UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA
V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA
Ústav informatiky

DOPYTOVANIE NAD RDF DÁTAMI S UŽÍVATEĽSKOU PREFERENCIOU

DIPLOMOVÁ PRÁCA

Apríl 2006 Bc. Veronika Vaneková

Vyhlásenie

Vyhlasujem, že som túto diplomovú prácu vypracovala samostatne, na základe vedomostí získaných štúdiom a s pomocou uvedenej literatúry.

_____

Veronika Vaneková

Poďakovanie

Chcela by som sa poďakovať vedúcemu diplomovej práce prof. RNDr. Petrovi Vojtášovi, DrSc. a konzultantovi Mgr. Petrovi Gurskému za rady a pripomienky počas písania práce. Tiež ďakujem všetkým svojím priateľom a rodine za technickú a morálnu podporu.

Abstrakt

V tejto diplomovej práci sa zaoberáme výrokovým jazykom RDF a jeho fuzzyfikáciou. Skúmame možnosti, ako priradiť RDF dátam ohodnotenie z intervalu [0,1] na základe užívateľských preferencií. Uvádzame podrobnejšie niektoré dôkazy z oblasti formálnej sémantiky RDF. Praktická časť obsahuje návrh a implementáciu programu, ktorý je založený na databázovom systéme Sesame a slúži na ohodnocovanie RDF dát. Užívateľské preferencie spracuje v podobe fuzzy množín.

Abstract

In this diploma thesis we deal with the assertional language RDF and with its fuzzyfication. We analyze the possibility of adding evaluation or rating to RDF data according to user preference. We introduce some proofs concerning RDF formal semantics in more detail. The software part contains the design and implementation of software based on RDF database system Sesame. The software can be used for rating RDF data. It processes the user preference in the form of fuzzy sets.

Obsah

1 Úvod a motivácia

Jeden z najnovších výskumov, ktoré prebiehajú v oblasti informatiky, sa týka návrhu a vývoja sémantického webu. Sémantický web je v istom zmysle rozšírením súčasného webu. Súčasné vyhľadávače dokážu spracovať informácie iba na základe ich syntaxe, teda spôsobu ich zápisu. Nevedia rozlíšiť význam (sémantiku) slov a dokumentov. To spôsobuje problémy, ak má hľadané slovo viac významov. Musíme starostlivo vyberať kľúčové slová, aby sme odfiltrovali nezaujímavé informácie.

Ak napríklad hľadáme na internete cez vyhľadávač slovo „eclipse“, získame astronomické stránky či informácie o zatmení slnka, ale takisto odkazy na vývojové prostredie Javy a reklamy na audio techniku či internetové pripojenie. Nechcené výsledky zbytočne znižujú prehľadnosť a nútia nás prezerať množstvo stránok. Ak sa ich chceme zbaviť, upravíme ďalšie hľadanie. Vytipujeme si slová, ktoré presnejšie vystihujú okruh nášho záujmu. Napríklad „eclipse java“ nás pravdepodobne zbaví astronomických stránok, ale „sun eclipse“ nás nezbaví všetkých odkazov na Javu, keďže Java je produktom firmy Sun. V tomto prípade musíme opäť meniť kľúčové slová na základe intuície a nájdených výsledkov, až kým nenájdeme, čo chceme. Niektoré prípady sú však zvlášť komplikované.

Práve tento problém má vyriešiť sémantický web. Základná myšlienka spočíva v tom, že k dátam pripojíme špecifikáciu významu vo vhodnom tvare. Pôjde o metadáta, teda „dáta o dátach“. Metadáta budú štrukturované a tým umožníme ich automatické spracovanie. Budú slúžiť skôr pre programy než pre ľudí, ktorí stránky čítajú, ale ak ich zobrazíme graficky, sú ľahko pochopiteľné. Programy teda budú môcť pracovať so sémantickým obsahom dát a nie iba s ich syntaktickou formou. Cieľom je vyvinúť vyhľadávacie programy alebo agenty, ktoré dokážu „rozumieť“ obsahu dokumentov a využiť to pri vyhľadávaní a spracovaní informácií.

Treba poznamenať, že „význam“ je veľmi široký pojem. To isté slovo môže mať viacero významov. Niektoré slová ich majú viac než desať, ak sa v prirodzenej reči vyskytujú vo frázach a v obrazných, prenesených označeniach. Zatiaľ nie je reálne, aby programy zvládli rozlíšiť všetky významové odtienky nejakého jazyka. „Význam“ tu berieme ako interpretáciu, prepojenie dát so skutočným svetom.

Na špecifikáciu významu dokumentov slúži formát RDF (Resource Description Framework). Podľa definície v (Klyne, Carroll 2004) je RDF štruktúra určená na reprezentáciu informácií a metadát na webe. V RDF vyjadrujeme logické tvrdenia a výroky pomocou presnej formálnej abecedy, je to teda výrokový jazyk. Je určený pre prístup a použitie cez internet a poskytuje základ pre pokročilejšie výrokové jazyky s podobným zameraním. Koncepcie a ciele RDF zdôrazňujú všeobecnosť a presnosť pri tvorení výrokov o ľubovoľnom predmete, skôr než prispôsobovanie sa nejakému konkrétnemu spôsobu spracovania. Vo formáte RDF zapisujeme dáta alebo metadáta, teda informácie o konkrétnych zdrojoch, organizáciách a dokumentoch. Tieto metadáta je potom možné spracovať pomocou aplikácií, vyhľadávačov či sieťových agentov. Od internetových dokumentov určených priamo pre užívateľov prechádzame k informáciám pre rôzne sieťové procesy. Pre tieto procesy poskytuje RDF akýsi spoločný jazyk.

RDF zavádza minimálne obmedzenia, čo sa týka jeho používania a rozširovania. Aplikácie, ktoré nepotrebujú komunikovať s inými, môžu prispôsobiť sémantiku RDF pre vlastné potreby a obmedziť syntax. Takto vzniká množstvo verzií použitia,

ale hlavnou výhodou RDF je práve všeobecnosť a možnosť prenosu či zdieľania dát. Preto je RDF základom pri budovaní sémantického webu.

Aplikácie sémantického webu môžu naraziť na nepresné alebo približné užívateľské požiadavky. Potom aj výsledky, ktoré dostaneme na základe takýchto požiadaviek, budú mať rôznu mieru nepresnosti. Podobne aj zdroje dát nemusia byť dôveryhodné a to sa odrazí na celkovej hodnote výsledkov. A tu sa dostávame ku viachodnotovej (fuzzy) logike. Fuzzy logika rozširuje Booleovskú logiku tak, aby sa dali spracovať pravdivostné hodnoty, ktoré sú medzi „true" a „false". Teda pracuje s približnými informáciami, čím sa približuje prirodzenému ľudskému spôsobu uvažovania. Aj z tohto dôvodu je zaujímavé skúmať fuzzy logiku vo vzťahu k sémantickému webu.

Hlavným cieľom tejto práce je preskúmať možnosti RDF pri spracovaní dát. Zaujíma nás, či je možné v RDF zapísať ohodnotené dáta, ako im pridať ohodnotenie a ako potom takéto dáta spracovať. Žiadna implementácia fuzzy RDF dosiaľ neexistuje, je však pravdepodobné, že návrh sémantického webu sa tejto otázke v budúcnosti nevyhne. Preto skúmame možnosť spojenia princípov fuzzy logiky s dátovou štruktúrou RDF. Používame RDF ako základnú štruktúru pre deduktívne logické programovanie.

Chceme dosiahnuť kombináciu, spojenie RDF a fuzzy logiky. V nasledujúcej kapitole najskôr popíšeme jazyk RDF, jeho abstraktnú a konkrétnu syntax. V kapitole 3 sa budeme podrobnejšie zaoberať sémantikou RDF, napríklad interpretáciami. Pojmy z oblasti sémantiky zadefinujeme formálnejšie než uvádza špecifikácia (Hayes 2004). Tiež odstránime niektoré nepresnosti špecifikácie a dotiahneme do konca slovne naznačené dôkazy. V kapitole 4 popíšeme spracovanie RDF dát, predovšetkým dopytovanie. Potom prejdeme na úplne odlišnú tému, fuzzy logiku v kapitole 5. Uvedieme niekoľko dôležitých definícií pre pochopenie tematiky a potom analyzujeme možnosti fuzzyfikácie RDF. Tu sa pokúsime spojiť dve rozličné témy, RDF z kapitol 2 až 4 a fuzzy logiku z kapitoly 5.

2 Základy RDF

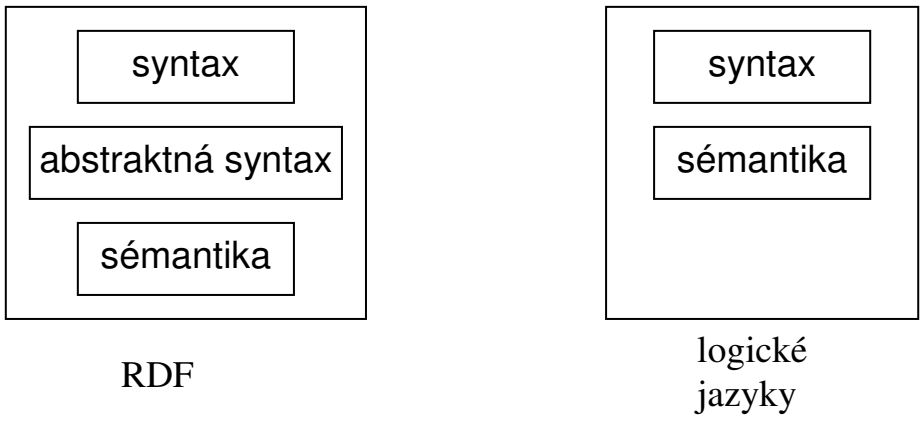
2.1 Ciele a rozdelenie

Návrh RDF je zameraný na splnenie týchto cieľov:

- jednoduchý dátový model – RDF dáta vždy tvoria graf bez ohľadu na ich konkrétnu syntax.
- formálna sémantika a odvodzovanie – presne definovaná sémantická implikácia umožňuje odvodzovať nové grafy pomocou pravidiel.
- XML syntax – je odporúčaným spôsobom zápisu RDF (Beckett 2004). Formát XML je vhodný na výmenu informácií medzi aplikáciami.
- podpora dátových typov XML Schemy - RDF používa hodnoty reprezentované podľa dátových typov XML Schemy a tak umožňuje výmenu informácií medzi RDF a inými XML aplikáciami.
- abeceda identifikátorov URI (Uniform Resource Identifier) – pomocou URI označujeme všetky popisované zdroje, entity. V RDF dátach sa vyskytuje aj iný druh hodnôt, literál.
- možnosť vytvárať tvrdenia o ľubovoľných zdrojoch – vo všeobecnosti nepredpokladáme úplné informácie o akomkoľvek zdroji. RDF dokonca

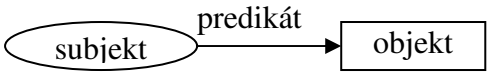
povolí každému vytvárať nezmyselné alebo protirečivé tvrdenia. Aplikácie môžu byť navrhnuté tak, aby nepripustili sporné informácie.

Každý logický jazyk sa skladá zo syntaxe a sémantiky. Syntax určuje, ako správne zostaviť výroky a formuly, zatiaľ čo sémantika sa zaoberá ich významom, interpretáciou. Syntax a sémantiku môžeme považovať za úrovne abstrakcie – syntax je konkrétna úroveň a sémantika abstraktná úroveň. RDF však k tejto štruktúre pridáva akúsi medzivrstvu, ktorá sa nazýva abstraktná syntax.



Obrázok 1. Úrovne abstrakcie RDF a iných logických jazykov

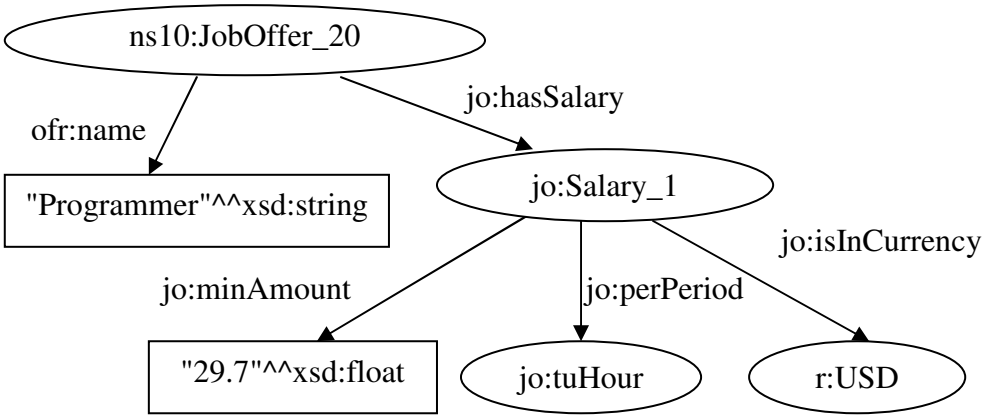
Abstraktná syntax určuje, že všetky dáta či metadáta v RDF sú vo forme trojíc (tvrdení), ktoré pozostávajú z troch častí: subjekt, predikát (nazývaný aj vlastnosť) a objekt. Množinu takýchto trojíc nazývame *RDF graf*. Subjekty a objekty sú uzly grafu, pričom môžu byť aj prázdne, predikáty tvoria orientované hrany od subjektu k objektu.



Obrázok 2. RDF trojica

Objekty z tvrdení môžu vystupovať ako subjekty v iných tvrdeniach, alebo môžu byť listami grafu. RDF trojicu môžeme spojiť s inými trojicami, ale pritom si zachová svoj význam bez ohľadu na celkovú zložitosť grafu. Tieto trojice sa dajú zakódovať rôznymi spôsobmi, čo je vecou *konkrétnej syntaxe*. Špecifikácia RDF odporúča zápis vo formáte XML. Ale je možné použiť aj ľubovoľnú inú konkrétnu syntax, ktorá kóduje RDF trojice a vyhovuje podmienkam abstraktnej syntaxe.

Sémantika popisuje význam RDF grafov. Subjekt v RDF grafe je vec, ktorú chceme popísať. Nazývame ho tiež *zdroj* a má svoj jednoznačný URI identifikátor. Medzi URI patria napríklad aj URL adresy, ale vo všeobecnosti URI nemusia odkazovať na skutočné internetové stránky. O identifikátoroch URI povieme viac kapitole 2.2.1. Predikát v RDF grafe je vlastnosť, prípadne vzťah či charakteristika priradená subjektu. Objekt je hodnota tejto vlastnosti pre daný subjekt.
Najskôr ukážeme príklad RDF grafu na ilustráciu problematiky.



Obrázok 3. Príklad RDF grafu

Na obrázku 3 sú URI referencie označené elipsami a literály zase obdĺžnikmi. Tento graf popisuje subjekt, pracovnú ponuku, s jej vlastnosťami name a hasSalary. Samotný objekt predikátu hasSalary je ďalším subjektom s vlastnosťami minAmount, perPeriod a isInCurrency. V grafe sú URI referencie v skrátenom tvare kvôli prehľadnosti. Prefix ofr: by sme nahradili URI referenciou http://www.fiit.stuba.sk/nazou/ontologies/offer-job# a podobne.

Tvrdenie (subjekt, predikát, objekt) znamená, že medzi subjektom a objektom trojice platí nejaký vzťah určený predikátom. Trojicu <subjekt, predikát, objekt> by sme v klasickom predikátovom počte zapísali ako predikát(subjekt, objekt). Podobný systém platí aj vo vetách prirodzeného jazyka. V Slovenčine základný typ vety, ktorý vypovedá o vlastnostiach nejakej veci, obsahuje tri vetné členy: podmet, prísudok a predmet. Dajú sa stotožniť so subjektom, predikátom a objektom RDF. Napríklad trojice z obrázku 3 prepíšeme do Slovenčiny takto:

ns10:JobOffer_20 má meno "Programmer"
ns10:JobOffer_20 má plat jo:Salary_1
jo:Salary_1 má minimálnu hodnotu "29.7"
jo:Salary_1 má menu r:USD
jo:Salary_1 má obdobie jo:tuHour

V Prologu by tieto trojice vyzerali nasledovne:

```
name(JobOffer_20, Programmer).
hasSalary(JobOffer_20, Salary_1).
minAmount(Salary_1, 29.7).
isInCurrency(Salary_1, USD).
perPeriod(Salary_1, tuHour).
```

2.2 Abstraktná syntax

V tejto kapitole prejdeme na definície pojmov ako URI, literál, trojica či RDF graf.

2.2.1 URI referencie

Definícia. *URI referencia* v RDF grafe je konečný reťazec znakov Unicode, ktorý:

- neobsahuje riadiace znaky (#x00 - #x1F, #x7F - #x9F)
- je zakódovaný v 8-bitovom formáte UTF-8. Oktetom, ktoré nezodpovedajú povoleným US-ASCII znakom, musí predchádzať únikový znak %.
- nepovolené oktety sú ošetrené URI únikovým mechanizmom, teda konvertujú sa na %HH, kde HH je dvojmiestne hexadecimálne číslo zodpovedajúce hodnote oktetu.

