri procesu ROD izvaja nenehno ob

vsaki iteraciji. Implementacija omenjenega pristopa je realizirana v obliki indikatorja popolnosti ontologije OC, ki je podrobneje predstavljen v poglavju 3.3. Z uporabo omenjene metrike je učinek dvojen, in sicer (1) pomoč pri hitrejšem in lažjem napredovanju med posameznimi koraki procesa ROD in (2) izboljšana kakovost razvite ontologije, saj uporabnika že sproti opozarja na napake. Prednost indikatorja OC je v njegovi dinamični naravi izračunavanja, saj se v vsakem koraku procesa ROD razvoja prilagodi, tako da se uteži nastavijo trenutnim potrebam. Posledično to pomeni, da v začetnih fazah pri zajemu poslovnega slovarja ni potrebe po preverjanju odvečnih primerkov, obstoju neuporabljenih konceptov ali inverznih lastnosti, temveč je poudarek na opisu komponent TBox in RBox v naravnem jeziku in obstoju povezave med vsemi koncepti.

5.2.3 *HIPOTEZA 3*

S pomočjo pristopa ROD je mogoče izdelati aplikacijo semantičnega spleta, ki je neodvisna od tehnologije in izvajalne platforme ter uporablja podatke iz različnih virov.

Za potrditev aplikabilnosti pristopa ROD in indikatorja OC je bil omenjen proces preverjen na problemski domeni trgovanja s finančnimi inštrumenti na finančnih trgih ter je podrobneje predstavljen v poglavju 4.2. Primerjava je bila izvedena med pristopom ROD in ad hoc pristopom, kjer se je spremljala rast števila aksiomov in popolnost ontologije skozi posamezne iteracije. Rezultati so pokazali, da je bila omenjena problemska domena s procesom ROD implementirana 23 % hitreje oz. je bilo potrebno 7 iteracij manj, kot pri ad hoc pristopu. Prav tako je bil opazen bolj konstanten trend kakovosti ontologije, saj se je le-ta pri pristopu ROD konvergirala k 100 %, medtem ko je pri ad hod pristopu močno alternirala. Rezultat razvoja je predstavljen v prilogi (Priloga 1: Primer trgovanja s finančnimi inštrumenti), v kateri so vse podrobnosti ontologije FITS. Omenjeno ontologijo sestavljajo številne medsebojno povezane ontologije, ki so vse zajete v standardnih jezikih semantičnega spleta (OWL, SWRL), zato je poskrbljeno tudi za neodvisnost od izvajalne platforme in stroja za sklepanje. Razvita ontologija je popolnoma funkcionalna in primerna za uporabo v aplikaciji semantičnega spleta, saj črpa podatke iz ostalih spletnih virov, ki so dostopni preko vmesnikov. Ravno tako pa je možno neposredno dodajanje drugih virov semantičnega spleta. Pri problemski domeni finančnega trgovanja pa je dosežena tudi zelo velika stopnja modularnosti, saj lahko uporabnik poljubno dodaja svoje interpretacije pravil za finančno trgovanje, medtem ko lahko temelji tudi na obstoječih.

5.2.4 *HIPOTEZA 4*

Prototip orodja za podporo procesu ROD omogoča uporabniku sledenje korakom procesa za hiter razvoj in nenehno evaluacijo razvite ontologije.

Prototip orodja IntelliOnto je bil razvit za potrebe lažje in hitrejše uporabe hitrega pristopa za razvoj ontologij (ROD). Gre za spletno aplikacijo, ki omogoča vnos naprednih omejitev ontologije na dva načina: (1) sprehod po semantični mreži in dodajanje omejitev (poglavje 4.1.1.1.1) in (2) vnos omejitev v obliki atomov pravil (poglavje 4.1.1.1.2). Pri sprehodu po semantični mreži uporabniku omogočimo pregled obstoječe baze znanja in neposredne manipulacije z obstoječim znanjem ter možnost dodajanja novih aksiomov v

uporabniku prijazni obliki. Vnos omejitev v obliki atomov pravil pa predstavlja vmesnik, ki omogoča dodajanje poslovnih pravil problemske domene. V okviru uporabniškega vmesnika je vključen tudi indikator popolnosti, ki nam v obliki radarskega grafa sproti prikazuje popolnost razvite ontologije in nam pomaga s priporočili, kaj lahko naredimo, da izboljšamo našo ontologijo. Indikator OC je povezan tudi s samim procesom razvoja, saj nam glede na popolnost razvite ontologije predlaga prehod v naslednjo fazo razvoja in nas skozi celoten proces razvoja tudi vodi.

5.3 PRISPEVKI ZNANOSTI

Na podlagi predstavljenih rezultatov v predhodnem poglavju podajamo strnjen pregled prispevkov omenjene doktorske disertacije:

1. **Inovativen model za hiter razvoj ontologij (ROD)**, ki je primeren za uporabnike brez podrobnega tehničnega znanja in znanja načrtovanja ontologij. V okviru procesa razvoja so podprte predrazvojne, razvojne in porazvojne aktivnosti, od zajema poslovnega slovarja do uporabe razvite ontologije v obliki funkcionalne komponente v informacijskem sistemu.
2. **Indikator popolnosti ontologije (OC)**, ki se uporablja za lažje sledenje korakom procesa ROD in skrbi za nenehno evaluacijo razvite ontologije ter z dinamičnim prilagajanjem uteži v izračunu uporabniku pomaga s priporočili, s katerimi lahko le-ta izboljša ontologijo.
3. **Prototip orodja IntelliOnto** za boljšo podporo procesu ROD, ki predstavlja grafični vmesnik k razvoju ontologij in neposredno vključuje proces ROD ter indikator popolnosti ontologije (OC).
4. Razvita **ontologija FITS** za podporo trgovanju s finančnimi inštrumenti, ki s svojo modularno zasnovo omogoča ponovno uporabo.
5. **Identifikacija načinov uporabe funkcionalnih komponent ontologije** v informacijskih sistemih, kjer je možna uporaba celotne razvite ontologije ali samo shematskega dela (TBox + RBox) v različnih primerih uporabe.
6. **Opredelitev vmesnikov za neposredno povezovanje delno strukturiranih virov** v ontologijo, kjer lahko s pomočjo generičnega pristopa regularnih izrazov v ontologijo vključimo poljuben delno strukturiran vir; od dokumentov, spletnih strani, podatkovnih baz, datotek CSV ipd.

5.4 MOŽNOSTI ZA NADALJNJE DELO

5.4.1 *RAZŠIRITEV INDIKATORJA POPOLNOSTI ONTOLOGIJE*

Indikator popolnosti ontologije v trenutni različici podpira 31 semantičnih testov, ki so razdeljeni v naslednje kategorije: opis (10), razdelitev (10), odvečnost (6), doslednost (1) in anomalija (4). Za boljše rezultate bi bilo potrebno dodati še dodatne semantične teste, predvsem v kategoriji doslednosti. Zaradi modularne zasnove indikatorja popolnosti je omenjeno opravilo zelo enostavno izvedljivo, saj z dodatni testi zgolj prilagodimo obstoječe uteži in uporabnik pri razvoju ne opazi razlik v podpori, pojavijo se zgolj dodatne možnosti priporočil izboljšanja ontologije.

5.4.2 *IZBOLJŠANJE PODPORE ORODJA*

Trenutna podpora razvoju je v obliki prototipa orodja IntelliOnto, ki vključuje osnovne funkcionalnosti in temelji na spletnem okolju. Za bolj popolno podporo bi bilo v orodje potrebno dodati vse funkcionalnosti pristopa ROD, saj lahko le na ta način popolnoma podpremo hiter razvoj ontologij, ki je primeren za uporabnike brez naprednega tehničnega znanja in predhodnega znanja načrtovanja ontologij. Smiselno bi bilo tudi podpreti najbolj uporabljano orodje za razvoj ontologij Protégé in za to okolje pripraviti razširitveni zavihek, ki bi omogočal podporo procesu ROD kar znotraj omenjenega orodja. V obeh primerih je zagotovo ključnega pomena vključitev podpore indikatorja popolnosti ontologije, ki je v trenutni različici orodja IntelliOnto že prisotna.

