Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Šega

POSLOVNIM UPORABNIKOM PRIJAZEN ZAJEM PRAVIL NA SEMANTIČNEM SPLETU

Diplomsko delo univerzitetnega študija

Ljubljana, 2008

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Šega

POSLOVNIM UPORABNIKOM PRIJAZEN ZAJEM PRAVIL NA SEMANTIČNEM SPLETU

Diplomsko delo univerzitetnega študija

Mentor: prof. dr. Marjan Krisper

Ljubljana, 2008

ZAHVALA

Za strokovno pomoč in vodenje pri izdelavi diplomske naloge se zahvaljujem mentorju prof. dr. Marjanu Krisperju in g. Dejanu Lavbiču.

Zahvaljujem se sestri Sanji Šega za lektoriranje diplomske naloge ter staršem, ki so me v času študija spodbujali in mi nudili vso potrebno podporo.

KAZALO

Seznam slik

Seznam tabel

KRATICE, OKRAJŠAVE, SIMBOLI

Semantic Web	Semantični splet - evolucija obstoječega spleta, kjer je informacijam dodan semantični pomen.
OWL Web Ontology Language	Jezik za zapis ontologij na semantičnem spletu.
SWRL Semantic Web Rule Language	Jezik za zapis pravil na semantičnem spletu.
Unicode	Standard za računalniško predstavitev znakov.
URI Universal Resource Identifier	Enotni identifikator vira.
XML eXtensible Markup Language	Razširljiv označevalni jezik.
RDF Resource Description Framework	Ogrodje za opis virov.
Shema RDF	Preprost jezik za opisovanje razredov virov in relacij med njimi znotraj RDF modela.
URL Uniform Resource Locator	Internetni naslov.
OIL	Ontology Interchange Language – jezik za izmenjavo ontologij
DAML+OIL DARPA Agent Markup Language+OIL	Jezik za označevanje DARPA agentov.
RuleML Rule Markup Language	Jezik za označevanje pravil.
ORL OWL Rules Language	Jezik za zapis OWL pravil.
FI	Finančni instrument.
YUI Yahoo User Interface	Odprtokodna zbirka orodij in kontrolnikov, napisanih v JavaScript jeziku.
AJAX Asynchronous JavaScript and XML	Skupina tehnik, ki se uporabljajo za izdelavo bogato interaktivnih spletnih aplikacij.
KAON2 KArlsruhe ONtology 2	Infrastruktura za upravljanje z OWL-DL in SWRL ontologijami
API	Aplikacijski programski vmesnik.
HTML HyperText Markup Language	Označevalni jezik za določitev strukture dokumentov, ki se prenašajo po svetovnem spletu in pregledujejo s spletnimi brskalniki.
WWW World Wide Web	Svetovni splet.
WWWC ali W3C World Wide Web Consortium	Konzorcij WWW – mednarodna organizacija za standardizacijo svetovnega spleta.

POVZETEK

Semantični splet predstavlja evolucijo obstoječega spleta, kjer je informacijam dodan semantični pomen. To bo v prihodnosti omogočilo razvoj avtomatiziranih programskih agentov, ki bodo namesto uporabnika izvajali vsakovrstne opravke.

Implementacija semantičnega spleta temelji na množici tehnologij in standardov, razporejenih v plasti. Pomemben del predstavljata koncepta predstavitve znanja v obliki ontologij in pravil nad temi ontologijami. Čeprav obstaja že več orodij, ki podpirajo gradnjo ontologij in zajem pravil, jih večina od uporabnikov zahteva strokovno poznavanje tehnologij.

V razvoju je orodje IntelliOnto, ki bo gradnjo ontologij in zajem pravil približalo poslovnim uporabnikom. Naloga diplomskega dela je bila razviti sistem poslovnim uporabnikom prijaznega zajema pravil in ta sistem implementirati v obliki modula za zajem pravil aplikacije IntelliOnto.

Razviti sistem omogoča uporabnikom zajem pravil brez predhodnega znanja abstraktne sintakse. Temelji na uporabi obrazcev za zajem atomov pravila. S pomočjo teh obrazcev se atom pravila predstavi kot trditev v naravnem jeziku. Ogrodje obrazca je vezni tekst, ki povezuje nastavke argumentov. Ta tekst se po potrebi pridobi iz ontologije. Nastavki argumentov nakazujejo zahtevani tip argumenta, v obliki namigov pa tudi dodatne omejitve pri vnosu. Sam vnos argumentov poteka preko kontekstno ustreznega menija. Ko je celotno pravilo zajeto, se samodejno pretvori v ustrezno sintakso in pošlje spletni storitvi. Le-ta pravilo iz abstraktne sintakse pretvori v SWRL sintakso in ga doda ontologiji.

Razviti in implementirani so bili trije načini zajema pravil. Prvi predvideva zajem enega pravila naenkrat v obliki »pogoji => posledice«. Drugi omogoča zajem pravil z uporabo odločitvene tabele in uporabniku olajša zajem paketa pravil, katerih telesa so sestavljena iz različnih kombinacij pogojev odločitvene tabele. Tretji način pa temelji na gradnji pravil v obliki odločitvenega drevesa, pri čemer vejitve potekajo glede na veljavnost atoma kretnice.

Ključne besede: poslovna pravila, semantični splet, ontologija, OWL, SWRL

ABSTRACT

The Semantic Web is an evolution of the current Web where information is given semantic meaning. This will enable development of automatic software agents which will carry out various tasks for the users.

The implementation of Semantic Web is based on a set of technologies and standards, which form layers. The most important part of the Semantic Web are the two concepts of knowledge representation in the form of ontologies and rules over these ontologies. Although there are already several tools that support ontology development and rules editing, most of them require the users to have expert knowledge of the technologies.

A tool called IntelliOnto is being developed that will bring ontology building and rule editing closer to business users. The aim of this graduate thesis was to develop a system for user-friendly rule editing and to implement this system as a rule editing module for IntelliOnto application.

The system that was developed enables rules editing without requiring the users to be familiar with abstract syntax. It is based on forms for editing rule atoms. With the help of these forms, any atom of the rule can be presented as a sentence in the natural language. The framework of the form is a text that binds the anchors for arguments. This text is acquired from the ontology where necessary. The text of the argument anchors provides users with information about the type of the argument that is required, while hints of these anchors show additional restrictions when in place. Arguments are entered with the help of the menu with contextually appropriate choices. When the rule is fully edited, it is automatically transformed into the required syntax and sent to a web service. This web service transforms the rule from the abstract into the SWRL syntax and adds it to the ontology.

Three approaches to rule editing were developed and implemented. The first one is used for editing one rule at a time in the form of “antecedents => consequents”. The second approach enables rule editing by using a decision table. This approach is especially useful for editing rule packages that contain rules which have bodies consisting of different combinations of condition atoms, entered into decision table. The third approach is based on rule editing in the form of a decision tree, in which branching is done according to validity of the condition atom.

Keywords: business rules, Semantic Web, ontology, OWL, SWRL

1. UVOD

Tim Berners-Lee, oče svetovnega spleta in direktor Konzorcija WWW, je semantični splet opisal kot:

»… razširitev trenutnega spleta, v katerem je informacijam dodan dobro definiran pomen, kar omogoča boljše sodelovanje dela računalnikov in ljudi.« [1]

Zamislimo si recimo spletno aplikacijo, dostopno vsem uporabnikom socialnih omrežij širom spleta. Aplikacija omogoča ustvarjanje novih in uporabo že obstoječih ontologij. Ena od obstoječih ontologij opisuje tudi domeno trgovanja s finančnimi instrumenti. Na tej ontologiji lahko vsak uporabnik zgradi svojo strategijo trgovanja, ki jo zapiše s pravili.

Vsak dan se podatki iz realnega sveta dodajo v ontologijo, odločitvena logika pa, na podlagi teh podatkov in uporabnikove strategije, na navidezni borzi upravlja z njegovim portfeljem. Uporabnik svojo strategijo seveda izboljšuje in poskuša zaslužiti čim več navideznega denarja. Pri tem si lahko pomaga s strategijami ostalih uporabnikov, katerih elemente lahko enostavno doda svoji strategiji. Ves čas se seveda vodi spletna lestvica največjih zaslužkarjev, na kateri lahko uporabnik preveri svojo uspešnost.

Le zabava? Ni nujno. Uporabnik si lahko namreč po želji omisli programskega agenta, ki na podlagi omenjene ontologije in uporabnikove strategije opravlja z njegovim portfeljem v realnem svetu.

Zanimiva ideja, ki pa zna kaj kmalu postati tudi realnost. Tehnologije semantičnega spleta so od njegove predstavitve leta 2001 namreč precej napredovale. Na programske agente bo sicer potrebno še nekaj časa počakati, orodja za izdelavo ontologij in zajem pravil pa že obstajajo.

Preden pa bodo koncepti in tehnologije semantičnega spleta splošno sprejeti, je potrebno orodja za delo z njimi poenostaviti in približati poslovnim uporabnikom. Aplikacija IntelliOnto, ki je trenutno še v zgodnji fazi razvoja, že od samega začetka nastaja prav s tem namenom. [9]

V okviru tega diplomskega dela sem za potrebe aplikacije IntelliOnto razvil modul za zajem pravil. Pri tem sem se bolj kot na samo tehnično implementacijo osredotočil na logično plat zajema pravil. Moj namen je bil namreč razviti tak sistem zajema, da ga bo zlahka osvojil tudi navaden poslovni uporabnik.

Med razvojem sem poskušal najti odgovore na vprašanja:

- Kako uporabniku olajšati zajem pravil?
- Kako pravila predstaviti v čim bolj razumljivi obliki?
- Kako preprečiti napake pri zajemu pravil?
- Kateri načini zajema pravil bi ta postopek poslovnemu uporabniku še dodatno olajšali?

Izhajal sem iz dejstva, da večina poslovnih uporabnikov dobro pozna svoje strokovno področje, probleme pa imajo pri zapisu tega znanja z neko formalno sintakso. Ta problem lahko rešimo tako, da jim omogočimo zajem pravil na višjem nivoju, tako da znanje abstraktne sintakse več ni potrebno.

Podana ideja sistema je naslednja: če je gradnja pravil v abstraktni sintaksi podobna pisanju prostega spisa, bo gradnja pravil v aplikaciji IntelliOnto bolj podobna izpolnjevanju obrazca. Na podlagi tega obrazca bo sistem samodejno ustvaril pravila v abstraktni sintaksi, ta pa bodo s pomočjo enega od obstoječih orodij prevedena v SWRL obliko. Na ta način se izognemo »pravopisnim in slogovnim« napakam kot posledici uporabnikovega nepoznavanja sintakse, s pravilno zastavljenim sistemom pa ne omejimo izraznih možnostih zapisa pravil.

Razvoj sistema je sledil naslednjim smernicam:

- Pravila je potrebno predstaviti v lahko razumljivi obliki.
- Kjer je le mogoče, se uporablja naravni jezik.
- Vnos argumentov poteka z izbiranjem možnosti s seznama, pri čemer se prikažejo le smiselne izbire.
- Uporabniku se vnos pravil dodatno olajša s ponudborelevantnih namigov.
- Na voljo je več načinov vnosa pravil, s katerimi se olajša vnos paketov pravil določene oblike.

Diplomska naloga v drugem poglavju podrobneje opiše tehnologije semantičnega spleta, predvsem uporabljena jezika OWL za zapis ontologij in SWRL za zapis pravil na semantičnem spletu. V tretjem poglavju se naloga ukvarja s problematiko poslovnemu uporabniku prijaznega zajema pravil in razvije sistem, ki tak zajem omogoča. V četrtem poglavju je predstavljena problemska domena trgovanja s finančnimi instrumenti, v petem poglavju pa implementacija te domene v okviru razvitega sistema za uporabnikom prijazen zajem pravil. Diplomska naloga je povzeta z zaključkom v šestem poglavju.

2. RAZISKOVALNA PODROČJA

2.1. Semantični splet in ontologije

Na svetovnem spletu je danes na voljo osupljiva množica informacijskih storitev. Večina od nas je že kdaj preko spleta opravila nakup, rezervirala knjigo v knjižnici, ali pa vsaj preverila tedenski kino spored. Vendar pa se poraja problem, da je informacijskih storitev enostavno preveč, da bi jih lahko vse razumeli in obvladovali. Pri iskanju informacij si pomagamo s spletnimi iskalniki, vendar pa moramo potem vseeno ročno pregledati zadetke, da najdemo stran z informacijo, ki smo jo želeli najti.

Potrebujemo torej nov pristop, nove tehnike, ki nam bodo pomagale pri procesiranju vseh teh informacij. Tehnike, ki nam bodo našle le tiste stvari, ki jih iščemo, ostale pa izpustile. Tehnike, ki bodo izluščile in povzele tiste stvari, ki so pomembne, in nam pomagale razumeti zveze med njimi. [2]

Težava razvoja takšnih tehnik je v tem, da so spletne strani oblikovane tako, da so berljive in razumljive ljudem, ne računalnikom. Semantični splet je vizija informacije, ki je razumljiva računalnikom, tako da lahko le-ti opravijo večino težaškega dela pri iskanju, deljenju in kombiniranju informacij na spletu. [24]

2.1.1. Semantični splet

Tim Berners-Lee, izumitelj svetovnega spleta (World Wide Web – WWW), si je le-tega najprej zamislil s precej bogatejšimi opisi dokumentov in povezavami med njimi. Vendar pa so bile te ideje spričo potrebe po preprostem, uporabnem in robustno delujočem sistemu, ki bi ga lahko uporabljal vsakdo, postavljene na stran. Rezultat je preprostejši, bolj človeško naravnan splet, ki ga poznamo danes.

Vendar pa ideja ni zamrla. Tim Berners-Lee je svojo prvotno vizijo še nadgradil in jo objavil v članku z naslovom »The Semantic Web«, ki je bil objavljen maja 2001 v reviji Scientific American. [1] V tem članku predstavi privlačno vizijo sveta, v katerem ljudem ni potrebno garaško brskati po informacijah na spletu in se med seboj usklajevati za izvedbo rutinskih opravil, kot so na primer načrtovanje sestankov, iskanje dokumentov in lociranje storitev. Vse to namesto njih stori kar sam splet. To pa se lahko omogoči tako, da se priskrbi zadosten koncept spletnih virov, obenem pa se priskrbijo orodja za uporabo tega konteksta, tako da lahko stroji (ali »programski agenti« - programi, ki delajo namesto ljudi) najdejo prave stvari in lahko sprejemajo odločitve. [10]

Spletu je torej potrebno dodati pomen – opisati razpoložljive spletne vire in povezave med njimi. Temeljni princip semantičnega spleta je tako uporaba metapodatkov. [2] Le-ti v spletnih straneh obstajajo že sedaj, vendar so le del HTML kode.Z njimi opisujemo format, v katerem naj bo informacija predstavljena. Imajo torej sintaktični pomen, ne pa semantičnega. Tako lahko določimo, da se neka beseda izpiše v rdeči barvi, ne moremo pa določiti, da ta beseda predstavlja na primer ime avtorja strani. Za potrebe semantičnega spleta pa morajo biti

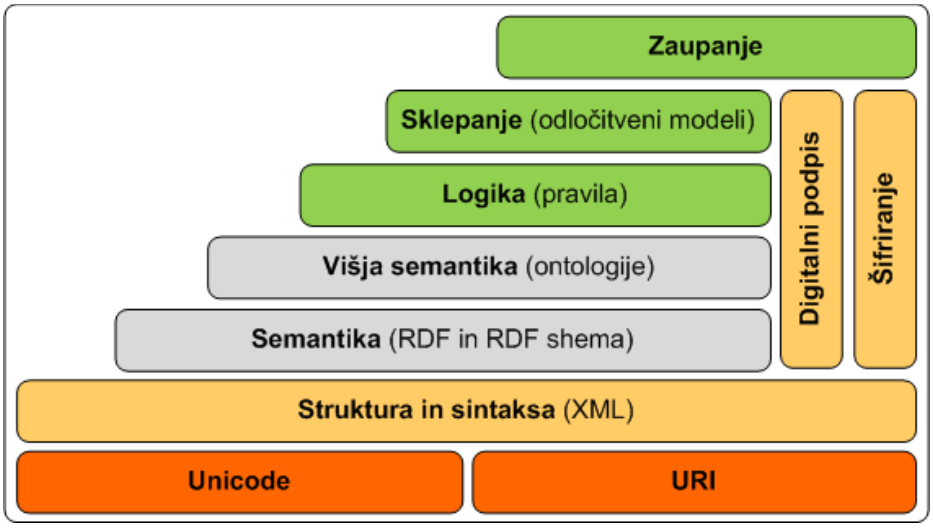
tudi metapodatki semantični, torej morajo opisovati bodisi vsebino strani (glavno tematiko, povezavo z ostalimi stranmi, …), bodisi entitete na strani (pomen predstavljenih informacij).

Z vzpostavljenim sistemom semantičnih metapodatkov se odpre veliko novih možnosti: [2]

- Iskanje ne samo po tekstu, pač pa tudi po pomenu. Če na primer iščemo dokumente, ki se navezujejo na besedno zvezo »dr. Danilo Türk«, lahko med rezultati najdemo tudi povsem relevanten dokument, ki se navezuje na besedno zvezo »predsednik Republike Slovenije«.
- Boljša predstavitev informacij. Namesto linearnega seznama zadetkov nam lahko iskanje vrne rezultate strnjene po pomenu. Iskanje besede »Jaguar« bi tako vrnilo vsaj dva skupka rezultatov – en bi se navezoval na avtomobilsko znamko, drugi pa na vrsto velikih mačk.
- Lažje odkrivanje in povezovanje semantično opisanih spletnih storitev.

2.1.2. Tehnologije in standardi semantičnega spleta

Semantični splet temelji na arhitekturi sklada plasti tehnologij in standardov [1, 3, 4, 10]:



Slika 1: Sklad plasti tehnologij in standardov semantičnega spleta

- **Unicode in URI**

  Unicode, standard za računalniško predstavitev znakov, in URI (Universal Resource Identifier – enotni identifikator vira), standard za identificiranje in lociranje virov, predstavljata temelja tehnologij v višje ležečih plasteh.

- **Plast strukture in sintakse (XML)**

  XML (eXtensible Markup Language - razširljivi označevalni jezik) in sorodni standardi (npr. naslovni prostori in Sheme XML) predstavljajo splošen način strukturiranja podatkov na spletu.

Z uporabo XML lahko vsakdo ustvari svoje zaznamke, t.j. skrite oznake, ki označujejo spletne strani ali sekcije teksta na strani. Skripti ali programi lahko s pridom uporabijo te zaznamke, vendar pa mora programer vedeti, kaj je ustvarjalec strani z določenim zaznamkom hotel povedati. XML torej uporabnikom omogoča dodajanje notranje strukture njihovim dokumentom, ne pove pa, kaj ta struktura pomeni.

- **Plast semantike (RDF in RDF Shema)**

  Prva plast, ki uresničuje ideje semantičnega spleta, zajema tehnologiji RDF (Resource Description Framework - ogrodje za opis virov) in RDF Shema.

  RDF je preprosto ogrodje za predstavitev metapodatkov. Ti so zakodirani v sete trojčkov, ki imajo podoben pomen kot osebek, povedek in predmet v osnovnem stavku. Trojčki so lahko zapisani z uporabo XML zaznamkov. Z uporabo RDF tako dokument vsebuje trditve, da imajo neke stvari (ljudje, predmeti, spletne strani, …) relacije (na primer »je sestra«, »je avtor«) z določenimi vrednostmi (neka druga oseba, predmet, spletna stran, …). Ta struktura se izkaže kot naraven način za opis velike večine podatkov, ki jih procesirajo stroji. Osebek in predmet sta oba identificirana z URI, uporabljenim na enak način kot v povezavi na spletni strani (internetni naslov – Uniform Resource Locator ali URL – je najpogostejši tip URI). Tudi povedki so identificirani z URI, kar vsakomur omogoča definiranje novega konceptaoz. novega povedka le z definiranjem njegovega URI nekje na spletu.

  RDF trojčki formirajo mrežo informacij o povezanih stvareh. Ker RDF za zakodiranje teh informacij v dokument uporablja identifikatorje URI, le-ti zagotovijo, da koncepti niso le besede v dokumentu, ampak so vezani na enotno definicijo, ki jo lahko vsak najde na spletu.

  Kaj pa se zgodi v primeru, ko dva dokumenta uporabljata različna identifikatorja URI za isto informacijo? Programski agent, ki želi primerjati ali kombinirati informacije teh dveh dokumentov, mora nekako vedeti, da ta dva izraza pomenita isto stvar.

  RDF Shema je preprost jezik za opisovanje razredov virov in relacij med njimi znotraj RDF modela. Ponuja preprosto ogrodje za sklepanje, na podlagi katerega lahko pridemo do zaključkov o tipih virov.

- **Plast višje semantike (ontologije)**

  Ontologije predstavljajo bogatejši način predstavitve znanja kot RDF Schema. Z jeziki za zapis ontologij lahko predstavimo kompleksnejše omejitve tipov virov in njihovih relacij.