Definícia. Dve URI referencie sú *ekvivalentné*, ak sú ich reťazce Unicode rovné znak po znaku.

Definícia. Množinu všetkých URI referencií označujeme URIREF.

Adresy URL sú špeciálnym prípadom URI, pričom poznáme ich protokol (napríklad HTTP, FTP). URI sú však iba identifikátory a teda nie je nutné uvádzať protokol, hoci sa to bežne robí.

URI referencie delíme na absolútne a relatívne. V prípade konkrétnej syntaxe je možné definovať základnú URI a povoliť relatívne URI referencie ako skratky. URI môžu obsahovať aj identifikátory fragmentov (#), napríklad
http://www.w3.org/1999/02/22-rdf-syntax-ns#type

Prvú časť URI, http://www.w3.org/1999/02/22-rdf-syntax-ns#, potom môžeme zadefinovať napríklad ako prefix rdf: a pridať ho ku všetkým relatívnym URI referenciám. Nazveme ho *prefix menného priestoru*. Potom http://www.w3.org/1999/02/22-rdf-syntax-ns#type zapíšeme ako rdf:type. Detaily však závisia od konkrétnej syntaxe. Viac o URI referenciách je v článku (Berners-Lee, Fielding, Masinter 1998).

2.2.2 Dátové typy

Dátové typy označujú typ použitej hodnoty, napríklad celé čísla, čísla s plávajúcou desatinnou čiarkou a dátumy. Väčšinou sa používajú dátové typy z XML Schemy, ale je možné použiť aj iné. Niektoré typy z XML Schemy zase nie sú vhodné pre účely RDF.

Definícia. *Dátový typ* označíme $D = (L, H, lh, uriD)$, kde $uriD \subseteq URIREF$ je množina URI referencií, ktoré označujú tento dátový typ. Väčšinou býva jednoprvková. L je množina reťazcov Unicode, nazývame ju *lexikálny priestor*. Množina H je *priestor hodnôt* dátového typu. Zobrazenie $lh: L \rightarrow H$ nazývame *lexikálno-hodnotové zobrazenie*. Toto zobrazenie nie je prosté – každý prvok priestoru hodnôt môže mať viac prvkov lexikálneho priestoru (alebo aj žiadne), ktoré sa na tento prvok zobrazia. Nazývame ich lexikálne reprezentácie hodnoty.

Definícia. Množinu všetkých dátových typov nazveme DATATYPE.

Príklad. V lexikálno-hodnotovom zobrazení dátového typu xsd:boolean má každý prvok priestoru hodnôt (tu ich označujeme T a F) dve lexikálne reprezentácie:

Priestor hodnôt	{T, F}
Lexikálny priestor	{"0", "1", "true", "false"}
Lexikálno-hodnotové zobrazenie	lh("true")=T, lh("1")=T, lh("0")=F, lh("false")=F
URI dátového typu	{xsd:boolean}

V RDF nie sú zabudované vlastné definície čísel, dátumov či iných bežných hodnôt. RDF sa podriaďuje dátovým typom, ktoré sú zadefinované inde a odkazuje

na ne cez URI referencie. Najbežnejšie sa pre tento účel používajú dátové typy zadefinované v XML Scheme (Biron, Malhotra 2001). Neexistuje ani štandardný postup definovania nových dátových typov vnútri RDF, ale XML Schema umožňuje rozširovanie existujúcich dátových typov.

Preddefinovaný je iba dátový typ rdf:XMLLiteral, ktorý sa používa na vkladanie XML do RDF literálov. Tento dátový typ má priradenú URI referenciu http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral.

Definícia. Každý reťazec s, ktorý patrí do lexikálneho priestoru pre rdf:XMLLiteral, nazveme *správne napísaný literálový reťazec XML*. Príslušnú hodnotu nazveme *XML hodnotou* literálu. XML literál, ktorý má správne napísaný literálový reťazec, nazveme *správne napísaný XML literál*; iné XML literály budú *nesprávne napísané*.

2.2.3 Literály

Literály sú konštanty spolu s nepovinným označením svojho dátového typu alebo jazyka. Označujú hodnoty ako celé čísla, desatinné čísla, reťazce či dátumy. Literál môže vystupovať ako objekt RDF trojice, ale nie ako subjekt či predikát. Poznáme dva druhy literálov, jednoduché a typované. Jednoduché literály kódujú text v prirodzenom jazyku.

Definícia. *Jednoduchý literál* je reťazec znakov Unicode v úvodzovkách, za ktorým môže nasledovať tag jazyka. Teda jl = "$a_1...a_n$" alebo jl = "$a_1...a_n$"@xy, kde "$a_1...a_n$" nazývame reťazec literálu, a_i je znak v Unicode a @xy je tag jazyka. Znaky xy sú skratkou daného jazyka, napríklad en pre angličtinu, sk pre slovenčinu.

Definícia. *Typovaný literál* je reťazec v úvodzovkách, za ktorým povinne nasleduje URI referencia dátového typu oddelená znakmi ^^. Typovaný literál označuje prvok z priestoru hodnôt dátového typu, ktorý získame po aplikovaní lexikálno-hodnotového zobrazenia na reťazec literálu.
tl = "$a_1...a_n$"^^$b_1...b_n$, kde a_i,b_i sú znaky Unicode, "$a_1...a_n$" nazývame reťazec literálu. URI dátového typu je uri_typu(tl) = $b_1...b_n \in$ URIREF.
typ("$a_1...a_n$"^^$b_1...b_n$) = dt $\in$ DATATYPE je typ literálu, ak platí dt = (L, H, lh, T) a $b_1...b_n \in$ T, čiže dt je dátový typ prislúchajúci URI referencii $b_1...b_n$.

Definícia. Množinu jednoduchých literálov označíme JL, množinu typovaných literálov TL. Množina všetkých literálov potom je LIT=JL $\cup$ TL.

Príklad. Nasledujúce typované literály sú definované pre typ xsd:boolean. L = {"true", "false", "1", "0"}, H = {T, F}, uriD = xsd:boolean.

Typovaný Literál	Lexikálno-hodnotové zobrazenie	Hodnota
"true"^^ xsd:boolean	lh("true")=T	T
"1"^^xsd:boolean	lh("1")=T	T
"false"^^xsd:boolean	lh("false")=F	F
"0"^^xsd:boolean	lh("0")=F	F

Tabuľka 1. Literály typu xsd:boolean

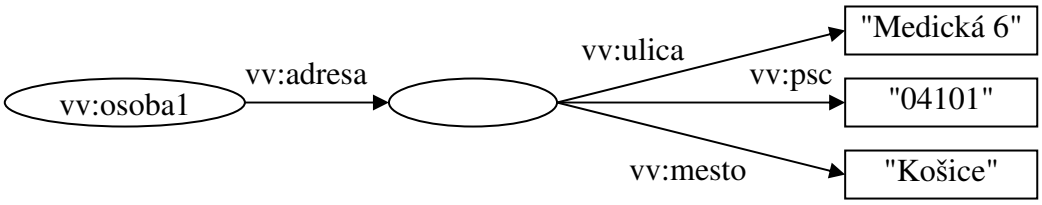
Definícia. Dva literály sa rovnajú vtedy a len vtedy, ak platia nasledujúce podmienky:

- Reťazce týchto dvoch literálov sú zhodné znak po znaku.
- Buď majú obidva literály tagy jazyka, alebo ich nemá ani jeden.

- Ak majú literály tagy jazyka, tak sa tagy rovnajú.
- Buď majú obidva literály URI dátového typu, alebo ich nemá ani jeden.
- Ak majú literály URI dátového typu, tak sa tieto URI zhodujú znak po znaku.

2.2.4 Prázdne uzly

Prázdne uzly sú uzly RDF grafu, ktoré nemajú svoju URI referenciu. Ak sa zdroj, ktorý popisujeme, vyskytuje na internete alebo má nejaké zmysluplné pomenovanie, je vhodné prideliť mu URI referenciu. Niektoré zdroje však nemajú nijaké meno a ostanú prázdne. Napríklad ak zadávame adresu, dôležité sú jej časti ulica, PSČ a mesto, ale samotnú adresu nemusíme nijako inak pomenovať.



Obrázok 4. Adresa ako prázdny uzol

RDF neudáva nijaké odporúčanie o vnútornej štruktúre označení prázdnych uzlov. Môžu byť označené ako _:node1, prípadne iným spôsobom. Majú označenie napriek tomu, že sú „prázdne“, aby sme vedeli určiť ich totožnosť.

Definícia. Množina prázdnych uzlov v RDF grafoch je nekonečná a spočítateľná množina reťazcov Unicode. Dva prázdne uzly sú zhodné, ak sa zhodujú ich reťazce znak po znaku.

Definícia. Množinu prázdnych uzlov označíme PRAZDNE. Množiny URIREF, LIT a PRAZDNE sú po dvojiciach disjunktné.

2.2.5 RDF grafy

Definícia. *RDF trojica* je usporiadaná trojica v tvare <subjekt, predikát, objekt>, kde:

- subjekt $\in$ URIREF $\cup$ PRAZDNE, teda subjekt je URI referencia alebo prázdny uzol,
- predikát $\in$ URIREF, teda predikát je URI referencia,
- objekt $\in$ URIREF $\cup$ LIT $\cup$ PRAZDNE, teda objekt je URI referencia, literál alebo prázdny uzol.

Definícia. *RDF graf* G je konečná množina RDF trojíc.

Definícia. Množinu uzlov RDF grafu G označujeme uzly(G) a definujeme ju uzly(G) = {subjekt : <subjekt, predikát, objekt> $\in$ G }$\cup${objekt : <subjekt, predikát, objekt> $\in$ G}.

Definícia. Množinu URI referencií, ktoré označujú hrany v grafe G, nazývame hrany(G) a definujeme ju hrany(G) ={predikát : <subjekt, predikát, objekt> $\in$ G}.

Je zjavné, že RDF grafy sa líšia svojou definíciou od klasických grafov z matematiky. Ale je možné transformovať ich na klasické grafy. Množina subjektov a objektov RDF trojíc udáva množinu uzlov grafu, orientované hrany sú dvojice <subjekt, objekt> získané z RDF trojíc <subjekt, predikát, objekt>. Ak máme daný RDF graf G = {<s_1,p_1,o_1>, ..., <s_n,p_n,o_n>: n$\in$N}, zostrojíme z neho klasický graf v tvare G'=(V,E), E$\subseteq$V$\times$V, kde V je množina vrcholov a E množina hrán:

V= uzly(G)
E= {<subjekt, objekt> : <subjekt, predikát, objekt> ∈ G}.

2.3 Konkrétna syntax

RDF má predpísanú iba abstraktnú syntax. Aplikácie môžu používať ľubovoľnú konkrétnu syntax, ktorá vyhovuje podmienkam abstraktnej. Tento fakt spôsobil, že vzniklo množstvo druhov syntaxe a stále vznikajú nové. Niektoré sú zamerané na jednoduchosť zápisu, iné na rýchlosť spracovania. V nasledujúcich podkapitolách sa zameriame na štyri najčastejšie používané druhy RDF syntaxe.

2.3.1 RDF-XML

Táto syntax je založená na štruktúre jazyka XML. Presná špecifikácia je v článku (Beckett 2004). Aby sme mohli zakódovať RDF graf do XML, musíme reprezentovať uzly a predikátové hrany pomocou XML elementov, atribútov, obsahu elementov a hodnôt atribútov. RDF graf tu chápeme ako súhrn ciest z koreňa do listov v tvare (uzol, predikátová hrana, uzol, predikátová hrana, …, uzol). Tieto cesty pokrývajú celý graf. Vo formáte RDF/XML sa tieto cesty zmenia na postupnosti elementov vnorených vnútri iných elementov. Prvý uzol sa stane vonkajším elementom, ďalšia predikátová hrana sa zmení na vnorený element a tak ďalej.

Syntax RDF/XML odporúča konzorcium W3C ako štandard. Jej výhodou je všeobecne známy formát a množstvo existujúcich nástrojov na spracovanie XML dokumentov. Takto zakódované RDF je možné vložiť do HTML dokumentov alebo ich pripojiť v samostatnom súbore ako metadáta, čiže informácie o obsahu daného HTML dokumentu.

RDF graf s jedinou trojicou <ns10:JobOffer_20, ofr:name, "Programmer"> by vo formáte XML vyzeral takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:ns10="http://www.fiit.stuba.sk/nazou/ontologies/offer-job-inst#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:ofr=" http://www.fiit.stuba.sk/nazou/ontologies/offer#">

<rdf:Description rdf:about="http://www.fiit.stuba.sk/nazou/ontologies/offer-job-
inst#JobOffer_20">
        <ofr:name>Programmer</ns2:name>
</rdf:Description>
</rdf:RDF>
```

Problémy spôsobuje fakt, že XML kóduje stromy, zatiaľ čo RDF všeobecné grafy. Preto je nutné mnohé RDF grafy rozdeliť na časti, ktoré sú stromami a tak ich zakódovať.

2.3.2 Notation 3 alebo N3

Notation 3 alebo **N3** (Berners-Lee 2001) je textová syntax, existuje však nezávisle od RDF a toto je len jedno z jej využití. Notation 3 povoľuje premenné a kvantifikátory, do RDF pridáva pravidlá a tvrdenia o tvrdeniach. Dokonca povoľuje aj také dáta, ktoré narušia sémantiku RDF grafu. Preto je nutné syntax kontrolovať vzhľadom na RDF. Existuje množstvo N3 parserov v rôznych programovacích

jazykoch. Napríklad v Jave je vyvinuté API na spracovanie RDF pod názvom Jena a obsahuje aj N3 parser. Dátové súbory sú vo formáte UTF-8 a majú príponu .n3.

Základná štruktúra RDF trojíc v N3 je:

```
subjekt predikát objekt .
```

URI referencie sú uzavreté v špicatých zátvorkách, jednotlivé zložky RDF trojíc sú oddelené medzerami. Každá trojica je ukončená medzerou a bodkou. Používajú sa tu prefixy menných priestorov a relatívne URI referencie tak, ako sme ich popísali v kapitole 2.2.1. Prefix zviažeme s URI referenciou s použitím kľúčového slova @prefix. Kľúčové slová môžu byť explicitne uvedené na začiatku dokumentu za príkazom @keywords. Ak ich neuvedieme, za kľúčové slovo sa považuje každé slovo začínajúce znakom zavináč @.

Prázdne uzly sa označujú podobne ako relatívne URI, čiže prefix:meno, ale prefix menného priestoru je v tomto prípade iba znak _. Prázdny uzol z obrázku 4 by sme označili napríklad _:adresa1.

Užitočným nástrojom je písanie viacnásobných objektov oddelených čiarkami pre ten istý subjekt a predikát, pričom subjekt a predikát stačí uviesť raz. Podobne sa dajú napísať rôzne predikáty pre ten istý subjekt a oddelia sa bodkočiarkami. Nasledujúci príklad obsahuje RDF graf z obrázku 3 zakódovaný vo formáte N3.

```
@prefix jo: <http://www.fiit.stuba.sk/nazou/ontologies/offer-job#> .
@prefix ns10: <http://www.fiit.stuba.sk/nazou/ontologies/offer-job-inst#> .
@prefix r: <http://www.fiit.stuba.sk/nazou/ontologies/region#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ofr: < http://www.fiit.stuba.sk/nazou/ontologies/offer#>

ns10:JobOffer_20 ofr:name "Programmer" .
ns10:JobOffer_20   jo:hasSalary   jo:Salary_1 .
jo:Salary_1   a   jo:Salary ;
jo:perPeriod   jo:tuHour ;
jo:isInCurrency   r:USD ;
jo:minAmount   "29.7"^^xsd:float .
```

Popíšeme ešte dve ďalšie špeciálne možnosti formátu N3: kvantifikátory a cesty v RDF grafoch.

Základná syntax pre *kvantifikátory* je:

```
{ @forSome <#g>. trojica } .
{ @forAll <#g>. trojica } .
```

kde trojica je ľubovoľná, len musí obsahovať premennú <#g>. Do vnorených zátvoriek môžeme napísať viac kvantifikátorov za sebou.

Cesty v RDF grafoch: ak máme RDF trojicu

```
s p o .
```

potom objekt o sa dá v N3 zjednodušene zapísať s!p (tento spôsob pripomína objektovo orientované programovanie). Subjekt s zase vyjadríme ako o^p. Tieto výrazy môžeme kombinovať a tak prechádzať aj zložitými grafmi.

2.3.3 N-Triples

N-Triples (Beckett, Barstow 2001) je riadkový textový formát v 7-bitovom US-ASCII. Je podmnožinou formátu Notation 3 a teda sa na jeho čítanie a spracovanie

dajú použiť nástroje N3. Dáta vo formáte N-Triples sa ukladajú do súborov s príponou .nt. Základná syntax RDF trojíc v N-Triples je:
subjekt predikát objekt .
Každý riadok teda obsahuje jednu trojicu <subjekt, predikát, objekt> a je ukončený medzerou a bodkou. Všetky identifikátory subjektov a predikátov sú v plnom formáte URI a zvyčajne sú to dlhé reťazce. Subjekt sa musí písať v každom riadku, aj keď je rovnaký pre viac trojíc. Z týchto vlastností vidieť, že formát N-Triples je určený na generovanie, čítanie a spracovanie pomocou programov a nie manuálne. Dá sa pomerne jednoducho spracovať, no je ťažko prehľadný pre užívateľov.
Ako príklad dát vo formáte N-Triples uvedieme prvé dva riadky transformované z predchádzajúceho príkladu. Oproti N3 tu chýbajú definície prefixov a trojice nie sú skrátené.