5.4.3 *INTEGRACIJA MODELA ZA HITER RAZVOJ ONTOLOGIJ S SOCIALNIMI OMREŽJI*

V uspeh svetovnega spleta prav gotovo ne moremo dvomiti, pri prehodu na t. i. splet 2.0 pa se je zgodil zanimiv premik v smeri dodajanja novih vsebin, saj so predvsem tehnično neizobraženi uporabniki prevzeli pobudo pri soustvarjanju vsebine na svetovnem spletu s pojavom različnih storitev socialnega spleta, kot so forumi, spletni dnevniki, spletna mesta za druženje, storitve takojšnjega sporočanja, souporaba multimedijskih vsebin ipd.

Slika 207: Vpliv socialnih omrežij

Z vodenim pristopom k opredelitvi določenih problemskih domen lahko s pomočjo folksonomij že dosežemo dobre rezultate, ideja za nadaljnje delo na omenjeni doktorski disertaciji pa je prenos pristopa za hiter razvoj ontologij na socialni splet. S pripravo aplikacije za najbolj priljubljena socialna omrežja, kot sta npr. Facebook in LinkedIn, bi lahko dosegli kritično maso uporabnikov, ki praviloma niso tehnično dobro izobraženi, lahko pa bi s pomočjo pristopa ROD razvijali aplikacije semantičnega spleta.

5.4.4 *INTEGRACIJA PRISTOPA S PRISTOPI SAMODEJNEGA UČENJA ONTOLOGIJ*

Pristop predstavljen v omenjeni doktorski disertaciji, temelji na načinu opredelitve ontologije, v katerem ima osrednjo vlogo uporabnik, ki prenese znanje v formalizirano obliko. Obstajajo pa tudi številni pristopi za samodejno učenje ontologij, ki temeljijo na metodah umetne inteligence in odkrivanja zakonitosti v podatkih. Za prihodnje izboljšave omenjenega pristopa bi bilo zagotovo koristno vključiti omenjene pristope v proces ROD, kar bi ga še dodatno poenostavilo in približalo poslovnim uporabnikom. Vloga metod samodejnega učenja ontologij bi bila zgolj podporne narave, saj bi pristop še vedno temeljil na poslovnem uporabniku, ki bi opredelil problemsko domeno, te omenjene metode pa bi mu zgolj olajšale delo.

LITERATURA

ABOU-ZEID, E. S., 2003, What can ontologists learn from knowledge management? *Journal of Computer Information Systems*, **43**, pp. 109-117.

ALLEMANG, D. & HENDLER, J., 2008, *Semantic Web for Working Ontologist: Effective Modeling in RDFS and OWL* (Elsevier).

ALVARADO, M., ROMERO-SALCEDO, M. & SHEREMETOV, L., 2004, A corporative memory based on the user profiles. *Expert Systems with Applications*, **26**, pp. 87-94.

AVISON, D. & FITZGERALD, G., 2003, *Information Systems Development: Methodologies, Techniques and Tools, 3rd edition* (Maidenhead, UK: McGraww-Hill).

BACLAWSKI, K., KOKAR, M. M., WALDINGER, R. & KOGUT, P. A., 2002, Consistency Checking of Semantic Web Ontologies. In *1st International Semantic Web Conference on The Semantic Web*, Sardinia, Italy.

BAJEC, M. & KRISPER, M., 2005, A methodology and tool support for managing business rules in organisations. *Information Systems*, **30**, pp. 423-443.

BAUMEISTER, J. & SEIPEL, D., 2005, Smelly Owls - Design Anomalies in Ontologies. In *18th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2005)*, USA, Clearwater Beach.

BECHHOFER, S. & GOBLE, C., 2001, Thesaurus construction through knowledge representation. *Data & Knowledge Engineering*, **37**, pp. 25-45.

BELLIFEMINE, F., CAIRE, G., TRUCCO, T., RIMASSA, G. & MUNGENAST, R., 2006, JADE Administrator's guide. Available online at: http://jade.tilab.com/doc/administratorsguide.pdf).

BERNARAS, A., LARESGOITI, I. & CORERA, J., 1996, Building and resusing ontologies for electrical network applications. In *European Conference on Artificial Intelligence (ECAI '96)*.

BLOEHDORN, S., HAASE, P., SURE, Y. & VOELKER, J., 2005, D.6.6.1 Report on the integration of ML, HLT and OM, pp. 37.

BRAMBILLA, M., CELINO, I., CERI, S. & CERIZZA, D., 2006, A Software Engineering Approach to Design and Development of Semantic Web Service Applications. In *5th International Semantic Web Conference*, Athens, USA.

BRANK, J., GROBELNIK, M. & MLADENIĆ, D., 2005, A Survey of Ontology evaluation techniques. In *Conference on Data Mining and Data Warehouses (SiKDD)*, Slovenia, Ljubljana.

BREWSTER, C., ALANI, H., DASMAHAPATRA, S. & WILKS, Y., 2004, Data driven Ontology evaluation. In *International Conference on Language Resources and Evaluation*, Portugal, Lisbon.

CAPLINSKAS, A., LUPEIKIENE, A. & VASILECAS, O., 2003, The role of ontologies in reusing domain and enterprise engineering assets. *Informatica*, **14**, pp. 455-470.

CHAKRABARTI, S., SRIVASTAVA, S., SUBRAMANYAM, M. & TIWARI, M., 2000, Using Memex to archive and mine community Web browsing experience. *Computer Networks*, **33**, pp. 669-684.

COLE, C., MANDELBLATT, B. & STEVENSON, J., 2002, Visualizing a high recall search strategy output for undergraduates in an exploration stage of researching a term paper. *Information Processing & Management*, **38**, pp. 37-54.

CORCHO, O., FERNANDEZ-LOPEZ, M. & GOMEZ-PEREZ, A., 2003, Methodologies, tools and languages for building ontologies. Where is their meeting point? *Data & Knowledge Engineering*, **46**, pp. 41-64.

CUI, Z., JONES, D. & O'BRIEN, P., 2001, Issues in Ontology-based Information Integration. In *E-Business & The Intelligent Web*, Seattle, USA.

DACONTA, M. C., OBRST, L. J. & SMITH, K. T., 2003, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management* (Indianapolis, USA: John Wiley & Sons).

DAHLGEN, K., 1995, A linguistic ontology. *International Journal of Human-Computer Studies*, **43**, pp. 809-818.

DATE, C. J., 2000, *What Not How: The Business Rules Approach to Application Development* (Reading, USA: Addison Wesley).

DAVIES, J., STUDER, R. & WARREN, P., 2006, *Semantic Web Technologies - trends and research in ontology-based systems* (Chichester, England: John Wiley & Sons).

DENNY, M., 2002, Ontology Building: A Survey of Editing Tools *XML.com*.

DONG, Y. & LI, M. S., 2004, HyO-XTM: a set of hyper-graph operations on XML Topic Map toward knowledge management. *Future Generation Computer Systems*, **20**, pp. 81-100.

ĐURIĆ, D., GAŠEVIĆ, D. & DEVEDŽIĆ, V., 2005, Ontology Modeling and MDA. *Journal of Object Technology*, **4**.

ERDMANN, M. & STUDER, R., 1999, Ontologies as conceptual models for XML documents. In *Modeling and Management (KAW'99), 12th Workshop for Knowledge Acquisition*, Banff, Canada.

FAHAD, M. & QUADIR, M. A., 2008, Ontological errors - Inconsistency, Incompleteness and Redundancy. In *International Conference on Enterprise Information Systems (ICEIS) 2008*, Barcelona, Spain.

FENSEL, D., 2001, *Ontologies: Silver bullet for Knowledge Management and Electronic Commerce* (Berlin, Germany: Springer-Verlag).

FERNANDEZ-LOPEZ, M., GOMEZ-PEREZ, A., SIERRA, J. P. & SIERRA, A. P., 1999, Building a chemical ontology using Methontology and the Ontology Design Environment. *Intelligent Systems*, **14**.

FIRAT, A., 2003, Information Integration Using Contextual Knowledge and Ontology Merging *Sloan School of Management*, pp. 151 (Massachusetts Institute of Technology).

FRANKEL, D. S., 2003, *Model Driven Architecture: Applying MDA to Enterprise Computing* (Wiley).

FURNAS, G. W., 1986, Generalized Fisheye Views, Human Factors in Computing Systems. In *ACM Conference on Human Factors in Computing Systems (CHI'98)*.