- **Plast logike**

  Logična plast dodaja podporo pravilom, s katerimi lahko na primer opišemo relacije, ki jih z jeziki za zapis ontologij ne moremo opisati. Plast ontologij in logike daje podlago plasti sklepanja.

- **Plast sklepanja**

  Ta plast zajema sisteme za (avtomatsko) sklepanje, ki jih uporabljamo nad strukturo ontologije in pravil, da izvlečemo nove zaključke. Z uporabo takšnega sistema se lahko programski agent odloči, ali nek vir ustreza njegovim zahtevam in obratno.

- **Plast zaupanja**

  Najvišja plast piramide zagotavlja koncepte zaupanja, ki jih lahko podpira semantični splet. Ta komponenta za zdaj še ne sega dlje od vizije, da bi ljudem omogočili postavljanje vprašanj o stopnji, do katere lahko zaupajo informacijam na spletu, da bi s tem dobili zagotovilo kvalitete. Ko pa bo ta plast realizirana, se bo med drugim posluževala tehnologij digitalnega podpisa in šifriranja, s pomočjo katerih bo potrjevala, da izpeljani zaključki prihajajo iz zaupanja vrednih virov.

2.1.3. Ontologije

Splošno sprejeta definicija ontologije je: [6]

»Ontologija je eksplicitna in formalna specifikacija konceptualizacije neke domene.«

Ta definicija poudarja dve stvari:

- Konceptualizacija je formalna, kar omogoča sklepanje z računalnikom.
- Ontologija je v praksi načrtovana za neko konkretno domeno.

Ontologije sestavljajo razredi (tudi koncepti), relacije (tudi lastnosti) in primerki (tudi objekti, instance). [2, 21]

- **Primerki**

  Primerki so osnovni gradniki ontologije. Primerki ontologije so lahko konkretni objekti, na primer ljudje, živali, rastline, predmeti itd., lahko pa so tudi abstraktni primerki, na primer števila ali besede.

- **Razredi**

  Razredi so abstraktne skupine, množice ali zbirke objektov. Klasificirajo lahko primerke, druge razrede, ali kombinacijo obojih. Med razredi lahko velja relacija podrejenosti. Tako je lahko nek razred podrazred nekega drugega razreda – nadrazreda. S pomočjo te relacije lahko formiramo hierarhijo razredov, tipično z nekim vseobsegajočim razredom na vrhu (na primer razred »Karkoli«).

  Pomembna posledica hierarhične razporeditve je dedovanje relacije – vse, kar mora veljati za starševski razred, mora veljati tudi za razrede njegovih otrok. Večina ontologij

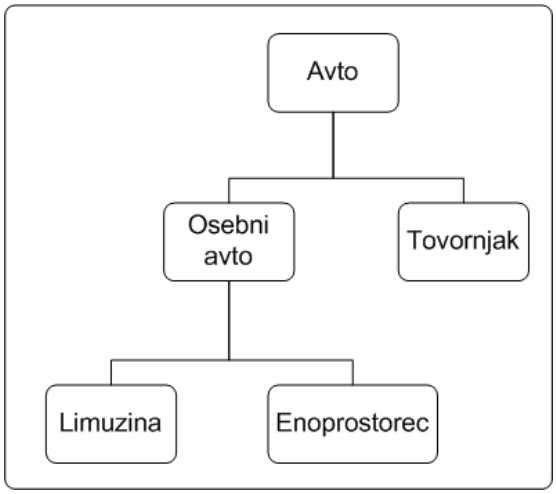
omogoča, da imajo razredi več staršev. Tako je lahko razred »HišnaMačka« otrok tako razreda »Mačka« kot tudi razreda »HišniLjubljenček«.

- **Relacije**

  Relacije v ontologiji določajo, v kakšni zvezi so objekti med seboj. Če imamo na primer v ontologiji razreda »HišniLjubljenček« in »Človek«, so primerki teh dveh razredov lahko med seboj povezani z relacijo <ima lastnika>. Ta primer tudi pokaže, da imajo relacije smer izraza. Inverzen izraz izraža isto dejstvo, vendar pa z obratno frazo v naravnem jeziku.

  Večina moči ontologij izhaja iz možnosti opisovanja relacij. Množica relacij opisuje »semantiko« domene. Relacije so lahko različnih tipov, na primer:

  - relacije med razredi,
  - relacije med primerki,
  - relacije med primerkom in razredom,
  - relacije med primerki in konkretnimi vrednostmi, ki pripadajo osnovnim podatkovnim tipom (tem relacijam rečemo tudi lastnosti).



Slika 2: Primer preproste ontologije

Kako torej ontologije rešujejo problem različnih identifikatorjev za isto stvar? Recimo, da želimo ugotoviti, ali je avtor dveh spletnih strani ista oseba. Pojavi pa se problem, da na nobeni strani avtor ni naveden z imenom, pač pa le s spletnim poštnim naslovom. Povrhu vsega je na eni strani naslov avtorjevega spletnega nabiralnika označen z identifikatorjem »poštniNaslov«, na drugi pa z »mailAddress«. Programski agent v takem primeru ne more vedeti, da podatka, označena s tema dvema identifikatorjema, pomenita isto stvar, torej spletni poštni naslov. Če pa imamo na voljo ontologijo, v kateri sta oba identifikatorja zajeta kot primerka razreda »SpletniPoštniNaslov«, lahko programski agent sklepa o istopomenskosti podatkov in opravi primerjavo.

Z uporabo ontologije torej omogočimo programskim agentom sklepanje o podatkih, na katere naletijo. Še bolj pa lahko znanje, zapisano v ontologiji, obogatimo z uporabo pravil.

2.1.4. Jeziki za zapis ontologij

Najbolj pogosto uporabljani jeziki za zapis ontologij so RDF + RDFS, DAML+OIL in OWL. [9]

2.1.4.1. RDF + RDFS

Podatkovni model RDF je ekvivalenten semantičnemu mrežnemu formalizmu. [5, 15] Sestavljajo ga trije tipi objektov:

- *Viri* so opisani z RDF izrazi in so vedno poimenovani z identifikatorji URI in morebitnimi sidrnimi identifikatorji.
- *Lastnosti* definirajo specifične pojave, karakteristike, atribute ali relacije, ki se uporabljajo za opis virov.
- *Trditve* določijo vrednost lastnosti določenega vira (ta vrednost je lahko druga RDF trditev).

Podatkovni model RDF ne zagotavlja mehanizmov za definiranje relacij med lastnostmi (atributi) in viri – to je naloga RDFS. Le-ta ponuja primitive za definiranje modelov znanja bližje pristopom, ki temeljijo na ogrodju.

RDF(S) se uporablja kot reprezentacijski format v mnogih orodjih in projektih, kot so Amaya, Protégé, Mozilla, SilRI, …

2.1.4.2. DAML+OIL

OIL (Ontology Interchange Language – jezik za izmenjavo ontologij), razvit v projektu OntoKnowledge, dovoljuje medobratovalnost med spletnimi viri. [5] Zgrajen je nad jezikom RDF(S) in ima štiri plasti:

- **Core OIL** vsebuje zbirko OIL primitivov s direktno preslikavo v RDF(S) primitive,
- **Standard OIL** je popoln OIL model, ki uporablja dodatne primitive,
- **Instance OIL** doda primerke razredov in vlog prejšnjemu modelu,
- **Heavy OIL** je plast za prihodnje razširitve jezika OIL.

Jezik DAML+OIL (DARPA Agent Markup Language + OIL – jezik za označevanje DARPA agentov) je razvil združeno komisijo ZDA in EU v projektu agencije DARPA s ciljem omogočiti semantično medobratovalnost v jeziku XML. Tudi DAML+OIL je zgrajen nad jezikom RDF(S), že njegovo ime pa pove, da je v tesni povezavi z jezikom OIL.

DAML+OIL ontologije podpirajo orodja OILEd, OntoEdit, Protege in WebODE.

2.1.4.3. OWL

Zgodnejši jeziki za zapis ontologij so se uporabljali za razvoj orodij in ontologij za specifične skupine uporabnikov (znanstvenike, vlado, posamezna podjetja, …), niso pa bili združljivi z arhitekturo svetovnega, še manj pa semantičnega spleta.

Zato je pod okriljem Konzorcija WWW (World Wide Web Consortium – WWWC ali W3C) nastal spletni jezik za zapis ontologij (Web Ontology Language ali OWL). [3, 15, 19, 21, 24]

OWL uporablja identifikatorje URI za poimenovanje ter spletno opisno ogrodje, ki ga zagotavlja RDF, in tako ontologijam doda nove zmogljivosti:

- možnost distribucije preko več sistemov,
- nadgradljivost po potrebah spleta,
- združljivost s spletnimi standardi za dostopnost in internacionalizacijo,
- odprtost in razširljivost.

Pri razvoju OWL jezika so izhajali iz jezika RDFS. Le-ta sicer omogoča zapis:

- razredov in hierarhije med njimi,
- relacij in hierarhije med njimi,
- omejitev domene in zaloge vrednosti,

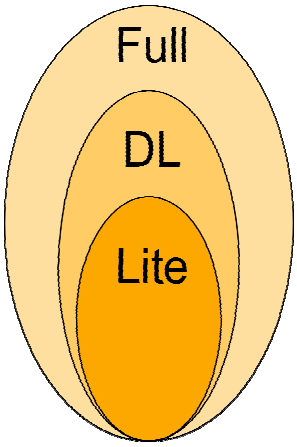
ne podpira pa:

- značilnosti relacij (inverznosti, tranzitivnosti, ...),
- lokalne omejitve zaloge vrednosti,
- definicije kompleksnih razredov,
- kardinalnosti,
- aksiomov o praznem preseku.

OWL razširi RDFS na kompleten jezik za predstavitev znanja in podatkov na spletu, tako da doda:

- logične izraze (in, ali, ne),
- (ne)enakost,
- lokalne relacije,
- obvezne/neobvezne relacije,
- obvezne vrednosti,
- naštevne razrede,
- simetrijo, inverznost.

Jezik OWL definira tri podjezike Vsak od teh podjezikov je razširitev svojega preprostejšega predhodnika tako v tem, kaj je lahko legalno izraženo, kot tudi v tem, kaj je lahko pravilno izpeljano.



Slika 3: Podjeziki jezika OWL

- **OWL Lite**

  OWL Lite je namenjen tistim uporabnikom, ki potrebujejo predvsem klasifikacijsko hierarhijo in preproste omejitve. Tako recimo podpira kardinalnost, vendar dovoljuje le kardinalne vrednosti 0 in 1. Ker je lažje razviti orodja za podporo OWL Lite jeziku kot za podporo drugima dvema OWL podjezikoma, OWL Lite ponuja hiter način migracije tezavrov in ostalih taksonomij. OWL Lite ima tudi nižjo formalno kompleksnost kot OWL DL.

- **OWL DL**

  OWL-DL je ime dobil zaradi povezave z *opisno logiko*, raziskovalnim področjem ki se ukvarja z logiko, ki tvori formalne temelje jeziku OWL. OWL DL je namenjen tistim uporabnikom, ki hočejo kar največjo izraznost ob ohranitvi računske popolnosti (vsi sklepi so zagotovljeno izračunljivi) in odločljivosti (vsi izračuni bodo končani v končnem času). OWL DL vsebuje vse konstrukte jezika OWL, vendar pri njihovi uporabi obstajajo določene omejitve. Tako je razred lahko podrazred večih razredov, ne more pa biti nek razred primerek drugega razreda.

- **OWL-Full**

  OWL Full je namenjen uporabnikom, ki želijo maksimalno izraznost in sintaktično svobodo jezika RDF, ne potrebujejo pa zagotovljene izračunljivosti. V OWL Full lahko recimo razred obravnavamo kot zbirko primerkov, obenem pa tudi kot primerek sam po sebi. Malo verjetno je, da bo lahko kdaj kak program podpiral popolno sklepanje za vsako značilnost jezika OWL Full.

Konstrukti OWL

OWL konstrukt	opisna logika	primer
presek (intersectionOf)	C1∩ ... ∩Cn	Človek∩Moški
unija (unionOf)	C1∪ ... ∪Cn	Doktor∪Odvetnik
komplement (complementOf)	¬C	¬Moški
eden od (oneOf)	{o1, ..., on}	{janez, marija}
vse vrednosti iz (allValuesFrom)	∀P.C	∀imaOtroka.Doktor
nekatere vrednosti iz (someValuesFrom)	∃P.C	∃ imaOtroka.Odvetnik
vrednost (value)	∃P.{o}	∃državljan.Slovenija
minimalna kardinalnost (minCardinality)	≥nP.C	≥2 imaOtroka.Odvetnik
maksimalna kardinalnost (maxCardinality)	≤nP.C	≤1 imaOtroka.Moški
kardinalnost (cardinality)	=nP.C	=1imaStarša.Ženska

Tabela 1: Konstrukti jezika OWL

OWL privzema »domnevo odprtega sveta«. To pomeni, da če s trenutnim znanjem neke domneve ni mogoče potrditi, ne moremo sklepati, da je domneva napačna.

Sintaksa OWL

OWL sintaksa temelji na RDF in XML sintaksah. Primeri:

- **Razred**

```
<owl:Class rdf:about="#Trgovalni_dan">
    <rdfs:comment xml:lang="si">Trgovalni podatki v okviru trgovalnega
dne.</rdfs:comment>
    <rdfs:label xml:lang="si">trgovalni dan</rdfs:label>
    <rdfs:label xml:lang="en">trading day</rdfs:label>
</owl:Class>
```

- **Relacije**

  OWL pozna dva tipa relacij:

  - **Lastnost** določa relacijo med primerki in RDF konkretnimi vrednostmi ali podatkovnimi tipi XML Sheme.

```
<owl:DatatypeProperty rdf:ID="razlog_za_akcijo">
  <rdfs:label xml:lang="si">razlog za izvedbo akcije</rdfs:label>
  <rdfs:label xml:lang="en">trade reason</rdfs:label>
  <rdfs:comment xml:lang="si">Razlog za izvedbo akcije nakupa ali
prodaje.</rdfs:comment>
  <rdfs:comment xml:lang="en">Reason for trading.</rdfs:comment>
  <rdfs:domain rdf:resource="#Trgovalni_dan"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

- **Objektna relacija** določa relacijo med primerki dveh razredov.

```
<owl:ObjectProperty rdf:ID="prejsnjiTrgovalniDan">
  <rdfs:range rdf:resource="#Trgovalni_dan"/>
  <rdfs:label xml:lang="si">prejšnji trgovalni dan</rdfs:label>
  <owl:inverseOf>
    <owl:FunctionalProperty rdf:ID="naslednjiTrgovalniDan"/>
  </owl:inverseOf>
  <rdfs:label xml:lang="en">Trading day has previous trading
day.</rdfs:label>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalPrope
rty"/>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:comment xml:lang="en">Trgovalni dan ima prejšnji trgovalni
dan.</rdfs:comment>
  <rdfs:domain rdf:resource="#Trgovalni_dan"/>
  <rdfs:label xml:lang="en">previous trading day</rdfs:label>
</owl:ObjectProperty>
```

- **Primerki**

```
<Borza rdf:ID="LJSE">
  <lokacija      rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Slovenia, Ljubljana</lokacija>
</Borza>
```

2.2. Poslovna pravila

2.2.1. Uvod

Poslovna pravila so v zadnjem desetletju postala zelo popularna in širše uporabljena, predvsem na področju informacijskih sistemov. [9] Glavni prednosti poslovnih pravil sta gotovo fleksibilnost in zmožnost spreminjanja poslovne logike. Poslovna pravila lahko definiramo kot deklaracijo politike ali pogojev, ki jim mora biti zadoščeno, in njihova vloga je definiranje, kako se znotraj organizacije izvajajo operativne odločitve.

Enotna metodologija razvoja informacijskih sistemov [14] poslovna pravila razdeli v tri skupine: definicije, omejitve in izpeljave.

- **Definicije**

  Definicije so posebna kategorija poslovnih pravil. Njihova vsebina pravzaprav ni tipična za poslovna pravila, pod katerimi si mnogi predstavljajo zgolj neke omejitve. Definicije neposredno ničesar ne omejujejo, temveč definirajo koncepte in dejstva iz poslovnega okolja. Definicije povedo, da v poslovnem okolju bodisi nekaj obstaja kot pomemben koncept ali pa v razmerju do drugih konceptov. Delimo jih na termine in dejstva.

- **Termin** je beseda ali fraza, ki ima za organizacijski sistem specifičen pomen. Termine lahko naprej delimo na *poslovne termine* in *skupne termine*.
- **Dejstva** opredeljujejo razmerja med termini iz poslovnega okolja in lahko zajemajo tudi več terminov. Dejstva so različnih vrst. Dejstva lahko delimo tudi na *osnovna* in *izpeljana dejstva*. Osnovna dejstva so nekaj, kar vemo o poslovnem okolju, izpeljana dejstva pa je mogoče iz osnovnih izpeljati.

Slika 4: Skupine poslovnih pravil

- **Omejitve**

Druga velika skupina poslovnih pravil so omejitve. Njihova vloga je določanje predpisov, ki morajo veljati v zvezi s strukturo ali delovanjem organizacijskega sistema. Delimo jih na omejitve strukture, omejitve operacij in sprožilce.

- **Omejitve strukture** so določila, ki se nanašajo na strukturne lastnosti organizacijskega sistema. Strukturne omejitve velikokrat nastopajo v povezavi z dejstvi in določajo omejitve v zvezi z njimi.
- **Omejitve operacij** določajo pogoje, ki morajo biti izpolnjeni pred ali po operaciji, da bi se ta pravilno izvedla. Operacija je v tem kontekstu generičen pojem in se nanaša na dogajanje, ki lahko nastopa na različnih ravneh abstrakcije (proces, aktivnost, operacija, funkcija itd.). Takšne omejitve so ključnega pomena za izvedbo operacije in so neodvisne od prožitvenega dogodka. Z drugimi besedami, njihova vsebina ne določa dogodka, ki operacijo dejansko proži.
- **Sprožilci** pa se od omejitev operacij ravno v tem razlikujejo. Njihova vsebina določa operacije, ki se sprožijo, če se zgodi določen dogodek in če so poleg tega izpolnjeni vsi potrebni pogoji. Sprožilci so odvisni od prožitvenega dogodka, saj pogoje preverjajo le v primeru, če se določen dogodek dejansko zgodi.

- **Izpeljave**

Zadnja skupina poslovnih pravil so izpeljave. Delimo jih na izračune in logične izpeljave:

- **Izračuni** so v bistvu algoritmi, ki predpišejo postopek izračuna za neko poslovno kategorijo.
- **Logične izpeljave** bazirajo na logičnem sklepu in so največkrat oblike: "če velja A, potem velja tudi B".

2.2.2. Pravila na semantičnem spletu

Splošno sprejetje jezika OWL sicer omogoča izmenjavo znanja v obliki ontologij med aplikacijami semantičnega spleta, še vedno pa obstaja potreba po nekem dodatnem mehanizmu za definiranje znanja. Ta dodatni mehanizem omogočajo pravila.

Pravila semantičnega spleta nimajo povsem enakega pomena kot poslovna pravila v informacijskih sistemih. V okviru semantičnega spleta pravila dopolnjujejo in razširjajo ontologije. [15]

Če preprosta pravila in aksiome še lahko določamo z jeziki za zapisovanje ontologij, za bolj zapletena pravila ti jeziki ne zadostujejo več. V tem primeru lahko nad ontologijo uporabimo pravila in tako pridemo do zaključkov, izrazimo omejitve, določimo politiko, določimo odzive na dogodke ali spremembe, odkrijemo novo znanje, pretvarjamo podatke itd. [17]

Ko imamo enkrat vzpostavljeno ontologijo in sistem pravil, lahko programska orodja višjih plasti semantičnega spleta izvajajo napredne semantične operacije. Te operacije pa lahko pravila uporabljajo tudi na enak način kot poslovna pravila v informacijskih sistemih.

2.2.3. Jeziki za zapis pravil na semantičnem spletu

V zadnjem času se je pojavilo kar nekaj predlogov jezikov za predstavitev pravil na semantičnem spletu. Med njimi prevladujejo RuleML (Rule Markup Language), OWL Rules Language (ORL) in Semantic Web Rule Language (SWRL). [8, 11]

2.2.3.1. RuleML

RuleML jeziki imajo potencial, da postanejo glavni akterji na področju uporabe pravil na semantičnem spletu in v porazdeljenih sistemih, saj omogočajo kreiranje, procesiranje, objavljanje in medsebojno komunikacijo pravil. Sintaksa RuleML temelji na XML, čeprav obstaja del njegove sintakse tudi v RDF obliki.

Cilj RuleML je zagotoviti ponovno uporabljivost in izmenjavo na višji ravni. Namesto da ustvarimo še en jezik za opis pravil, RuleML ponuja množico izpeljanih modularnih jezikov, ki temeljijo na skupnem podatkovnem modelu.

Zavedati se moramo, da imamo v poslovni domeni opravka z jezikoslovno bogato in zapleteno izrazno obliko, ki jo lahko v formalni matematični obliki zapišemo le v omejenem obsegu. Pri RuleML je zato uporabljen pristop, kjer imamo množico jezikov za opis najpomembnejših tipov pravil.

Osnovni tipi pravil, ki jih predlaga RuleML, so:

- **Pravila integritete**

  Integritetno pravilo sestavlja logični izraz, zapisan v poljubnem logičnem jeziku (npr. predikatna logika). Pravilo predstavlja izjavo, ki mora biti resnična v vseh nastalih stanjih in prehodih med stanji v diskretnem dinamičnem sistemu, za katerega je definirano. Priljubljena jezika za izražanje integritete sta SQL in OCL.

- **Izpeljana pravila**

  Izpeljano pravilo je sestavljeno iz enega ali več pogojev in sklepa. Zelo znane oblike izpeljanih pravil lahko najdemo v obliki Prolog pravil in SQL pogledov. To vrsto pravil lahko prikažemo tudi v UML razrednih diagramih s pomočjo OCL konstant.

- **Odzivna pravila**

  Odzivno pravilo sestavlja prožilni dogodek, pogoj, odzivna akcija in včasih tudi popogoj. Odzivna pravila delimo glede na prisotnost popogojev v dve skupini: ECA5, ki nimajo popogojev, in ECAP6, kjer so popogoji prisotni. Pogosto uporabljana primera teh pravil sta SQL prožilec in Outlook pravilo.

- **Akcijska pravila**

  Akcijsko pravilo preprosto sestavljata pogoj in akcija, ki se ob izpolnjenem pogoju izvede. S to vrsto pravil lahko implementiramo tudi izpeljana pravila. Na področju poslovnih pravil so akcijska pravila ena najpomembnejših in tudi najpogosteje uporabljanih ter predstavljajo ključno komponento v RuleML družini jezikov. Primeri znanih izvajalnih okolij akcijskih pravil so JESS, iLOG Rules/JRules, Fair Isaac/Baze Advisor itd.

2.2.3.2. ORL

Številne pomanjkljivosti jezika OWL izhajajo iz dejstva, da je jezik relativno bogat pri izražanju razredov, medtem ko se veliko slabše odreže pri relacijah. Preprosto ne obstaja konstruktor, s katerim bi lahko določili povezave med sestavljenimi relacijami in drugimi, mogoče tudi sestavljenimi, relacijami. Enostaven primer, ki opisuje ta problem, je povezava med relacijami *jeStarš, jeBrat in jeStric*, ki jo v osnovni obliki OWL jezika ne moremo zapisati. [8]

Eden od načinov reševanja omenjenega problema je razširitev jezika OWL z bolj izraznim jezikom za opis relacij. Dodajanje pravil jezikom za predstavitev znanja, ki temeljijo na opisni logiki, ni ravno nov pristop. Kar nekaj predhodnih sistemov (npr. Classic), ki temeljijo na opisni logiki, je svojo izrazno moč povečalo z uporabo jezika za predstavitev pravil. Vendar so pravila v teh sistemih imela šibkejšo semantično moč, kot jo imamo pri vpeljavi relacij med razredi in nadrazredi, saj so bile relacije prisotne le med posameznimi primerki in niso vplivale na sklepanje na podlagi razredne hierarhije.

ORL razširja OWL v sintaktičnem in semantičnem smislu: osnovna sintaksa ORL pravil je razširitev abstraktne sintakse OWL DL in OWL Lite, pravila pa imajo tudi formalen pomen v povezavi z razširitvijo OWL DL.

Prednosti ORL pravil so naslednja:

- Glede na OWL ne predstavljajo bistvenih sprememb, tako da so tudi relativno enostavna za uporabo.
- OWL pravila imajo veliko izrazno moč, saj temeljijo na izrazni moči OWL pri obeh elementih pravil - vzroku (telo) in posledici (glava).

Obljubljajo zelo elegantno rešitev raznovrstnega semantičnega procesiranja. Raziskovalnim skupinam, ki se ukvarjajo s semantičnim spletom, je jasno, da pri realizaciji ideje semantičnega spleta ne bo dovolj zgolj ena tehnika, model ali metoda. Potreba po različnih načinih uporabe ne bo predstavljala izbire, ampak bo nujno prisotna.

2.2.3.3. SWRL

Čeprav z uporabo OWL jezika pridobimo veliko izrazno moč na semantičnem spletu, vseeno obstajajo določene omejitve pri izražanju. Te omejitve pa lahko odpravimo z uporabo SWRL jezika za predstavitev pravil na semantičnem spletu. [8, 11 20]

SWRL je nastal na temeljih ORL. Ideja za SWRL se je pojavila šele leta 2004 in je eden novejših predlogov jezika za opis pravil na semantičnem spletu. Glavna ideja, ki stoji za SWRL, je razširitev OWL DL s pravili, kjer želimo obdržati čim večjo združljivost z obstoječo sintakso in semantiko.

Kot v veliko drugih jezikih za zapis pravil imajo tudi v SWRL pravila obliko parov pogoj-posledica. V SWRL terminologiji se pogoju pravi *telo pravila*, posledici pa *glava pravila*. Telo in glavo sestavlja konjunkcija enega ali več *atomov*. Za zdaj SWRL še ne podpira kompleksnejših kombinacij atomov. [12]

SWRL pravila se nanašajo na OWL primerke, primarno v okviru OWL razredov in relacij. Vzemimo primer SWRL pravila, s katerim želimo izraziti, da če ima oseba otroka moškega spola, to pomeni, da ima sina. Za to bi bilo potrebno zajeti koncepte »Oseba«, »Moški«, »Otrok«, in »Sin«. Koncepta »Oseba« in »Moški« lahko zajamemo kot OWL razred *Oseba* s podrazredom *Moški*. Relaciji »Otrok« in »Sin« pa lahko izrazimo kot OWL relaciji *imaOtroka* in *imaSina*, ki sta pripeta *Osebi*. Pravilo v SWRL bi potem bilo:

```
Oseba(?x1) ∧ imaOtroka(?x1, ?x2) ∧ Moški(?x2) → imaSina(?x1, ?x2)
```

Izvršitev tega pravila bi vsem primerkom x1, ki zadostujejo telesu pravila, nastavila relacijo *imaSina* na x2.

SWRL pravila se lahko nanašajo tudi eksplicitno na OWL primerke. Naslednji primer je izpeljanka prejšnjega in preverja, ali ima konkretni primerek po imenu Miha sina:

```
Oseba(Miha) ∧ imaOtroka(Miha, ?x2) ∧ Moški(?x2) → imaSina(Miha, ?x2)
```

SWRL podpira tudi konkretne vrednosti podatkov. Če ima primerek lastnost *jeStar,* lahko tako preverjamo, če ima Miha 20 let starega sina:

```
Oseba(Miha) ∧ imaOtroka(Miha, ?x2) ∧ Moški(?x2) ∧ jeStar(?x2, 20) →
                    ima20LetStaregaSina(Miha, ?x2)
```

Podprti so tudi nizi, zajeti znotraj enojnih narekovajev.

SWRL podpira tudi koncepta »isti kot« in »različen od«. SWRL *sameAs* atom lahko preveri, če sta dva OWL primerka Miha in Marko isti primerek:

```
sameAs(Miha, Marko)
```

Podobno lahko *differentFrom* atom preveri, ali sta dva OWL primerka različna.

SWRL vsebuje tudi atom, ki preverja, ali je neki primerek, lastnost ali spremenljivka določenega tipa. Ti atomi, ki se jim pravi »atomi zaloge vrednosti« (data range atoms), morajo imeti na začetku kvalifikator imenskega prostora »xsd:«. Naslednji primer preveri, ali je spremenljivka *x* tipa nepredznačeni integer:

```
xsd:unsignedInt(?x)
```

Druga oblika atoma zaloge vrednosti se uporablja za izražanje »eden od« relacije v SWRL. Naslednji SWRL atom nakazuje, da mora biti spremenljivka *x* ena od vrednosti 3, 4 ali 5:

```
[3, 4, 5](?x)
```

Jezik SWRL vsebuje tudi množico vgrajenih predikatov, ki precej razširijo njegovo izrazno moč. Pred vsakim atomom vgrajenega predikata mora biti kvalifikator imenskega prostora »swrlb:«. Vgrajeni predikati (built-in) lahko sprejmejo dva ali več argumentov. Najpreprostejši vgrajeni predikati so primerjalni operatorji. Z naslednjim pravilom preverjamo, če je Miha starejši od Marka:

```
        jeStar(Miha, ?starost1) ∧ jeStar(Marko, ?starost2) ∧
swrlb:greaterThan(?starost2, ?starost1) → jeStarejši(Miha, Marko)
```

SWRL vsebuje tudi kompleksnejše matematične (seštevanje, deljenje) vgrajene predikate, predikate za delo z nizi, datumi, seznami, … [12]

Pomembna omejitev SWRL jezika je ta, da je monoton - to pomeni. da ne podpira negacije atomov. [20] Pravilo

```
Oseba(?x1) ∧ ¬imaBrata(?x2) → Edinec(?x1)
```

tako ni mogoče.

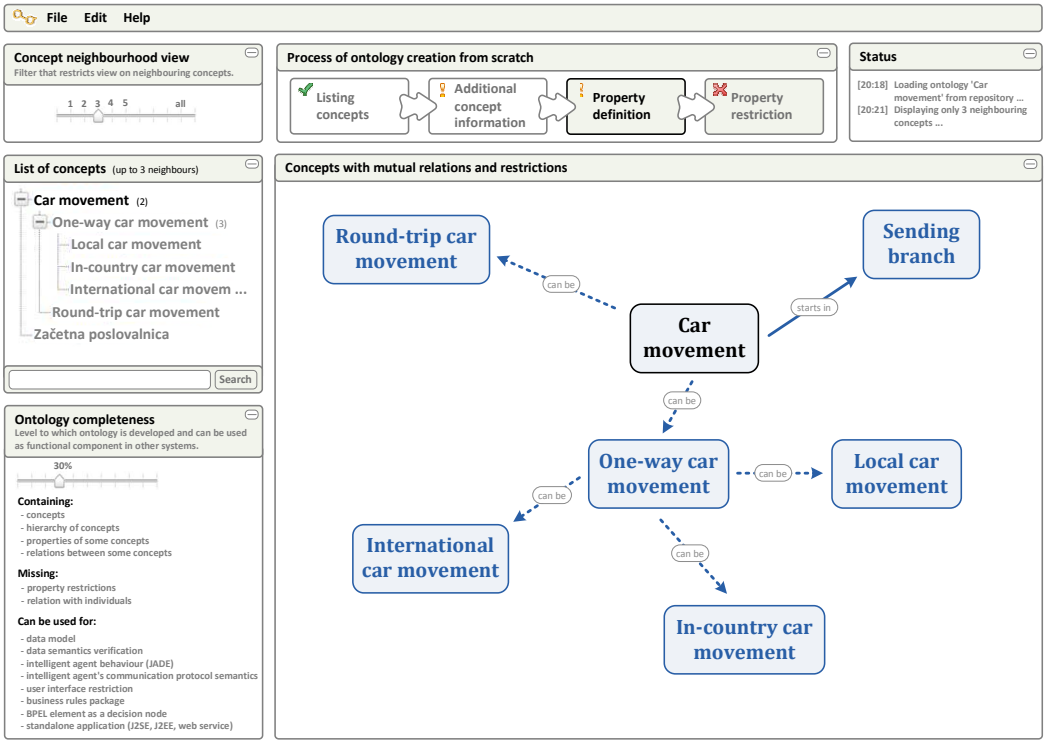
2.3. IntelliOnto

Na žalost so danes aplikacije, ki temeljijo na ontologijah in tehnologijah semantičnega spleta, omejene na akademsko okolje, v poslovnem okolju pa širše uporabe ni zaznati. Sklepamo lahko, da dobršen del problema predstavljajo obstoječe metodologije, saj je obstoj preverjenih procedur dober indikator zrelosti tehnologije. Velik problem je tudi pomanjkanje preprostih pristopov k razvoju ontologij ter orodij, ki bi take pristope podpirala. [9]

Uporaba pristopa upravljanja s poslovnimi pravili se zdi primeren način za poenostavitev razvoja in uporabe ontologij v poslovnih aplikacijah. V razvoju je model hitrega razvoja ontologij (ROD - Rapid Ontology Development), ki temelji na pristopu upravljanja s poslovnimi pravili.

Vzporedno z razvojem modela hitrega razvoja ontologij poteka tudi razvoj orodja IntelliOnto, ki bo izkoriščalo tehnike tega modela.

Orodje IntelliOnto je spletna aplikacija, ki uporablja več odprtokodnih tehnologij za manipulacijo ontologij (Protege, JENA, KAON2, …) in tehnologije Spleta 2.0 za uporabniški vmesnik (AJAX).



Slika 5: Uporabniški vmesnik orodja IntelliOnto

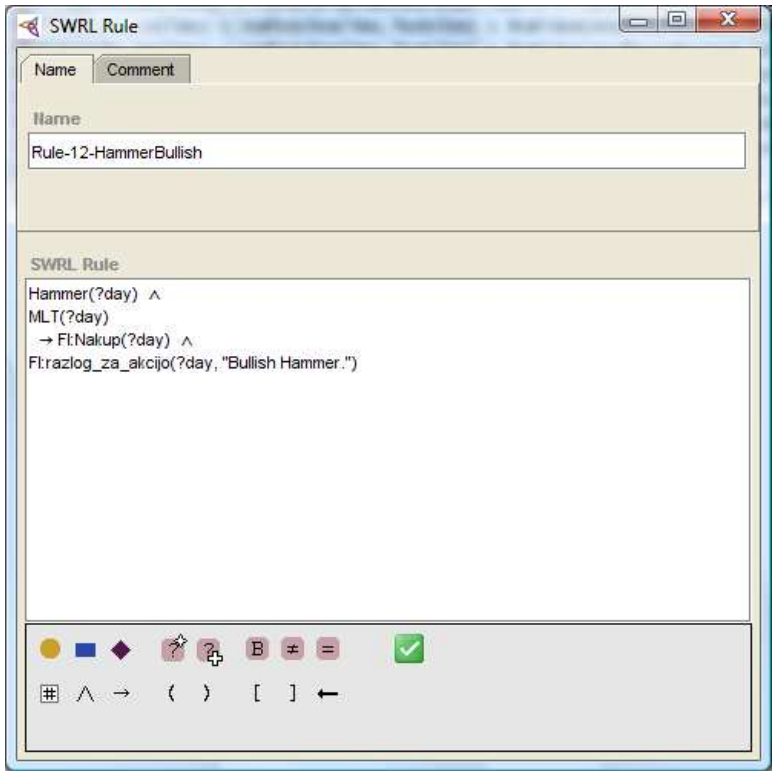
Gradnja ontologije poteka v osrednjem delu uporabniškega vmesnika – risalni plošči za manipulacijo razredov in pravil. Uporabniku omogoča enostavno ustvarjanje razredov, relacij, odvisnosti in omejitev na način, podoben naravnemu jeziku.

Orodje podpira tudi izbiro jezika za zapis ontologije oz. definiranje svojega jezika, integracijo z obstoječimi ontologijami (ponovna uporaba ontoloških elementov), izvoz v obliki funkcionalne komponente, upravljanje z verzijami, spreminjanje in ponovno postavitev.

Orodje IntelliOnto je trenutno še v zgodnji fazi razvoja.

3. ZAJEM PRAVIL, PRIJAZEN POSLOVNIM UPORABNIKOM

Tudi če bi pravila zajemal računalniški strokovnjak, bi bilo neposredno zapisovanje v SWRL obliko nesmiselno in nesmotrno. Zato bolj ali manj vsa obstoječa orodja za zajem pravil omogočajo vnos v abstraktni, »človeku razumljivi sintaksi« (human readable syntax). Te sintakse se računalniški strokovnjak hitro priuči in zajem pravil mu ne povzroča večjih težav. Ko je pravilo zajeto, ga orodje avtomatsko pretvori v SWRL obliko.



Slika 6: Primer zajema pravila v SWRLTab modulu orodja Protege

Kadar pa pravila zajema poslovni uporabnik, je tudi ta sintaksa lahko preveč zapletena. Poglejmo recimo naslednji primer:

```
                Financni_instrument(?financniInstrument) ∧
              (Kotira_na_borzi(?financniInstrument,#LJSE#) ∨
              Kotira_na_borzi(?financniInstrument,#NYSE#)) ∧
                 seTrgujeNaTrgovalniDan(?trgovalniDan) ∧
                         Prodaja(?trgovalniDan) ∧
                  razlog_za_nakup(?trgovalniDan,?razlog)
                                  =>
stringConcat (?printString,#Razlog za nakup #,?financniInstrument) ∧
          stringConcat (?printString,?printString,# je: #) ∧
          stringConcat (?printString,?printString,?razlog) ∧
                       PrintOutput(?printString)
```

Če želimo, da uporabnik to pravilo zapiše v abstraktni sintaksi, predpostavljamo, da uporabnik:

- pozna imena razredov,
- razume in zna uporabljati spremenljivke,
- razume in zna uporabljati konkretne vrednosti,
- pozna imena relacij in njihove domene/doseg,
- pozna imena vgrajenih predikatov,
- pozna pomen in vrstni red argumentov v vgrajenih predikatih,
- razume in zna uporabljati gnezdenje atomov z uporabo oklepajev.

Da bi si povprečen poslovni uporabnik pridobil predpostavljeno znanje, bi bilo potrebno zelo veliko časa in truda. Ker je to za marsikoga nesprejemljivo in bi lahko uporabnike odvrnilo od sicer zelo uporabne tehnologije, sem preučil, kako bi bilo mogoče zajem pravil poenostaviti do te mere, da bi se ga lahko poslovni uporabnik priučil z manj truda in v precej krajšem času.

V skladu s smernicami, navedenimi v uvodu, sem razvil sistem zajemanja pravil, ki ga opisujem v naslednjem poglavju.

3.1. Predlagan sistem zajemanja poslovnih pravil

V osnovi je vsako pravilo sestavljeno iz enega ali več **pogojev** in ene ali več **posledic**. Vsak pogoj ali posledica je **atom** pravila.

(Miha is Oseba)

Vsak atom je obdan z oklepaji, znotraj katerih je **obrazec** atoma, ki se razlikuje glede na tip atoma. Obrazec sestavljajo **argumenti** in **vezni tekst**, ki smiselno povezuje argumente glede na tip izbranega atoma. Vezni tekst je navadno fiksen, razen kadar je atom tipa *relacija*. Takrat ga pridobimo iz ontologije na podlagi izbrane relacije.

V obrazcu lahko spreminjamo samo vrednosti argumentov. Argumenti so lahko primerki razredov, spremenljivke ali konkretne vrednosti. Primerke razredov in obstoječe spremenljivke izbiramo iz seznama, pri čemer se upoštevajo domena, doseg in ostale omejitve ontologije. Ko so argumenti vnešeni, je obrazec pravzaprav smiselna trditev v naravnem jeziku.

Med posameznimi atomi v telesu pravila in posameznimi atomi v glavi pravila je privzet operator IN. Operator ALI med atomi v glavi že po definiciji pravil ne more nastopati, med atomi v telesu pa ni podprt iz naslednjih razlogov:

- Vsako pravilo, ki vsebuje operator ALI, lahko zapišemo kot dve pravili s samimi operatorji IN.

  Pravilo