```
< http://www.fiit.stuba.sk/nazou/ontologies/offer-job-inst#JobOffer_20>
   < http://www.fiit.stuba.sk/nazou/ontologies/offer#name>
     "Programmer"^^http://www.w3.org/2001/XMLSchema#string .
<http://www.fiit.stuba.sk/nazou/ontologies/offer-job-inst#JobOffer_20>
   <http://www.fiit.stuba.sk/nazou/ontologies/offer-job#hasSalary>
     <http://www.fiit.stuba.sk/nazou/ontologies/offer-job#Salary_1> .
```

2.3.4 Turtle

Turtle (Terse RDF Triple Language) (Beckett 2006) je najčitateľnejší textový RDF formát. Je rozšírením N-Triples, pričom preberá niektoré užitočné vlastnosti Notation 3, ale v ničom nenarušuje dátový model RDF. Teda z nášho hľadiska obsahuje len výhody Notation 3 a odstraňuje nevýhody. Nasledujúce vlastnosti z Notation 3 nie sú zahrnuté do Turtle (zoznam nie je úplný):

- príkaz @keywords
- kvantifikátory @forAll, @forSome
- použitie logických spojok => (implikácia), = (ekvivalencia), <= (obrátená implikácia)

Turtle sa odlišuje od N-Triples v tom, že rozširuje kódovanie z ASCII na UTF-8. Ďalej preberá nasledujúce funkcie z Notation 3:

- príkaz @prefix
- použitie skratiek menných priestorov a relatívnych URI
- písanie viacnásobných objektov oddelených čiarkami pre ten istý subjekt a predikát
- písanie rôznych predikátov oddelených bodkočiarkami pre ten istý subjekt
- použitie zátvoriek ( ) na označenie prázdnych uzlov (v Turtle sa označujú aj štýlom _:xx)
- predikát rdf:type alebo v plnej verzii http://www.w3.org/1999/02/22-rdf-syntax-ns#type sa dá nahradiť predikátom a.

Dátové súbory v Turtle majú príponu .ttl. Jednoduché dáta ako v predchádzajúcom príklade budú syntakticky zhodné s N3. Tieto dva formáty by sa odlišovali napríklad v prípade RDF grafov s prázdnymi uzlami.

3 Sémantika RDF

Sémantika RDF je popísaná v dokumente (Hayes 2004), ktorý vypracovalo Konzorcium W3C. Zaoberá sa významom trojíc a grafov, prípadne odvodzovacími pravidlami. Význam RDF grafov síce vieme niekedy určiť intuitívne, ale nie vždy sa to dá, zoberme si napríklad trojicu

ex:a ex:b "c" .

O význame tejto trojice vieme povedať iba to, že ex:a má vlastnosť ex:b s hodnotou "c", ale nevieme určiť, k čomu sa vzťahujú URI referencie ex:a, ex:b a literál "c". Presne takéto určenie vzťahu medzi zložkami RDF grafov a skutočným svetom je úlohou sémantiky.

Pri popise sémantiky RDF predpokladáme, že jazyk logiky sa vzťahuje ku svetu a že tvrdenia, ktoré vytvoríme, hovoria niečo o svete. Pod tvrdeniami máme na mysli výroky, ktoré vyhlásime za pravdivé. Tvrdenia teda vyjadrujú minimálne podmienky, ktoré kladieme na svet. Vždy existuje viac možných svetov, ktoré vyhovujú našim tvrdeniam a nazývame ich *interpretácie*. Význam trojice vždy závisí od toho, ako interpretujeme URI a literály.

Zámerom tejto kapitoly je vytvoriť presný popis vlastností, ktoré musí interpretácia spĺňať, pričom zachováme čo najviac všeobecnosti. Formálna sémantika slúži na to, aby sme vedeli rozpoznať korektnosť odvodzovacích pravidiel, teda zistiť, kedy zachovávajú pravdivosť výrokov. Potom vieme z RDF grafov korektne odvodiť iné grafy.

V pôvodnej špecifikácii (Hayes 2004) sú definície aj vety uvedené väčšinou len slovne. Dôkazy sú v dodatkoch, obyčajne iba naznačené. Podrobnejšie sú vypracované dôkazy viet, ktoré sa týkajú sémantickej implikácie. Definície aj vety tu rozširujeme o formálne označenia jednotlivých množín a funkcií, ktoré využívame ďalej. Upravujeme niektoré nekonzistentné pojmy, napríklad pojem „meno“, ktorý je v špecifikácii definovaný dvomi rôznymi spôsobmi. Meníme aj označenia množín tak, aby vyhovovali slovenským názvom. Podľa týchto zmien upravujeme aj dôkazy a uvádzame ich podrobne.

RDF grafy, ktoré používame v príkladoch, zapisujeme vo formáte Turtle kvôli jednoduchosti. Používame štandardné prefixy rdf:, rdfs:, a xsd:, navyše prefixy uvedené v nasledujúcom odseku. Z príkladov vynechávame ich definície, lebo by boli všade rovnaké a znižovali by prehľadnosť:

```
@prefix rdf: < http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix r: <http://www.fiit.stuba.sk/nazou/ontologies/region#> .
@prefix ofr: <http://www.fiit.stuba.sk/nazou/ontologies/offer#> .
@prefix jo: <http://www.fiit.stuba.sk/nazou/ontologies/offer-job#> .
@prefix ns10: <http://www.fiit.stuba.sk/nazou/ontologies/offer-job-inst#> .
@prefix vv: <http://s.ics.upjs.sk/~vanekova/predicates/> .
```

3.1 Základné pojmy a definície

Definícia. *RDF graf* alebo jednoducho *graf* G je množina RDF trojíc. (Opakujeme túto definíciu z kapitoly 2.2.5.)

Definícia. *Podgraf* G' RDF grafu G je podmnožina trojíc v pôvodnom grafe. G'⊆G. *Vlastný podgraf* G' je vlastná podmnožina trojíc v grafe G, teda G'⊂G.

Definícia. *Konštantný (ground)* RDF graf G je taký graf, ktorý neobsahuje prázdne uzly. uzly(G) ∩ PRAZDNE = ∅.

Definícia. *Meno (uzlu, hrany)* je URI referencia, jednoduchý literál alebo typovaný literál. Sú to výrazy, ktorým musí interpretácia priradiť význam. Prázdne uzly nemajú meno.

$$meno(x) = \begin{cases} \{x\}, \; ak \; x \in URIREF \cup LIT \\ \varnothing, \; ak \; x \in PRAZDNE \end{cases}$$

Definícia. Množinu mien nazývame *abeceda*. Abeceda grafu je množina mien, ktoré sa vyskytujú ako subjekty, predikáty alebo objekty v ktorýchkoľvek trojiciach grafu. abeceda(G) = U{meno(s)∪meno(p)∪meno(o): <s,p,o>∈ G}.

URI referencie, ktoré sa vyskytujú iba v typovaných literáloch, nepatria do abecedy. Definíciu môžeme rozšíriť aj o URI dátových typov, ale pre účely tejto práce to nie je potrebné.

Definícia. Nech M: PRAZDNE → LIT∪PRAZDNE∪URIREF je inštančné zobrazenie z množiny prázdnych uzlov do množiny literálov, prázdnych uzlov a URI referencií. Potom *inštancia grafu G vzhľadom na M* je graf, v ktorom sme prázdne uzly x nahradili hodnotami M(x).

$$inst_M(G) = \{< s', p, o' >: \; < s, p, o > \in G\}$$

$$s' = \begin{cases} M(s), \; ak \; s \in PRAZDNE \\ s, \; inak \end{cases} \qquad o' = \begin{cases} M(o), \; ak \; o \in PRAZDNE \\ o, \; inak \end{cases}$$

To, že množina PRAZDNE patrí do oboru hodnôt M, má svoj zmysel. Pri vytváraní inštancie nemusíme nahradiť všetky prázdne uzly literálmi alebo URI.

Definícia. Graf F je *inštancia grafu G*, ak (∃M) $inst_M$(G)=F. Skrátene to budeme označovať ako F = inst(G).

Lema 1.

1) G=inst(G). *(Každý graf je inštanciou seba samého.)*

2) inst(inst(G))=inst(G). *(Inštancia inštancie* G *je tiež inštancia* G.*)*

3) *Ak* H *je inštancia* G*, tak každá trojica v grafe* H *je inštanciou niektorej trojice z* G. *Formálne ak* H=inst(G)*, potom* (∀<s,p,o>∈ H) (∃<s',p,o'>∈ G): {<s,p,o>} = inst({<s',p,o'>}).

Dôkaz.

1) Zobrazenie M nadefinujeme ako identitu. Potom $inst_M$(G) bude obsahovať presne tie isté trojice ako G, množina RDF trojíc je grafom a teda G=$inst_M$(G).
2) Využijeme výsledok G=inst(G) a opäť zostrojíme inštancie pre obidve strany.
3) Vidieť z definície inštancie.

□

Definícia. Inštancia grafu G *vzhľadom na abecedu* A je inštancia, v ktorej všetky mená, nahradené za prázdne uzly pôvodného grafu, patria do A. Obor hodnôt zobrazenia M je množina PRAZDNE∪A.

Definícia. *Vlastná* inštancia grafu je inštancia, v ktorej bol prázdny uzol nahradený menom, alebo boli dva prázdne uzly z grafu zobrazené do rovnakého uzla. H=VInst(G) ak $H=inst_M(G) \wedge ((\exists\ x \in uzly(G) \cap PRAZDNE)\ (M(x) \in URIREF \cup LIT) \vee (\exists\ x,y \in uzly(G) \cap PRAZDNE)\ (x \neq y \wedge M(x)=M(y)))$. *Nevlastná* inštancia je inštancia, ktorá nie je vlastná. Označíme ju H=NInst(G).

Lema 2. *Inštancia, ktorá vznikla zobrazením prázdneho uzla na iný prázdny uzol nevyskytujúci sa v pôvodnom grafe, je inštanciou pôvodného grafu a pôvodný graf je jej inštanciou.*

Dôkaz.
Pôvodný graf je G. Zobrazenie M bude identita okrem jedného prvku $x \in PRAZDNE$, pre ktorý M(x)=z, $z \in PRAZDNE$, $z \notin uzly(G)$. Potom $H=inst_M(G)$. Opačným smerom definujeme zobrazenie M' ako inverzné ku M. Okrem prípadu M'(z)=x je M' identické zobrazenie. Teda aj $G=inst_{M'}(H)$.

□

Tento proces môžeme opakovať, takže ľubovoľné bijektívne zobrazenie nad množinou prázdnych uzlov definuje graf, ktorý je inštanciou pôvodného grafu a pôvodný graf je jeho inštanciou. Takéto dva grafy, z ktorých každý je inštanciou toho druhého, ale nie vlastnou, a odlišujú sa iba označením svojich prázdnych uzlov, považujeme za ekvivalentné. Takéto grafy môžeme spracovať ako identické; umožní nám to ignorovať dôsledky vyplývajúce z „premenovania" identifikátorov uzlov. Je to tiež v súlade s intuitívnou predstavou, že prázdne uzly nemajú konkrétne meno.

Definícia. Grafy G a H sú *ekvivalentné*, ak G=NInst(H) a H=NInst(G).

Definícia. RDF graf G je *neredundantný* (*lean*), ak neexistuje jeho inštancia, ktorá je jeho vlastným podgrafom. $(\forall\ H=inst(G))\ \neg(H \subset G)$. *Redundantný* graf je taký, ktorý nie je neredundantný.

Redundantné grafy obsahujú svoje vlastné inštancie a vyjadrujú rovnaký obsah ako ich neredundantné podgrafy. Napríklad graf

_:y vv:adresa _:x .
vv:osoba1 vv:adresa _:x .

obsahuje jednu zbytočnú trojicu, pretože substitúcia M(_:y) = vv:osoba1 má za výsledok jednoprvkový vlastný podgraf

vv:osoba1 vv:adresa _:x .

Naproti tomu pre nasledujúci graf takéto zobrazenie neexistuje, teda je neredundantný.

vv:osoba1 vv:adresa _:x .
_:x vv:mesto _:x .

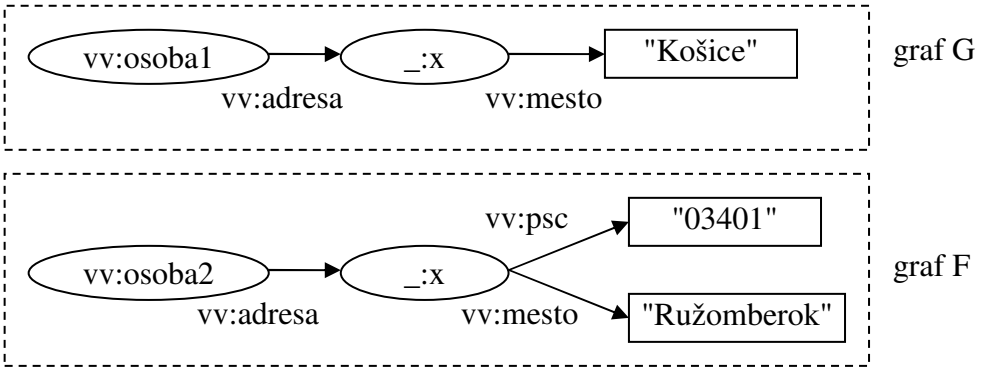
Definícia. *Zmiešanie* (*merge*) množiny RDF grafov S je definované nasledovne. Ak grafy nemajú nijaké spoločné prázdne uzly, tak zmiešanie grafov je ich zjednotenie. Ak obsahujú spoločné prázdne uzly, tak ich zmiešaním je zjednotenie množiny grafov, ktoré získame z pôvodných grafov nahradením za ekvivalentné grafy bez spoločných prázdnych uzlov. Formálne ak S je množina RDF grafov, potom

$$zmiešanie(s) = \begin{cases} \bigcup S,\ ak\ uzly(G) \cap uzly(F) \cap PRAZDNE = \varnothing\ pre\ F, G \in S,\ F \neq G \\ \bigcup \{inst_{M_G}(G), G \in S\},\ inak \end{cases}$$

$$M_G : PRAZDNE \cap uzly(G) \rightarrow A \subseteq PRAZDNE$$
$$(\forall H \in S, H \neq G)\ A \cap uzly(inst_{M_H}(H)) = \varnothing$$

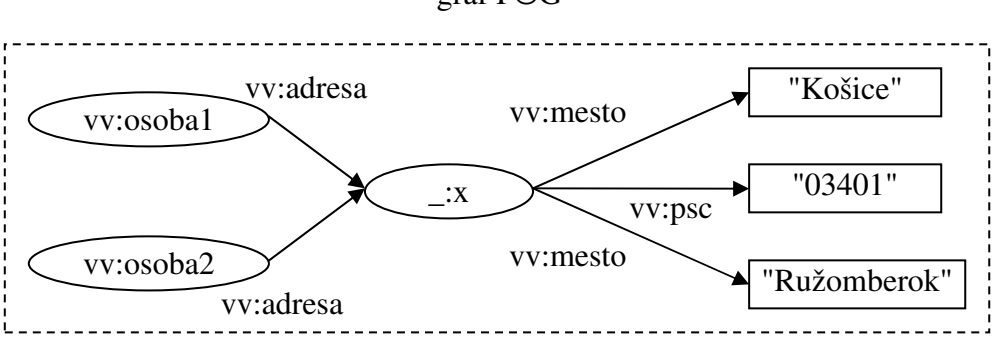
Ľahko sa vidí, že ľubovoľné dve zmiešania sú ekvivalentné. Preto budeme hovoriť všeobecne o zmiešaní množiny grafov.

Príklad. Zmiešanie ilustrujeme na príklade. Máme dva RDF grafy označené F a G:



Obrázok 5. RDF grafy F a G pred zmiešaním

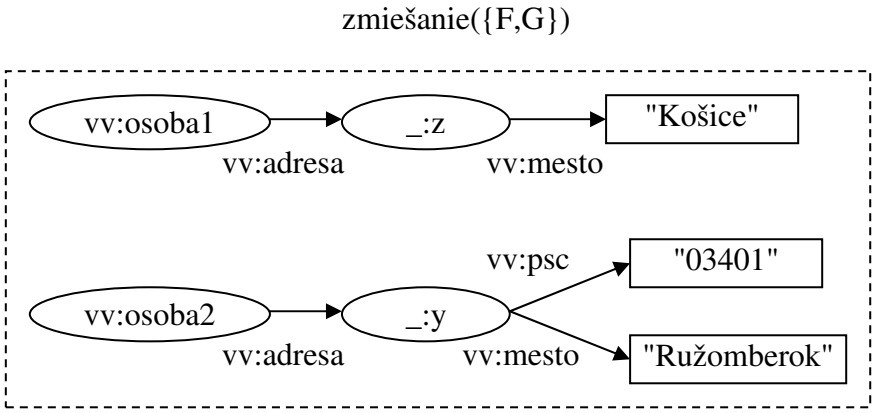
Ak zostrojíme zjednotenie, vo výslednom grafe budeme mať iba jeden prázdny uzol _:x. V obidvoch grafoch bol náhodou označený rovnako a tak sme tieto dva prázdne uzly považovali za zhodné.



Obrázok 6. Zjednotenie grafov F a G

Jediným problémom pri zjednotení grafov sú prázdne uzly. Ak máme v dvoch grafoch tú istú URI referenciu alebo literál, vieme, že označujú tú istú entitu a môžeme ich bez obáv stotožniť. Pri prázdnych uzloch to nie je možné. Ich označenia nemajú globálny význam. Slúžia len na to, aby sme v rámci jedného grafu vedeli určiť, či sú dva prázdne uzly zhodné alebo nie.

V prípade zmiešania grafov F a G z príkladu najskôr premenujeme prázdne uzly tak, aby boli množiny prázdnych uzlov v týchto grafoch disjunktné. Napríklad použijeme inštančné zobrazenia M_F(_:x)=_:y, M_G(_:x)=_:z. Okrem prípadu _:x budú M_F a M_G identické zobrazenia. Zostrojíme príslušné inštancie $inst_{M_G}$(G) a $inst_{M_F}$(F). Ich zjednotenie bude zmiešaním grafov F a G, ako ukazuje nasledujúci obrázok.



Obrázok 7. Zmiešanie grafov F a G

Obrázok 7 pripomína obrázok 5, kde boli pôvodné grafy F a G. Líši sa v označení prázdnych uzlov a tiež v tom, že teraz ide o jeden RDF graf, aj keď nie je súvislý. Túto skutočnosť sme zdôraznili čiarkovanou čiarou, ktorá na obrázkoch ohraničuje jednotlivé grafy.

Prázdne uzly tu chápeme ako premenné s existenčným kvantifikátorom pre ten RDF graf, v ktorom sa vyskytujú, ale s rozsahom kvantifikátora na celý graf. Takže žiadny prázdny uzol z jedného grafu sa nemôže dostať do oboru pôsobenia kvantifikátorov druhého grafu. Zmiešanie použijeme vtedy, keď grafy pochádzajú z rôznych zdrojov a nemôžeme zaručiť, že nejaký prázdny uzol v jednom grafe značí rovnakú entitu ako prázdny uzol v inom grafe.

Vo všeobecnosti nezískame zmiešanie množiny grafov jednoduchým spojením príslušných dokumentov vo formáte Turtle či N-Triples a zostrojením nového grafu z výsledného dokumentu. Ak by pôvodné dokumenty používali niektoré rovnaké identifikátory uzlov, zmiešaný dokument by popisoval graf, v ktorom sa niektoré uzly „nechtiac“ stotožnili. Pred zmiešaním dokumentov je nevyhnutné overiť, či dva alebo viaceré dokumenty nepoužívajú tie isté identifikátory uzlov a v každom takom dokumente ich rôzne premenovať.

3.2 Interpretácie

Intuitívna predstava sémantiky spočíva v tom, že každé tvrdenie kladie požiadavku na svet. Tvrdíme, že svet je usporiadaný tak, aby bol výrok pravdivý. Inými slovami, tvrdením kladieme obmedzenie na to, aký môže svet byť. Práve tento „možný svet“ je interpretácia výroku, v tomto prípade RDF grafu. Nepredpokladáme, že by tvrdenie obsahovalo dostatok informácií na to, aby sme mohli definovať jednoznačnú interpretáciu. Výrokmi nevieme obmedziť interpretácie na jediný možný svet, takže neexistuje nič také ako jednoznačná interpretácia RDF grafu. Vo všeobecnosti čím väčší je RDF graf, tým viac vyjadruje o svete, tým menšia je množina interpretácií, v ktorej je graf pravdivý a tým menej je možných svetov, v ktorých môže byť pravdivý.

Podľa interpretácie je možné zistiť pravdivostnú hodnotu (true alebo false) ktorejkoľvek konštantnej RDF trojice. Najskôr definujeme univerzum, teda množinu všetkých vecí, ktoré graf popisuje, potom množinu vlastností. Každú URI referenciu pomocou zobrazenia spojíme so zdrojom alebo s vlastnosťou, ktorú táto referencia popisuje. Ak URI popisuje vlastnosť, tak uvedieme, akú hodnotu má táto vlastnosť

pre každú vec v univerze. Ak popisuje dátový typ, tak uvedieme, že ten dátový typ definuje zobrazenie z množiny lexikálnych hodnôt do množiny hodnôt daného dátového typu. Tieto informácie stačia na rozhodnutie pravdivostnej hodnoty každej konštantnej trojice a teda aj každého konštantného RDF grafu. O grafoch, ktoré obsahujú aj prázdne uzly, napíšeme ďalej. Ak by sme vynechali ktorúkoľvek týchto informácií, mohlo by sa stať, že správne vytvorená trojica by nemala vymedzenú pravdivostnú hodnotu.

Interpretácie závisia od množiny mien (z grafu), ktorú nazývame abeceda interpretácie. Môžeme teda skôr hovoriť o interpretácii abecedy RDF než samotného RDF grafu. Niektoré interpretácie môžu priradiť symbolom konkrétnej abecedy špeciálne významy. Interpretácie, ktoré pre nejakú abecedu zdieľajú špeciálne významy, nazývame vzhľadom na túto abecedu „rdf-interpretácie", „rdfs-interpretácie" a pod. Interpretáciu bez podmienok na abecedu (vrátane samotnej abecedy RDF) nazveme *jednoduchá interpretácia*, alebo len interpretácia.

Význam literálov závisí od formy reťazca, ktorý obsahujú. Jednoduché literály bez URI referencie dátového typu vždy označujú seba samé. V prípade typovaných literálov však úplné určenie významu závisí od dostupnosti informácií o dátovom type, ktorý nie je definovaný vnútri RDF. Význam typovaného literálu je hodnota zobrazená z jeho reťazca znakov pomocou lexikálno-hodnotového zobrazenia (pozrite kapitolu 2.2.2 - Dátové typy). Osobitný význam majú literály typu rdf:XMLLiteral. Každá interpretácia definuje zobrazenie z množiny typovaných literálov na zdroje.

Definícia. *Entita* alebo *zdroj* je všeobecný termín pre prvky množiny U nazývanej *univerzum*. Zdroje sú všetky veci, o ktorých chceme tvoriť výroky.

Definícia. *Jednoduchá interpretácia* abecedy A je $(U, V, I^P, I^{REF}, I^{LIT}, LH)$ kde platia nasledujúce podmienky a označenia:

1. $U \neq \emptyset$ je neprázdna množina zdrojov, nazýva sa doména alebo univerzum interpretácie I.
2. $V \neq \emptyset$ nazývame množina vlastností interpretácie I.
3. Zobrazenie $I^P : V \rightarrow \mathcal{P}(U \times U)$ t.j. obor hodnôt je množina množín párov <x,y>, kde $x,y \in U$.
4. Zobrazenie I^{REF}: $URIREF \cap A \rightarrow (U \cup V)$ je interpretácia URI referencií z abecedy A.
5. Zobrazenie I^{LIT}: $TL \cap A \rightarrow U$ je interpretácia typovaných literálov z abecedy A.
6. Množinu $HL \subset U$ nazývame množina hodnôt literálov. Obsahuje všetky jednoduché literály z abecedy A.

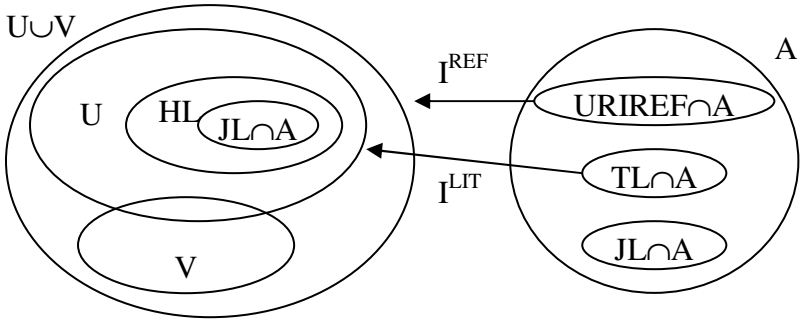
Označenia sú pomerne zložité a preto je tu mnemotechnická pomôcka:

U - Univerzum,
V - Vlastnosti,
HL - Hodnoty Literálov.
I^{LIT} - Interpretácia typovaných LITerálov,
I^{REF} - Interpretácia URI REFerencií
I^P - Interpretácia Predikátov

Zobrazenie $I^P(x)$ nazývame *rozšírenie* vlastnosti x. Je to množina dvojíc označujúcich subjekty a objekty, pre ktoré je táto vlastnosť (predikát) splnená, čiže je to binárne relačné rozšírenie. Každému predikátu vlastne priradí dvojice <subjekt, objekt>, ktoré s týmto predikátom tvoria pravdivú trojicu.

Zobrazenie I^{REF} interpretuje URI referencie, teda každej URI priradí zdroj alebo vlastnosť, ktorú URI referencia označuje. I^{LIT} zase interpretuje typované literály. Predpoklad, že HL je podmnožinou U znamená, že hodnoty literálov považujeme za reálne „existujúce“ hodnoty. Tým vlastne tvrdíme, že hodnoty literálov sú zdroje, ale nestotožňujeme ich s URI referenciami. HL môže obsahovať okrem jednoduchých literálov aj iné prvky. Existuje technický dôvod, prečo je oborom hodnôt zobrazenia I^{LIT} množina U, a nie HL. Keď interpretujeme dátové typy, typované literály môžu byť vnútorne nekonzistentné, ako napríklad literál "abc"^^xsd:integer a takéto nesprávne napísané literály sa musia zobraziť cez I^{LIT} na neliterálové hodnoty.

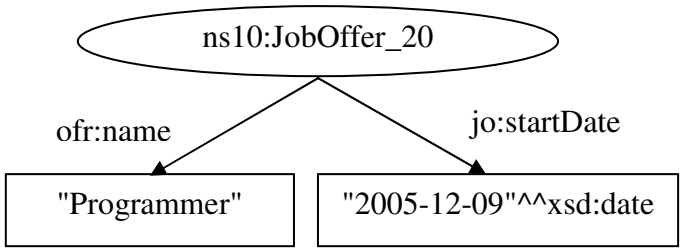
Nasledujúci obrázok znázorňuje vzťahy medzi uvedenými množinami. Všimnite si, že nijako neobmedzujeme vzťah medzi množinami U a V, hoci najčastejšie platí $V \subset U$.



Obrázok 8. Interpretácia abecedy

Vľavo na obrázku sú množiny, ktoré definujeme pri interpretácii, vpravo je abeceda označená A. Všimnite si, že súčasťou univerza U sú všetky hodnoty jednoduchých literálov. Typované literály aj URI sa zobrazia na ľubovoľné prvky U.

Príklad. Je daný nasledujúci RDF graf, ktorý označíme G_1:



Obrázok 9. Príklad RDF grafu

Abeceda tohto grafu bude A = abeceda(G_1) = {ns10:JobOffer_20, ofr:name, jo:startDate, "2005-12-09"^^xsd:date, "Programmer"}. Definujeme interpretáciu tejto abecedy:
(U_1, V_1, I^P_1, I^{REF}_1, I^{LIT}_1, HL_1):
U_1 = {pracovná ponuka, "Programmer", 9.decembra 2005, názov, dátum}
V_1 = {názov, dátum}
I^P_1(názov) = {<pracovná ponuka, "Programmer">}
I^P_1(dátum) = {<pracovná ponuka, 9.december 2005>}
I^{REF}_1(ns10:JobOffer_20) = pracovná ponuka, I^{REF}_1(jo:name) = názov, I^{REF}_1(jo:startDate) = dátum
I^{LIT}_1("2005-12-09"^^xsd:date)=9.december 2005

HL_1={"Programmer"}

V tejto interpretácii je V podmnožinou U, teda vlastnosti sú tiež zdroje. Táto interpretácia je v súlade s prirodzeným očakávaním o význame jednotlivých mien z abecedy, hlavne ak sme abecedu sami vytvorili. Interpretácia je však „možný svet" a možné je skutočne všeličo. Závisí to od nás, ako abecedu interpretujeme. Iná interpretácia môže byť:

$(U_2, V_2, I^P_2, I^{REF}_2, I^{LIT}_2, HL_2)$:

U_2 = {niečo, dátum, "Programmer"}

V_2 = {1}

$I^P_2(1) = \emptyset$

I^{REF}_2(ns10:JobOffer_20) = niečo, I^{REF}_2(jo:name) = 1, I^{REF}_2(jo:startDate) = 1

I^{LIT}_2("2005-12-09"^^xsd:date) = dátum

HL_2={"Programmer"}

Pravdivostná hodnota konštantného RDF grafu (bez prázdnych uzlov) závisí od interpretácie jeho abecedy. Induktívne zadefinujeme *význam*, teda sémantickú hodnotu každého RDF výrazu od literálov a URI cez trojice až po RDF grafy. Mená v grafe označujú entity z univerza U, význam trojíc je ich pravdivostná hodnota.

Definícia. Nech $(U, V, I^P, I^{REF}, I^{LIT}, HL)$ je jednoduchá interpretácia abecedy A. Potom interpretáciu (význam) I konštantného grafu G definujeme indukciou:

1. $I(x) = x$ pre každý jednoduchý literál $x \in HL$,
2. $I(x) = I^{LIT}(x)$, pre každý typovaný literál $x \in A \cap TL$,
3. $I(x) = I^{REF}(x)$, pre každú URI referenciu $x \in A \cap URIREF$,
4. nech <s,p,o> je konštantná trojica, potom I(<s,p,o>) = true, ak $s,p,o \in A$ a platí $\langle I(s),I(o)\rangle \in I^P(I(p))$, inak I(<s,p,o>) = false,
5. nech G je konštantný graf, potom I(G) = false ak I(<s,p,o>) = false pre nejakú trojicu $\langle s,p,o\rangle \in G$, inak I(G) =true.

Definícia. Interpretácia I *splňuje* graf G, ak I(G) = true. Hovoríme tiež, že graf G je pravdivý v interpretácii I.

Ak by abeceda RDF grafu obsahovala mená, ktoré nepatria do A, potom by interpretácia týmto menám nepriradila sémantickú hodnotu. Niektorej trojici v grafe (a teda aj celému grafu) by tieto podmienky priradili hodnotu false. Základný význam ľubovoľného grafu spočíva v tom, že všetky mená v grafe sa vzťahujú k nejakým prvkom univerza. Do predpokladov definície pridáme podmienku A = abeceda(G). Posledná podmienka hovorí aj o tom, že prázdny graf (prázdna množina trojíc) je triviálne pravdivý. Všimnite si, že význam jednoduchých literálov vždy patrí do HL (literály znamenajú samy seba) a tiež to, že významy subjektu a objektu každej pravdivej trojice musia patriť do U.

Príklad. Máme interpretáciu $(U_1, V_1, I^P_1, I^{REF}_1, I^{LIT}_1, HL_1)$ abecedy A z predchádzajúceho príkladu. Potom sú v nej splnené nasledujúce trojice:

ns10:JobOffer_20 ofr:name "Programmer" .
ns10:JobOffer_20 jo:startDate "2005-12-09"^^xsd:date .

I(<ns10:JobOffer_20, ofr:name, "Programmer">) = true
ak platí
$\langle I(\text{ns10:JobOffer\_20}), I(\text{"Programmer"})\rangle \in I^P_1(I(\text{ofr:name}))$
teda ak <pracovná ponuka, "Programmer"> $\in I^P_1$(názov), čo je splnené.

V interpretácii (U_2, V_2, I^P_2, I^{REF}_2, I^{LIT}_2, HL_2) by tieto trojice neboli pravdivé, pretože:
I(<ns10:JobOffer_20, ofr:name, "Programmer">) = true
ak platí
<I(ns10:JobOffer_20), I("Programmer")> ∈ I^P_2(I(ofr:name))
teda ak <niečo, "Programmer"> ∈ I^P_2(1), čo neplatí. V tomto prípade rozšírenie predikátu ofr:name je prázdna množina, a teda trojica nemôže byť splnená.
I(<ns10:JobOffer_20, jo:startDate, "2005-12-09"^^xsd:date>) = true
ak platí
<I(ns10:JobOffer_20), I("2005-12-09"^^xsd:date)> ∈ I^P_2(I(jo:startDate))
teda ak <niečo, dátum> ∈ I^P_2(1), čo neplatí. Predikáty ofr:name aj jo:startDate sme zobrazili na tú istú vlastnosť 1.

Špeciálny prípad tvoria prázdne uzly v grafoch. Naznačujú existenciu nejakej entity bez toho, aby uviedli jej meno. Interpretácia abecedy nedáva prázdnym uzlom žiadny význam, pretože abeceda môže obsahovať iba URI referencie a literály. Ak chceme brať do úvahy všetky grafy, nielen konštantné, pozmeníme definíciu interpretácie. Predpokladajme, že I je interpretácia, Z je zobrazenie z nejakej množiny prázdnych uzlov do univerza U tejto interpretácie. Definujme rozšírenú interpretáciu I+Z rovnako ako I okrem toho, že prázdne uzly interpretuje pomocou hodnôt zo Z.

Definícia. Množinu prázdnych uzlov grafu G označujeme prazdne(G) = uzly(G) ∩ PRAZDNE.

Definícia. Nech Z: PRAZDNE→U, potom interpretácia I+Z grafu G je rozšírená oproti I o nasledujúci prípad: ak x je prázdny uzol a je definované Z(x), potom [I+Z](x) = Z(x).

Ďalej predefinujeme bod 5) definície I: ak G je RDF graf, potom I(G) = true ak [I+Z](G) = true pre nejaké zobrazenie Z: prazdne(G)→U, inak I(G)= false.