GALLAIRE, H., MINKER, J. & NICOLAS, J. M., 1984, Logic and Databases. *Computational surveys*, **16**.

GANGEMI, A., CATENACCI, C., CIARAMITA, M. & LEHMANN, J., 2006, Modelling ontology evaluation and validation. In *3nd European Semantic Web conference (ESWC 2006)*.

GÓMEZ-PÉREZ, A., 1999, Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases. In *12th Banff Workshop on Knowledge Acquisition, Modelling, and Management (KAW '99)*.

GÓMEZ-PÉREZ, A., 2002, OntoWeb - Deliverable 1.3 - A survey on ontology.

GOMEZ-PEREZ, A., FERNANDEZ-LOPEZ, M. & CORCHO, O., 2003, Ontological Engineering. *Advanced Information and Knowledge Processing*.

GREENSPAN, S. J., MYLOPOULOS, J. & BORGIDA, A., 1982, Capturing more world knowledge in the requirements specification. In *International Conference on Software Engineering*, Tokyo, Japan.

GRUBER, T. R., 1993, A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, **5**, pp. 199-220.

GUARINO, N., 1998, Formal Ontology and Information Systems. In *FOIS'98*, Trento, Italy.

GUPTA, J. N. D. & SHARMA, S. K., 2003, *Intelligent Enterprises of the 21st Century* (IGI Global).

HAASE, P., HARMELEN, F. V., ZHISHENG, H., STUCKENSCHMIDT, H. & SURE, Y., 2005, A Framework for handling inconcistency in changing ontologies. In *4th International Semantic Web conference (ISWC 2005)*, Ireland, Galway.

HAASE, P. & STOJANOVIC, L., 2005, Consistent evolution of OWL ontologies. In *2nd European Semantic Web conference (ESWC 2005)*, Greece, Heraklion.

HARTEL, F. W., DE CORONADO, S., DIONNE, R., FRAGOSO, G. & GOLBECK, J., 2005, Modeling a description logic vocabulary for cancer research. *Journal of Biomedical Informatics*, **38**, pp. 114-129.

HARTMANN, J., SURE, Y., GIBOIN, A. et al., 2004, D1.2.3 Methods for ontology evaluation.

HEFLIN, J. & HENDLER, J., 2000, Searching the web with SHOE *Artificial Intelligence for Web Search*, pp. 36-40 (Menlo Park, USA, AAAI Press).

HOLSAPPLE, C. W. & JOSHI, K. D., 2002, A collaborative approach to ontology design. *Communications of the ACM*, **45**, pp. 42-47.

HORROCKS, I., ANGELE, J., DECKER, S. et al., 2003, Where are the rules? *IEEE Intelligent Systems*, **18**, pp. 76-83.

INŠTITUT ZA SLOVENSKI JEZIK FRANA RAMOVŠA, 2002, Slovar slovenskega knjižnega jezika.

JARRAR, M. & MEERSMAN, R., 2002, Formal ontology engineering in DOGMA approach. In *International Conference on Ontologies, Databases and Applications of Semantics (ODBase 02)*.

JOVANOVIĆ, J. & GAŠEVIĆ, D., 2005, Achieving knowledge interoperability: An XML/XSLT approach. *Expert Systems with Applications*, **29**, pp. 535-553.

KARDASIS, P. & LOUCOPOULOS, P., 2004, Expressing and organising business rules. *Information and Software Technology*, **46**, pp. 701-718.

KESSELER, M., 1996, A schema based approach to HTML authoring. *World Wide Web Journal*, **96**.

KOTIS, K. & VOUROS, G., 2003, Human centered ontology management with HCONE. In *IJCAI '03 Workshop on Ontologies and Distributed Systems*.

LABROU, Y. & FININ, T. W., 1999, Yahoo! as an ontology: Using Yahoo! categories to describe documents. In *1999 ACM CIKM International Conference on Information and Knowledge Management*, Kansas City, USA.

LAVBIČ, D., 2005, Data semantics and ontologies. *Uporabna informatika*, **13**, pp. 121-135.

LAVBIČ, D., ŠAŠA, A. & KRISPER, M., 2006, Knowledge Management in Multi-Agent Systems. In *Knowledge Management in Organisations (KMO'2006)*, Maribor, Slovenia.

LENART, D. B. & GUHA, R. V., 1989, *Building large Knowledge-Based Systems* (Boston, USA: Addison-Wesley Longman Publishing).

LOZANO-TELLO, A. & GÓMEZ-PÉREZ, A., 2004, ONTOMETRIC: A method to choose the appropriate ontology. *Journal of Database Management*, **15**, pp. 1-18.

MAEDCHE, A. & STAAB, S., 2002, Measuring Similarity between Ontologies. In *European Conference on Knowledge Acquisition and Management (EKAW 2002)*, Spain, Madrid.

MARGOLIS, B. & SHARPE, J., 2007, *SOA for Business Developer: Concepts, BPEL, and SCA* (MC Press).

MILLER, G. A., 1995, WordNet: a lexical database for English. *Communications of the ACM*, **38**, pp. 39-41.

MORGENSTERN, M., 1983, Active Database as a paradigm for enhanced computing environments. In *9th International Conference on Very Large Databases*, Florence, Italy.

MYLOPOULOS, J., 1998, Information modeling in the time of the revolution. *Information Systems*, **23**, pp. 127-155.

NICOLA, A. D., NAVIGLI, R. & MISSIKOFF, M., 2005, Building an eProcurement ontology with UPON methodology. In *15th e-Challenges Conference*, Ljubljana, Slovenia.

NIJSSEN, G. M. & HALPIN, T. A., 1989, *Conceptual Schema and Relational Database Design - A Fact Oriented Approach* (New York, USA: Prentice Hall).

NOY, N. F., GUHA, R. & MUSEN, M. A., 2005, User ratings of ontologies: Who will rate the raters? In *Symposium on Knowledge Collection from Volunteer Contributors*, USA, Stanford.

NOY, N. F. & MCGUINNESS, D. L., 2001, Ontology development 101: A guide to creating your first ontology (Stanford Knowledge Systems Laboratory).

OLSEVICOVA, K., 2006, Development of Topic Maps E-Learning Portal. *Electronic Journal of e-Learning*, **4**, pp. 59-66.

OMG, 2008, Semantics of Business Vocabulary and business Rules (SBVR), v1.0. Available online at: http://www.omg.org/spec/SBVR/1.0/).

PARK, J. & HUNTING, S., 2002, *XML Topic Maps: Creating and Using Topic Maps for the Web* (Boston, USA: Addison-Wesley).

PASSIN, T. B., 2004, *Explorer's Guide to the Semantic Web* (Greenwich, London: Manning Publicastions).

PORZEL, R. & MALAKA, R., 2004, A Task-based Approach for Ontology Evaluation. In *ECAI 2004, Workshop on Ontology Learning and Population*, Valencia, Spain.

PRINCETON UNIVERSITY, 2006, WordNet: A lexical database for the English language.

QUINE, W. V. O., 1963, *From a Logical Point of View - 9 Logico-Philosophical Essays* (New York, USA: Harper & Row).

RAO, J., DIMITROV, D., HOFMANN, P. & SADEH, N., 2006, A Mixed Initiative Semantic Web Framework for Process Composition. In *5th International Semantic Web Conference*, Athens, USA.

SANJUAN, E. & IBEKWE-SANJUAN, F., 2006, Text mining without document context. *Information Processing & Management*, **42**, pp. 1532-1552.

SARKAR, M. & BROWN, M. H., 1994, Graphical Fisheye Views. *Communications of the ACM*, **37**, pp. 73-84.

SCHREIBER, G., AKKERMANS, H., ANJEWIERDEN, A. et al., 1999, *Knowledge Engineering and Management - The CommonKADS Methodology* (London, England: The MIT Press: Cambridge, Massachusetts).

SCHWEIGER, R., HOELZER, S., RUDOLF, D., RIEGER, J. & DUDECK, J., 2003, Linking clinical data using XML topic maps. *Artificial Intelligence in Medicine*, **28**, pp. 105-115.

SEIDENBERG, J. & RECTOR, A., 2006, Web Ontology Segmentation: Analysis, Classification and Use. In *15th Internation World Wide Web Conference*, May, Edinburgh, Scotland.