  ```
  (a ∧ (b ∨ c) => d)
  ```

tako lahko zapišemo kot pravili

```
(a ∧ b => d)
```

in

```
(a ∧ c => d).
```

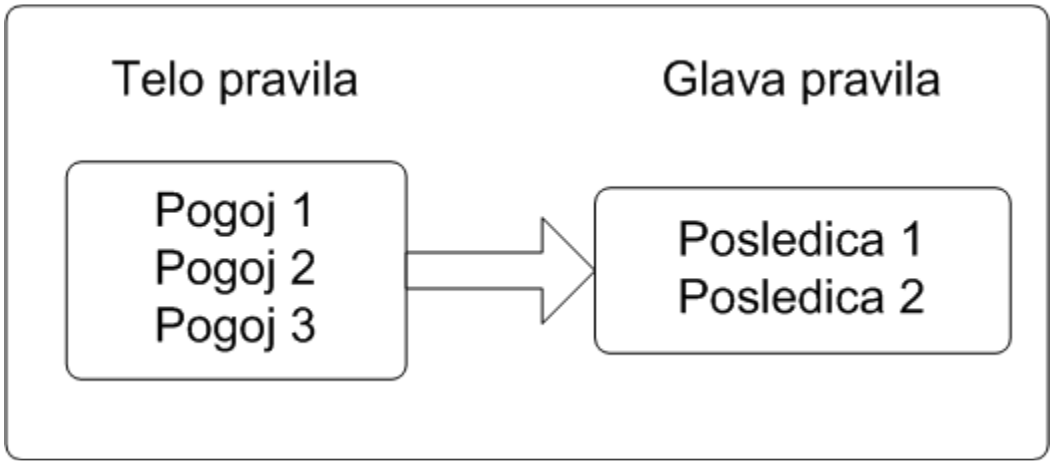
- Gradnja pravil brez operatorja ALI je enostavnejša za razumevanje, hkrati pa je manj možnosti za napake.
- Sistem gradnje pravil brez operatorja ALI je lažji za implementacijo.

3.2. Načini zajema pravil

Čeprav so v bistvu vsa pravila oblike »Telo => Glava«, zna biti zajem večjega števila pravil na ta način počasen in mukotrpen. Zato moj sistem poleg tega osnovnega načina zajema pravil omogoča še dva dodatna načina, z uporabo katerih zajem olajšamo in pospešimo. To sta zajem s pomočjo odločitvene tabele in zajem s pomočjo odločitvenega drevesa.

3.2.1. Zajem »Telo => glava«

To je osnoven način zajema pravil. Določimo atome telesa pravila in atome glave pravila.



Slika 7: Zajem pravila na način "Telo => Glava"

Rezultat je eno samo pravilo v abstraktni sintaksi.

3.2.2. Zajem z odločitveno tabelo

Osnova tega načina zajema pravil je odločitvena tabela naslednje oblike:

Atomi telesa (pogoji)	Permutacije veljavnosti pogojev
Atomi glave (posledice)	Izbrane posledice, ki veljajo pri vsaki od kombinacij pogojev

Tabela 2: Oblika odločitvene tabele

V levem zgornjem delu tabele so atomi telesa (pogoji), v levem spodnjem delu pa atomi glave (posledice). V desnem zgornjem delu so vse možne permutacije veljavnosti pogojev. Če je atomov pogojev N, potem je permutacij (in s tem stolpcev) 2^N. V spodnjem desnem delu v vsakem stolpcu za vsako posledico označimo, ali naj pri tej permutaciji velja.

Pogoj 1	DA	DA	DA	DA	NE	NE	NE	NE
Pogoj 2	DA	DA	NE	NE	DA	DA	NE	NE
Pogoj 3	DA	NE	DA	NE	DA	NE	DA	NE
Posledica 1							DA	
Posledica 2		DA						
Posledica 3							DA	

Tabela 3: Primer odločitvene tabele

Rezultat take tabele je eno »Telo => Glava« pravilo za vsak stolpec, kjer je z vrednostjo DA označena vsaj ena posledica (torej največ 2^N pravil).

Ta način od popolne uporabnosti loči ena bistvena pomanjkljivost, ki pa je povezana z že omenjeno značilnostjo SWRL jezika – monotonostjo. Le-ta namreč prepoveduje negiranje atomov. Vrednost NE pri nekem pogoju tako pomeni le to, da pravilnost tega pogoja za veljavnost telesa pravila ni pomembna, ne pa da je nujna napačnost tega pogoja.

Iz drugega stolpca v zgornjem primeru (Pogoj 1 – DA, Pogoj 2 – DA, Pogoj 3 – NE, Posledica 1 - NE , Posledica 2 – DA, Posledica 3 - NE) tako dobimo pravilo

```
Pogoj1 ∧ Pogoj2 => Posledica2
```

in ne

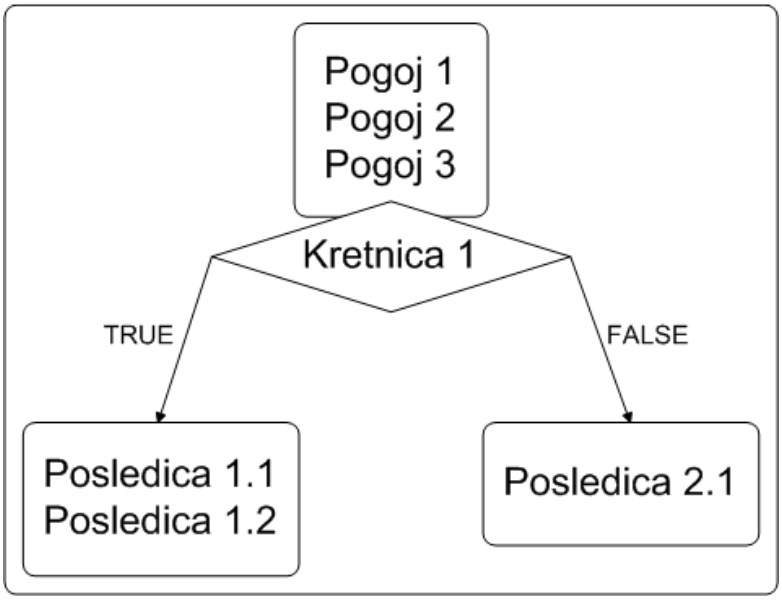
```
Pogoj1 ∧ Pogoj2 ∧ ¬(Pogoj3) => Posledica2.
```

Zajem pravil na ta način je tako uporaben predvsem takrat, kadar želimo naenkrat zajeti več pravil, katerih unija pogojev v telesih je razmeroma majhna, presek pa morda najdemo tudi med posledicami v glavah pravil. Pri večjem številu različnih pogojev v telesih ta način zajema postane nepregleden, saj število permutacij in s tem število stolpcev v tabeli eksponentno raste. Število posledic v glavah na zajem ne vpliva negativno, saj število vrstic raste linearno.

3.2.3. Zajem z odločitvenim drevesom

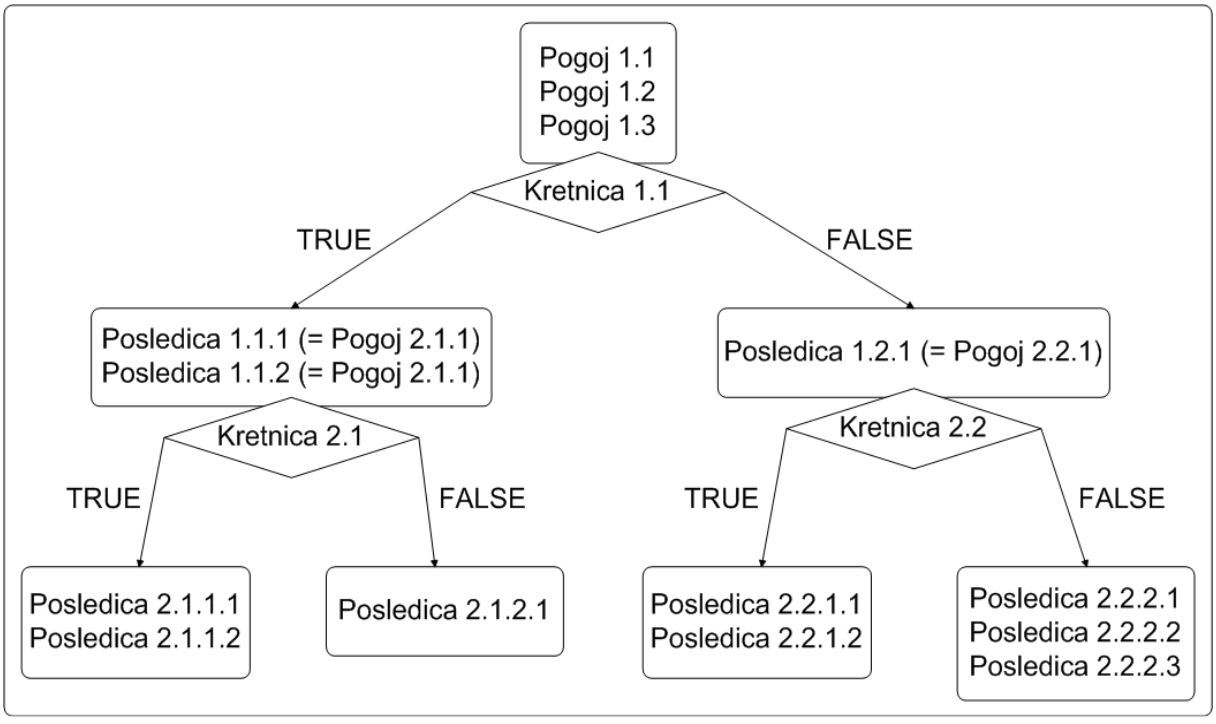
Zajem pravil v obliki odločitvenega drevesa pride v poštev takrat, kadar se telesi dveh pravil razlikujeta le v enem pogoju – **kretnici** - katerega pravilnost pa odloča o glavi pravila.

Zajem takih pravil poteka z gradnjo binarnega odločitvenega drevesa:



Slika 8: Zajem pravila z enonivojskim odločitvenim drevesom

Drevo lahko gradimo tudi naprej:



Slika 9: Zajema pravila z dvonivojskim odločitvenim drevesom

Tudi pri tem načinu zajema se srečamo s problemom monotonosti - kako realizirati kretnice? Rešitev je v omejitvi tipov atomov, ki jih lahko uporabimo kot kretnice, na *zaloge vrednosti*

in *vgrajene predikate*. Pri teh dveh tipih atomov lahko negirano kretnico predstavimo kot enega ali več monotonih pogojev. Najlažje je pri vgrajenih predikatih za primerjanje. Tako je recimo negacija atoma

```
equal(?x, ?y)
```

atom

```
notEqual(?x, ?y).
```

Malce bolj zapleteno je na primer pri negaciji atoma

```
inDataRange(?x, #oče, mati, sin#),
```

saj ga moramo nadomestiti kar s tremi atomi:

```
notEqual(?x, #oče#) ∧ notEqual(?x, #mati#) ∧ notEqual(?x, #sin#).
```

Na prvi pogled se zdi, da bi lahko navedeno rešitev problema monotonosti uporabili tudi pri načinu zajema pravil z odločitveno tabelo, vendar pa se ob natančnejši analizi izkaže, da to ne bi bilo praktično. V telesu pravila bi se namreč nahajali dve vrsti atomov – taki, ki jih lahko negiramo, in taki, ki jih ne moremo. Uporabnik bi tako moral biti ob zajemu pravil neprestano pozoren na vrsto atoma, kar pa je vse prej kot uporabniku prijazno. Zato sem se odločil, da negacijo omogočim le za potrebe kretnic.

Rezultat zajema pravil z odločitvenim drevesom je po eno pravilo za vsako glavo v drevesu. Pri gradnji pravila na nižjem nivoju uporabimo za telo pravila kar glavo pravila na višjem nivoju. Pogoj atoma kretnice oz. atomov njegove negacije dodamo telesu pravila.

4. PROBLEMSKA DOMENA TRGOVANJA S FINANČNIMI INSTRUMENTI

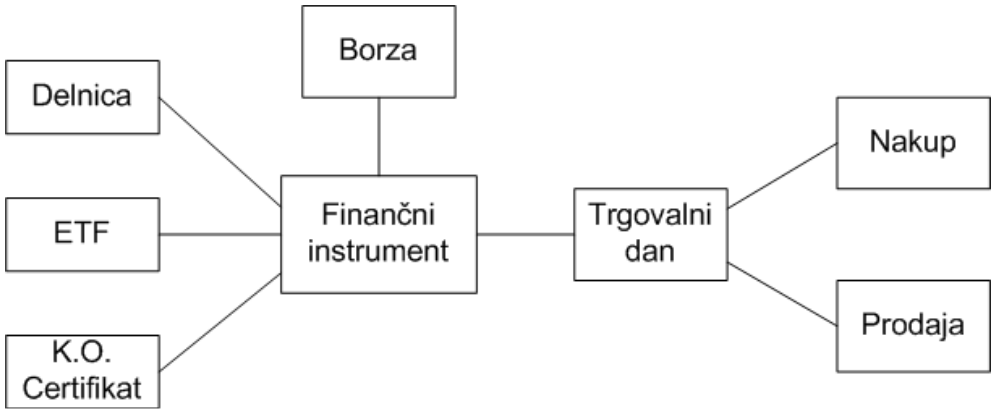
Kot problemsko domeno za predstavitev razvitega sistema zajema poslovnih pravil sem izbral domeno trgovanja s finančnimi instrumenti (v nadaljevanju FI). Postavil sem se v vlogo vlagatelja, ki bi rad uporabljal aplikacijo IntelliOnto za avtomatizacijo trgovanja s FI.

Osnovna podlaga avtomatizacije je ontologija, ki opisuje trg. Eden glavnih razredov v tej ontologiji je razred *Trgovalni_dan*, ki vsebuje podatke o kupčijah z nekim FI na nek dan. Primerki razreda *Trgovalni_dan* lahko pripadajo enemu ali več podrazredom. Pripadnost podrazredom je določena s strategijo trgovanja, zapisano s poslovnimi pravili. Določanje pripadnosti poteka avtomatsko z zagonom stroja za sklepanje nad ontologijo in poslovnimi pravili. Če ta proces določi, da nek primerek pripada podrazredu *Nakup* ali *Prodaja*, to pomeni signal uporabniku, naj kupi oz. proda FI.

V tem poglavju bom opisal ontologijo ter strategijo trgovanja s FI, ki sem ju implementiral.

4.1. Ontologija

Ontologija trgovanja s FI je sestavljena iz dveh delov. Prvi del je osnovna ontologija, ki opisuje samo domeno trgovanja s FI. Drugi del ontologije dopolnjuje osnovno ontologijo z razredi in relacijami, ki so potrebne za implementacijo izbrane strategije trgovanja s FI.



Slika 10: Shema osnovne ontologije.

V naslednjih treh podpoglavjih bom naštel in navedel vse razrede in relacije ontologije. Večina razredov in relacij ima poleg navedenega slovenskega opisa vsaj še slovensko oznako ter angleški opis in oznako, ki pa jih nisem navajal. Ti podatki se ponekod s pridom uporabijo kot dodatna pomoč uporabniku pri zajemu pravil.

4.1.1. Razredi osnovne ontologije

- **Borza**
  - opis: borza
- **Financni_instrument**
  - opis: Poljubna oblika investiranja.
- **Delnica**
  - Podmnožica razreda *Financni_instrument*.
  - opis: Lastniški vrednostni papir.
- **ETF**
  - Podmnožica razreda *Financni_instrument*.
  - opis: Sklad, s katerim se trguje na borzi.
- **KO_certifikat**
  - Podmnožica razreda *Financni_instrument*.
  - opis: Špekulativni finančni instrument, vezan na nek drug finančni instrument.
- **Trgovalni_dan**
  - opis: Trgovalni podatki v okviru trgovalnega dne.
- **Nakup**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Trgovalni dan z nakupnim signalom.
- **Prodaja**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Trgovalni dan s prodajnim signalom.

4.1.2. Relacije osnovne ontologije

4.1.2.1. Objektne relacije:

- **kotiraNaBorzi**
  - opis: Finančni instrument kotira na borzi.
  - domena: *Financni_instrument*
  - doseg: *Borza*
  - inverzna relacija: *imaKotirajocFI*
- **imaKotirajocFI**
  - opis: Borza ima kotirajoč finančni instrument.
  - domena: *Borza*
  - doseg: *Financni_instrument*
  - inverzna relacija: *kotiraNaBorzi*
- **seTrgujeNaTrgovalniDan**
  - opis: Finančni instrument se trguje na trgovalni dan.
  - domena: *Financni_instrument*
  - doseg: *Trgovalni_dan*
  - inverzna relacija: *veljaZaFinancniInstrument*
- **veljaZaFinancniInstrument**
  - opis: Finančni instrument se trguje na trgovalni dan.

  - domena: *Trgovalni_dan*
  - doseg: *Financni_instrument*
  - inverzna relacija: *seTrgujeNaTrgovalniDan*
- **temeljiNaFI**
  - opis: ETF oz. K.O. certifikat temelji na finančnem instrumentu.
  - domena: *ETF, KO_certifikat*
  - doseg: *Financni_instrument*
  - inverzna relacija: *jeDelFI*
- **jeDelFI**
  - opis: Finančni instrument je del ETF-a oz. K.O. certifikata.
  - domena: *Financni_instrument*
  - doseg: *ETF, KO_certifikat*
  - inverzna relacija: *temeljiNaFI*
- **naslednjiTrgovalniDan**
  - opis: Trgovalni dan ima naslednji trgovalni dan.
  - domena: *Trgovalni_dan*
  - doseg: *Trgovalni_dan*
  - inverzna relacija: *prejsnjiTrgovalniDan*
- **prejsnjiTrgovalniDan**
  - opis: Trgovalni dan ima prejšnji trgovalni dan.
  - domena: *Trgovalni_dan*
  - doseg: *Trgovalni_dan*
  - inverzna relacija: *naslednjiTrgovalniDan*

4.1.2.2. Lastnosti:

- **ISIN**
  - opis: Enolična oznaka vrednostnega papirja.
  - domena: *Financni_instrument*
- **KO_meja**
  - opis: Knock-out meja.
  - domena: *KO_certifikat*
- **lokacija**
  - opis: Lokacija, kjer se borza nahaja (npr. Slovenija, ZDA, Kitajska).
  - domena: *Borza*
- **datum**
  - opis: Datum.
  - domena: *Trgovalni_dan*
- **open**
  - opis: Cena vrednostnega papirja ob prvem sklenjenem poslu.
  - domena: *Trgovalni_dan*
- **close**
  - opis: Cena vrednostnega papirja ob zadnjem sklenjenem poslu.
  - domena: *Trgovalni_dan*
- **low**
  - opis: Najnižja cena vrednostnega papirja pri sklenjenem poslu.
  - domena: *Trgovalni_dan*

- **high**
  - opis: Najvišja cena vrednostnega papirja pri sklenjenem poslu.
  - domena: *Trgovalni_dan*
- **volume**
  - opis: Število sklenjenih poslov.
  - domena: *Trgovalni_dan*
- **razlog_za_akcijo**
  - opis: Razlog za izvedbo akcije nakupa ali prodaje.
  - domena: *Trgovalni_dan*

4.1.3. Razredi, potrebni za implementacijo strategije

- **BodyColor**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Trgovalni dnevi glede na barvo telesa;
- **BlackBody**
  - Podmnožica razreda *BodyColor*.
  - opis: Cena je bila v okviru trgovalnega dne v povprečju rastoča.
- **WhiteBody**
  - Podmnožica razreda *BodyColor*.
  - opis: Cena je bila v okviru trgovalnega dne v povprečju padajoča.
- **ExtremeValues**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Trgovalni dnevi glede na najnižje oz. najvišje vrednosti.
- **MLT**
  - Podmnožica razreda *ExtremeValues*.
  - opis: Danes je najnižja vrednost.
- **MHT**
  - Podmnožica razreda *ExtremeValues*.
  - opis: Danes je najvišja dosežena cena v zadnjih treh dneh.
- **Marubozu**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Svečnik, ki nima senc.
- **Doji**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Svečnik, ki ima majhno telo (manj kot 10% velikosti svečnika).
- **Hammer**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Svečnik, ki se pojavi po negativnem trendu, in ima majhno telo, dolgo spodnjo senco in kratko zgornjo senco.
- **HangingMan**
  - Podmnožica razreda *Trgovalni_dan*.
  - opis: Svečnik, ki se pojavi po pozitivnem trendu, in ima majhno telo, dolgo spodnjo senco in kratko zgornjo senco.

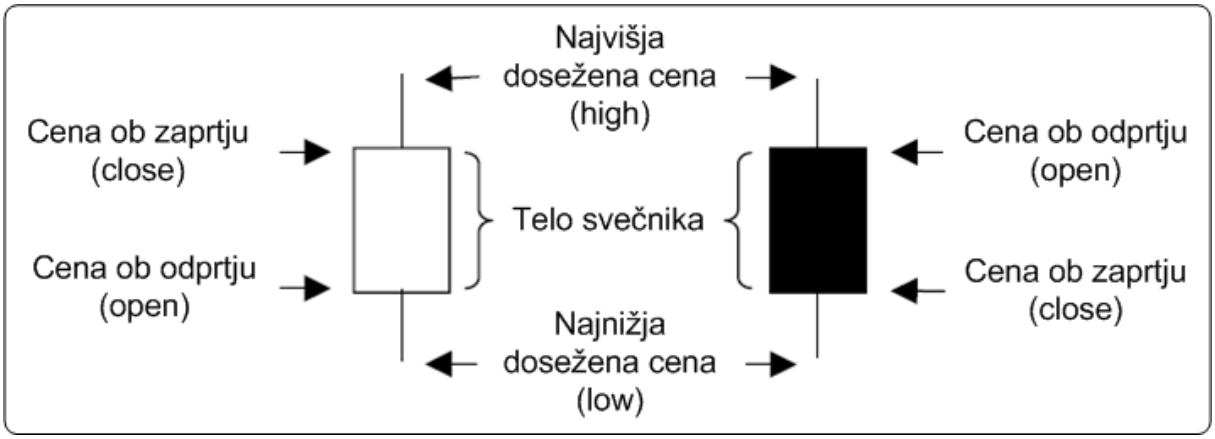
4.2. Strategija trgovanja s finančnimi instrumenti

Strategija trgovanja s FI, ki sem jo izbral za predstavitev delovanja razvitega modula za zajem poslovnih pravil, temelji na tehnični analizi grafikona japonskih svečnikov.

4.2.1. Tehnična analiza grafikona japonskih svečnikov

Tehnično analizo grafikona japonskih svečnikov so razvili v 17. stoletju na Japonskem, in sicer za potrebe trgovanja z rižem. Ta analiza dogajanje na trgu obravnava kot večen boj med biki (kupci) in medvedi (prodajalci). Kadar so biki močnejši, se cene FI zvišujejo, kadar pa so močnejši medvedi, se cene FI znižujejo. Analiza tako skozi življenjsko dobo FI zaznava medvedje in bikovske trende. [7,13, 16, 22]

Osnovna enota grafikona je svečnik, ki prikazuje podatke o trgovanju s FI na določen trgovalni dan.



Slika 11: Grafični prikaz belega in črnega svečnika

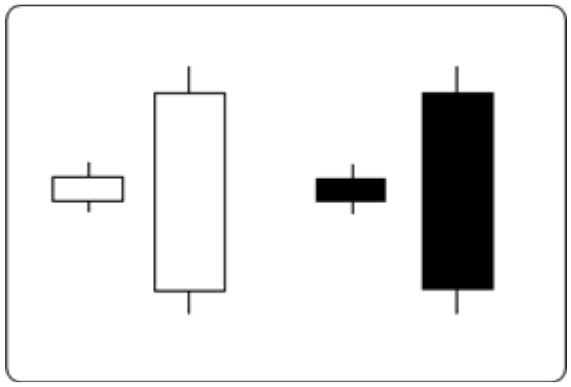
Vsak svečnik zaznamujejo štiri vrednosti:

- cena ob odprtju,
- cena ob zaprtju,
- najvišja dosežena cena,
- najnižja dosežena cena.

Kadar je cena ob zaprtju višja od cene ob odprtju, je telo svečnika bele barve (white body). Če pa je obratno, je telo črne barve (black body). Črtam nad telesi se reče sence. Glede na obliko ima vsak svečnik svoj pomen in ime.

4.2.1.1. Nekaj oblik svečnikov

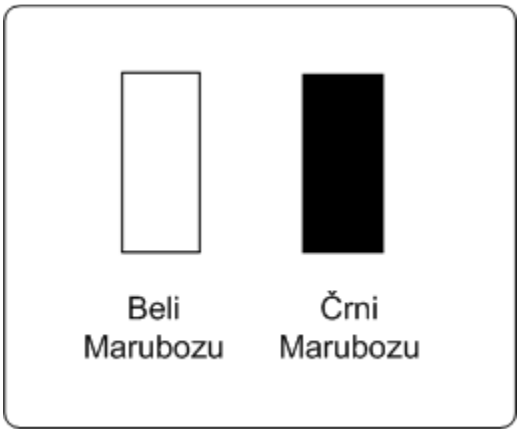
Dolgi oz. kratki svečniki



Slika 12: Grafični prikaz dolgega in kratkega svečnika

Dolgi beli svečniki nakazujejo močan nakupni pritisk. Daljši kot je bel svečnik, višje je bila cena ob zaprtju od cene ob odprtju. To nakazuje, da so bili kupci agresivni in so tako cene od odprtja do zaprtja bistveno narasle. .

Dolgi črni svečniki nakazujejo močan prodajni pritisk. Daljši kot je črn svečnik, nižje je bila cena ob zaprtju od cene ob odprtju. To nakazuje, da so bili bolj agresivni prodajalci in so tako cene od odprtja do zaprtja bistveno padle.**Marubozu**



Slika 13: Grafični prikaz belega in črnega Marubozu svečnika

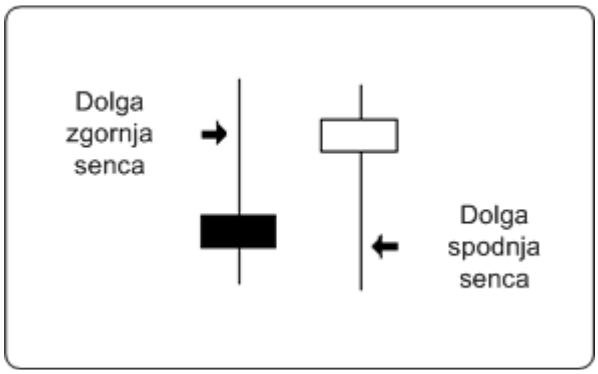
Še pomenljivejša sta črni in beli Marubozu svečnik, saj nimata zgornje in spodnje sence.

Beli Marubozu svečnik nastopi, kadar je cena ob odprtju enaka najnižji doseženi ceni in cena ob zaprtju enaka najvišji doseženi ceni. To nakazuje, da je presežek povpraševanja nad ponudbo nadzoroval trgovanje od prve do zadnje kupčije.

Črni Marubozu svečnik pa nastopi, kadar je cena ob odprtju enaka najvišji doseženi ceni in cena ob zaprtju enaka najnižji doseženi ceni. To nakazuje ravno obratno od belega Marubozu

svečnika – trgovanje je od prve do zadnje kupčije nadzoroval presežek ponudbe nad povpraševanjem.

Dolge oz. kratke sence



Slika 14: Grafični prikaz dolge zgornje in dolge spodnje sence

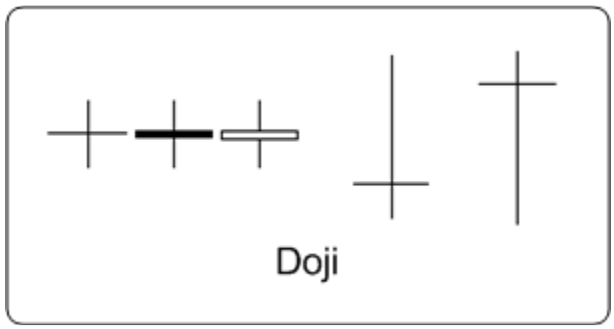
Zgornje in spodnje sence svečnikov lahko razkrijejo dragocene informacije o dogajanju prek celega trgovalnega dne. Zgornje sence predstavljajo najvišjo doseženo ceno in spodnje sence najnižjo doseženo ceno trgovalnega dne.

Svečniki s kratkimi sencami nakazujejo da je bila večina kupčij sklenjenih blizu cen ob odprtju in zaprtju. Svečniki z dolgimi sencami pokažejo da je kupčevanje potekalo tudi precej nad oz. pod ceno ob odprtju in ceno ob zaprtju.

Svečniki z dolgimi zgornjimi sencami in kratkimi spodnjimi sencami nakazujejo, da so do neke točke v trgovalnem dnevi prevladovali kupci, kar je botrovalo zvišanju cen. Kasneje je prišlo do preobrata, prodajalci so bili prisiljeni v znižanje cen in šibka cena ob zaprtju je ustvarila dolgo zgornjo senco.

Ravno obratno svečniki z dolgimi spodnjimi sencami in kratkimi zgornjimi sencami nakazujejo, da so do neke točke v trgovalnem dnevu prevladovali prodajalci, kar je botrovalo znižanju cen. Kasneje je prišlo do preobrata, kupci s svojim povpraševanjem zvišali cene in visoka cena ob zaprtju je ustvarila dolgo spodnjo senco.

Doji



Slika 15: Grafični prikaz različnih oblik Doji svečnika

Doji so pomembni svečniki in vsebujejo informacije tako sami po sebi kakor s svojo prisotnostjo v številnih pomembnih vzorcih. Doji nastopijo, kadar sta cena ob odprtju in cena ob zaprtju praktično enaki. Dolžina zgornje in spodnje sence lahko variira.

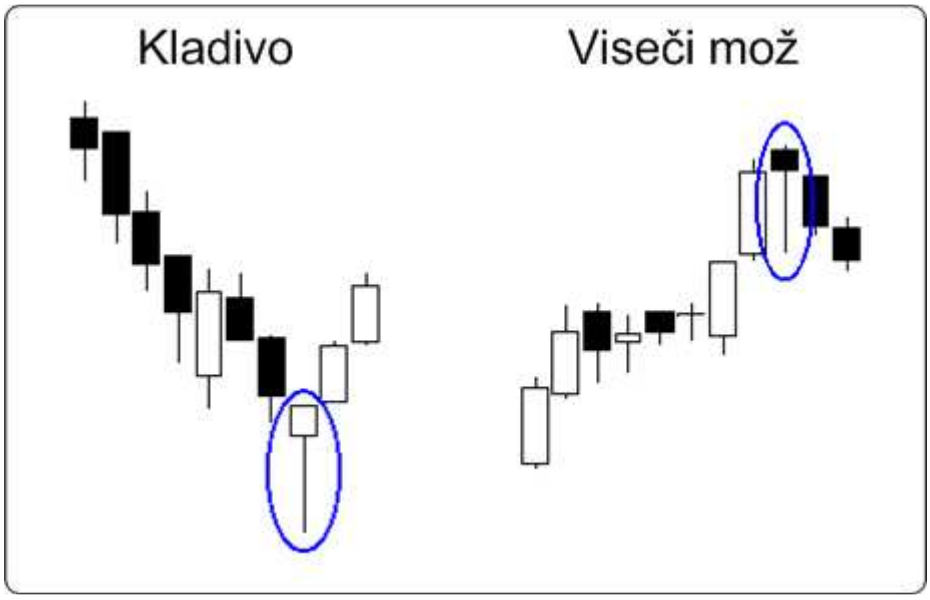
4.2.1.2. Vzorci svečnikov

Nekateri svečniki že samostojno razkrijejo marsikatero informacijo, na podlagi katere se lahko odločimo za nakup ali prodajo FI. Svojo pravo izrazno moč pa dobijo šele, ko jih obravnavamo skupaj z njegovimi predhodniki in nasledniki. Tak vzorec vsebuje od enega do največ pet svečnikov. Večina vzorcev se uporablja za ugotavljanje spremembe smeri trenda, obstajajo pa seveda tudi takšni, ki potrjujejo nadaljevanje obstoječega trenda. Svetovna literatura navaja okoli štirideset različnih vzorcev spremembe trenda in vsaj šestnajst vzorcev, ki napovedujejo nadaljevanje trenda.

Značilnosti vseh teh vzorcev se dajo zapisati kot skupek poslovnih pravil. Če ta pravila veljajo, potem je bil vzorec najden in to je lahko signal za vlagatelja, naj kupi oz. proda FI, ali pa naj bo pozoren na spremembo trenda.

Za predstavitev delovanja modula sem si izbral samo en vzorec:

Kladivo in Viseči mož



Slika 16: Grafični prikaz vzorcev Kladivo in Viseči mož

Kladivo in Viseči mož sta enaka po obliki. Oba imata majhno telo (črno ali belo), dolgo spodnjo senco in kratko ali celo neobstoječo zgornjo senco. Bistveno pa se razlikujeta v pomenu, ki temelji na predhodnem trendu cen FI.

Kladivo je vzorec bikovskega preobrata in se pojavi po dolgotrajnejšem negativnem trendu cen. Najnižja dosežena cena dolge spodnje sence nakazuje, da so prodajalci s svojo

agresivnostjo med trgovalnim dnem zbili ceno. Proti koncu trgovalnega dne pa so kupci močno prevzeli pobudo in trgovalni dan se je končal z precej visoko ceno. Čeprav se to zdi zadosten razlog za nakup, Kladivo potrebuje dodatno bikovsko potrditev. Najboljša taka potrditev je dolg bel svečnik.

Za razliko od Kladiva je Viseči mož vzorec medvedjega preobrata. S svojim pojavom po dolgotrajnejšem pozitivnem trendu cen nakazuje, da se pritisk prodajalcev povečuje. Najnižja dosežena cena dolge spodnje sence priča o tem, da so prodajalci s svojo agresivnostjo cene potisnili nizko, kar pa je kljub poznejšemu preobratu s strani kupcev signal za previdnost. Vendar pa tudi Viseči mož pred ukrepanjem potrebuje medvedjo potrditev, najraje v obliki dolgega črnega svečnika.

4.2.2. Moja strategija trgovanja s FI

Na podlagi opisanih oblik ter vzorcev svečnikov sem razvil svojo strategijo trgovanja s FI. Sem previden vlagatelj, zato se za nakup FI odločam le ob zanesljivih napovedih spremembe trenda iz negativnega v pozitivni. Po drugi strani se za prodajo odločim že ob manjših signalih, ki namigujejo na negativen trend v prihodnosti.

Za nakup se odločim takrat, kadar se pojavijo naslednji vzorci:

- beli Marubozu,
- belo Kladivo s potrditvijo v obliki dolgega belega svečnika.

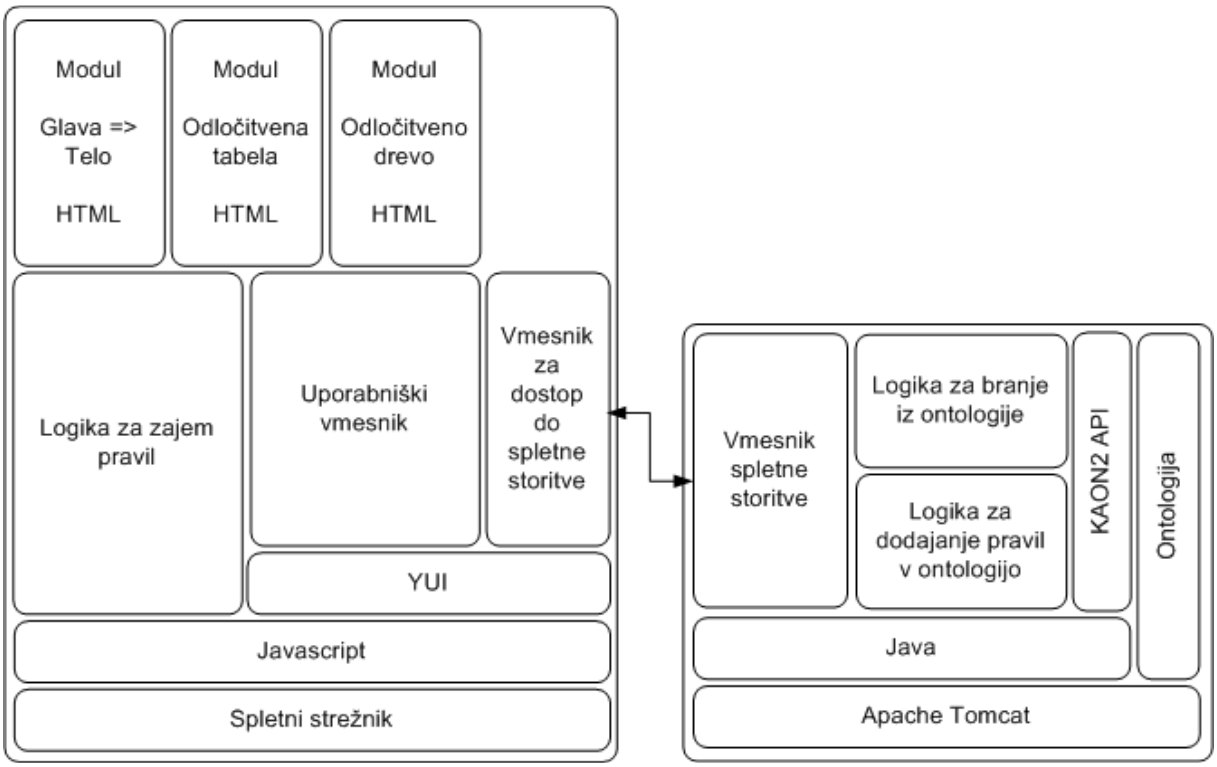
Za prodajo se odločim takrat, kadar se pojavijo naslednji vzorci:

- črni Marubozu,
- črni Viseči mož,
- beli Viseči mož s potrditvijo v obliki Doji ali črnega svečnika.

5. Implementacija

5.1. Arhitektura aplikacije

Ker je aplikacija IntelliOnto še v začetnih stadijih razvoja, sem svoj modul implementiral kot samostojno aplikacijo, ki pa jo bo lahko razmeroma preprosto preoblikovati v modul aplikacije IntelliOnto.



Slika 17: Struktura aplikacije

Aplikacija, ki sem jo razvil, je sestavljena iz dveh delov - spletne aplikacije in spletne storitve.

5.1.1. Spletna aplikacija

Spletna aplikacija skrbi za zajem pravil. Sestavljajo jo naslednji elementi:

- **HTML moduli**

  HTML moduli vsebujejo ogrodje uporabniškega vmesnika, torej HTML tabelo, gumbe, oddelek za prikaz niza s pravilom v abstraktni sintaksi itd. Vsebujejo tudi globalne

spremenljivke in nekatere funkcije, katerih vsebina je odvisna od samega načina zajema pravil, na primer funkcijo za gradnjo niza s pravili v abstraktni sintaksi.

- **Uporabniški vmesnik**

  Funkcije uporabniškega vmesnika skrbijo za grafični prikaz atomov znotraj ogrodja, določenega v HTML modulu. Skrbijo tudi za vsebino in prikazovanje menija za vnos argumentov ter kontekstnega menija. Te funkcije temeljijo na komponentah knjižnice YUI.

- **Logika za zajem pravil**

  Funkcije tega elementa skrbijo za:

  - logično plat zajema pravil, na primer za ustvarjanje objektov, zajem vrednosti in tipa vnešenih argumentov, generiranje niza z zajetim pravilom v abstraktni sintaksi itd.,
  - klice funkcij uporabniškega vmesnika,
  - klice funkcij vmesnika za dostop do ontologije.

- **Vmesnik za dostop do spletne storitve**

Funkcije tega vmesnika omogočajo klice funkcij spletne storitve. S klici teh funkcij pridobivamo podatke o ontologiji, ali pa spletni storitvi pošljemo niz s pravili v abstraktni sintaksi, da se pretvorijo v SWRL obliko in dodajo ontologiji.

Klici funkcij spletne storitve potekajo asinhrono, saj so implementirani s komponento *ConnectionManager* knjižnice YUI, ki temelji na tehnologiji AJAX.

- **YUI**

  YUI (Yahoo User Interface) knjižnica je odprtokodna zbirka orodij in kontrolnikov, napisanih v JavaScript jeziku. Knjižnica nam omogoča razvoj močno interaktivnih spletnih aplikacij. Uporablja moderne tehnologije, kot so DOM (Document Object Model – dokumentni objektni model) skriptiranje, DHTML (Dynamic HTML – dinamični HTML) in AJAX (Asynchronous JavaScript and XML – asinhroni JavaScript in XML). [23]

  V svoji aplikaciji uporabljam naslednje gradnike YUI knjižnice:

  - **Overlay Container**: z uporabo tega gradnika sem implementiral meni za vnos argumentov atoma.
  - **Context Menu**: z uporabo tega gradnika sem implementiral kontekstni meni za dodajanje, ponastavljanje in brisanje atomov.
  - **Connection Manager**: z uporabo tega gradnika sem implementiral klice funkcij spletne storitve.

Poleg naštetih gradnikov sem uporabil tudi orodje »yahoo-dom-event« za zaznavanje klikov na sidra atomov ter servisiranje teh klikov.

- **JavaScript**

  Vsa programska koda spletne aplikacije je napisana v jeziku JavaScript.

- **Spletni strežnik**

  Spletna aplikacija je nameščena na poljubnem spletnem strežniku

5.1.2. Spletna storitev

Spletna storitev preko funkcij svojega vmesnika omogoča spletni storitvi branje podatkov iz ontologije oz. pisanje podatkov vanjo. Manipulacija ontologije poteka z uporabo KAON2 API.

KAON2 (Karlsruhe ontology 2) je infrastruktura za upravljanje z OWL-DL in SWRL ontologijami. Poleg različnih orodij za gradnjo ontologij, strežnika za ontologije, stroja za sklepanje itd. ponuja tudi javansko knjižnico KAON2 API. Ta knjižnica vsebuje vmesnik, preko katerega lahko enostavno manipuliramo z ontologijami, iz njih prebiramo podatke ali jih vanjo zapisujemo. [18]

Kadar želi spletna aplikacija iz ontologije prebrati podatke, pokliče ustrezno funkcijo vmesnika spletne storitve. Funkcije logike za branje iz ontologije z uporabo KAON2 API pridobijo potrebne podatke iz ontologije in jih pravilno strukturirajo. Klicana funkcija vmesnika nato vrne zahtevane podatke spletni aplikaciji.

Edina funkcija vmesnika, ki omogoča pisanje v ontologijo, sprejme kot parameter niz z zajetimi poslovnimi pravili v abstraktni sintaksi. Logika za dodajanje pravil v ontologijo ta niz najprej razdeli na posamezna pravila, nato pa vsako pravilo z uporabo KAON2 API pretvori v SWRL obliko in ga zapiše v ontologijo.

Spletna storitev je implementirana v Javi, nahaja pa se na Apache Tomcat 6.0 strežniku.

5.2. Podrobnosti implementacije

5.2.1. Atomi

Na atome v aplikaciji lahko gledamo z dveh vidikov. S programsko-logičnega vidika obravnavamo atome kot objekte razreda atom. Zanimajo nas lastnosti tega objekta, njegove metode, povezovanje med atomi itd. Z vizualnega vidika pa obravnavamo atome kot HTML kodo, ki nam v spletni aplikaciji predstavlja atom. Tukaj nas zanima predvsem, kako kreirati HTML kodo, da bi na čim lepši in uporabnejši način prikazali podatke logičnega atoma.

5.2.2. Razred atom

Vsak objekt razreda atom ob kreaciji dobi enolično oznako oblike *Atom* + zaporedna številka atoma. Na novo ustvarjeni atomi so vedno tipa *empty*. Vsak atom vsebuje podatke, ki se uporabljajo za grafično predstavitev atoma. Na podlagi teh podatkov in tipa atoma se s klicem metode *makeDiv()* ustvari začetna HTML koda za vizualno predstavitev atoma. V procesu zajema poslovnega pravila se ti podatki dopolnjujejo in spreminjajo.

Ko so zajeti vsi potrebni podatki, lahko pokličemo metodo *createRuleString()*, ki vrne niz s pravilom tega atoma v abstraktni sintaksi. Vsak atom vsebuje tudi podatke o domeni, dosegu in drugih omejitvah, ki vplivajo na izbiranje argumentov.

Atom lahko vsebuje tudi kazalec na naslednji atom, s čimer na preprost način gradimo sezname. Ob končni pretvorbi pravila v abstraktno sintakso tako zlahka sestavljamo pravila posameznih atomov.

5.2.3. Vizualna predstavitev atomov

Obrazec atoma, opisan v tretjem poglavju, sem realiziral s preprosto HTML kodo oblike