Tieto pravidlá nemenia definíciu interpretácie abecedy. Tá pozostáva z rovnakých hodnôt U, V, I^P, I^{REF}, I^{LIT}, HL. Jednoducho sme zmenili pravidlá vytvárania významu pre danú interpretáciu, takže tá istá interpretácia ohodnotí konštantné grafy aj grafy s prázdnymi uzlami. Obor hodnôt zobrazenia Z je univerzum U, teda každému prázdnemu uzlu priradíme nejakú entitu. Z sa líši od vyššie definovaného inštančného zobrazenia M: PRAZDNE → PRAZDNE∪LIT∪URIREF. Pri vytváraní inštancie nahrádzame prázdne uzly v grafe, pri interpretácii im priradíme význam, ale nezmeníme samotný graf.

3.3 Sémantická implikácia

Definícia. Množina RDF grafov S *(jednoducho) sémanticky implikuje* graf E, ak v každej interpretácii, v ktorej je pravdivý každý graf z S, je pravdivý aj graf E. Označujeme S⊨E ak (∀I) ((∀G∈ S I(G)=true) → I(E)=true). Ekvivalentne hovoríme, že E je *sémantickým dôsledkom* S.

Definícia. Graf G (jednoducho) sémanticky implikuje graf E, označujeme G⊨E, ak (∀I) (I(G)=true → I(E)=true).

Lema 3. *Sémantická implikácia grafov má nasledujúce vlastnosti:*

1. Reflexívnosť: (∀G) G⊨G.
2. Tranzitívnosť: (∀F,G,H) ak F⊨G ∧ G⊨H, potom F⊨H.

Dôkaz. Vyplýva priamo z druhej definície a tranzitívnosti obyčajnej implikácie.

□

Sémantická implikácia je teda závislá od interpretácie. Zatiaľ používame len jednoduché interpretácie a teda aj jednoduchú sémantickú implikáciu. Ak A sémanticky implikuje B, tak v každej interpretácii, kde je pravdivé A, je pravdivé aj B. Teda graf A už obsahuje rovnaký „význam" ako graf B; mohli by sme povedať, že význam B je nejako obsiahnutý vo význame A. Ak sa grafy A a B navzájom sémanticky implikujú, ich význam je rovnaký, kladú rovnaké požiadavky na svet. Najzaujímavejšie je to v prípade, keď A a B sú syntakticky rôzne grafy.

Definícia. Proces, ktorý vytvorí graf E z iného grafu G (prípadne množiny grafov S), je *(jednoducho) korektný*, ak $G \models E$, prípadne $S \models E$, inak je *nekorektný*.

Ak je proces nekorektný, neznamená to, že jeho záverom bude nepravdivý graf, ani korektnosť procesu nezaručuje pravdivosť výsledného grafu. Pravdivosť výsledku totiž závisí od pravdivosti predpokladov (RDF grafov) v interpretácii. Ale korektnosť znamená zachovanie pravdivosti pri odvodzovaní: ak máme v nejakej interpretácii pravdivé predpoklady, korektný proces z nich nikdy neodvodí nepravdivý záver. Nasledujúce vety o rôznych vlastnostiach sémantickej implikácie platia iba pre jednoduchú sémantickú implikáciu, nie pre rdf-sémantickú implikáciu, ktorú uvedieme ďalej.

Veta o prázdnom grafe. *Každý graf sémanticky implikuje prázdnu množinu trojíc, ale táto množina sémanticky implikuje iba samu seba.* $(\forall G)\ G \models \emptyset$, $(\forall G \neq \emptyset)\ \emptyset \not\models G$.

Dôkaz. Prvá časť: prázdna množina trojíc $\emptyset$ je konštantný graf, lebo neobsahuje prázdne uzly. Z bodu 5) definície interpretácie konštantného grafu vieme, že $I(\emptyset)=true$ akk neexistuje trojica $\langle s,p,o\rangle \in \emptyset$, pre ktorú $I(\langle s,p,o\rangle)=false$. To platí, lebo $\emptyset$ neobsahuje nijaké trojice. Je pravdivá v ľubovoľnej interpretácii I, teda $(\forall I)\ I(\emptyset)=true$. Pre ľubovoľný graf G a ľubovoľnú interpretáciu I dostaneme $I(\emptyset)=true$. Z toho dostávame $(\forall G)\ G \models \emptyset$ podľa definície sémantickej implikácie.

Druhá časť: $(\forall G \neq \emptyset)\emptyset \not\models G$. Nech G je ľubovoľný neprázdny graf a trojica $\langle s,p,o\rangle \in G$. Potom interpretácia I, pre ktorú $I^P(I(p)) = \emptyset$ nesplňuje G, ale splňuje $\emptyset$. Našli sme interpretáciu, v ktorej je pravdivé $\emptyset$, ale nie G. Teda $\emptyset \not\models G$.

□

Veta o podgrafe. *Graf sémanticky implikuje všetky svoje podgrafy. Ak* $E \subseteq G$, *potom* $G \models E$.

Dôkaz. Triviálny, vychádza z definícií podgrafu a sémantického dôsledku. Ak pre nejakú interpretáciu I platí $I(G)=true$, potom $(\exists Z)\ (\forall \langle s,p,o\rangle \in G)\ ([I+Z](\langle s,p,o\rangle)=true)$, teda pre nejaké zobrazenie Z sú všetky jeho trojice sú pravdivé v $[I+Z]$. Potom aj $(\forall \langle s,p,o\rangle \in E \subseteq G)\ ([I+Z](\langle s,p,o\rangle)=true)$ a teda $I(E)=true$.

□

Veta o inštanciách. *Graf je sémantickým dôsledkom každej svojej inštancie. Ak* $E=inst(G)$, *potom* $E \models G$.

Dôkaz. Nech $I(E)=true$ a $E=inst_M(G)$. Potom pre nejaké zobrazenie Z na množine prázdnych uzlov z E platí, že $(\forall <s,p,o> \in E)$ $([I+Z](<s,p,o>)=true)$. Hľadáme zobrazenie Z', pre ktoré by platilo $[I+ Z'](G)=true$.

Pre každý prázdny uzol $b \in prazdne(G)$ definujeme $Z'(b)=[I+Z](M(b))$. Teda $M(b)$ je prázdny uzol alebo meno dosadené v inštancii za b. Potom $[I+Z'](G)=[I+Z](E)=true$, takže I splňuje G. Lenže I bolo ľubovoľné, a teda $E \models G$.

□

Veta o zmiešaní. *Pre množinu RDF grafov* S *platí* $S \models zmiešanie(S)$ *a* $(\forall G \in S)$ $zmiešanie(S) \models G$.

Dôkaz. Z definície sémantickej implikácie a zmiešania. Všetky prvky S sú pravdivé vtedy a len vtedy, ak sú pravdivé všetky trojice v zmiešaní S.
Ak platí $(\forall F,G \in S,\ F \neq G)$ $uzly(G) \cap uzly(F) \cap PRAZDNE = \emptyset$, teda ak grafy z S nemajú spoločné prázdne uzly, potom $zmiešanie(S)=US$ a teda z vety o podgrafe triviálne platí $(\forall G \in S)$ $zmiešanie(S) \models G$. Ak $(\forall G \in S)$ $I(G)=true$, potom aj $I(US)=true$ a je triviálne splnené $S \models zmiešanie(S)$.
Ak grafy z S obsahujú spoločné prázdne uzly, potom podľa definície $zmiešanie(S) = U \{inst_{M_G}(G), G \in S\}$, kde inštančné zobrazenie M_G je pre každý graf $G \in S$ rôzne a platí M_G: $prazdne(G) \rightarrow A$, $A \subseteq PRAZDNE$, $(\forall H \in S, H \neq G)$ $A \cap uzly(inst_{M_H}(H)) = \emptyset$.

Potom podľa vety o inštanciách $(\forall G \in S)$ $inst_{M_G}(G) \models G$. Podľa vety o podgrafe $(\forall G \in S)$ $zmiešanie(S) \models inst_{M_G}(G)$. Z tranzitívnosti dostávame $zmiešanie(S) \models G$.

Chceme ešte dokázať $S \models zmiešanie(S)$. Označme si $E_G = inst_{M_G}(G)$. Vieme, že $(\forall G \in S)$ $E_G = inst(G)$, ale aj $G = inst(E_G)$. V grafe G sme iba nahradili prázdne uzly, ekvivalencia grafov E_G a G vyplýva z Lemy 1. Podľa vety o inštanciách platí $(\forall G \in S)$ $G \models E_G$. Teda $(\forall G \in S)(\forall I)$ $(I(G)=true \rightarrow I(E_G)=true)$. Ak pre nejakú interpretáciu I platí, že $(\forall G \in S)$ $I(G)=true$, potom aj všetky $I(E_G)=true$ a aj $I(UE_G)=I(zmiešanie(S))=true$. Teda $(\forall I)$ $((\forall G \in S$ $I(G)=true) \rightarrow I(zmiešanie(S))=true)$. $S \models zmiešanie(S)$.

□

Z tejto vety vyplýva, že s množinou grafov môžeme zaobchádzať ako s jedným grafom, keď sa jedná o splňovanie a sémantickú implikáciu. V ďalšom texte aj výraz „graf“ môžeme nahradiť výrazom „množina grafov“. Znamená to tiež, že množiny grafov sú ekvivalentné svojmu zmiešaniu, t.j. jednému grafu, aspoň čo sa týka sémantickej implikácie.

Veta o interpolácii. $G \models E$ *akk* $(\exists P \subseteq G)$ $P=inst(E)$. *Teda* G *sémanicky implikuje* E *vtedy a len vtedy, ak existuje podgraf* G*, ktorý je inštanciou* E.

Dôkaz.

$\Leftarrow$ Vyplýva z vety o podgrafe a inštancii. Ak $F \subseteq G$, $F=inst(E)$, potom $G \models F$ a $F \models E$. Použijeme tranzitívnosť sémantickej implikácie.
$\Rightarrow$ Použijeme Herbrandovskú konštrukciu na zostrojenie interpretácie grafu. Pre daný neprázdny graf G je jeho *Herbrandovská interpretácia*, označená $Herb(G)$, definovaná pomocou tejto interpretácie abecedy:
$HL_{Herb(G)} = uzly(G) \cap JL$;

$U_{Herb(G)}$ = uzly(G);
$V_{Herb(G)}$ = hrany(G);
$I^P_{Herb(G)}$ = {<s,o>: <s,p,o>∈ G}

$I^{REF}_{Herb(G)}$ a $I^{LIT}_{Herb(G)}$ sú identické zobrazenia na príslušných častiach abecedy grafu G.

Je ľahko vidieť, že [Herb(G)+Z], kde Z je identické zobrazenie na množine prázdnych uzlov z G, splňuje každú trojicu z G, a teda Herb(G) splňuje G.

Teraz hľadáme podgraf G, ktorý je inštanciou E. Podľa predpokladov G ⊨ E. Ak G=inst(E), potom za podgraf vezmeme celý graf G. Ak G nie je inštanciou E, potom zostrojíme interpretáciu Herb(G). Vieme, že Herb(G) splňuje G, a teda Herb(G) splňuje aj E, t.j. pre nejaké zobrazenie Z z prázdnych uzlov do $U_{Herb(G)}$, [Herb(G)+Z] splňuje každú trojicu <s,p,o>∈ E. Teda G musí obsahovať trojicu <[Herb(G)+Z](s), p, [Herb(G)+Z](o)>. Táto trojica je inštanciou predošlej trojice pri zobrazení [Herb(G)+Z]. V tomto prípade môžeme [Herb(G)+Z] použiť ako inštančné zobrazenie, keďže množina $U_{Herb(G)}$=uzly(G) a $V_{Herb(G)}$=hrany(G). Keby graf G danú trojicu neobsahoval, nemohol by sémanticky implikovať E. Stačilo by nadefinovať interpretáciu I, v ktorej by množina $I^P(I(p))$ neobsahovala dvojicu <I(s),I(o)>. Množina všetkých takých trojíc <[Herb(G)+Z](s), p, [Herb(G)+Z](o)> je podgrafom G a ten je inštanciou E.

□

Herbrandovské interpretácie spracúvajú URI referencie a typované literály rovnakým spôsobom ako jednoduché literály, t.j. ich významy sú ich vlastné syntaktické tvary. To samozrejme nemusí byť zámerom RDF, ale potvrdzuje to, že každý RDF graf má splniteľnú jednoduchú interpretáciu.

Ak chceme zistiť, či množina RDF grafov sémanticky implikuje iný graf, stačí overiť, či existuje inštancia implikovaného grafu, ktorá je podmnožinou zmiešania pôvodnej množiny grafov. V skutočnosti nemusíme priamo zostrojiť zmiešanie. Vychádzame z implikovaného grafu E a prázdne uzly berieme ako premenné v procese hľadania zhodného podgrafu. Ku prázdnym uzlom z E viažeme „zodpovedajúce“ mená v grafoch z množiny S, t.j. v grafoch, ktoré môžu sémanticky implikovať E. Veta o interpolácii ukazuje, že tento proces je korektný. Taktiež je úplný, ak je takým algoritmus hľadania zhodného podgrafu. Potom je sémantická implikácia v RDF rozhodnuteľná, teda existuje konečný algoritmus, ktorý pre každú konečnú množinu S a graf E rozpozná, či S sémanticky implikuje E alebo nie.

Priamym dôsledkom vety o interpolácii je kritérium neplatnosti sémantickej implikácie:

Veta o anonymite. *Nech* E *je neredundantný graf, nech* E' = VInst(E) *je jeho vlastná inštancia. Potom* E⊭E'.

Dôkaz. Sporom. Nech E⊨E', potom podgraf E je inštanciou E' (z predchádzajúcej vety), ale E'=VInst(E) a teda daný podgraf E je aj vlastnou inštanciou E. Potom E nemôže byť neredundantný graf a dostávame spor s predpokladom. Teda E⊭E'.

□

Ak do RDF grafu pridáme nové trojice, v novom grafe budú prirodzene platiť nové sémantické implikácie. Nové trojice vlastne pridávajú grafu význam. Zaujíma nás však, či takéto rozšírenia z grafov žiadny význam neodstránia, teda či všetky sémantické implikácie, ktoré platili pred zmenou, platia aj po nej. Táto vlastnosť sa

nazýva monotónnosť a je pre nás dôležitá. Ak by neplatila, nebolo by korektné odvodzovať nové trojice a pridávať ich do grafu, aby sme neporušili sémantiku.

Veta o monotónnosti. *Nech* $S \subseteq S'$ *a nech* $S \models E$. *Potom aj* $S' \models E$.

Dôkaz. Vyplýva z vety o podgrafe a zmiešaní. $S \subseteq S'$, teda $S' \models S$. Z predpokladov platí $S \models E$, z tranzitívnosti dostaneme $S' \models E$.

□

Vlastnosť, ktorá zaručuje, že z konečnej množiny predpokladov sa vždy dá odvodiť konečný dôsledok, sa nazýva *kompaktnosť*. Kompaktnosť je pre jednoduchú sémantickú implikáciu triviálna. Pre zložitejšie sémantické rozšírenia však už triviálna nie je.

Veta o kompaktnosti. *Ak* $S \models E$*, pričom* E *je konečný graf, potom existuje* S'*, konečná podmnožina* S*, taká, že* $S' \models E$.

Dôkaz. Podľa vety o interpolácii je S', podgraf S, tiež inštanciou E. Teda S' je konečný a sémanticky implikuje E.

□

3.4 Rdf-interpretácia

Jednoduché interpretácie a jednoduchá sémantická implikácia zachytávajú sémantiku RDF grafov bez toho, aby venovali pozornosť konkrétnym významom mien v grafe. Aby sme získali úplný význam grafu, ktorý používa určitú abecedu, obvykle musíme pridať ďalšie sémantické podmienky, ktoré priradia URI referenciám a typovaným literálom v grafe význam. V tejto kapitole popíšeme interpretácie, ktoré musia splniť sémantické podmienky pre abecedu RDF. Potom použijeme aj silnejšiu verziu sémantickej implikácie, ktorú nazveme *rdf-sémantická implikácia*.

Definícia. *Abeceda RDF,* alebo rdfA, je množina URI referencií v mennom priestore rdf:. rdfA = {rdf:type, rdf:Property, rdf:XMLLiteral, rdf:nil, rdf:List, rdf:Statement, rdf:subject, rdf:predicate, rdf:object, rdf:first, rdf:rest, rdf:Seq, rdf:Bag, rdf:Alt, rdf:value, rdf:_n, $n \in N$}

Definícia. *Axiomatické trojice RDF* sú nasledujúce trojice:

rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:nil rdf:type rdf:List .
rdf:_n rdf:type rdf:Property . kde $n \in N$.

Definícia. Nech G je RDF graf, abeceda(G)=A, nech $(U, V, I^P, I^{REF}, I^{LIT}, HL)$ je jednoduchá interpretácia abecedy rdfV∪A. Potom *rdf-interpretácia* grafu G

(označujeme ju I) je interpretácia, ktorá splňuje všetky axiomatické trojice RDF a vyhovuje nasledujúcim podmienkam:

1. $x \in V$ akk $\langle x, I(rdf:Property)\rangle \in I^P(I(rdf:type))$
2. Ak "$a_1...a_n$"^^rdf:XMLLiteral $\in A$, $a_1...a_n$ je správne napísaný literálový reťazec XML, potom
   I^{LIT}("$a_1...a_n$"^^rdf:XMLLiteral) je XML hodnota pre $a_1...a_n$;
   I^{LIT}("$a_1...a_n$"^^rdf:XMLLiteral) $\in$ HL;
   $\langle I^{LIT}$("$a_1...a_n$"^^rdf:XMLLiteral), I(rdf:XMLLiteral)$\rangle \in I^P(I(rdf:type))$
3. Ak "$a_1...a_n$"^^rdf:XMLLiteral $\in A$, $a_1...a_n$ je nesprávne napísaný literálový reťazec XML, potom
   I^{LIT}("$a_1...a_n$"^^rdf:XMLLiteral) $\notin$ HL;
   $\langle I^{LIT}$("$a_1...a_n$"^^rdf:XMLLiteral), I(rdf:XMLLiteral)$\rangle \notin I^P(I(rdf:type))$.

Prvá podmienka definuje množinu vlastností V ako množinu prvkov univerza, ktoré majú vlastnosť I(rdf:type) s hodnotu I(rdf:Property). Všimnite si, že táto podmienka vyžaduje, aby V bolo podmnožinou U, čiže aby vlastnosti patrili do univerza. Tretia podmienka vyžaduje, aby sa nesprávne napísané XML literály nezobrazili na žiadne literálové hodnoty. Tak štandardne ošetríme výskyt nesprávne napísaných literálov. Každá rdf-interpretácia je zároveň jednoduchou interpretáciou.

Definícia. S *rdf-sémanticky implikuje* E, označujeme $S \models_{RDF} E$, ak každá rdf-interpretácia, ktorá splňuje každý graf z S, splňuje aj E.

Táto definícia je podobná definícii jednoduchej sémantickej implikácie, ale týka sa len rdf-interpretácií, nie všetkých jednoduchých interpretácií. Ľahko sa vidí, že vety z predošlej kapitoly neplatia pre rdf-sémantickú implikáciu. Napríklad trojica rdf:type rdf:type rdf:Property . je axiómou, je pravdivá v každej rdf-interpretácii, a teda je aj rdf-sémantickým dôsledkom prázdneho grafu. To je kontrapríklad ku vete o prázdnom grafe.

3.5 Odvodzovacie pravidlá

Teraz uvedieme niekoľko odvodzovacích pravidiel, pomocou ktorých vieme z existujúcich trojíc odvodiť nové trojice. Pravidlá sú vo forme tabuliek, prvý stĺpec obsahuje meno pravidla, druhý predpoklady a tretí stĺpec trojice, ktoré pomocou tohto pravidla odvodíme. Implementácie môžu aplikovať pravidlá na pôvodné grafy a hľadať vyhovujúce dôsledky, alebo spracovať výsledný graf ako navrhovaný sémantický dôsledok a aplikovať pravidlá opačným smerom pri hľadaní vyhovujúcich predpokladov.

Pravidlá sú korektné podľa definície korektnosti z kapitoly 3.3: graf rdf-sémanticky implikuje každý väčší graf získaný aplikovaním týchto pravidiel na seba. Aplikovaním takýchto pravidiel získame zjednotenie grafu s jeho dôsledkom (namiesto zmiešania), a teda sa zachovajú všetky prázdne uzly, ktoré predtým v grafe boli.

Trojice uvádzame vo formáte Turtle, pričom používame nasledujúce označenia:

- $p \in$ URIREF je ľubovoľný predikát trojice.
- $s \in$ URIREF$\cup$PRAZDNE je ľubovoľný subjekt trojice.
- $o \in$ URIREF$\cup$PRAZDNE$\cup$LIT je ľubovoľný objekt trojice.
- $lit \in$ LIT je ľubovoľný literál.
- _:$n \in$ PRAZDNE je identifikátor prázdneho uzla, n je prirodzené číslo.

3.5.1 Pravidlá jednoduchej sémantickej implikácie

Vetu o interpolácii z kapitoly 3.3 môžeme aplikovať v podobe nasledujúcich pravidiel sémantickej implikácie. Pravidlá vytvárajú *generalizácie*, t.j. grafy, ktorých inštanciou je pôvodný graf. Najskôr potrebujeme zvoliť generalizačné zobrazenie gen: URIREF∪PRAZDNE∪LIT → PRAZDNE. Oproti vytváraniu inštancií je to opačný postup.

Meno pravidla	Predpoklady	Odvodené trojice
gen1	s p o .	s p _:n . kde _:n = gen(o)
gen2	s p o .	_:n p o . kde _:n = gen(s)

Tabuľka 2. Pravidlá generalizácie

Generalizačné zobrazenie nám zaručí, že z rôznych URI referencií alebo literálov nevytvoríme rovnaký prázdny uzol. Potom vieme, že pôvodný graf je vlastnou inštanciou výsledného grafu, ktorý sme získali pridaním trojice s prázdnym uzlom. Napríklad graf

vv:osoba1 vv:adresa vv:adresa1 .
vv:osoba2 vv:pozna vv:osoba1 .

sa dá rozšíriť nasledovne:

_:x vv:adresa vv:adresa1 . pravidlo gen2 použije nový uzol _:x pridelený vv:osoba1
vv:osoba2 vv:pozna _:x . pravidlo gen1 použije ten istý uzol _:x pridelený vv:osoba1
_:x vv:adresa _:y . pravidlo gen1 použije nový uzol _:y priradený vv:adresa1

Ale nebolo by korektné odvodiť nasledovné:
_:x vv:pozna vv:osoba1 . pravidlo gen2, pretože _:x nie je priradený vv:osoba2

Tieto pravidlá dovoľujú množenie prázdnych uzlov a vytvárajú redundantné grafy. Napríklad jeden prázdny uzol môžeme neustále nahrádzať iným prázdnym uzlom.

Ľahko sa vidí, že S je inštanciou E vtedy a len vtedy, ak sa E dá odvodiť z S aplikovaním týchto pravidiel vo vhodnom poradí. Podľa vety o interpolácii S jednoducho sémanticky implikuje E akk vieme odvodiť (graf obsahujúci) E z grafu S aplikovaním pravidiel gen1 a gen2. Je tiež jasné, že obyčajné aplikovanie pravidiel nie je efektívny vyhľadávací proces, lebo nemusí nikdy skončiť a bude produkovať ľubovoľne mnoho ekvivalentných trojíc.

Nasledujúce pravidlo je špeciálnym prípadom pravidla gen1 a vzťahuje sa len na literály:

Meno pravidla	Predpoklady	Odvodené trojice
lgen	s p lit .	s p _:n . kde _:n = gen(lit)

Tabuľka 3. Pravidlo generalizácie literálov

Toto pravidlo nahradí literál novým prázdnym uzlom. Prázdne uzly v RDF trojiciach môžu byť na mieste subjektu, kým literály nie. Aplikovaním tohto pravidla

môžeme „literálom“ priradiť vlastnosti a s použitím iných pravidiel o nich odvodzovať závery.

3.5.2 Pravidlá RDF-sémantickej implikácie

Meno pravidla	Predpoklady	Odvodené trojice
rdf1	s p o .	p rdf:type rdf:Property .
rdf2	s p lit . kde lit je správne napísaný XML literál.	_:n rdf:type rdf:XMLLiteral . kde _:n = gen(lit) podľa pravidla lgen.

Tabuľka 4. Pravidlá rdf-sémantickej implikácie

Tieto pravidlá sú úplné v nasledujúcom zmysle:

Veta o rdf-sémantickej implikácii. *Množina grafov* S *rdf-sémanticky implikuje* E *vtedy a len vtedy, ak existuje graf* F*, ktorý sa dá odvodiť z* S *a z axiomatických trojíc RDF aplikovaním pravidiel lgen, rdf1, rdf2, a ktorý navyše jednoducho sémanticky implikuje* E.

Dôkaz.
⇐ Ak S alebo E sú prázdne, potom je dôkaz triviálny. Predpokladajme teda, že obidve sú neprázdne. Z predpokladov máme graf F odvodený z S pravidlami lgen, rdf1, rdf2 taký, že F⊨E. V tomto smere musíme iba overiť, že použité pravidlá sú rdf-korektné, teda všetky odvodené trojice budú rdf-sémantickým dôsledkom pôvodného grafu.