SENICA, B., 2008, Primerjava metodologij in orodij za razvoj in uporabo ontologij *Laboratorij za informatiko*, pp. 95 (Ljubljana, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani).

SPENCE, R. & APPERLEY, M. D., 1982, Data Base Navigation: An Office Environment for the Professional. *Behaviour & Information Technology*, **1**, pp. 43-54.

SPREEUWENBERG, S. & GERRITS, R., 2005, Business Rules in the Semantic Web, are there any or are they different? In *Industry forum*.

STAAB, S., BRAUN, C., BRUDER, I. et al., 1999, A system for facilitating and enhancing web search. In *International Working Conference on Artificial and Natural Neural Networks: Engineering Applications of Bio-Inspired Artificial Neural Networks (IWANN '99)*.

STAAB, S., SCHNURR, H. P., STUDER, R. & SURE, Y., 2001, Knowledge process and ontologies. *IEEE Intelligent Systems*, **16**.

STRUCK, D. L., 1999, Business rule continuous requirements environment (Colorado Springs, Colorado, Colorado Technical University).

SURE, Y., 2003, Methodology, Tools & Case Studies for Ontology based Knowledge Management *Institute AIFB*, pp. 332 (University of Karlsruhe).

SWARTOUT, B., PATIL, R., KNIGHT, K. & RUSS, T., 1997, Toward distributed use of large-scale ontologies. In *Symposium on Ontological Engineering of AAAI*, Stanford, USA.

SWARTOUT, B., RAMESH, P., KNIGHT, K. & RUSS, T., 1997, Towards distributed use of largescale ontologies. In *Symposium on Ontological Engineering of AAAI*, Stanford, USA.

TANAKA, A. K., 1992, On conceptual design of active databases (Georgia, USA, Georgia Institute of Technology).

USCHOLD, M. & GRUENINGER, M., 1996, Ontologies: principles, methods and applications. *Knowledge Sharing and Review*, **11**.

USCHOLD, M. & KING, M., 1995, Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing (IJCAI '95)*, Montreal, Canada.

VALARAKOS, A. G., KARKALETSIS, V., ALEXOPOULOU, D. et al., 2006, Building an allergens ontology and maintaining it using machine learning techniques. *Computers in Biology and Medicine*, **36**, pp. 1155-1184.

VEALE, T., 2006, An analogy-oriented type hierarchy for linguistic creativity. *Knowledge-Based Systems*, **19**, pp. 471-479.

VÖLKER, J., VRANDECIC, D. & SURE, Y., 2005, Automatic Evaluation of Ontologies (AEON). In *International Semantic Web Conference*.

VRANDEČIČ, D. & GANGEMI, A., 2006, Unit tests for ontologies. In *1st International Workshop on Ontology content and evaluation in Enterprise*, Montpellier, France.

W3C, 2000, Semantic Web development - Technical proposal. Available online at: http://www.w3.org/2000/01/sw/DevelopmentProposal).

W3C, 2003, Semantic Web Activity Statement. Available online at: http://www.w3.org/2001/sw/Activity).

WARE, C., 2004, *Information visualization: Perception for design, 2nd edition* (San Francisco, USA: Morgan Kaufmann).

WATERSON, A. & PREECE, A., 1999, Verifying ontological commitment in knowledge-based systems. *Knowledge-Based Systems*, **12**, pp. 45-54.

WELTY, C. & GUARINO, N., 2001, Supporting ontological analysis of taxonomic relationships. *Data & Knowledge Engineering*, **39**, pp. 51-74.

WIDOM, J. & CERI, S., 1996, *Active Database Systems - Triggers and Rules for Advanced Database Processing* (San Francisco, USA: Morgan Kaufmann).

WIEDERHOLD, G., 1992, Mediators in the architecture of future information systems. *IEEE Computer*, **25**, pp. 38-49.

WOOLDRIDGE, M., 2002, *An Introduction to MultiAgent Systems* (Chichester, England: John Wiley & Sons).

YU, A. C., 2006, Methods in biomedical ontology. *Journal of Biomedical Informatics*, **39**, pp. 252-266.

PRILOGA 1: PRIMER TRGOVANJA S FINANČNIMI INŠTRUMENTI

Problemsko domeno trgovanja s finančnimi inštrumenti sestavljajo številne ontologije (FITS) (Slika 208), ki so med seboj povezane na takšen način, da si lahko vsak uporabnik poljubno sestavi svoj pogled na finančno trgovanje.

Slika 208: Povezanost ontologij pri trgovanju s finančnimi inštrumenti

Osnovni koncepti, s katerimi se srečamo pri finančnem poslovanju, so predstavljeni v ontologiji **FI** (http://amor.fri.uni-lj.si/dejan/SW/FI.owl) (Slika 209).

Slika 209: Jedrna ontologija, v kateri so predstavljeni osnovni koncepti finančnega trgovanja

Za vsak finančni inštrument, ki ga želimo v sistemu uporabiti, pripravimo ontologijo elementa vira. Slika 210 prikazuje ontologijo za delnico Google Inc. (GOOG) **FI-Yahoo-Finance-CSV-GOOG** (http://amor.fri.uni-lj.si/dejan/SW/FI/FI-Yahoo-Finance-CSV-GOOG.owl), katere podatke pridobivamo iz vira Yahoo Finance in v obliki datoteke CSV.

Slika 210: Ontologija za delnico Google Inc. (GOOG) iz vira Yahoo Finance v obliki datoteke CSV

Podobno kot prejšnji primer je ontologija za delnico Apple Inc. (AAPL) **FI-Yahoo-Finance-CSV-AAPL** (http://amor.fri.uni-lj.si/dejan/SW/FI-Yahoo-Finance-CSV-AAPL.owl) (Slika 211), kjer podatke prav tako pridobivamo iz vira Yahoo Finance in v obliki datoteke CSV.

Slika 211: Ontologija za delnico Apple Inc. (AAPL) iz vira Yahoo Finance v obliki datoteke CSV

Ko imamo zbrane vse ontologije virov, jih s pomočjo ontologije vira združimo v celoto (Slika 212). Tu je prikazana ontologija vira Yahoo Finance v obliki datoteke CSV **FI-Yahoo-Finance-CSV** (http://amor.fri.uni-lj.si/dejan/SW/FI-Yahoo-Finance-CSV.owl). Naloga te ontologije je, da združi vse finančne inštrumente določenega vira in jih pripravi za uporabo v jedrni ontologiji.

Slika 212: Ontologija vira Yahoo Finance v obliki datoteke CSV

Naslednja ontologija (Slika 213) je zopet ontologija elementa vira in velja za delnico Krka d. d. (KRKG) **FI-AmiBroker-AQH-KRKG** (http://amor.fri.uni-lj.si/dejan/SW/FI-AmiBroker-AQH-KRKG.owl), kjer podatke pridobivamo iz AmiBroker aplikacije v zapisu AQH.

Slika 213: Ontologija za delnico Krka d. d. (KRKG) iz vira AmiBroker v obliki zapisa AQH

Slika 214 prikazuje ontologijo vira AmiBroker v obliki zapisa AQH **FI-AmiBroker-AQH** (http://amor.fri.uni-lj.si/dejan/SW/FI-Yahoo-Finance-CSV.owl).

Slika 214: Ontologija vira AmiBroker v AQH zapisu

Za zapis določene strategije trgovanja se uporablja ločena ontologija, predvsem z namenom omogočanja modularne zasnove, saj si lahko uporabnik izbere in nadgradi poljubno strategijo. Slika 215 prikazuje enostavno trgovalno strategijo **FI-Strategy-Simple** (http://amor.fri.uni-lj.si/dejan/SW/FI-Strategy-Simple.owl). Pravila v tej ontologiji zahtevajo, da finančni inštrument kupimo, ko se zaključna cena v treh zaporednih dneh poviša, in ga prodamo, ko zaključna cena v treh zaporednih dneh pade.

Slika 215: Enostavna trgovalna strategija

Bolj smiselna je strategija enostavnih drsečih povprečij **FI-Strategy-SMA** (http://amor.fri.uni-lj.si/dejan/SW/FI-Strategy-SMA.owl) (Slika 216). V tej ontologiji je opredeljeno krajše in daljše drseče povprečje. V primeru preseka obeh drsečih povprečij se generira signal za trgovanje. Če krajše drseče povprečje prečka daljše drseče povprečje, potem gre za prodajo in v obratnem primeru za nakup.