```
(vezniTekst1 <A>argument1</A> vezniTekst2 <A>argument2</A> vezniTekst3 …),
```

ki da ob prikazu naslednji rezultat:

(vezniTekst1 argument1 vezniTekst2 argument2 vezniTekst3 …).

Vidimo, da je vezni tekst kar navaden tekst, argumenti pa so HTML sidra (<a> tag). Vezni tekst je pri večini tipov atoma fiksen, pri nekaterih pa je pridobljen iz ontologije. Prav tako je od tipa atoma odvisno število sider in njihov tekst, ki nakazuje tip argumenta, ki ga mora uporabnik izbrati.

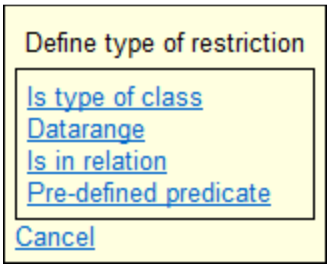
Morda bi kdo oporekal uporabi sidra, češ da bi bil za nalogo vnosa argumentov morda bolj primeren kak drug HTML element, ki bi bolje nakazoval, da se bo ob kliku prikazal seznam oz. meni (npr. polje s spustnim seznamom). Vendar pa sem se za uporabo sidra odločil predvsem zaradi njegovega videza. Sidro namreč še vedno nakazuje, da se bo ob kliku nekaj zgodilo, v osnovi pa je navaden tekst, kar veliko pripomore k berljivosti in s tem k udobju zajemanja pravil.

Z mislijo na udobje uporabnika sem dodal tudi t.i. namige (tooltips), ki se pojavijo ob kratkem postanku nad sidri, in ponujajo dodatne informacije o zahtevanem ali že izbranem argumentu.

5.2.3.1. Meni

Vnos argumentov torej začnemo s klikom na sidro. Pri tem se pokliče funkcija, ki najprej prepreči normalno izvajanje sidrnega elementa (torej skok na drugo spletno stran), nato pa prikaže meni.

Za implementacijo menija sem uporabil *Overlay* element knjižnice YUI. Njegova uporabna lastnost je, da lahko dinamično spreminjamo njegovo vsebino (HTML kodo), prikazujemo pa ga lahko lebdečega nad trenutno vsebino strani (seveda ga lahko tudi kadarkoli skrijemo).



Slika 18: Primer menija

Vsebino menija delimo na glavo, telo in nogo. V glavi vedno piše, kaj trenutno izbiramo, v nogi pa je sidro za zaprtje menija (kadar smo na osnovni strani menija) oz. za vrnitev na prejšnjo stran menija.

Vnos argumentov poteka preko sider v telesu menija. Navadno moramo najprej izbrati podtip argumenta. Pri tem skočimo na podstran menija, kjer se prikaže bodisi seznam argumentov, bodisi obrazec za vnos nove spremenljivke, zaloge vrednosti ali konstante. Pri slednjih hkrati z vnosom tudi izberemo vnešeni argument. Ponekod so na sidrih za izbiro argumenta oz. podtipa argumenta na voljo tudi namigi, ki olajšajo izbiro.

Ko izberemo argument, se tekst sidra, za katerega smo prikazali meni, spremeni na vrednost izbranega argumenta. Če smo vnesli novo spremenljivko, se njenemu imenu na začetku in koncu doda znak '?', če pa smo vnesli novo konstanto, se le-ta obda z znakoma '#'. Tako uporabnik že na prvi pogled ve, kakšen je podtip argumentov v atomu.

Obenem s spremembo teksta sidra se vrednost argumenta zapiše tudi v pripadajoči objekt atoma. Za potrebe ustvarjanja pravila atoma v abstraktni sintaksi se zapiše tudi podtip argumenta.

5.2.3.2. Tipi atomov in njihove značilnosti

Po kratki analizi sem ugotovil, da vsak atom spada v enega naslednjih tipov:

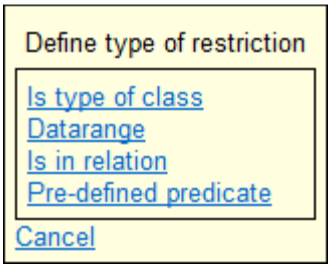
- **prazen atom,**
- **atom razreda,**
- **atom zaloge vrednosti,**
- **atom primerkovne relacije,**
- **atom podatkovne relacije,**
- **5 tipov atomov vgrajenih predikatov:**
  - atom vgrajenega predikata 2-1,
  - atom vgrajenega predikata 2-2,
  - atom vgrajenega predikata 2-3,
  - atom vgrajenega predikata 3-1,
  - atom vgrajenega predikata 4-1.

V nadaljevanju bom podrobneje opisal njihove obrazce, menije in morebitne posebne funkcionalnosti.

Prazen atom

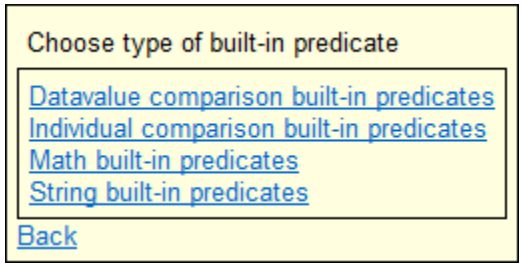
Obrazec: (choose restriction)

Prazen atom je osnovni tip, ki služi izbiri dejanskega tipa atoma. Ob kliku na argument se pojavi meni, kjer izbiramo med naslednjimi možnostmi:



Slika 19: Meni za izbiro tipa atoma

Če izberemo »Is type of class«, se tip atoma spremeni v *atom razreda*. Če izberemo »Datarange« se tip atoma spremeni v *atom zaloge vrednosti*.
Če izberemo »Is in relation«, se odpre podstran menija, kjer so naštete vse relacije ontologije. Ko izberemo eno izmed njih, aplikacija ugotovi, kakšnega tipa je relacija (primerkovna ali podatkovna), in tip atoma se ustrezno spremeni v *primerkovna relacija* ali v *podatkovna relacija*. Obenem se v atom zapišejo podatki o veznem tekstu obrazca.
Če izbiremo »Pre-defined predicate«, se najprej odpre podstran menija, kjer izberemo vrsto vgrajenega predikata:



Slika 20: Meni za izbiro vrste vgrajenega predikata

Ob izbiri vrste vgrajenega predikata se odpre podstran menija, kjer je seznam vseh vgrajenih predikatov izbrane vrste. Ko izberemo enega izmed njih, aplikacija ugotovi, kakšnega tipa je izbrani vgrajeni predikat, in tip atoma se spremeni v ustrezen tip vgrajenega predikata.

Ko je tip atoma spremenjen, se s klicem *makeDiv()* funkcije spremeni tudi obrazec atoma.

Atom razreda

Obrazec: (choose individual is choose class)

Atom razreda se uporablja za izražanje pripadnosti primerka nekemu razredu.

Izpolnjevanja tega obrazca se lahko lotimo na dva načina. Če želimo najprej vnesti prvi argument (primerek), potem nam meni ponudi tri možnosti:

- Existing variable
- New variable
- Value

Če izberemo tretjo možnost, potem nam podmeni ponudi seznam le tistih primerkov, ki še ne pripadajo nobenemu razredu. Po izbiri primerka izberemo še vrednost drugega argumenta (razred), in sicer s seznama vseh razredov, ki nam ga ponudi meni.

Malce bolj zapleteno je, če izberemo eno od prvih dveh možnosti. Ker vse primerkovne spremenljivke v aplikaciji pripadajo nekemu razredu, moramo ob izbrani eni od prvih dveh možnosti na podstrani menija najprej izbrati, kateremu razredu pripada oz. bo pripadala spremenljivka. Ko izberemo razred, se na podstrani v prvem primeru prikaže seznam obstoječih spremenljivk, pripadajočih izbranemu razredu, v drugem primeru pa se prikaže obrazec za vnos nove spremenljivke, ki bo pripadala izbranemu razredu.

Ob izbiri obstoječe oz. vnosu nove spremenljivke se prvi argument nastavi na novo vrednost. Obenem pa se drugi argument (razred) nastavi na razred, ki mu pripada izbrana oz. ustvarjena spremenljivka. S tem se uporabniku olajša delo, ni mu namreč potrebno posebej vnašati drugega argumenta, obenem pa se preprečijo morebitne logične napake.

Če se lotimo izpolnjevanja obrazca na drugi način, potem najprej določimo vrednost drugemu argumentu (razred). Če nato za prvi argument izberemo obstoječo ali novo spremenljivko, korak izbire razreda, ki mu spremenljivka pripada, izpustimo. Privzame se namreč, da spremenljivka pripada razredu iz drugega argumenta. Če za prvi argument izberemo primerek, izbira poteka enako kot pri prvem načinu.

Ne glede na to, kateri način smo izbrali pri izpolnjevanju obrazca, lahko pride do logičnih napak v primeru, če naknadno spremenimo vrednost drugega argumenta na nek nov razred. Če je spremenljivka iz prvega argumenta uporabljena samo v tem atomu, potem zadostuje, da tej spremenljivki nastavimo pripadnost na novo vrednost drugega argumenta. Če pa spremenljivka nastopa še v katerem drugem atomu, lahko pride do neskladnosti. V tem primeru prav tako spremenimo pripadnost spremenljivke, vendar pa v vseh ostalih atomih argumente, kjer je bila izbrana ta spremenljivka, ponastavimo na prvotno vrednost. S tem zagotovimo logično skladnost celotnega pravila, ki ga gradimo.

Obrazec atoma v abstraktni sintaksi:

```
argument2(argument1)
```

Atom zaloge vrednosti

Obrazec: (choose data is in datarange choose datarange)

Atom zaloge vrednosti uporabimo takrat, ko hočemo določiti, da je podatkovna spremenljivka oz. konstanta zajeta v neki zalogi vrednosti ali v enem izmed podatkovnih tipov.

Vnos prvega (podatkovnega) argumenta je podoben vnosu primerkovnega argumenta, opisanega pri atomu razreda. Osnovni meni je enake oblike:

- Existing variable
- New variable
- Value

Razlika je v tem, da podatkovne spremenljivke ne pripadajo nobenemu razredu, zato je ob izbiri ene izmed prvih dveh možnosti korak izbire razreda spuščen, tako da nam meni takoj ponudi izbiro obstoječe spremenljivke s seznama oz. obrazec za vnos nove spremenljivke.

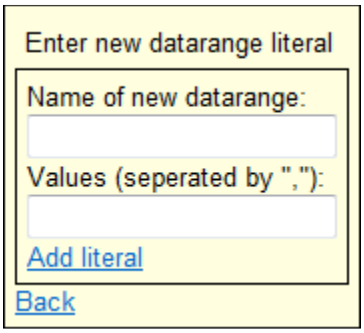
Razlikuje se tudi vnos vrednosti, saj pri podatkovnih argumentih ne izbiramo primerkov, pač pa vpisujemo konstanto. Zato se nam ob izbiri možnosti »Value« prikaže obrazec za vnos konstante.

Vnos argumenta zaloge vrednosti je precej specifičen. Osnovni meni je sicer podoben ostalim, razlikuje pa se po izbiri ene od možnosti:

- Existing datarange
- New datarange
- Datatype

Seznam zalog vrednosti je ob zagonu aplikacije prazen. Napolnimo ga z vnosom novih zalog vrednosti preko izbire »New datarange«. V obrazec, ki se nam prikaže na podstrani menija, moramo vnesti dva podatka:

- ime zaloge vrednosti,
- konstante, zajete v zalogi vrednosti, ločene z znakom ','.



Slika 21: Obrazec za vnos nove zaloge vrednosti

Ob izbiri »Existing datarange« izbiramo s seznama obstoječih zalog vrednosti. Pri tem je tekst sider v seznamu enak imenu zaloge vrednosti, konstante, zajete v tej zalogi vrednosti, pa se prikažejo kot namig tega sidra. Tako lažje najdemo zalogo vrednosti, ki jo iščemo.

V obeh primerih se v tekst argumenta zapiše ime zaloge vrednosti, v namig argumenta pa konstante v tej zalogi vrednosti.

Slika 22: Namig argumenta zaloge vrednosti

Tretja možnost osnovnega menija je »Datatype«, ki jo uporabimo, kadar želimo določiti, da spremenljivka pripada enemu izmed podatkovnih tipov. Izbiramo lahko med naslednjimi podatkovni tipi:

- Boolean,
- Integer,
- Byte,
- Short integer,
- Long integer,
- Float,
- Double float,
- String,
- Time,
- Date,
- Datetime,
- Duration

Vsa sidra vsebujejo namige z opisom podatkovnega tipa. Ti namigi se ob izbiri enega izmed tipov prenesejo na sidro vnešenega argumenta.

Slika 23: Namig sidra v meniju

Obrazec atoma v abstraktni sintaksi:

```
inDataRange(argument1,argument2)
```

Atom primerkovne relacije

Obrazec: (choose individual <vezni tekst – oznaka relacije> choose individual)

Atom primerkovne relacije izraža obstoj povezave med dvema primerkoma. Vezni tekst atoma je odvisen od relacije, ki jo izberemo, in se pridobi iz oznake, pripisane tej relaciji v ontologiji. Ta značilnost, skupaj s položajem veznega teksta na sredini med obema argumentoma, sledi načelu uporabe čim bolj naravnega jezika v atomih. Tako je npr. atom

(argument1 ima kotirajoč finančni instrument argument2)

precej bolj berljiv in laže razumljiv od istega atoma v abstraktni sintaksi:

```
imaKotirajocFI(argument1,argument2).
```

Uporabniku prijazen je tudi vnos obeh primerkovnih argumentov, saj upošteva morebitno domeno in obseg relacije. Obstoj le-teh se ob kreaciji atoma zapiše v namiga sider obeh argumentov, vpliva pa na potek vnosa argumentov.

Slika 24: Namig s prikazom omejitve zaradi domene argumenta

Oba primerkovna argumenta vnašamo preko istega menija kot pri vnosu primerkovnega argumenta atoma razreda in tudi tu sta možna dva poteka vnosa. Če je vrednost argumenta omejena z domeno oz. dosegom, potem se pri izbiri obstoječe ali vnosu nove spremenljivke privzame, da spremenljivka pripada razredu domene oz. dosega. Če pa argument ni omejen, moramo pred izbiro obstoječe oz. vnosom nove spremenljivke na podstrani menija izbrati razred, kateremu spremenljivka pripada oz. bo pripadala.

Enako velja pri vnosu primerka. Za razliko od atoma razreda pri relacijah ne obstaja omejitev na primerke, ki ne pripadajo nobenemu razredu. Izbiramo lahko torej med tistimi primerki, ki pripadajo razredu domene oz. dosega. Kadar pa domena oz. doseg ni določen, moramo pred izbiro primerka izbrati tudi razred, med primerki katerega želimo izbirati.

Pri vnosu argumentov relacije, kjer domena in/ali doseg nista določena, obstaja tudi varovalka, ki prepreči podvojene argumente in s tem nesmiselne atome.

Obrazec atoma v abstraktni sintaksi:

```
imePrimerkovneRelacije(argument1,argument2)
```

Atom podatkovne relacije

Obrazec: (<vezni tekst – oznaka relacije> of choose individual is choose data)

Atom primerkovne relacije izraža povezavo med primerkom in neko podatkovno vrednostjo. Tudi pri tem tipu atoma sem obliko obrazca prilagodil naravnemu jeziku. Podatkovna relacija pravzaprav predstavlja nek atribut primerka, to pa navadno izrazimo kot »(atribut) X primerka Y je Z«. »X«, v našem primeru prvi del veznega teksta (torej oznako relacije), dobimo iz ontologije.

Vnos primerkovnega argumenta je enak kot pri atomu primerkovne relacije, upošteva se tudi domena (dosega pri podatkovnih relacijah ni). Tudi pri vnosu podatkovnega argumenta ni nobene posebnosti, vnese se preko istega menija, ki se uporablja tudi pri vnosu podatkovnega argumenta pri atomu zaloge vrednosti.

Obrazec atoma v abstraktni sintaksi:

```
imePodatkovneRelacije(argument1,argument2)
```

Atomi vgrajenih predikatov

SWRL pozna veliko vgrajenih predikatov. Sam sem implementiral le osemnajst najuporabnejših, ostale pa se lahko na razmeroma preprost način v aplikacijo vgradinaknadno.

Vgrajeni predikati se med seboj razlikujejo po številu in tipu argumentov ter položaju vmesnega teksta. Iz osemnajstih vgrajenih predikatov, ki sem jih implementiral, sem tako izpeljal kar pet tipov atomov. Prva številka označuje število argumentov, ki nastopajo v atomu, druga pa različico atoma glede na položaj veznega teksta in/ali tip argumentov:

- **atom vgrajenega predikata 2-1**

  obrazec atoma: (podatkovniArgument1 <vezni tekst 1> podatkovniArgument2)

- **atom vgrajenega predikata 2-2**

  obrazec atoma: (<vezni tekst 1> podatkovniArgument1 <vezni tekst 2> podatkovniArgument2)

- **atom vgrajenega predikata 2-3**

  obrazec atoma: (primerkovniArgument1 <vezni tekst 1> primerkovniArgument1)

- **atom vgrajenega predikata 3-1**

  obrazec atoma: (podatkovniArgument1 <vezni tekst 1> podatkovniArgument2 <vezni tekst 2> podatkovniArgument3)

- **atom vgrajenega predikata 4-1**

  obrazec atoma: (podatkovniArgument1 <vezni tekst 1> podatkovniArgument2 <vezni tekst 2> podatkovniArgument3 <vezni tekst 3> podatkovniArgument4)

Na srečo je precej izbranih vgrajenih predikatov podobnih - če bi implementiral še vse preostale vgrajene predikate, bi bilo število tipov gotovo še precej večje. Kljub vsemu pa problem implementacije vgrajenih predikatov ni prevelik. Sicer jih je veliko, so pa vsi dobro definirani, se ne spreminjajo in jih je končno mnogo. To pomeni, da lahko atome vseh vgrajenih predikatov oblikujemo neposredno v kodi, ne da bi morali upoštevati nedoločenost podatkov iz ontologije. Tako jih lahko definiramo precej bolj natančno kot druge tipe atomov.

Obrazci atomov vgrajenih predikatov so tako precej bolj natančno določeni. Poleg svojega veznega teksta ima namreč večina tudi konkretno poimenovane argumente (začetni tekst teh argumentov), kar uporabniku še dodatno olajša vnašanje.

Implementiral sem naslednje vgrajene predikate, razdeljene v štiri skupine:

Vgrajeni predikati za primerjavo podatkov	
je enak (is equal)	Obrazec: (Datavalue1 is equal to Datavalue2) Preverjanje enakosti podatkov.
je različen (is not equal)	Obrazec: (Datavalue1 is not equal to Datavalue2) Preverjanje različnosti podatkov.
je manjši (is less than)	Obrazec: (Datavalue1 is less than Datavalue2) Primerjanje velikosti podatkov.
je manjši ali enak (is less than or equal)	Obrazec: (Datavalue1 is less than or equal to Datavalue2) Primerjanje velikosti podatkov.
je večji (is greater than)	Obrazec: (Datavalue1 is greater than Datavalue2) Primerjanje velikosti podatkov.
je večji ali enak (is greater than or equal)	Obrazec: (Datavalue1 is greater than or equal to Datavalue2) Primerjanje velikosti podatkov.

Tabela 4: Vgrajeni predikati za primerjavo podatkov

Vgrajeni predikati za primerjavo primerkov	
je isti (is same)	Obrazec: (Individual1 is same as Individual2) Preverjanje identičnosti podatkov.
ni isti (is different)	Obrazec: (Individual1 is different from Individual2) Preverjanje identičnosti podatkov.

Tabela 5: Vgrajeni predikati za primerjavo primerkov

Matematični vgrajeni predikati	
seštej (add)	Obrazec: (Sum is Number1 plus Number2) Seštevanje števil.
odštej (subtract)	Obrazec: (Difference is Number1 minus Number2) Odštevanje števil.
množi (multiply)	Obrazec: (Product is Number1 multiplied by Number2) Množenje števil.
deli (divide)	Obrazec: (Quotient is Number1 divided by Number2) Deljenje števil.
absolutna vrednost (absolute value)	Obrazec: (absolute value of Number1 is Number2) Izračun absolutne vrednosti.

Tabela 6: Matematični vgrajeni predikati

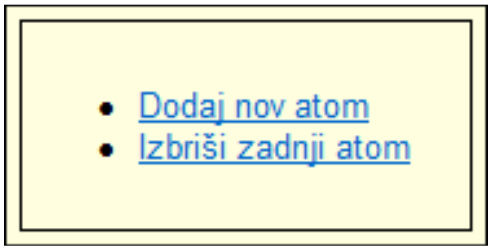
Vgrajeni predikati za delo z nizi	
stikanje nizov (string concatenation)	Obrazec: (Concatenated string is concatenation of String1 and String2) Stikanje nizov.
podniz (substring)	Obrazec: (Substring is substring of Original string starting with index fromIndex and ending with index toIndex) Določanje podnizov.
dolžina niza (string length)	Obrazec: (length of String is Length) Ugotavljanje dolžine niza.
niz z velikimi črkami (string in upper case)	Obrazec: (String in upper case is UpperCasedString) Pretvorba niza v niz s samimi velikimi črkami.
niz z malimi črkami (string in lower case)	Obrazec: (String in lower case is LowerCasedString) Pretvorba niza v niz s samimi malimi črkami.

Tabela 7: Vgrajeni predikati za delo z nizi

5.2.4. Implementacija načinov zajema pravil

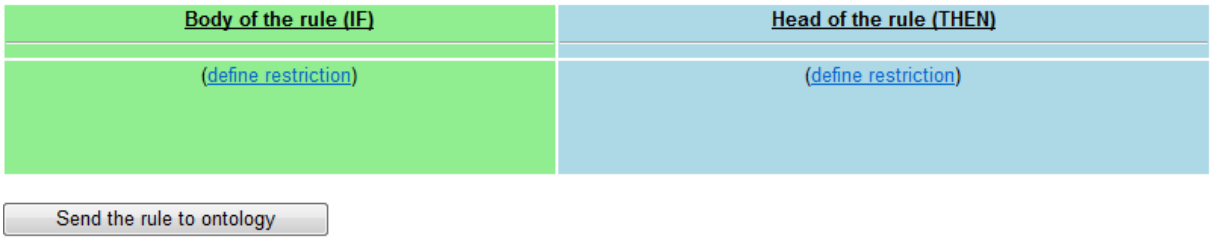
Implementiral sem vse tri načine zajema pravil, ki sem jih predlagal v tretjem poglavju. Izhodišče vsakega je posebna HTML stran, na kateri so definirani grafični vmesniki. Vsi trije načini uporabljajo skupne funkcije za prikaz obrazcev atomov ter vnašanje argumentov preko menija.

Dodajanje atomov je implementirano preko kontekstnega menija, do katerega dostopamo z desnim klikom na izbrano območje. Kontekstni meni je implementiran z uporabo YUI elementa »ContexMenu«.



Slika 25: Kontekstni meni za dodajanje in brisanje atomov

5.2.4.1. Modul »Telo => Glava«



Slika 26: Izgled modula "Telo => Glava"

Zajem poteka v okviru HTML tabele. Na levi strani gradimo telo pravila, na desni pa glavo pravila. Gradnjo začnemo s po enim praznim atomom na vsaki strani.

Grafični nosilec atoma (torej nosilec HTML kode, ki opisuje obrazec atoma), je HTML element **oddelek** ( ). Le-ta se uporablja za logično razdelitev HTML kode in ima svojo identifikacijsko oznako, tako da ga lahko kadarkoli naslovimo in spremenimo njegovo vsebino.

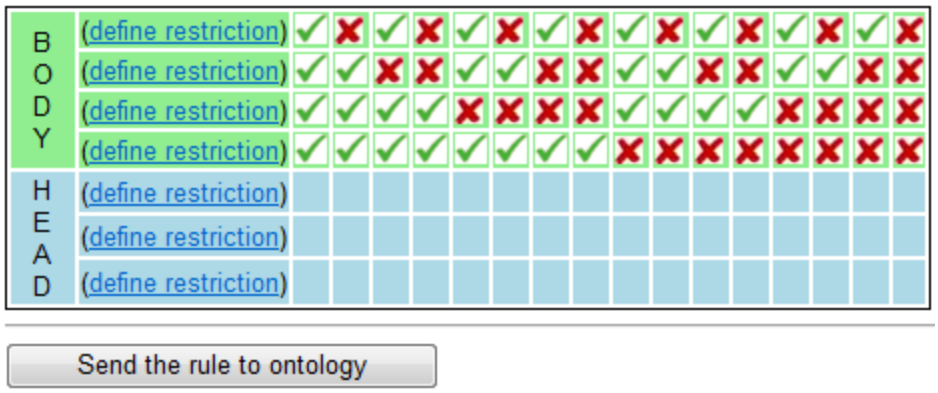
Ob desnem kliku na telo oz. glavo se prikaže kontekstni meni, preko katerega lahko dodajamo in brišemo atome. Vsakič, ko dodamo nov atom, se poleg logičnega atoma ustvari tudi nov oddelek. HTML obrazec atoma se zapiše v vsebino oddelka. Ustvari se tudi oddelek s tekstom operatorja (AND). Nato se najprej oddelek operatorja in nato oddelek z obrazcem atoma kot otroka dodata celici, ki vsebuje telo oz. glavo.

Brisanje atoma poteka v obratni smeri – najprej se izbrišeta oba oddelka, nato pa se izbriše še logični atom. Zadnjega atoma glave oz. telesa ni mogoče izbrisati, pač pa samo ponastaviti na osnovno stanje, torej prazen atom.

Ob vsakem vnosu argumenta se preveri, če je pravilo popolno (če noben atom ni prazen in če so v vseh atomih vnešeni vsi argumenti). Če je, potem se ustvari pravilo v abstraktni sintaksi in se prikaže na dnu strani. Tako uporabnik takoj ve, kdaj je pravilo pravilno vnešeno.

S klikom na gumb »Send the rule to ontology« se niz s pravilom v abstraktni sintaksi pošlje spletni storitvi, le-ta pa pravilo pretvori v SWRL kodo in jo doda v ontologijo.

5.2.4.2. Modul »odločitvena tabela«



Slika 27: Izgled modula "odločitvena tabela"

Tudi pri tem modulu poteka zajem v okviru HTML tabele, le da tukaj gradimo telo pravila v zgornjem delu, glavo pravila pa v spodnjem delu tabele. Gradnjo začnemo s po enim praznim atomom v telesu in glavi.

Pri tem načinu atomov ne dodajamo v eno samo celico tabele, pač pa ima vsak atom svojo vrstico. Zato je lahko grafični nosilec atoma kar druga celica vsake vrstice. Kadar izbrišemo atom, izbrišemo kar celotno vrstico. Tudi tu atome dodajamo preko kontekstnega menija, dosegljivega z desnim klikom na zgornji oz. spodnji del tabele. Ker način temelji na vrsticah, grafične predstavitve operatorjev ni.

Bistvo tega načina je določanje, katere posledice veljajo ob določeni permutaciji veljavnosti pogojev. Vsaka izmed permutacij je ponazorjena v svojem stolpcu tabele. Če pogoj v permutaciji velja, potem je v celici na križišču vrstice pogoja in stolpca permutacije slika z zeleno kljukico, če pa pogoj v permutaciji ni pomemben, je v tej celici slika z rdečim križcem.

Kako se torej dodajajo stolpci? Ob N pogojih imamo 2^N permutacij, zato začnemo z dvema stolpcema za začetni pogoj - v enem je kljukica, v drugem je križec. Ko dodamo nov pogoj in s tem novo vrstico, vsem obstoječim vrsticam, torej tudi vrsticam telesa dodamo stolpce permutacij, dokler jih nimajo 2^N. Obenem tudi celice, ki se ustvarijo v vsaki vrstici telesa, polnimo s slikami kljukic in križcev, in sicer po enostavni logiki – v i-ti vrstici se izmenjuje 2^(i-1) kljukic s 2^(i-1) križci. Če iz telesa izbrišemo atom (in s tem vrstico), enostavno izbrišemo zadnjo polovico stolpcev. Zaradi eksponentno rastočega števila stolpcev je število pogojev pri tem načinu omejeno na pet. Število posledic je neomejeno.

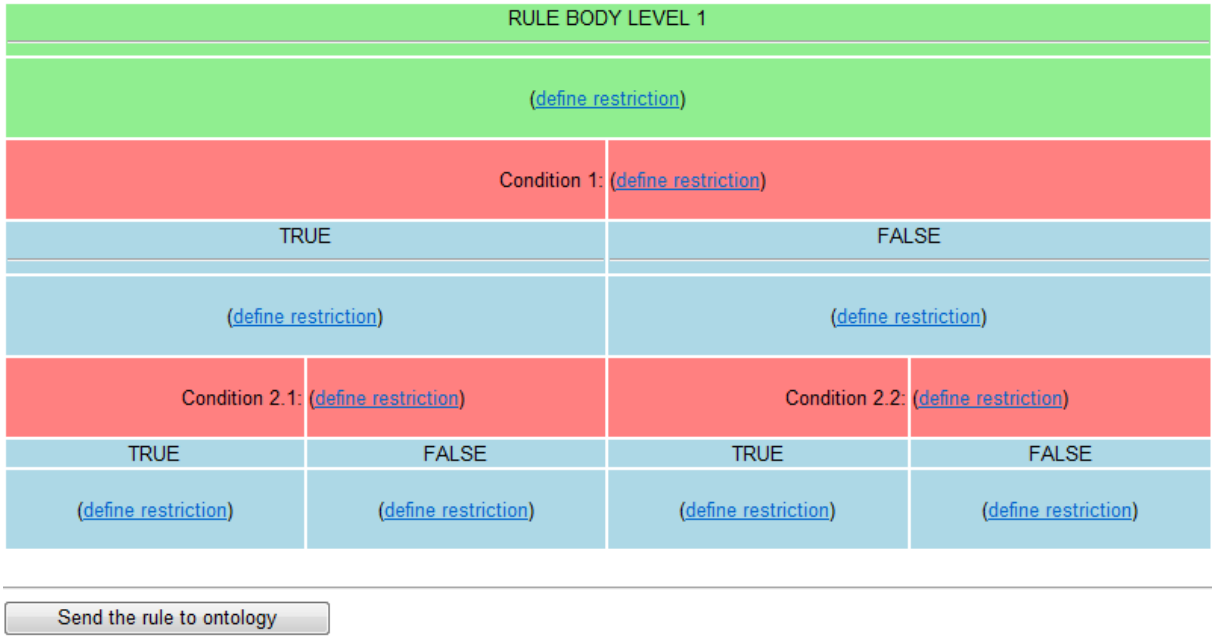
Ko imamo tako grafično prikazane permutacije pogojev, s klikom na celico v vrsti posledice določimo, da ta posledica velja ob permutaciji veljavnosti pogojev v stolpcu te celice. Grafično se to prikaže s sliko kljukice v kliknjeni celici. Ob ponovnem kliku na celico to določilo in sliko kljukice izbrišemo.

Tudi pri tem modulu se ob vsakem vnosu argumenta poskuša generirati pravilo v abstraktni sintaksi. Generiranje poteka tako, da se za vsak stolpec, v katerem je v telesu označena vsaj ena posledica, ustvari svoje pravilo v abstraktni sintaksi. Pogoj se v pravilo doda le, če je v tem stolpcu označen s kljukico. V glavo pravila se dodajo vse posledice, označene s kljukico. Če aplikacija ob tem postopku naleti na prazen ali neizpolnjen atom, je generacija pravila neuspešna. V nasprotnem primeru se vsa pravila zapišejo v en niz in se prikažejo na dnu strani.

S klikom na gumb »Send the rule to ontology« se niz s pravilom v abstraktni sintaksi pošlje spletni storitvi, le-ta pa vsako pravilo posebej pretvori v SWRL obliko in jo doda v ontologijo.

5.2.4.3. Modul »odločitveno drevo«

Najbolj naraven način implementacije odločitvenega drevesa bi bila implementacija z risanjem dreves na neki risalni površini, kamor bi dodajali telesa, kretnice in glave pravil ter jih s črtami povezovali med seboj. Ker pa bi to zahtevalo obsežen razvoj grafičnega okolja, ki sam po sebi ne sodi v okvir te diplomske naloge, sem odločitveno drevo implementiral na drugačen, preprostejši, vendar pa nič manj nazoren način.



Slika 28: Izgled modula "odločitveno drevo"

Zajem odločitvenega drevesa v tem modulu poteka preko HTML tabele in tudi tu atome dodajamo in brišemo preko kontekstnega menija.

Na vrhu tabele določimo telo pravila. Pod telesom pravila je kretnica, ki na začetku vsebuje prazen kretnični atom. Posamezna kretnica obsega natanko en atom –atomov ji ne moremo dodajati, lahko pa preko kontekstnega menija ponastavimo obstoječega.

Pod kretnico se tabela (in s tem drevo) razdeli na dva dela. Leva stran stopi v veljavo, kadar je kretnica resnična, desna stran pa takrat, kadar je kretnica neresnična. Vsaka stran se najprej začne z glavo pravila prvega nivoja, ki pa je obenem tudi telo drugega nivoja. Pod tem telesom drugega nivoja se nahaja kretnica drugega nivoja. Tu nastopi druga delitev glede na resničnost kretnice – vsaka stran vsebuje svojo glavo pravila na drugem nivoju.

Po vrsticah tabele imamo tako:

- telo pravila prvega nivoja,
- kretnico pravila prvega nivoja,
- dve glavi pravila prvega nivoja, ki sta obenem telesi pravil drugega nivoja,
- dve kretnici pravil drugega nivoja,
- štiri glave pravil drugega nivoja.

Pomanjkljivost te implementacije je nezmožnost zajema več kot dveh nivojev, saj imamo na tretjemu nivoju že osem vzporednih celic z glavami pravil tretjega nivoja, kar v širini ene strani težko elegantno prikažemo. Vendar pa sta za potrebe zajema pravil moje strategije dva nivoja povsem dovolj.

Oblika teles in glav je enaka kot pri načinu »Telo => Glava«. Vsako telo oz. glavo gradimo znotraj celice tabele. Grafični nosilci atomov so oddelki, ki jih dodajamo celicam. Med oddelke atomov seveda dodajamo oddelke s tekstom operaterja.

Tudi pri tem načinu se med zajemom pravil poskuša generirati celotno drevo pravil v abstraktni sintaksi za prikaz na dnu strani. Drevo pravil, zapisano v abstraktni sintaksi, je sestavljeno iz nizov pravil vsakega nivoja v abstraktni sintaksi, ki jih generiramo s sprehodom po drevesu. Sprehajamo se tako, da vedno najprej sledimo levim vejam, ko pridemo do konca, pa se vrnemo en nivo višje ter sledimo še desni veji.

Kretnične pogoje dodamo telesu pravila. Če gradimo pravilo s predpostavko, da je kretnica resnična, potem telesu dodamo kar nespremenjeni atom kretnice. Če pa predpostavljamo neresničnost kretnice (torej sledimo desni veji), potem moramo telesu pravila dodati negiran atom kretnice.

Negacija atoma kretnice je odvisna od tipa in vsebine atoma. Nekateri vgrajeni predikati so v paru z drugim vgrajenim predikatom in se medsebojno negirajo. Taki pari so zajeti v tabeli 8.

Če atom kretnice vsebuje enega izmed naštetih vgrajenih predikatov, potem se pri kreiranju negiranega predikata uporabi njegov par iz tabele. Če pa atom kretnice ne vsebuje nobenega izmed naštetih vgrajenih predikatov, to pomeni, da je negacija tega atoma bolj komplicirana. V tem primeru se atom ne negira v sami spletni aplikaciji, pač pa se v niz s pravilom v abstraktni sintaksi doda kar nespremenjen atom kretnice v abstraktni sintaksi, vendar se ga na začetku označi s klicajem ('!'). Vsi tako označeni atomi se negirajo šele v okviru spletne storitve, preden se pravilo pretvori v SWRL kodo. Z izognitvijo negiranju takih atomov v samem modulu se izognemo predolgim in morda nejasnim pravilom v abstraktni sintaksi.

vgrajeni predikat	negacija vgrajenega predikata
je enak (is equal)	je različen (is not equal)
je različen (is not equal)	je enak (is equal)
je manjši (is less than)	je večji ali enak (is greater than or equal)
je manjši ali enak (is less than or equal)	je večji (is greater than)
je večji (is greater than)	je manjši ali enak (is less than or equal)
je večji ali enak (is greater than or equal)	je manjši (is less than)
je isti (is same)	je različen (is different)
je različen (is different)	je isti (is same)

Tabela 8: Pari vgrajenih predikatov glede na negacijo

Ker sta rezultat vsake kretnice dve pravili, je skupno število generiranih pravil dvakrat večje od števila kretnic v drevesu. Drevo pravila z dvema nivojema je v abstraktni sintaksi torej oblike:

```
telo1 ∧ kretnica1 => glava1
glava1 ∧ kretnica2.1 => glava2.1
glava1 ∧ negiranaKretnica2.1 => glava2.2
telo1 ∧ negiranaKretnica1 => glava2
glava2 ∧ kretnica2.2 => glava2.3
glava2 ∧ negiranaKretnica2.2 => glava2.4
```

S klikom na gumb »Send the rule to ontology« se niz s pravili v abstraktni sintaksi pošlje spletni storitvi, le-ta pa vsako pravilo posebej pretvori v SWRL kodo in jo doda v ontologijo. Pred tem seveda še negira vse atome, ki so označeni s klicajem.

5.3. Implementacija strategije trgovanja

Poslovna pravila, na podlagi katerih poteka prepoznavanje vzorca svečnikov, sem zajel s pomočjo razvitih modulov.

5.3.1. Določanje barve in velikosti svečnika

Poslovna pravila

- **Barva svečnika**
  - Če je cena ob zaprtju višja ali enaka od cene ob odprtju, je svečnik bele barve.
  - Če je cena ob zaprtju nižja od cene ob odprtju, je svečnik bele barve.

- **Velikost telesa**
  - Če je svečnik bele barve, potem je velikost telesa enaka razliki med ceno ob zaprtju in ceno ob odprtju.
  - Če je svečnik črne barve, potem je velikost telesa enaka razliki med ceno ob odprtju in ceno ob zaprtju.

Zajem v modulu »Odločitveno drevo«

Slika 29: Zajem pravil za določitev barve in velikosti telesa

Generirano pravilo v abstraktni sintaksi

```
Trgovalni_dan(?day) & open(?day,?openPrice) & close(?day,?closePrice) &
greaterThanOrEqual(?closePrice,?openPrice)     =>     WhiteBody(?day)     &
subtract(?bodySize,?closePrice,?openPrice) & realBodySize(?day,?bodySize)

Trgovalni_dan(?day) & open(?day,?openPrice) & close(?day,?closePrice) &
lessThan(?closePrice,?openPrice)        =>        BlackBody(?day)        &
subtract(?bodySize,?openPrice,?closePrice) & realBodySize(?day,?bodySize)
```

Generirano pravilo v SWRL sintaksi

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#candleColor"/>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Trgovalni_dan" />
      <ruleml:var>day</ruleml:var>6
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="open">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>openPrice</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="close">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>closePrice</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:builtinAtom  swrlx:builtin="&swrlb;#greaterThanOrEqual">
      <ruleml:var>closePrice</ruleml:var>
      <ruleml:var>openPrice</ruleml:var>
    </swrlx:builtinAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="WhiteBody" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:builtinAtom  swrlx:builtin="&swrlb;#subtract">
      <ruleml:var>bodySize</ruleml:var>
      <ruleml:var>closePrice</ruleml:var>
      <ruleml:var>openPrice</ruleml:var>
    </swrlx:builtinAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="realBodySize">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>bodySize</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
  </ruleml:_head>
</ruleml:imp>


<ruleml:imp>
  <ruleml:_rlab ruleml:href="#candleColor"/>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Trgovalni_dan" />
```

```
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="open">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>openPrice</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="close">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>closePrice</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
    <swrlx:builtinAtom  swrlx:builtin="&swrlb;#lessThan">
      <ruleml:var>closePrice</ruleml:var>
      <ruleml:var>openPrice</ruleml:var>
    </swrlx:builtinAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="BlackBody" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:builtinAtom  swrlx:builtin="&swrlb;#subtract">
      <ruleml:var>bodySize</ruleml:var>
      <ruleml:var>openPrice</ruleml:var>
      <ruleml:var>closePrice</ruleml:var>
    </swrlx:builtinAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="realBodySize">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>bodySize</ruleml:var>
    </swrlx:datavaluedPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

5.3.2. Trgovalni dnevi z najnižjo oz. najvišjo doseženo ceno obdobja

5.3.2.1. Trgovalni dan z najnižjo doseženo ceno obdobja

Poslovno pravilo

- Najnižji doseženi ceni dveh predhodnih trgovalnih dni sta višji ali enaki najnižji doseženi ceni na ta trgovalni dan.

Zajem v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is Trgovalni_dan) AND (?day-1? is Trgovalni_dan) AND (?day-2? is Trgovalni_dan) AND (?day? has previous trading day ?day-1?) AND (?day-1? has previous trading day ?day-2?) AND (low price of ?day? is ?lowPrice?) AND (low price of ?day-1? is ?lowPrice-1?) AND (low price of ?day-2? is ?lowPrice-2?) AND (?lowPrice? is less than or equal to ?lowPrice-1?) AND (?lowPrice-1? is less than or equal to ?lowPrice-2?)	(?day? is MLT)

Slika 30: Zajem pravila za določitev trgovalnega dne z najnižjo doseženo ceno obdobja

Generirano pravilo v abstraktni sintaksi