Slika 216: Trgovalna strategija enostavnih drsečih povprečij

Zadnja od trgovalnih strategij je najbolj zahtevna, in sicer gre za trgovalno strategijo japonskih svečnikov **FI-Strategy-Japanese** (http://amor.fri.uni-lj.si/dejan/SW/FI-Strategy-Japanese.owl). Pri tej trgovalni strategiji so opredeljeni številni vzorci trgovalnih dni, ki v primeru, da se pojavijo v določenem zaporedju, nakazujejo trgovalni signal.

Slika 217: Trgovalna strategija japonskih svečnikov

ONTOLOGIJA FI

RAZREDI

FI:**StockExchangeMarket**

```
  rdfs:label          "Borza@si"
                      "Stock exchange market@en"
  rdfs:comment        "Market for trading stock and their derivates
                      at agreed price.@en"
```

FI:**TradingDay**

```
  rdfs:label          "Trgovalni dan@si"
                      "Trading day@en"
  rdfs:comment        "Trgovalni podatki v okviru trgovalnega
                      dne.@si"
```

FI:**BuyTradingDay**

```
  rdfs:label          "Nakupni trgovalni dan@si"
                      "Buy trading day@en"
  rdfs:comment        "Trgovalni dan z nakupnim signalom@si"
                      "Trading day with buy signal@en"
  rdfs:subClassOf     FI:TradingDay
```

```
FI:SellTradingDay
  rdfs:label        "Prodajni trgovalni dan@si"
                    "Sell trading day@en"
  rdfs:comment      "Trgovalni dan s prodajnim signalom@si"
                    "Trading day with sell signal@en"
  rdfs:subClassOf   FI:TradingDay

FI:FinancialInstrument
  rdfs:label        "finančni inštrument@si"
                    "financial instrument@en"
  rdfs:comment      "Poljubna oblika investiranja.@si"
                    "Any form of funding medium.@en"

FI:ETF
  rdfs:label        "ETF@si"
                    "ETF@en"
  rdfs:comment      "Sklad, s katerim se trguje na borzi.@si"
                    "Exhange traded fund.@en"
  rdfs:subClassOf   FI:FinancialInstrumet

FI:KOCertificate
  rdfs:label        "K.O. certifikat@si"
                    "K.O. certificate@en"
  rdfs:comment      "Špekulativen finančni inštrument, vezan na
                    nek drug finančni inštrument.@si"
                    "Speculative financial instrument, based on
                    underlying instrument.@en"
  rdfs:subClassOf   FI:FinancialInstrument

FI:InterestRateInstrument
  rdfs:label        "Interest rate instrument@en"
  rdfs:subClassOf   FI:FinancialInstrument

FI:EquityRelatedInstrument
  rdfs:label        "Equity related instrument@en"
  rdfs:subClassOf   FI:FinancialInstrument
  owl:disjointWith  FI:InterestRateInstrument

FI:Deposit
  rdfs:label        "Depozit@si"
                    "Deposit@en"
  rdfs:subClassOf   FI:InterestRateInstrument
```

```
FI:Loan
  rdfs:label        "Posojilo@si"
                    "Loan@en"
  rdfs:subClassOf   FI:InterestRateInstrument
  owl:disjointWith  FI:Deposit

FI:Stock
  rdfs:label        "Delnica@si"
                    "Stock@en"
  rdfs:comment      "Lastniški vrednostni papir@si"
                    "Share of ownership in a corporation@en"
  rdfs:subClassOf   FI:EquityRelatedInstrument

FI:EquityLinkedDerivate
  rdfs:label        "Equity linked derivate@en"
  rdfs:subClassOf   FI:EquityRelatedInstrument

FI:ForwardTypeDerivate
  rdfs:label        "Forward type derivate@en"
  rdfs:subClassOf   FI:EquityLinkedDerivate

FI:OptionTypeDerivate
  rdfs:label        "Option type derivate@en"
  rdfs:subClassOf   FI:EquityLinkedDerivate
```

LASTNOSTI

```
FI:location
  rdf:type       owl:DatatypeProperty
                 owl:FunctionalProperty
  rdfs:label     "lokacija@si"
                 "location@en"
  rdfs:comment   "Lokacija, kjer se nahaja borza (npr. Slovenija,
                 ZDA, Kitajska).@si"
                 "Location of stock exchange market (e.g.
                 Slovenia, USA, China).@en"
  rdfs:domain    FI:StockExchangeMarket
  rdfs:range     xsd:string

FI:symbol
  rdf:type       owl:DatatypeProperty
                 owl:FunctionalProperty
  rdfs:label     "simbol@si"
                 "symbol@en"
  rdfs:domain    FI:FinancialInstrument
  rdfs:range     xsd:string
```

FI:**ISIN**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"ISIN@si" "ISIN@en"
rdfs:comment	"Enolična oznaka vrednostnega papirja.@si" "International Securities Identification Number.@en"
rdfs:domain	FI:FinancialInstrument
rdfs:range	xsd:string

FI:**name**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"ime@si" "name@en"
rdfs:domain	FI:FinancialInstrument
rdfs:range	xsd:string

FI:**tradedAt**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
rdfs:label	"kotira na@si" "traded at@en"
rdfs:comment	"Finančni inštrument kotira na borzi.@si" "Financial instrument is traded at stock exchange market.@en"
rdfs:domain	FI:FinancialInstrument
rdfs:range	FI:StockExchangeMarket

FI:**hasTraded**

rdf:type	owl:ObjectProperty
rdfs:label	"ima kotirajoč@si" "has traded@en"
rdfs:comment	"Borza ima kotirajoč finančni inštrument.@si" "Stock exchange market has traded financial instrument.@en"
owl:inverseOf	FI:tradedAt

FI:**isBasedOn**

rdf:type	owl:ObjectProperty
rdfs:label	"temelji na@si" "is based on@en"
rdfs:comment	"ETF ali K.O. certifikat temelji na finančnem inštrumentu.@si" "ETF or K.O. certificate is based on financial instrument.@en"
rdfs:domain	FI:ETF FI:KOCertificate
rdfs:range	FI:FinancialInstrument

FI:**isPartOf**

rdf:type	owl:ObjectProperty
rdfs:label	“je del @si” “is part of @en”
rdfs:comment	“Finančni inštrument je del ETF-a oz. K.O. certifikata.@si” “Financial instrument is part of ETF or K.O. certificate.@en”
rdfs:domain	FI:FinancialInstrument
rdfs:range	FI:ETF FI:KOCertifikat
owl:inverseOf	FI:isBasedOn

FI:**KOBarrier**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"K.O. mejo@si" "K.O. barrier@en"
rdfs:comment	"knock-out meja@si" "knock-out barrier@en"
rdfs:domain	FI:KOCertificate
rdfs:range	xsd:float

FI:**validFor**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
rdfs:label	"velja za@si" "is valid for@en"
rdfs:comment	"Trgovalni dan velja za finančni inštrument.@si" "Trading day is valid for financial instrument.@en"
rdfs:domain	FI:TradingDay
rdfs:range	FI:FinancialInstrument

FI:**isTradedOn**

rdf:type	owl:ObjectProperty
rdfs:label	"se trguje na@si" "is traded on@en"
rdfs:comment	"Finančni inštrument se trguje na trgovalni dan.@si" "Financial instrument is traded on trading day.@en"
rdfs:domain	FI:FinancialInstrument
rdfs:range	FI:TradingDay
owl:inverseOf	FI:validFor

FI:**next**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
rdfs:label	"ima naslednji@si" "has next@en"
rdfs:comment	"Trgovalni dan ima naslednji trgovalni dan.@si" "Trading day has next trading day.@en"
rdfs:domain	FI:TradingDay
rdfs:range	FI:TradingDay

FI:**previous**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
rdfs:label	"ima prejšnji@si" "has previous@en"
rdfs:comment	"Trgovalni dan ima prejšnji trgovalni dan.@si" "Trading day has previous trading day.@en"
rdfs:domain	FI:TradingDay
rdfs:range	FI:TradingDay
owl:inverseOf	FI:next

FI:**date**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"datum@si" "date@en"
rdfs:domain	FI:TradingDay
rdfs:range	xsd:date

FI:**open**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"cena ob odprtju@si" "open price@en"
rdfs:comment	"Cena vrednostnega papirja ob prvem sklenjenem poslu.@si" "Financial instrument's price at first trade.@en"
rdfs:domain	FI:TradingDay
rdfs:range	xsd:float

FI:**close**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"cena ob zaprtju@si" "close price@en"
rdfs:comment	"Cena vrednostnega papirja ob zadnjem sklenjenem poslu.@si" "Financial instrument's price at last trade.@en"
rdfs:domain	FI:TradingDay
rdfs:range	xsd:float

FI:**low**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"najnižja cena@si" "low price@en"
rdfs:comment	"Najnižja cena vrednostnega papirja pri sklenjenem poslu.@si" "Lowest financial instrument's price at trade.@en"
rdfs:domain	FI:TradingDay
rdfs:range	xsd:float

FI:**high**

rdf:type	owl:DatatypeProperty owl:FunctionalProperty
rdfs:label	"najvišja cena@si" "high price@en"
rdfs:comment	"Najvišja cena vrednostnega papirja pri sklenjenem poslu.@si" "Highest financial instrument's price at trade.@en"
rdfs:domain	FI:TradingDay
rdfs:range	xsd:float

FI:**volume**

```
rdf:type        owl:DatatypeProperty
                owl:FunctionalProperty
rdfs:label      "število poslov@si"
                "volume@en"
rdfs:comment    "Število sklenjenih poslov.@si"
rdfs:domain     FI:TradingDay
rdfs:range      xsd:float
```

FI:**tradeReason**

```
rdf:type        owl:DatatypeProperty
rdfs:label      "razlog za izvedbo akcije@si"
                "trade reason@en"
rdfs:comment    "Razlog za izvedbo akcije nakupa ali
                prodaje.@si"
                "Reason for trading.@en"
rdfs:domain     FI:BuyTradingDay
                FI:SellTradingDay
rdfs:range      xsd:string
```

PRIMERKI

FI:**LJSE**

```
rdfs:label   "Ljubljana Stock Exchange"
FI:location  "Slovenia"
```

FI:**NASDAQ**

```
rdfs:label   "NASDAQ Stock Exchange"
FI:location  "USA"
```

FI:**NYSE**

```
rdfs:label   "New York Stock Exchange"
FI:location  "USA"
```

FI:**STE**

```
rdfs:label   "Boerse Stuttgart"
FI:location  "Germany"
```

ONTOLOGIJA FI-YAHOO-FINANCE-CSV-GOOG

RAZREDI

FI-Yahoo-Finance-CSV-GOOG:**Row**

LASTNOSTI

FI-Yahoo-Finance-CSV-GOOG:**next**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
rdfs:domain	FI-Yahoo-Finance-CSV-GOOG:Row
rdfs:range	FI-Yahoo-Finance-CSV-GOOG:Row

FI-Yahoo-Finance-CSV-GOOG:**previous**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
owl:inverseOf	FI-Yahoo-Finance-CSV-GOOG:next

FI-Yahoo-Finance-CSV-GOOG:**date**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:date

FI-Yahoo-Finance-CSV-GOOG:**open**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:float

FI-Yahoo-Finance-CSV-GOOG:**close**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:float

FI-Yahoo-Finance-CSV-GOOG:**low**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:float

FI-Yahoo-Finance-CSV-GOOG:**high**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:float

FI-Yahoo-Finance-CSV-GOOG:**volume**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:float

ONTOLOGIJA FI-YAHOO-FINANCE-CSV-AAPL

RAZREDI

FI-Yahoo-Finance-CSV-AAPL:**Row**

LASTNOSTI

FI-Yahoo-Finance-CSV-AAPL:**next**

```
  rdf:type       owl:ObjectProperty
                 owl:FunctionalProperty
  rdfs:domain    FI-Yahoo-Finance-CSV-AAPL:Row
  rdfs:range     FI-Yahoo-Finance-CSV-AAPL:Row
```

FI-Yahoo-Finance-CSV-AAPL:**previous**

```
  rdf:type       owl:ObjectProperty
                 owl:FunctionalProperty
  owl:inverseOf  FI-Yahoo-Finance-CSV-AAPL:next
```

FI-Yahoo-Finance-CSV-AAPL:**date**

```
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:date
```

FI-Yahoo-Finance-CSV-AAPL:**open**

```
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float
```

FI-Yahoo-Finance-CSV-AAPL:**close**

```
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float
```

FI-Yahoo-Finance-CSV-AAPL:**low**

```
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float
```

FI-Yahoo-Finance-CSV-AAPL:**high**

```
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float
```

FI-Yahoo-Finance-CSV-AAPL:**volume**

```
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float
```

ONTOLOGIJA FI-YAHOO-FINANCE-CSV

RAZREDI

FI-Yahoo-Finance-CSV-GOOG:Row
rdfs:subClassOf FI:TradingDay

FI-Yahoo-Finance-CSV-AAPL:Row
rdfs:subClassOf FI:TradingDay

LASTNOSTI

FI-Yahoo-Finance-CSV-GOOG:next
rdfs:subPropertyOf FI:next

FI-Yahoo-Finance-CSV-GOOG:previous
rdfs:subPropertyOf FI:previous

FI-Yahoo-Finance-CSV-GOOG:date
rdfs:subPropertyOf FI:date

FI-Yahoo-Finance-CSV-GOOG:open
rdfs:subPropertyOf FI:open

FI-Yahoo-Finance-CSV-GOOG:close
rdfs:subPropertyOf FI:close

FI-Yahoo-Finance-CSV-GOOG:low
rdfs:subPropertyOf FI:low

FI-Yahoo-Finance-CSV-GOOG:high
rdfs:subPropertyOf FI:high

FI-Yahoo-Finance-CSV-GOOG:volume
rdfs:subPropertyOf FI:volume

FI-Yahoo-Finance-CSV-AAPL:next
rdfs:subPropertyOf FI:next

FI-Yahoo-Finance-CSV-AAPL:previous
rdfs:subPropertyOf FI:previous

FI-Yahoo-Finance-CSV-AAPL:date
rdfs:subPropertyOf FI:date

FI-Yahoo-Finance-CSV-AAPL:open
rdfs:subPropertyOf FI:open

FI-Yahoo-Finance-CSV-AAPL:close
rdfs:subPropertyOf FI:close

FI-Yahoo-Finance-CSV-AAPL:low
rdfs:subPropertyOf FI:low

FI-Yahoo-Finance-CSV-AAPL:high
rdfs:subPropertyOf FI:high

FI-Yahoo-Finance-CSV-AAPL:volume
rdfs:subPropertyOf FI:volume

Pravila

FI-Yahoo-Finance-CSV-GOOG:Row(?day)
→ FI:isTradedOn(GOOG, ?day)

FI-Yahoo-Finance-CSV-AAPL:Row(?day)
→ FI:isTradedOn(AAPL, ?day)

Ontologija FI-AmiBroker-AQH-KRKG

Razredi

FI-AmiBroker-AQH-KRKG:**Row**

Lastnosti

FI-AmiBroker-AQH-KRKG:**next**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
rdfs:domain	FI-AmiBroker-AQH-KRKG:Row
rdfs:range	FI-AmiBroker-AQH-KRKG:Row

FI-AmiBroker-AQH-KRKG:**previous**

rdf:type	owl:ObjectProperty owl:FunctionalProperty
owl:inverseOf	FI-AmiBroker-AQH-KRKG:next

FI-AmiBroker-AQH-KRKG:**date**

rdf:type	owl:DatatypeProperty
rdfs:range	xsd:date

```
FI-AmiBroker-AQH-KRKG:open
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float

FI-AmiBroker-AQH-KRKG:close
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float

FI-AmiBroker-AQH-KRKG:low
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float

FI-AmiBroker-AQH-KRKG:high
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float

FI-AmiBroker-AQH-KRKG:volume
  rdf:type       owl:DatatypeProperty
  rdfs:range     xsd:float
```