```
Trgovalni_dan(?day) & Trgovalni_dan(?day-1) & Trgovalni_dan(?day-2) &
prejsnjiTrgovalniDan(?day-1,?day) & prejsnjiTrgovalniDan(?day-2,?day-1) &
low(?day,?lowPrice) & low(?day-1,?lowPrice-1) & low(?day-2,?lowPrice-2) &
lessThanOrEqual(?lowPrice,?lowPrice-1) & lessThanOrEqual(?lowPrice-
1,?lowPrice-2) => MLT(?day)
```

5.3.2.2. Trgovalni dan z najvišjo doseženo ceno obdobja

Poslovno pravilo

- Najvišji doseženi ceni dveh predhodnih trgovalnih dni sta nižji ali enaki od najvišje dosežene cene na ta trgovalni dan.

Zajem v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is Trgovalni_dan) AND (?day-1? is Trgovalni_dan) AND (?day-2? is Trgovalni_dan) AND (?day? has previous trading day ?day-1?) AND (?day-1? has previous trading day ?day-2?) AND (high price of ?day? is ?highPrice?) AND (high price of ?day-1? is ?highPrice-1?) AND (high price of ?day-2? is ?highPrice-2?) AND (?highPrice? is greater than or equal to ?highPrice-1?) AND (?highPrice-1? is greater than or equal to ?highPrice-2?)	(?day? is MHT)

Slika 31: Zajem pravila za določitev trgovalnega dne z najvišjo doseženo ceno obdobja

Generirano pravilo v abstraktni sintaksi

```
Trgovalni_dan(?day) & Trgovalni_dan(?day-1) & Trgovalni_dan(?day-2) &
prejsnjiTrgovalniDan(?day-1,?day) & prejsnjiTrgovalniDan(?day-2,?day-1) &
high(?day,?highPrice) & high(?day-1,?highPrice-1) & high(?day-2,?highPrice-
2) & greaterThanOrEqual(?highPrice,?highPrice-1) &
greaterThanOrEqual(?highPrice-1,?highPrice-2) => MLT(?day)
```

5.3.3. Marubozu

5.3.3.1. Beli Marubozu in proženje nakupnega signala

Poslovni pravili

- Najnižja dosežena cena je enaka ceni ob odprtju.
- Najvišja dosežena cena je enaka ceni ob zaprtju.

Zajem v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is WhiteBody) AND (low price of ?day? is ?lowPrice?) AND (open price of ?day? is ?openPrice?) AND (high price of ?day? is choose data) AND (close price of ?day? is choose data) AND (?lowPrice? is equal to ?openPrice?) AND (?highPrice? is equal to ?closePrice?)	(?day? is Marubozu) AND (?day? is Nakup) AND (trade reason of ?day? is #white Marubozu#)

Slika 32: Zajem pravila za prepoznavo belega Marubozu svečnika in proženje nakupnega signala

Generirano pravilo v abstraktni sintaksi

```
WhiteBody(?day)    &    low(?day,?lowPrice)    &    open(?day,?openPrice)    &
high(?day,?highPrice)          &          close(?day,?closePrice)          &
equal(?lowPrice,?openPrice)      &      equal(?highPrice,?closePrice)      =>
Marubozu(?day) & Nakup(?day) & razlog_za_akcijo(?day,#white Marubozu#)
```

5.3.3.2. Črni Marubozu in proženje prodajnega signala

Poslovni pravili

- Najnižja dosežena cena je enaka ceni ob zaprtju.
- Najvišja dosežena cena je enaka ceni ob odprtju.

Zajem v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is BlackBody) AND (low price of ?day? is ?lowPrice?) AND (open price of ?day? is ?openPrice?) AND (high price of ?day? is choose data) AND (close price of ?day? is choose data) AND (?lowPrice? is equal to ?closePrice?) AND (?highPrice? is equal to ?openPrice?)	(?day? is Marubozu) AND (?day? is Prodaja) AND (trade reason of ?day? is #black Marubozu#)

Slika 33: Zajem pravila za prepoznavo črnega Marubozu svečnika in proženje prodajnega signala

Generirano pravilo v abstraktni sintaksi

```
BlackBody(?day)    &    low(?day,?lowPrice)    &    open(?day,?openPrice)    &
high(?day,?highPrice)          &          close(?day,?closePrice)          &
equal(?lowPrice,?closePrice)      &      equal(?highPrice,?openPrice)      =>
Marubozu(?day) & Nakup(?day) & razlog_za_akcijo(?day,#black Marubozu#)
```

5.3.4. Doji

Poslovno pravilo

- Absolutna velikost telesa predstavlja največ deset odstotkov velikosti celotnega svečnika (torej razlike med najvišjo in najnižjo doseženo ceno).

Zajem v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is Trgovalni_dan) AND (real body size of ?day? is ?bodySize?) AND (low price of ?day? is ?lowPrice?) AND (high price of ?day? is ?highPrice?) AND (?candleSize? is ?highPrice? minus ?lowPrice?) AND (?bodySizeLimit? is ?candleSize? multiplied by #0.1#) AND (?bodySize? is less than or equal to ?bodySizeLimit?)	(?day? is Doji)

Slika 34: Zajem pravila za prepoznavo Doji svečnika

Generirano pravilo v abstraktni sintaksi

```
Trgovalni_dan(?day) & realBodySize(?day,?bodySize) & low(?day,?lowPrice) &
high(?day,?highPrice) & subtract(?candleSize,?highPrice,?lowPrice) &
multiply(?bodySizeLimit,?candleSize,#0.1#) &
lessThanOrEqual(?bodySize,?bodySizeLimit) => Doji(?day)
```

5.3.5. Kladivo

5.3.5.1. Prepoznavanje Kladiva

Poslovna pravila

- Najnižja dosežena cena svečnika je tudi najnižja dosežena cena zadnjih treh dni.
- Spodnja senca je vsaj dvakrat daljša od dolžine telesa.
- Zgornja senca je dolga največ 15 odstotkov dolžine telesa.

Realizacija v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is MLT) AND (real body size of ?day? is ?bodySize?) AND (low price of ?day? is ?lowPrice?) AND (high price of ?day? is ?highPrice?) AND (?minLowerShadow? is ?bodySize? multiplied by #2#) AND (?maxUpperShadow? is ?bodySize? multiplied by #0.15#)	(?day? is Hammer)

Slika 35: Zajem pravila za prepoznavo vzorca Kladivo

Generirano pravilo v abstraktni sintaksi

```
MLT(?day) & realBodySize(?day,?bodySize) & low(?day,?lowPrice) &
high(?day,?highPrice) & multiply(?minLowerShadow,?bodySize,#2#) &
multiply(?maxUpperShadow,?bodySize,#0.15#) => Hammer(?day)
```

5.3.5.2. Proženje nakupnega signala

Poslovni pravili

- Kladivo je bele barve.
- Naslednji trgovalni dan je potrditev pojava pozitivnega trenda v obliki belega svečnika.

Realizacija v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is Hammer) AND (?day? is WhiteBody) AND (?day? has next trading day ?day+1?) AND (?day+1? is WhiteBody)	(?day? is Nakup) AND (trade reason of ?day? is #bullish Hammer#)

Slika 36: Zajem pravila za proženje nakupnega signala ob prepoznanem vzorcu Kladivo

Generirano pravilo v abstraktni sintaksi

```
Hammer(?day) & WhiteBody(?day) & naslednjiTrgovalniDan(?day+1,?day) &
WhiteBody(?day+1) => Nakup(?day) & razlog_za_akcijo(?day,#bullish Hammer#)
```

Generirano pravilo v SWRL sintaksi

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#Hammer"/>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Hammer" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:classAtom>
      <owlx:Class owlx:name="WhiteBody" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasNextTradingDay">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>day+1</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:classAtom>
      <owlx:Class owlx:name="WhiteBody" />
      <ruleml:var>day+1</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Prodaja" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="razlog_za_akcijo">
      <ruleml:var>day</ruleml:var>
```

```
      <owlx:DataValue                    owlx:datatype="&xsd;string">Bullish
Hammer</owlx:DataValue>
    </swrlx:datavaluedPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

5.3.6. Viseči mož

5.3.6.1. Prepoznavanje Visečega moža

Poslovna pravila

- Najvišja dosežena cena svečnika je tudi najvišja dosežena cena zadnjih treh dni.
- Spodnja senca je vsaj dvakrat daljša od dolžine telesa.
- Zgornja senca je dolga največ 15 odstotkov dolžine telesa.

Realizacija v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is MHT) AND (real body size of ?day? is ?bodySize?) AND (low price of ?day? is ?lowPrice?) AND (high price of ?day? is ?highPrice?) AND (?minLowerShadow? is ?bodySize? multiplied by #2#) AND (?maxUpperShadow? is ?bodySize? multiplied by #0.15#)	(?day? is HangingMan)

Slika 37: Zajem pravila za prepoznavo vzorca Viseči mož

Generirano pravilo v abstraktni sintaksi

```
MHT(?day)    &    realBodySize(?day,?bodySize)    &    low(?day,?lowPrice)    &
high(?day,?highPrice)    &    multiply(?minLowerShadow,?bodySize,#2#)    &
multiply(?maxUpperShadow,?bodySize,#0.15#) => HangingMan(?day)
```

5.3.6.2. Prvo pravilo za proženje nakupnega signala

Poslovno pravilo

- Viseči mož je črne barve.

Realizacija v modulu »Telo => Glava«

Body of the rule (IF)	Head of the rule (THEN)
(?day? is HangingMan) AND (?day? is BlackBody)	(?day? is Prodaja) AND (trade reason of ?day? is #Black Hanging man#)

Slika 38: Zajem prvega pravila za proženje nakupnega signala ob prepoznanem vzorcu Viseči mož

Generirano pravilo v abstraktni sintaksi

```
HangingMan(?day) & BlackBody(?day) => Prodaja(?day) & razlog_za_akcijo(?day,#Black Hanging man#)
```

5.3.6.3. Drugo in tretje pravilo za proženje nakupnega signala

Poslovna pravila

- **Prvi par poslovnih pravil**
  - Viseči mož je bele barve.
  - Naslednji trgovalni dan je potrditev pojava negativnega trenda v obliki Doji svečnika.

- **Drugi par poslovnih pravil**
  - Viseči mož je bele barve.
  - Naslednji trgovalni dan je potrditev pojava negativnega trenda v obliki črnega svečnika.

Realizacija v modulu »Odločitvena tabela«

Slika 39: Zajem drugega in tretjega pravila za proženje nakupnega signala ob prepoznanem vzorcu Viseči mož

Generirano pravilo v abstraktni sintaksi

```
HangingMan(?day) & WhiteBody(?day) & naslednjiTrgovalniDan(?day+1,?day) &
BlackBody(?day+1) => Prodaja(?day) & razlog_za_akcijo(?day,#Hanging man and
Black Body#)
HangingMan(?day) & WhiteBody(?day) & naslednjiTrgovalniDan(?day+1,?day) &
Doji(?day+1) => Prodaja(?day) & razlog_za_akcijo(?day,#Hanging man and
Doji#)
```

Generirano pravilo v SWRL sintaksi

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#hangingMan1"/>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="HangingMan" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:classAtom>
      <owlx:Class owlx:name="WhiteBody" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasNextTradingDay">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>day+1</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:classAtom>
      <owlx:Class owlx:name="BlackBody" />
      <ruleml:var>day+1</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Prodaja" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="razlog_za_akcijo">
      <ruleml:var>day</ruleml:var>
      <owlx:DataValue  owlx:datatype="&xsd;string">Hanging  man  and  Black
Body</owlx:DataValue>
    </swrlx:datavaluedPropertyAtom>
  </ruleml:_head>
```

```
</ruleml:imp>

<ruleml:imp>
  <ruleml:_rlab ruleml:href="#hangingMan2"/>
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="HangingMan" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:classAtom>
      <owlx:Class owlx:name="WhiteBody" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasNextTradingDay">
      <ruleml:var>day</ruleml:var>
      <ruleml:var>day+1</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Doji" />
      <ruleml:var>day+1</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Prodaja" />
      <ruleml:var>day</ruleml:var>
    </swrlx:classAtom>
    <swrlx:datavaluedPropertyAtom  swrlx:property="razlog_za_akcijo">
      <ruleml:var>day</ruleml:var>
      <owlx:DataValue    owlx:datatype="&xsd;string">Hanging    man    and
Doji</owlx:DataValue>
    </swrlx:datavaluedPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

6. ZAKLJUČEK

Med procesom ustvarjanja diplomske naloge sem našel odgovore na vsa vprašanja, ki sem si jih zastavil na začetku. Končni rezultat je razvit in implementiran ter poslovnemu uporabniku prijazen sistem za zajem pravil na semantičnem spletu.

Osnovna prednost razvitega sistema v primerjavi z že obstoječimi sistemi za zajem pravil je v njegovi zasnovi, ki omogoča zajem pravil uporabnikom brez znanja abstraktne sintakse. Temelji na atomih pravil, ki so grafično predstavljeni z obrazci. Obrazci so sestavljeni tako, da ponudijo nastavke za vnos argumentov v obliki HTML sider in vmesnega teksta, kar daje atomu videz trditve v naravnem jeziku. To poslovnemu uporabniku pravila naredi precej bolj razumljiva in lažje berljiva.

Vnašanje argumentov poteka preko menija, ki ponuja kontekstno ustrezne izbire. S tem je onemogočen vnos argumentov napačnega tipa, vnos primerkov in spremenljivk, ki ne zadostujejo omejitvam domene oz. Dosega itd. Dodane so tudi omejitve, ki preprečujejo vnos nekaterih nesmiselnih vrednosti. Vsi ti ukrepi preprečujejo sintaktične in omejujejo logične napake pri zajemu pravil.

Razviti sistem predvideva tri načine zajema pravil. Prvi način je osnoven način zajema enega samega pravila oblike »Telo => Glava«. S tem načinom lahko zajamemo vsako pravilo, vendar pa je lahko počasen, kadar želimo zajeti veliko množico pravil.

Drugi način omogoča zajem pravil v obliki odločitvene tabele. Ta način je uporaben predvsem takrat, kadar želimo zajeti množico pravil, katerih telesa sestavlja majhno število permutirano veljavnih pogojev. Pri implementaciji tega načina sem naletel na dva problema:

- Ker število permutacij narašča eksponentno s številom pogojev v telesu, je v glavi telesa lahko največ pet pogojev, da lahko odločitveno tabelo še normalno prikažemo. Rešitev tega problema bi bila morda razdelitev telesa na dva dela. V prvem bi bilo največ pet pogojev, katerih veljavnosti permutiramo, v drugem pa ostali pogoji telesa, ki morajo vedno veljati.
- Druga težava je povezana z lastnostjo SWRL jezika. Ker je le-ta monoton (ne pozna negacije atomov), pri permutacijah ne moremo preverjati nepravilnosti pogojev, ampak jih lahko le ignoriramo. Ta težava precej okrni uporabnost tega načina zajema, vendar pa bi se ji lahko izognili le tako, da bi uporabili kak drug jezik za zapis pravil.

Tretji način podpira zajem pravil v obliki odločitvenih dreves. Z uporabo tega načina preprosto zajemamo pravila, kjer je glava odvisna od pravilnosti kretnice. Tudi tu se je pojavil problem monotonosti, ki je bil rešen z omejitvijo tipa atoma kretnice na zalogo vrednosti in na vgrajene predikate. Le-te namreč lahko negiramo z uporabo atomov točno nasprotnega pomena, ki morajo biti resnični, da je kretnica neresnična.

V tej diplomski nalogi sem pokazal, da je mogoče pravila zajemati na višjem nivoju, ki odpravi potrebo po znanju abstraktne sintakse. Nekaterim logičnim konstruktom se sicer ne moremo izogniti (npr. spremenljivkam), vendar pa poslovni uporabniki z njihovim razumevanjem ne bi smeli imeti velikih problemov.

7. LITERATURA

Članki v zbornikih konferenc, članki v revijah, knjige:

[1] T. Berners-Lee, J. Hendler, O. Lassila, »The Semantic Web,« *Scientific American*, maj 2001.

[2] J. Davies, R. Studer, P. Warren, *Semantic Web Technologies: Trends and Research in Ontology-based Systems*, Chichester: John Wiley & Sons, 2006.

[3] S. Decker, S. Melnik, F. van Harmelen, D. Fensel; M. Klein, J. Broekstra, M. Erdmann, I. Horrocks, »The Semantic Web: the roles of XML and RDF,« *IEEE Internet Computing*, št. 5, zv. 4, str. 63-73, 2000

[4] A. J. Gerber, A. Barnard, A. J. van der Merwe, »Towards a semantic web layered architecture,« v zborniku *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*, Innsbruck, Avstrija, 2007, str. 353-362.

[5] A. Gómez-Pérez, O. Corcho, "Ontology Specification Languages for the Semantic Web," *IEEE Intelligent Systems*, št. 1, zv. 17, str. 54-60, 2002.

[6] T. R. Gruber, »Ontolingua: A Mechanism to Support Portable Ontologies,« *Technical Report* KSL-91-66, Stanford University, Knowledge Systems Laboratory, 1993.

[7] M. Jelovčan, »*Valutni trg kot naložbena priložnost malega vlagatelja*«, diplomsko delo, Univerza v Ljubljani, Ekonomska fakulteta, jan. 2006.

[8] D. Lavbič, M. Bajec, M. Krisper, »Pravila na semantičnem spletu,« *Elektroteh. vestn.*, št. 5, str. 249-254, 2006.

[9] D. Lavbič, M. Krisper, »Rapid ontology development model based on business rules management approach for the use in business applications,« v zborniku *10th International Conference on Enterprise Information Systems*, Barcelona, Španija, maj 2008.

[10] B. Matthews, »Semantic Web Technologies,« *JISC Technology and Standards Watch Report*, CCLRC Rutherford Appleton Laboratory, 2005.

[11] M. V. Milovanović, »*Modeling rules on semantic web*«, magistrsko delo, Univerza v Beogradu, Fakulteta za organizacijske vede, Beograd, 2007.

[12] M. O'Connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, M. Musen, »Supporting Rule System Interoperability on the Semantic Web with SWRL,« v zborniku *Fourth International Semantic Web Conference (ISWC-2005)*, Galway, Irska, 2005.

[13] G. Perme, »*Tehnična analiza vrednostnih papirjev in njihovih trgov*«, diplomsko delo, Univerza v Ljubljani, Ekonomska fakulteta, jan. 2004.

[14] M. Silič, M. Colnar, M. Krisper, R. Rupnik, M. Bajec, I.Rozman, M. Heričko, T. Domanjko, M. B. Jurič, A. Živkovič, S. Beloglavec, M. Kožman, A. Novakovia, M. Stantič, S. Rubin, R. Tomažič, R. Jensterle, »*EMRIS - Enotna metodologija razvoja informacijskih sistemov*«. zv. 2, Vlada Republike Slovenije, Center za informatiko, 2000.

[15] N. Shadbolt, T. Berners-Lee, W. Hall, »The Semantic Web Revisited,« *IEEE Intelligent Systems*, št. 3, zv. 21, str. 96-101, 2006.

Ostali viri:

[16] (2008) Cndlestick Trading Forum. Dostopno na: http://www.candlestickforum.com/

[17] (2005) International Conference on Rules and Rule Markup Languages for the Semantic Web. Dostopno na: http://2005.ruleml.org/

[18] (2008) KAON2 – ontology Management for the Semantic Web. Dostopno na: http://kaon2.semanticweb.org/

[19] (2004) OWL Web Ontology Language Overview. Dostopno na: http://www.w3.org/2004/OWL/

[20] (2004) Proposal for a Semantic Web Rule Language (SWRL). Dostopno na: http://www.w3.org/Submission/SWRL/

[21] (2007) I. Savnik, prosojnice za predavanja pri predmetu Podatkovni modeli in jeziki, zv. 10. Dostopno na: http://www.pef.upr.si/~savnik/predmeti/PMJ/predavanja/10-OWL.pdf

[22] (2008) Stockcharts.com – Chart Analysis. Dostopno na: http://stockcharts.com/school/doku.php?id=chart_school:chart_analysis

[23] (2008) The Yahoo! user Interface Library. Dostopno na: http://developer.yahoo.com/yui/

[24] (2008) Wikipedia. Dostopno na: http://www.wikipedia.org/

IZJAVA

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Marjana Krisperja, ter soglašam, da je diplomsko delo v elektronski obliki dostopno v zbirki »Dela FRI«.

Ljubljana, oktober 2008 Matjaž Šega