ONTOLOGIJA FI-AMIBROKER-AQH

RAZREDI

```
FI-AmiBroker-AQH-KRKG:Row
  rdfs:subClassOf  FI:TradingDay
```

LASTNOSTI

```
FI-AmiBroker-AQH-KRKG:next
  rdfs:subPropertyOf  FI:next

FI-AmiBroker-AQH-KRKG:previous
  rdfs:subPropertyOf  FI:previous

FI-AmiBroker-AQH-KRKG:date
  rdfs:subPropertyOf  FI:date

FI-AmiBroker-AQH-KRKG:open
  rdfs:subPropertyOf  FI:open

FI-AmiBroker-AQH-KRKG:close
  rdfs:subPropertyOf  FI:close

FI-AmiBroker-AQH-KRKG:low
  rdfs:subPropertyOf  FI:low
```

FI-AmiBroker-AQH-KRKG:high

```
rdfs:subPropertyOf  FI:high
```

FI-AmiBroker-AQH-KRKG:volume

```
rdfs:subPropertyOf  FI:volume
```

PRAVILA

FI-AmiBroker-AQH-KRKG:Row(?day)

```
→  FI:isTradedOn(KRKG, ?day)
```

ONTOLOGIJA FI-STRATEGY-SIMPLE

PRAVILA

```
FI:nextTradingDay(?day1, ?day2) ∧ FI:nextTradingDay(?day2,
?day3) ∧
FI:close(?day1, ?close1) ∧ FI:close(?day2, ?close2) ∧
FI:close(?day3, ?close3) ∧ swrlb:greaterThan(?close3, ?close2) ∧
swrlb:greaterThan(?close2, ?close1)
  → FI:BuyTradingDay(?day3) ∧
    FI:tradeReason(?day3, "Today is the third consucetive day
    where the close price has risen.")
```

```
FI:nextTradingDay(?day1, ?day2) ∧ FI:nextTradingDay(?day2,
?day3) ∧
FI:close(?day1, ?close1) ∧ FI:close(?day2, ?close2) ∧
FI:close(?day3, ?close3) ∧ swrlb:lessThan(?close3, ?close2) ∧
swrlb:lessThan(?close2, ?close1)
  → FI:SellTradingDay(?day3)  ∧
    FI:tradeReason(?day3, "Today is the third consucetive day
    where the close price has fallen.")
```

ONTOLOGIJA FI-STRATEGY-SMA

LASTNOSTI

FI-Strategy-SMA:longSMA

```
rdf:type     owl:DatatypeProperty
             owl:FunctionalProperty
rdfs:label   "daljša drseča sredina@si"
             "long SMA@en"
rdfs:domain  FI:TradingDay
rdfs:range   xsd:float
```

FI-Strategy-SMA:shortSMA

```
rdf:type      owl:DatatypeProperty
              owl:FunctionalProperty
rdfs:label    "kratka drseča sredina@si"
              "short SMA@en"
rdfs:domain   FI:TradingDay
rdfs:range    xsd:float
```

PRAVILA

```
FI:next(?day1, ?day2) ∧ FI:next(?day2, ?day3) ∧
FI:next(?day3, ?day4) ∧ FI:next(?day4, ?day5) ∧
FI:close(?day1, ?close1) ∧ FI:close(?day2, ?close2) ∧
FI:close(?day3, ?close3) ∧ FI:close(?day4, ?close4) ∧
FI:close(?day5, ?close5) ∧
KAON2:ifTrue("($1 + $2 + $3 + $4 + $5)/float(5)", close1,
close2, close3, close4, close5, sSMA)
   →    FI-Strategy-SMA:shortSMA(?day5, ?sSMA)

FI:next(?day1, ?day2) ∧ FI:next(?day2, ?day3) ∧
FI:next(?day3, ?day4) ∧ FI:next(?day4, ?day5) ∧
FI:next(?day5, ?day6) ∧ FI:next(?day6, ?day7) ∧
FI:next(?day7, ?day8) ∧ FI:next(?day8, ?day9) ∧
FI:next(?day9, ?day10) ∧
FI:close(?day1, ?close1) ∧ FI:close(?day2, ?close2) ∧
FI:close(?day3, ?close3) ∧ FI:close(?day4, ?close4) ∧
FI:close(?day5, ?close5) ∧ FI:close(?day6, ?close6) ∧
FI:close(?day7, ?close7) ∧ FI:close(?day8, ?close8) ∧
FI:close(?day9, ?close9) ∧ FI:close(?day10, ?close10) ∧
KAON2:ifTrue("($1 + $2 + $3 + $4 + $5 + $6 + $7 + $8 + $9 +
$10)/float(10)", close1, close2, close3, close4, close5, close6,
close7, close8, close9, close10, sSMA)
   →    FI-Strategy-SMA:longSMA(?day10, ?lSMA)

FI:nextTradingDay(?day1, ?day2) ∧
FI-Strategy-SMA:shortSMA(?day1, ?sSMA1) ∧
FI-Strategy-SMA:shortSMA(?day2, ?sSMA2) ∧
FI-Strategy-SMA:longSMA(?day1, ?lSMA1) ∧
FI-Strategy-SMA:longSMA(?day2, ?lSMA2) ∧
swrlb:lessThan(?sSMA1, ?lSMA1) ∧ swrlb:greaterThan(?sSMA2,
?lSMA2)
   →    FI:BuyTradingDay(?day2)
```

```
FI:nextTradingDay(?day1, ?day2) ∧
FI-Strategy-SMA:shortSMA(?day1, ?sSMA1) ∧
FI-Strategy-SMA:shortSMA(?day2, ?sSMA2) ∧
FI-Strategy-SMA:longSMA(?day1, ?lSMA1) ∧
FI-Strategy-SMA:longSMA(?day2, ?lSMA2) ∧
swrlb:lessThan(?lSMA1, ?sSMA1) ∧ swrlb:greaterThan(?lSMA2,
?sSMA2)
  →    FI:SellTradingDay(?day2)
```

ONTOLOGIJA FI-STRATEGY-JAPANESE

RAZREDI

FI-Strategy-Japanese:JapanesePatterns

```
rdfs:label       "Japonski trgovalni vzorci@si"
                 "Japanese trading patterns@en"
rdfs:subClassOf  FI:TradingDay
```

FI-Strategy-Japanese:BlackBody

```
rdfs:label       "Črno telo svečnika@si"
                 "Black body@en"
rdfs:comment     "Črno telo svečnika, kjer je cena ob odprtju
                 večja od cene ob zaprtju.@si"
                 "Black body where close price is lower than
                 open price.@en"
rdfs:subClassOf  FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:Doji

```
rdfs:label       "Doji@si"
                 "Doji@en"
rdfs:subClassOf  FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:Engulfing

```
rdfs:label       "Engulfing@si"
                 "Engulfing@en"
rdfs:subClassOf  FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:LargeBody

```
rdfs:label       "Veliko telo svečnika@si"
                 "Large body@en"
rdfs:subClassOf  FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:MHT

```
rdfs:label       "Danes najvišja vrednost@si"
                 "Maximum high today@en"
rdfs:subClassOf  FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:MHY

```
rdfs:label        "Včeraj najvišja vrednost@si"
                  "Maximum high yesterday@en"
rdfs:subClassOf   FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:MLT

```
rdfs:label        "Danes najnižja vrednost@si"
                  "Minimum low today@en"
rdfs:subClassOf   FI-Strategy-Japanese:JapanesePatterns
```

FI-Strategy-Japanese:WhiteBody

```
rdfs:label        "Belo telo svečnika@si"
                  "White body@en"
rdfs:comment      "Belo telo svečnika, kjer je cena ob zaprtju
                  večja od cene ob odprtju.@si"
                  "White body where close price is higher than
                  open price.@en"
rdfs:subClassOf   FI-Strategy-Japanese:JapanesePatterns
```

PRAVILA

FI:**TradingDay(**?day**)** ∧ FI:**close(**?day, ?close**)** ∧ FI:**open(**?day, ?open**)** ∧ KAON2:**ifTrue("**$1 >= $2", ?close, ?open**)**

→ FI-Strategy-Japanese:**WhiteBody(**?day**)**

FI:**TradingDay(**?day**)** ∧ FI:**close(**?day, ?close**)** ∧ FI:**open(**?day, ?open**)** ∧ KAON2:**ifTrue("**$1 < $2", ?close, ?open**)**

→ FI-Strategy-Japanese:**BlackBody(**?day**)**

FI-Strategy-Japanese:**WhiteBody(**?day**)** ∧ FI:**close(**?day, ?close**)** ∧ FI:**open(**?day, ?open**)** ∧ KAON2:**ifTrue("**$1 >= $2*1.01", ?close, ?open**)**

→ FI-Strategy-Japanese:**LargeBody(**?day**)**

FI-Strategy-Japanese:**BlackBody(**?day**)** ∧ FI:**close(**?day, ?close**)** ∧ FI:**open(**?day, ?open**)** ∧ KAON2:**ifTrue("**$1 <= $2*0.99", ?close, ?open**)**

→ FI-Strategy-Japanese:**LargeBody(**?day**)**

FI-Strategy-Japanese:**WhiteBody(**?day**)** ∧ FI:**close(**?day, ?close**)** ∧ FI:**open(**?day, ?open**)** ∧
KAON2:**ifTrue("**($1 - $2) <= $1*0.0025", ?close, ?open**)**

→ FI-Strategy-Japanese:**Doji(**?day**)**

```
FI-Strategy-Japanese:BlackBody(?day) ∧ FI:close(?day, ?close) ∧
FI:open(?day, ?open) ∧
KAON2:ifTrue("($2 - $1) <= $1*0.0025", ?close, ?open)
  →   FI-Strategy-Japanese:Doji(?day)

FI-Strategy-Japanese:WhiteBody(?day) ∧ FI:close(?day, ?close) ∧
FI:open(?day, ?open) ∧ FI:high(?day, ?high) ∧ FI:low(?day, ?low) ∧
KAON2:ifTrue("($1 - $2) <= ($3 - $4)*0.1", ?close, ?open, ?high,
?low)
  →   FI-Strategy-Japanese:Doji(?day)

FI-Strategy-Japanese:BlackBody(?day) ∧ FI:close(?day, ?close) ∧
FI:open(?day, ?open) ∧ FI:high(?day, ?high) ∧ FI:low(?day, ?low) ∧
KAON2:ifTrue("($2 - $1) <= ($3 - $4)*0.1", ?close, ?open, ?high,
?low)
  →   FI-Strategy-Japanese:Doji(?day)

FI:next(?day1, ?day2) ∧ FI:next(?day2, ?day3) ∧
FI:next(?day3, ?day4) ∧ FI:next(?day4, ?day5) ∧
FI:low(?day1, ?low1) ∧ FI:low(?day2, ?low2) ∧
FI:low(?day3, ?low3) ∧ FI:low(?day4, ?low4) ∧ FI:low(?day5,
?low5) ∧
KAON2:ifTrue("($5 < $4) && ($5 < $3) && ($5 < $2) && ($5 < $1)",
?low1, ?low2, ?low3, ?low4, ?low5)
  →   FI-Strategy-Japanese:MLT(?day5)

FI:next(?day1, ?day2) ∧ FI:next(?day2, ?day3) ∧
FI:next(?day3, ?day4) ∧ FI:next(?day4, ?day5) ∧
FI:high(?day1, ?high1) ∧ FI:high(?day2, ?high2) ∧
FI:high(?day3, ?high3) ∧ FI:high(?day4, ?high) ∧
FI:high(?day5, ?high5) ∧
KAON2:ifTrue("($4 > $5) && ($4 > $3) && ($4 > $2) && ($4 > $1)",
?high1, ?high2, ?high3, ?high4, ?high5)
  →   FI-Strategy-Japanese:MHY(?day4)

FI:next(?day1, ?day2) ∧ FI:next(?day2, ?day3) ∧
FI:next(?day3, ?day4) ∧ FI:next(?day4, ?day5) ∧
FI:high(?day1, ?high1) ∧ FI:high(?day2, ?high2) ∧
FI:high(?day3, ?high3) ∧ FI:high(?day4, ?high) ∧
FI:high(?day5, ?high5) ∧
KAON2:ifTrue("($5 > $4) && ($5 > $3) && ($5 > $2) && ($5 > $1)",
?high1, ?high2, ?high3, ?high4, ?high5)
  →   FI-Strategy-Japanese:MHT(?day5)
```

```
FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:BlackBody(?day1) ∧
FI-Strategy-Japanese:BlackBody(?day2) ∧
FI:open(?day1, ?open1) ∧ FI:close(?day1, ?close1) ∧
FI:open(?day2, ?open2) ∧ FI:close(?day2, ?close2) ∧
KAON2:ifTrue("($2 < $1) && ($3 > $4)", ?close1, ?close2, ?open1,
?open2)
  →   FI-Strategy-Japanese:Engulfing(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:WhiteBody(?day1) ∧
FI-Strategy-Japanese:BlackBody(?day2) ∧
FI:open(?day1, ?open1) ∧ FI:close(?day1, ?close1) ∧
FI:open(?day2, ?open2) ∧ FI:close(?day2, ?close2) ∧
KAON2:ifTrue("($1 > $2) && ($3 < $4)", ?open2, ?close1, ?close2,
?open1)
  →   FI-Strategy-Japanese:Engulfing(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:WhiteBody(?day1) ∧
FI-Strategy-Japanese:WhiteBody(?day2) ∧
FI:open(?day1, ?open1) ∧ FI:close(?day1, ?close1) ∧
FI:open(?day2, ?open2) ∧ FI:close(?day2, ?close2) ∧
KAON2:ifTrue("($1 > $2) && ($3 < $4)", ?close2, ?close1, ?open2,
?open1)
  →   FI-Strategy-Japanese:Engulfing(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:BlackBody(?day1) ∧
FI-Strategy-Japanese:WhiteBody(?day2) ∧
FI:open(?day1, ?open1) ∧ FI:close(?day1, ?close1) ∧
FI:open(?day2, ?open2) ∧ FI:close(?day2, ?close2) ∧
KAON2:ifTrue("($1 > $2) && ($3 < $4)", ?close2, ?open1, ?open2,
?close1)
  →   FI-Strategy-Japanese:Engulfing(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:BlackBody(?day1) ∧
FI-Strategy-Japanese:Engulfing(?day2) ∧
FI-Strategy-Japanese:LargeBody(?day2) ∧
FI-Strategy-Japanese:WhiteBody(?day2) ∧
FI-Strategy-Japanese:MLT(?day2)
  →   FI-Strategy-Japanese:BuyTradingDay(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:Doji(?day1) ∧
FI-Strategy-Japanese:Engulfing(?day2) ∧
FI-Strategy-Japanese:LargeBody(?day2) ∧
FI-Strategy-Japanese:WhiteBody(?day2) ∧
FI-Strategy-Japanese:MLT(?day2)
  →   FI-Strategy-Japanese:BuyTradingDay(?day2)
```

```
FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:WhiteBody(?day1) ∧
FI-Strategy-Japanese:Engulfing(?day2) ∧
FI-Strategy-Japanese:LargeBody(?day2) ∧
FI-Strategy-Japanese:BlackBody(?day2) ∧
FI-Strategy-Japanese:MHT(?day2)
  →   FI-Strategy-Japanese:SellTradingDay(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:WhiteBody(?day1) ∧
FI-Strategy-Japanese:Engulfing(?day2) ∧
FI-Strategy-Japanese:LargeBody(?day2) ∧
FI-Strategy-Japanese:BlackBody(?day2) ∧
FI-Strategy-Japanese:MHY(?day2)
  →   FI-Strategy-Japanese:SellTradingDay(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:Doji(?day1) ∧
FI-Strategy-Japanese:Engulfing(?day2) ∧
FI-Strategy-Japanese:LargeBody(?day2) ∧
FI-Strategy-Japanese:BlackBody(?day2) ∧
FI-Strategy-Japanese:MHT(?day2)
  →   FI-Strategy-Japanese:SellTradingDay(?day2)

FI:next(?day1, ?day2) ∧ FI-Strategy-Japanese:Doji(?day1) ∧
FI-Strategy-Japanese:Engulfing(?day2) ∧
FI-Strategy-Japanese:LargeBody(?day2) ∧
FI-Strategy-Japanese:BlackBody(?day2) ∧
FI-Strategy-Japanese:MHY(?day2)
  →   FI-Strategy-Japanese:SellTradingDay(?day2)
```