- Pravidlo lgen: z trojice <s,p,lit> odvodí <s,p,gen(lit)>, pričom pre generalizačné zobrazenie platí gen: URIREF∪PRAZDNE∪LIT→PRAZDNE. Nadefinujeme si zobrazenie Z ako inverzné ku zobrazeniu gen. Pre každú rdf-interpretáciu I platí: ak [I+Z](<s,p,lit>)=true, potom [I+Z](<s,p,gen(lit)>)=true a teda {<s,p,lit>} rdf-sémanticky implikuje {<s,p,gen(lit)>}.
- Pravidlo rdf1: z trojice <s,p,o> odvodí <p, rdf:type, rdf:Property>. Ak nejaká interpretácia splňuje <s,p,o>, potom p∈V a z bodu 1 definície rdf-interpretácie dostaneme I(<p, rdf:type, rdf:Property>) = true.
- Pravidlo rdf2: z trojice <s,p,lit> odvodí <_:n, rdf:type, rdf:XMLLiteral>, kde lit je správne napísaný XML literál a gen(lit) = _:n. Z bodu 2 definície rdf-interpretácie vieme, že I({<I^{LIT}(lit), rdf:type, rdf:XMLLiteral>}) = true. Potom postupujeme analogicky ako v prípade pravidla lgen, ale zobrazenie Z pozmeníme tak, aby malo pre prázdny uzol _:n hodnotu rovnú XML hodnote literálu lit.

Vidíme teda, že S⊨$_{RDF}$F a zároveň z predpokladov platí F⊨E. Každá rdf-interpretácia je aj jednoduchou interpretáciou. Vezmeme teda ľubovoľnú rdf-interpretáciu I takú, že (∀G∈S) (I(G) = true). Potom aj I(F)=true, lebo S⊨$_{RDF}$F. Ale platí aj I(E) = true, lebo F⊨E. Teda aj S⊨$_{RDF}$E.

⇒ Podobne ako v dôkaze vety o interpolácii, aj tu použijeme Herbrandovskú konštrukciu a aplikujeme ju na „uzáver“, ktorý dostaneme opakovaným aplikovaním odvodzovacích pravidiel na graf, pokiaľ je to možné. V dôkaze ukážeme, že výsledná interpretácia je vhodná pre danú abecedu.

RDF uzáver množiny grafov S označíme C a skonštruujeme ho z S týmto postupom:

1. do S pridáme všetky axiomatické trojice RDF;
2. aplikujeme pravidlo lgen na každú trojicu, ktorá obsahuje správne napísaný XML literál, až kým sa graf aplikovaním pravidla už nemení;
3. aplikujeme pravidlo rdf1, až kým sa graf už nemení;
4. aplikujeme pravidlo rdf2, až kým sa graf už nemení.

C obsahuje presne jeden prázdny uzol _:n pridelený každému literálu v grafe S pravidlom lgen. Pravidlo rdf2 naviac pridá do C trojicu:

_:n rdf:type rdf:XMLLiteral .

V jednoduchých Herbrandovských interpretáciách všetky prvky abecedy znamenajú samých seba, ale sémantické podmienky pre rdf-interpretácie to nie vždy dovoľujú. Napríklad XML literály interpretujeme ako ich XML hodnoty. Pripomíname, že XML hodnoty literálov sú prvky priestoru hodnôt dátového typu rdf:XMLLiteral. Preto v týchto prípadoch rozlišujeme medzi skutočnými sémantickými hodnotami XML literálov a ich syntaktickými „náhradníkmi“, čiže prázdnymi uzlami vytvorenými pravidlom lgen.

Ak chceme zostrojiť RDF interpretáciu, musíme nahradiť XML literály aj ich náhradníkov príslušnými hodnotami literálov z univerza U. Dôkaz však vyžaduje, aby bola každá hodnota XML literálu jednoznačne asociovaná s lexikálnou hodnotou, ktorá ju označuje.

Ak lit je správne vytvorený XML literál, označíme jeho XML hodnotu ako xml(lit). Ďalej definujeme zobrazenie sur: U→abeceda(C)∪PRAZDNE. Je to takmer identické zobrazenie, ale keď ho použijeme na XML hodnoty literálov, dostaneme príslušný prázdny uzol v grafe pridelený tomuto XML literálu pravidlom lgen.

$$sur(x) = \begin{cases} \_:n,\ ak\ (\exists lit \in LIT)(xml(lit) = x \wedge gen(x) = \_:n) \\ x,\ inak \end{cases}$$

Definujeme interpretáciu abecedy uzáveru C a pomocou nej potom Herbrandovskú interpretáciu H:

$HL_H = (JL \cap C) \cup \{xml(x)$: x je správne napísaný XML literál z S $\}$

$U_H = HL_H \cup (abeceda(C) - JL)$

$V_H = \{x: \langle x, rdf:type, rdf:Property\rangle \in C\}$

Ak $x \in V_H$, potom $I^P_H(x) = \{\langle s,o\rangle: \langle sur(s), x, sur(o)\rangle \in C\}$

I^{REF}_H je identické zobrazenie nad URIREF∩S

Ak x je správne napísaný XML literál z S, potom $I^{LIT}_H(x) = xml(x)$, inak $I^{LIT}_H(x) = x$

Zobrazenie Z nad prázdnymi uzlami z C definujeme takto: Z(_:n)=xml(lit), ak _:n=gen(lit), inak Z(x)=x. Keďže C obsahuje všetky axiomatické trojice RDF, H ich splňuje triviálne. Potom vidíme, že [H+Z] splňuje C, S je podgrafom C a teda [H+Z] splňuje aj S.

H je rdf-interpretácia. Z konštrukcie sa ľahko vidí, že H vyhovuje prvým dvom sémantickým podmienkam RDF. Trojice odvodené pravidlom rdf2 vyžadujú, aby I^P_H(rdf:type) obsahovala dvojicu <xml(lit),rdf:XMLLiteral> pre každý správne napísaný XML literál lit. Tretia sémantická podmienka RDF je splnená triviálne. Nesprávne napísané XML literály v interpretácii H znamenajú samy seba, a nepatria do HL_H. Dvojica <lit, rdf:XMLLiteral> sa nemôže vyskytnúť v I^P_H(rdf:type), pretože literály

nevystupujú v pozícii subjektu. Preto je aj táto podmienka splnená a H je RDF-interpretácia.

Chceme dokázať, že $C \models E$. Keďže $S \models_{RDF} E$ a H je rdf-interpretácia, H splňuje E. Pre nejaké zobrazenie W: prazdne(E) $\rightarrow U_H$ platí, že [H+W] splňuje každú trojicu $\langle s,p,o \rangle \in E$, teda $I^P{}_H(p)$ obsahuje dvojicu <[H+W](s),[H+W](o)>. Teda C obsahuje trojicu <sur([H+W](s)), p, sur([H+W](o))>. Ale táto trojica je inštanciou <s,p,o> pri inštančnom zobrazení M(x) = sur(W(x)). Množina všetkých takýchto trojíc je podgrafom grafu C a zároveň je inštanciou grafu E. Teda podľa vety o interpolácii $C \models E$.

□

Najväčšie problémy v dôkaze očividne spôsobujú XML literály. Otázkou zostáva, či sú v RDF nevyhnutne potrebné. RDF dáta sú štruktúrované samy osebe a zrejme nie je ani vhodné, aby literály obsahovali XML tagy.

Táto veta okrem iného ukazuje, že ľubovoľný graf má vyhovujúcu rdf-interpretáciu a dôkaz ukazuje, ako ju zostrojiť z Herbrandovskej interpretácie uzáveru. Použijeme pritom vhodnú interpretáciu správne vytvorených XML literálov a povolíme možnú existenciu vlastností, ktoré nemajú rozšírenia. Ak je E konečné, potom odvodený podgraf C je tiež konečný.

Analógiu nachádzame napríklad vo výrokovom alebo predikátovom počte (Vojtáš 2005). Zaujíma nás korektnosť dôkazov (teda to, či každý dokázateľná formula je pravdivá) a úplnosť (či každá pravdivá formula je dokázateľná). Uvedená veta skúma vzťah medzi odvoditeľnosťou a sémantickou implikáciou. Teda namiesto dôkazu máme odvodzovanie a namiesto pravdivosti sémantickú implikáciu. Ak $S \models E$, potom môžeme graf S považovať za „teóriu“ a graf E za formulu, ktorá je v teórii pravdivá. Pri odvodzovaní zase vychádzame z axióm a „teórie“ S a snažíme sa odvodiť graf E.

Podobná veta (s ďalšími pravidlami a axiómami) sa dá dokázať aj pre rdfs-sémantickú implikáciu. RDFS je skratka pre RDF Schemu, ktorá zavádza do RDF pojmy ako trieda, podtrieda, podvlastnosť, doména a rozsah. Nemení pritom zápis RDF, ale rozširuje abecedu. Tu musíme do interpretácie zahrnúť aj URI referencie ako rdfs:Resource, rdfs:Literal, rdfs:Class, rdfs:subClassOf, rdfs:subPropertyOf. Potom aj axiomatických trojíc je podstatne viac a nie je také jednoduché zostrojiť uzáver aplikovaním pravidiel. Neuvádzame vetu o rdfs-sémantickej implikácii pre nedostatok priestoru. Obidva výsledky zaručujú, že s danými pravidlami vieme korektne odvodzovať nové trojice a nenarušíme pritom sémantiku pôvodných grafov. Vďaka tomu existujú programy, ktoré takéto odvodzovacie pravidlá využívajú.

Odvodzovanie podporuje napríklad databázový systém Sesame, ktorý popisujeme v nasledujúcej kapitole. Existujú aj programy špecializované na odvodzovanie nad RDF dátami, na odhaľovanie vzťahov medzi grafmi a hľadanie chýb. Jedným z nich je program s názvom Racer (Haarslev, Möller 2003).

Posledná veta tiež poskytuje návod, ako rozpoznať rdf-sémantickú implikáciu. Ak $S \models_{RDF} E$, potom z S a axiomatických trojíc sa dá odvodiť graf F taký, že $F \models E$. Problém tak redukujeme na rozpoznanie jednoduchej sémantickej implikácie medzi grafmi F a E. Všeobecný problém určenia jednoduchej sémantickej implikácie medzi dvoma RDF grafmi je rozhodnuteľný, ale NP-úplný. Môžeme ho previesť na problém hľadania podgrafu v ľubovoľnom orientovanom grafe, pričom do úvahy berieme iba prázdne uzly (Carroll 2003).

4 Spracovanie a využitie RDF dát

Vieme už tvoriť výroky vo forme RDF trojíc v niektorej konkrétnej syntaxi RDF. Ako pri všetkých dátach, aj tu však nejde iba o ich vytvorenie, ale aj o spracovanie a vyberanie podľa nejakých kritérií. Pokiaľ používame syntax RDF/XML, môžeme na to použiť nástroje na spracovanie XML, napríklad dopytovací jazyk XQuery, ktorý slúži na prehľadávanie stromovej štruktúry XML. Tento prístup má však viacero nevýhod, predovšetkým fakt, že RDF trojica sa dá zapísať v XML viacerými rôznymi spôsobmi. XML je zamerané na stromy a štrukturované informácie, zatiaľ čo zmyslom RDF je voľné tvorenie výrokov. RDF dáta tvoria graf, nie strom, a to tiež trochu komplikuje ich reprezentáciu v XML. Je jasné, že na spracovanie RDF dát potrebujeme dopytovací jazyk prispôsobený dátovému modelu RDF, ktorý by nebol viazaný na nijakú konkrétnu syntax. V ďalších podkapitolách sa pozrieme na dopytovacie jazyky a ich vlastnosti. V kapitole 4.3 skúmame využitie RDF a uvádzame niektoré aplikácie a jazyky založené na RDF.

4.1 Porovnanie RDF dopytovacích jazykov

Voľnosť implementácie RDF sa ešte viac než na konkrétnej syntaxi prejavila na dopytovacích jazykoch. Na webe je dostupných mnoho RDF dopytovacích jazykov a ďalšie sú ešte vo fáze návrhu. Všetky majú svoje výhody i nevýhody. Väčšinou sa svojou syntaxou podobajú na SQL až na to, že v prípade RDF nepracujeme s databázovými tabuľkami. Všetky dáta sú uložené v RDF grafe. Klauzula FROM teda úplne chýba alebo obsahuje vzorku, časť grafu, ktorú chceme nájsť. Vzorka môže obsahovať voľné premenné a výsledkom dopytu je tabuľka väzieb týchto premenných s nájdenými hodnotami. Iná možnosť je vziať za výsledok nájdené podgrafy pôvodného grafu.

V praktickej časti práce budeme používať dopytovací jazyk SeRQL, čo je súčasťou zadania. Teraz sa pozrieme na to, ako obstojí SeRQL v porovnaní s ostatnými jazykmi. Článok (Haase, Broekstra, Eberhart, Volz 2004) prináša najkomplexnejšie porovnanie vlastností a podporovaných funkcií RDF dopytovacích jazykov. Uvedieme časť výslednej tabuľky. Odstránili sme z nej vlastnosti ako podpora RDF kolekcií a kontajnerov, pretože pre nás nie sú podstatné. V tabuľke 5 znak - znamená, že jazyk nepodporuje danú vlastnosť, OK znamená plnú podporu a Čiast. čiastočnú podporu. Jazyky sme zoradili podľa počtu dosiahnutých bodov, pričom za plnú podporu vlastnosti rátame jeden bod a za čiastočnú podporu polovicu bodu.

	RDQL	Triple	N3	Versa	RQL	SeRQL	RDFQL
Zjednotenie	-	OK	OK	OK	OK	OK	OK
Rozdiel	-	-	-	Čiast.	OK	OK	OK
Voliteľné cesty	-	-	-	OK	Čiast.	OK	OK
Kvantifikácia	-	Čiast.	-	-	OK	OK	OK
Funkcia count	-	-	OK	OK	OK	-	OK
Rekurzia	-	OK	OK	OK	-	-	OK
Reifikácia	Čiast.	Čiast.	-	Čiast.	Čiast.	OK	OK
Prefixy menného priestoru	Čiast.	-	OK	-	OK	OK	OK

	RDQL	Triple	N3	Versa	RQL	SeRQL	RDFQL
Tagy jazyka	-	-	-	-	-	OK	OK
Lexikálny priestor	OK	OK	OK	OK	OK	OK	OK
Priestor hodnôt	Čiast.	-	-	-	OK	OK	OK
Sémantická implikácia	Čiast.	Čiast.	Čiast.	-	OK	OK	OK
Počet bodov	3	4.5	5.5	6	9	10	12

Tabuľka 5. Porovnanie RDF dopytovacích jazykov

Porovnávame nasledujúce vlastnosti jazykov:

- Zjednotenie, rozdiel a kvantifikácia sú zvyčajné množinové operácie na dátach.
- Funkciu count berieme tak, ako sa používa v SQL, teda na zistenie počtu výsledkov.
- Voliteľné cesty znamenajú obdobu „outer join“ v SQL. Niektoré trojice z dopytu označíme ako nepovinné, pokiaľ obsahujú iba doplnkové informácie a nevyžadujeme ich nutne vo výsledkoch.
- Rekurzívne dopyty sa môžu vyskytnúť vtedy, ak daná vlastnosť je tranzitívna. Každý RDF dopytovací jazyk musí obsahovať tranzitívnosť relácií z RDF Schemy, napríklad relácie subClassOf, ktorá je vzťahom triedy a jej podtriedy. Tu nás však zaujímajú tranzitívne relácie, ktoré sa nenachádzajú v RDF Scheme.
- Reifikácia je špeciálna vlastnosť RDF – znamená tvrdenie o inom tvrdení. Zovšeobecnenie pôvodného tvrdenia sa berie ako nový zdroj, ktorému môžeme priradiť ďalšie vlastnosti. O reifikácii povieme viac v kapitole 5.2.
- Prefixy menného priestoru sa používajú v dátach, ale aj v niektorých dopytovacích jazykoch. Napríklad namiesto URI referencie http://www.w3.org/1999/02/22-rdf-syntax-ns# použijeme iba prefix rdf: a príslušnú relatívnu URI referenciu. Dopyty sú potom oveľa prehľadnejšie.
- Tagy jazyka sú súčasťou jednoduchých literálov. Táto vlastnosť znamená, že dopytovací jazyk dokáže oddeliť reťazec literálu od tagu jazyka.
- Lexikálny priestor znamená možnosť pracovať s reťazcom literálu, napríklad porovnať ho s iným reťazcom. Práca s číselnými alebo inými hodnotami literálov podľa ich dátového typu je v riadku označenom ako Priestor hodnôt.
- Sémantickú implikáciu sme definovali v predošlej kapitole. Tu máme na mysli hľadanie inštancií v RDF grafoch.

Prvé tri priečky v porovnaní patria dopytovacím jazykom RQL, SeRQL a RDFQL. Ich názvy sa navzájom podobajú, čo je dosť zavádzajúce a znižuje to prehľadnosť tejto kapitoly. Napriek tomu ich aspoň stručne opíšeme. Porovnanie nezahŕňa jazyk SPARQL (Prud'hommeaux, Seaborne 2006), ktorý uvádza konzorcium W3C ako štandard pre RDF dopytovacie jazyky. SPARQL implementuje všetky vlastnosti, ktoré má jazyk RDFQL.

Implementáciu jazyka RDFQL (2) obsahuje program RDF Gateway, platforma pre vývoj aplikácií a softvérových agentov. RDF Gateway je server aj sieťový agent s vlastnou RDF databázou. RDFQL je akousi kombináciou SQL a JavaScriptu, namiesto obyčajných dopytov v ňom píšeme krátke programy, skripty. Je k dispozícii aj ako Java knižnica.

RQL a SeRQL sú súčasťou databázového systému Sesame. Sesame pracuje s RDF dátami v ktorejkoľvek syntaxi zo zoznamu v kapitole 2.3 a tiež obsahuje možnosť ich vzájomnej transformácie. Dá sa používať ako serverová aplikácia alebo ako knižnice príkazov v Jave. Obidva dopytovacie jazyky pripomínajú SQL a poskytujú výsledky dopytov v tvare RDF grafov alebo tabuliek substitúcií za voľné premenné v dopytoch.

SeRQL podporuje všetky testované vlastnosti okrem rekurzívnych dopytov a funkcie count. Rekurziu však môžeme nahradiť, ak použijeme na dáta odvodzovacie pravidlá a odvodíme chýbajúce vlastnosti. Najväčším nedostatkom SeRQL je absencia klauzuly ORDER BY, čo nás núti zoradiť získané výsledky iným spôsobom. V ďalšej kapitole sa pozrieme bližšie na dopyty a syntax SeRQL.

4.2 SeRQL

SeRQL (Sesame RDF Query Language) je dopytovací jazyk nad RDF alebo RDFS dátami. Skratkou RDFS označujeme RDF Schemu, v ktorej sa dajú definovať triedy. SeRQL je súčasťou systému Sesame. Presná špecifikácia je v dokumente (Broekstra, Kampman 2005).

4.2.1 Časti jazyka SeRQL

Hlavnými zložkami RDF trojíc sú URI referencie a literály. URI referencie môžeme v dopytoch zapísať v plnom tvare alebo skrátené. URI v plnom tvare sú uzavreté v špicatých zátvorkách < >. Ak sa často opakujú URI referencie so spoločným prefixom, je výhodné definovať ho ako prefix menného priestoru, a pomocou neho zapisovať skrátené URI. Začínajú prefixom, za ktorým nasleduje dvojbodka a potom časť URI referencie, ktorá už nie je obsiahnutá v mennom priestore. Prefix a dvojbodku nahradí dopytovací systém plným tvarom menného priestoru. Napríklad pre URI <http://www.openrdf.org/index.html> definujeme prefix sesame ako http://www.openrdf.org/. Potom zapíšeme skrátenú verziu URI ako sesame:index.html, pričom vynechávame zátvorky.

Pomocou funkcií namespace() a localName() vieme z URI referencií oddeliť URI menného priestoru a reťazec skrátenej (lokálnej) URI referencie. Napríklad namespace(<http://www.openrdf.org/index.html>) má hodnotu http://www.openrdf.org/ a localName(<http://www.openrdf.org/index.html>) má hodnotu index.html.

RDF literály sme podrobne popísali v kapitole 2.2.3. Pozostávajú z reťazca v úvodzovkách a nepovinnej časti, ktorá je buď tag jazyka (za znakom @) alebo URI dátového typu (za reťazcom ^^). Jednotlivé časti literálu vieme v SeRQL oddeliť funkciami label, lang a datatype. Napríklad label("computer"@en) je reťazec "computer", lang("computer"@en) je tag jazyka en, datatype("3.14"^^xsd:double) je dátový typ xsd:double.

Premenné sú reťazce, ktoré začínajú písmenom alebo znakom _ a pokračujú postupnosťou písmen, číslic a znakov -, _ alebo bodiek. Rozlišuje sa medzi veľkými a malými písmenami. Premenné nemôžu byť kľúčové slová jazyka SeRQL, čiže select, construct, from, where, using, namespace, true, false, not, and, or, like, label, lang, datatype, null, isresource, isliteral, sort, in, union, intersect, minus, exists, forall, distinct, limit, offset.

Prázdne uzly majú v Sesame svoje vnútorné označenie, hoci nemajú globálne platný identifikátor. Ich označenie platí len v lokálnom kontexte. V dopytoch môžeme používať aj označenia prázdnych uzlov, ale takéto dopyty znamenajú stratu

všeobecnosti. Aj keď použijeme rovnaký zdroj dát, môžu mať prázdne uzly úplne iné označenie. Identifikátory prázdnych uzlov sa dokonca môžu zmeniť aj po pridaní nových trojíc alebo po ich odobratí. Identifikátor prázdneho uzla má napríklad tvar _:bnode1.

4.2.2 Vyjadrenia cesty v grafe

V SeRQL píšeme podobné dopyty ako napríklad v SQL. Dáta však nie sú v tabuľke, ale v RDF grafe. Dopytujeme sa teda nad grafom a klauzula from tým nadobúda nový zmysel. Musíme v nej špecifikovať, akú časť grafu vlastne hľadáme. Dosiahneme to pomocou vyjadrenia cesty. Je to výraz, ktorý sa zhoduje s určitou cestou v RDF grafe. Hľadáme napríklad takúto časť RDF grafu (Ponuka, Plat a Velkost sú premenné):

Obrázok 10. Jednoduchá cesta v grafe

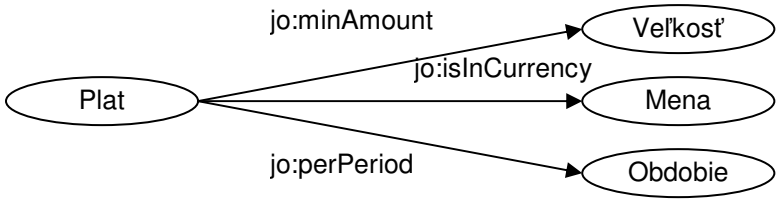
Cestu zapíšeme takto: {Ponuka} jo:hasSalary {Plat} jo:minAmount {Velkost}. Časti v zátvorkách { } znamenajú uzly grafu, ostatné sú hrany. Trojice sú vždy zapísané zľava doprava, teda {subjekt} predikát {objekt}. Výraz z príkladu môžeme tiež rozdeliť na dve trojice a oddeliť ich čiarkami:

{Ponuka} jo:hasSalary {Plat},
{Plat} jo:minAmount {Velkost}

Uzly môžu byť URI referencie, literály, premenné alebo identifikátory prázdnych uzlov. Hrany môžu byť iba URI alebo premenné. V príklade sú Plat, Ponuka a Velkost premenné a jo:hasSalary a jo:minAmount sú URI referencie. Tiež je možné nechať zátvorky pre uzol prázdne, napríklad:

{Ponuka} jo:hasSalary {} jo:moinAmount {Velkost}

Niekedy je vhodné použiť skrátený zápis cesty v grafe. Napríklad v prípade viacerých predikátov pre ten istý subjekt:



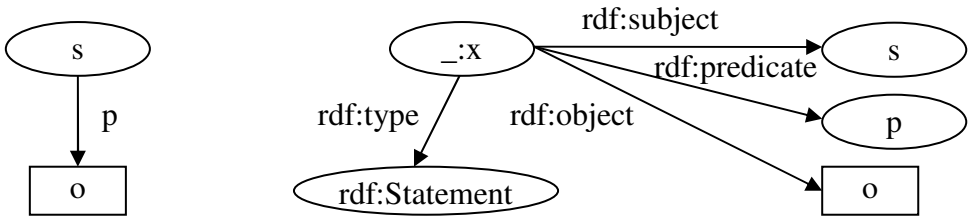
Obrázok 11. Graf s viacerými predikátovými hranami

{Plat} jo:minAmount {Velkost},
{Plat} jo:isInCurrency {Mena},
{Plat} jo:perPeriod {Obdobie}

Chceme skúmať rôzne vlastnosti toho istého subjektu. Nemusíme subjekt písať stále nanovo, ale použijeme nasledujúcu skratku:

{Plat} jo:minAmount {Velkost}; jo:isInCurrency {Mena}; jo:perPeriod {Obdobie}

Ďalšia skratka súvisí s reifikovanými trojicami. O reifikácii povieme viac v kapitole 5.2. Princíp je v tom, že vytvoríme „model“ nejakej RDF trojice pomocou iných štyroch trojíc. Tak získame subjekt (prázdny uzol) reprezentujúci pôvodnú trojicu a môžeme mu priradiť vlastnosti. Takýmto spôsobom vytvárame tvrdenia o iných RDF tvrdeniach. Obrázok ukazuje reifikáciu jednej RDF trojice. Pôvodná trojica je vľavo, výsledný graf vpravo. Prázdny uzol _:x reprezentuje pôvodnú trojicu. Takýto uzol sa nazýva *reifikovaná trojica* (hoci to nie je trojica, je to akýsi náhradník tejto trojice).



Obrázok 12. Cesty v grafe a reifikácia

SeRQL umožňuje vložiť celý zápis trojice do zátvoriek ako uzol. Označujeme tak reifikovanú trojicu. Teda { {s} p {o} } je uzol _:x, ktorý môžeme využiť v iných výrazoch ciest, napríklad:

```
{ {s} p {o} } pred {obj}
```

Toto je ekvivalentné nasledujúcemu výrazu:

```
{_:x} rdf:type {rdf:Statement},
{_:x} rdf:subject {s},
{_:x} rdf:predicate {p},
{_:x} rdf:object {o},
{_:x} pred {obj}
```

Niektorá trojica v dopyte môže byť dôležitá a iné len doplnkové, prípadne ani nemusia existovať. Potom ich môžeme vložiť do hranatých zátvoriek a SeRQL ich berie ako nepovinné. Premenné z nepovinných častí cesty ostanú nepriradené a budú mať hodnotu NULL. Tak neprídeme o dôležité informácie. Napríklad v grafe na obrázku 11 nás zaujíma hlavne výška platu a mena, obdobie niekedy nepoznáme a predpokladáme implicitnú hodnotu mesiac. Potom cestu vyjadríme nasledovne:

```
{Plat} jo:minAmount {Velkost}; jo:isInCurrency {Mena}; [jo:perPeriod {Obdobie}]
```

Takto nájdeme aj tie príklady platu, pre ktoré nepoznáme obdobie.

4.2.3 Klauzuly select a construct

SeRQL obsahuje dva druhy dopytov, ktoré sa líšia svojimi výsledkami. Prvý (s klauzulou select) vráti tabuľku hodnôt alebo množinu väzieb premenných a hodnôt. Druhý (s klauzulou construct) vráti podgraf, ktorý vyhovuje požiadavkám dopytu.

Dopyt sa skladá z týchto častí: SELECT (alebo CONSTRUCT), FROM, WHERE, LIMIT, OFFSET a USING NAMESPACE. Podobajú sa na klauzuly SQL, ale sú tu aj rozdiely.

Klauzula FROM je nepovinná a nasleduje za ňou vyjadrenie cesty v grafe. Tým určíme, ktoré časti RDF grafu chceme prehľadávať. Ak nezadáme FROM, výsledkom dopytu budú konštanty z klauzuly SELECT či graf z klauzuly CONSTRUCT.

LIMIT a OFFSET sú tiež nepovinné, umožňujú získať iba časť výsledkov dopytu. LIMIT znamená maximálny počet zobrazených výsledkov a OFFSET počet výsledkov, ktoré vynecháme pred začiatkom zobrazovania výsledkov.

V klauzule USING NAMESPACE môžeme definovať prefixy menných priestorov a v dopytoch používať iba skrátené verzie URI. Každý použitý prefix musí byť odlišný, napríklad:

```
USING NAMESPACE
   rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>,
   vv = <http://s.ics.upjs.sk/~vanekova/predicates/>
```

Nasledujúce prefixy sú v SeRQL predvolené a nemusíme ich písať v každom dopyte, iba ak ich chceme predefinovať:

```
rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>,
rdfs = <http://www.w3.org/2000/01/rdf-schema#>,
xsd = <http://www.w3.org/2001/XMLSchema#>,
owl = <http://www.w3.org/2002/07/owl#>,
serql = <http://www.openrdf.org/schema/serql#>
```

Za klauzulou SELECT nasledujú hodnoty oddelené čiarkami, ktoré budú zaradené do výsledku. Sú to vlastne názvy premenných špecifikovaných ďalej v klauzulách FROM a WHERE. Ako výsledok dostaneme možné hodnoty týchto premenných. Hviezdička * znamená všetky premenné použité v dopyte. Ak chceme odstrániť duplicity výsledkov, použijeme kľúčové slovo DISTINCT hneď za slovom SELECT.

```
SELECT *
FROM {S} rdfs:label {O}
```

Tento dopyt vráti všetky hodnoty premenných S a O. Môžeme zmeniť ich poradie:

```
SELECT O, S
FROM {S} rdfs:label {O}
```

Dopyt typu CONSTRUCT vráti RDF graf ako množinu trojíc. Preto za touto klauzulou musí nasledovať definícia cesty v grafe. Ak použijeme hviezdičku *, dostaneme na výstupe graf z klauzuly FROM. Môžeme tiež použiť DISTINCT na získanie rôznych trojíc. Tento dopyt definuje inverznú vlastnosť ku vlastnosti adresa:

```
CONSTRUCT {adresa} vv:obyvatel {osoba}
FROM {osoba} vv:adresa {adresa}
USING NAMESPACE
   vv = <http://s.ics.upjs.sk/~vanekova/predicates/>
```

WHERE je nepovinná klauzula a v nej definujeme obmedzujúce podmienky. Testujeme napríklad rovnosť alebo porovnávame hodnoty premenných. Používame operátory =, !=, <, >, <=, >=. SeRQL podporuje dátové typy ako xsd:float, xsd:double, xsd:decimal, xsd:long, xsd:nonPositiveInteger, xsd:byte, atď. Porovnávať však môžeme iba v rámci toho istého typu.
Hodnota NULL sa dá používať rovnako ako ostatné hodnoty.

Operátor LIKE sa používa na hľadanie zhodných reťazcov. Ako divokú kartu používame znak * pre ľubovoľný reťazec. LIKE rozlišuje veľké a malé písmená. Na zrušenie tejto vlastnosti zadáme IGNORE CASE. Tento dopyt nájde veľkosť platu a menu, ak je mena "SKK", "Skk", "skk" a pod.:

```
SELECT Velkost, Mena
FROM {Plat} jo:minAmount {Velkost}; jo:isInCurrency {Mena}
WHERE Mena LIKE "skk" IGNORE CASE
USING NAMESPACE
  jo = <http://nazou.fiit.stuba.sk/nazou/ontologies/offer-job#>
```

SeRQL obsahuje zabudované booleovské funkcie isResource() a isLiteral(), ktoré pre premennú v zátvorke zistia, či je to zdroj alebo literál. Nasledujúci dopyt nájde všetky literály v grafe:

```
SELECT DISTINCT lit
FROM {r} p {lit}
WHERE isLiteral(lit)
```

Funkcie isURI() a isBNode() podobne zistia, či je premenná v zátvorke URI referencia alebo identifikátor prázdneho uzla. Podmienky môžeme spájať logickými spojkami AND, OR a negovať pomocou NOT.

Dva dopyty rovnakého typu môžeme spojiť pomocou operácie UNION, UNION ALL, INTERSECT a MINUS. Predstavujú klasické množinové operácie zjednotenia, zjednotenia bez odstránenia duplicitných trojíc, prieniku a množinového rozdielu.

Vnorené dopyty: SeRQL obsahuje niekoľko typov vnorených dopytov. Ak vo vonkajšom dopyte priradíme hodnotu nejakej premennej, táto hodnota sa prenesie aj do vnútorného dopytu. Operátor IN testuje, či sa premenná alebo množina nachádza vo výsledkoch vnútorného dopytu. Kľúčové slová ANY a ALL slúžia ako existenčný a univerzálny kvantifikátor. ANY testuje, či je nejaká podmienka splnená aspoň pre jeden výsledok vnútorného dopytu. ALL znamená, že podmienka musí platiť pre každý výsledok vnútorného dopytu. Operátor EXISTS je tiež existenčný kvantifikátor, ktorý platí vtedy, ak má vnútorný dopyt aspoň jeden výsledok.

4.3 Praktické využitie RDF

Konzorcium W3C pôvodne vyvinulo RDF ako základ pre sémantický web. Napriek tomu je RDF vhodné aj pre celý rad iných aplikácií. Naskytá sa otázka, čím sa vlastne líši od XML, relačných databáz či dokonca UML a kde je vhodné ho použiť. V krátkosti objasníme každý z týchto rozdielov.

Rozdiel medzi relačnými databázami a RDF je v ich reprezentácii dát. Samozrejme, RDF môžeme uložiť do klasickej databázy s tromi stĺpcami pre subjekt, predikát a objekt. V praxi sa to tak často aj robí, napríklad v RDF databáze Sesame. Databázové tabuľky zase vieme previesť na RDF spôsobom popísaným ďalej v kapitole 5.1. Ale tabuľky sú vhodné vtedy, keď máme pevne štruktúrované údaje a keď chceme kontrolovať ich dátové typy alebo vylúčiť hodnoty NULL. Trojice použijeme vtedy, ak chceme tvoriť ľubovoľné výroky bez obmedzení, ak predpokladáme grafovú štruktúru dát, kde vlastnosti môžu, ale nemusia byť prítomné. V prípade RDF odpadá problém so zbytočnými prázdnymi hodnotami v tabuľkách.

Podobná otázka sa týka XML: ak RDF dáta bežne používame vo formáte XML, aký je medzi nimi rozdiel? Odlišujú sa svojím účelom a zameraním. XML je

prostriedkom pre tvorbu štruktúrovaných dát. Má síce presne definovanú syntax a možnosť vytvárať DTD a šablóny XSL, ale úplne chýba sémantická úroveň. Tagy iba ohraničujú rôzne údaje, ale ich interpretácia je ponechaná aplikácii, ktorá so súborom pracuje. Ak chceme dodať sémantiku, potrebujeme „metajazyk“ a práve túto funkciu plní RDF. Oproti XML má RDF presne definovanú abstraktnú syntax a sémantiku, ale iba niekoľko obmedzení pre konkrétnu syntax.

XML má stromovú štruktúru, zatiaľ čo RDF všeobecnejšiu grafovú. Teda vnorené elementy nejakého XML elementu sú v strome pod ním. Grafovú štruktúru v XML simulujeme pomocou identifikátorov URI. Tak môžeme spojiť dva rôzne elementy v RDF/XML bez toho, aby boli vnorené. Graf v podstate rozdelíme na viac stromov, ktoré sú prepojené URI referenciami. Samozrejme to znižuje prehľadnosť dokumentov vo formáte RDF/XML, lebo nevidíme na prvý pohľad, čo s čím súvisí.

Ak spracúvame XML dokument, musíme si pamätať postupnosť vnorení až kým nespracujeme vnútorné elementy. V RDF však má každá trojica vlastný ucelený význam a môžeme ju spracovať samu osebe bez ohľadu na ostatné trojice v dokumente. Trojice vieme kedykoľvek „spojiť“ do grafu vďaka tomu, že stotožníme uzly s rovnakými URI referenciami.

Rozdiely medzi RDF a UML sú tiež v ich zameraní. UML slúži na modelovanie programov a procesov, štruktúry tried a objektov, na vytváranie presne definovaných diagramov so špecifickým využitím pri návrhu softvéru (napríklad use-case, diagramy aktivít, sekvenčné diagramy). V jazyku RDF vieme modelovať čokoľvek z reálneho sveta. Môžeme sami určiť množinu zdrojov, vlastností a celú štruktúru grafu. RDF grafy obsahujú dáta alebo metadáta, nie však návrh programov. Grafová štruktúra nebola zvolená kvôli svojej vizuálnej, ale logickej stránke, kvôli tomu, že vyjadruje spojenie zdrojov, ich vlastností a hodnôt týchto vlastností.

RDF teda nie je určené ako náhrada alebo „vyšší stupeň“ databáz, XML alebo diagramov UML. Má svoju vlastnú oblasť využitia. Napríklad už v prvých návrhoch internetového prehliadača Mozilla (3) sa používali tabuľky TOC (table of contents) vo formáte RDF/XML. Tento formát sa využíval aj pri iných štruktúrovaných informáciách, napríklad pri obľúbených položkách a bočnej lište prehliadača.

RDF sa bežne používa tiež v jednej z verzií RSS, čo znamená RDF Site Summary alebo Rich Site Summary. Hlavnou myšlienkou je spojenie internetového dokumentu s jeho abstraktom, krátkou informáciou o obsahu dokumentu, o autorovi, dátume vytvorenia, téme a podobne. Tieto informácie sa dajú ľahko spracovať nejakým programom. Využíva sa to pri weblogoch (blogoch). RSS obmedzuje syntax RDF/XML pre vlastné účely na zjednodušenie spracovania. Dokument, ktorý chceme popísať, je zdrojom a priraďujeme mu vlastnosti ako napríklad:

- dc:title – názov dokumentu
- dc:creator – autor, entita zodpovedná za vytvorenie obsahu dokumentu
- dc:subject – téma obsahu dokumentu
- dc:public – entita zodpovedná za publikovanie dokumentu
- dc:date – dátum vytvorenia či zmeny dokumentu
- dc:identifier – jednoznačné označenie dokumentu v danom kontexte
- dc:language – jazyk obsahu dokumentu
- dc:rights – informácie o právach spojených s dokumentom

Prefix dc: alebo dcmi: reprezentuje URI http://purl.org/dc/elements/1.1/. Predikáty z tohto menného priestoru sú dielom skupiny Dublin Core Metadata Initiative (4). Jej cieľom je definovať model metadát na popisovanie internetových zdrojov. Tieto

popisy sa dajú využiť pri vyhľadávaní, v znalostných systémoch a podobne. Na adrese uvedenej vyššie nájdeme aktuálny zoznam predikátov a presný popis každého z nich. Sú to hlavne vlastnosti internetových dokumentov. Popis týchto vlastností sa dá ku dokumentu pripojiť v osobitnom súbore vo formáte RDF/XML alebo vložiť priamo do HTML.

Spomenieme ešte jednu open source aplikáciu, ktorá využíva RDF a nazýva sa FOAF, čo je skratka Friend-of-a-Friend. Umožňuje zverejniť informácie o sebe, popísať prepojenia na priateľov a známych. Spojením získame veľkú „sieť", RDF graf spojený predikátmi foaf:knows. Menný priestor http://xmlns.com/foaf/0.1/ sa tu nahrádza prefixom foaf:. Ďalšie predikáty v tomto mennom priestore sú napríklad:

- foaf:mbox – e-mailová adresa ako platná URI referencia
- foaf:surname – priezvisko osoby
- foaf:firstname – krstné meno osoby
- foaf:nick – prezývka osoby
- foaf:homepage – URL adresa domovskej stránky osoby
- foaf:phone – telefónne číslo osoby
- foaf:publications – odkaz na publikácie osoby

Na spracovanie týchto údajov môžeme použiť dopytovacie jazyky pre RDF, ale aj niektoré špecializované nástroje. Takto sa dá napríklad vytvoriť stránka s odkazmi na stránky kolegov alebo program, ktorý zisťuje „kto koho pozná".

V tejto práci najviac využijeme ontológie, ďalšiu možnosť využitia RDF. Pojem „ontológia" je všeobecne dosť nejasný a často aj kontroverzný. Pokúsime sa ho teda trochu objasniť.

Podľa stránky (Gruber 2003) „ontológia je špecifikácia konceptualizácie". Takáto stručná definícia potrebuje aj ďalšie vysvetlenie. Pod konceptualizáciou má autor na mysli abstraktný, zjednodušený pohľad na svet. Špecifikácia môže byť zapísaná v nejakom formálnom jazyku. Podľa toho však nevieme presne určiť, čo ešte je ontológia a čo už nie je, vymedziť jej hranice. Existujú aj presnejšie definície, napríklad definícia podľa dictionary.com: „ontológia je explicitná formálna špecifikácia o tom, ako reprezentovať objekty, koncepty a iné entity, ktoré existujú v nejakom okruhu záujmu, a vzťahy medzi nimi". Najpodrobnejšiu definíciu ontológie a jej štruktúry vytvoril J. Geller (Geller). Tu ju uvádzame v skrátenej verzii a bez príkladov.

Definícia.

1. Ontológia je graf, teda má grafovú štruktúru dát. Každý uzol predstavuje „koncept". Koncept je všetko, čo chceme popisovať. Koncepty sa často nazývajú „kategórie" alebo „triedy", v RDF je ekvivalentným termínom „zdroj". Konceptom zodpovedajú frázy alebo slová, väčšinou podstatné mená.
2. Uzly ontológie sú spojené hranami rôznych typov. Najdôležitejší typ hrany sa nazýva IS-A. Názov je z Angličtiny, znamená „je (niečím)". Namiesto hrán IS-A sa často používajú „AKO" (A kind of) alebo „SUBCLASS".
3. Uzly a hrany IS-A tvoria zakorenený orientovaný acyklický graf. Čím bližšie ku koreňu je nejaký uzol, tým všeobecnejší koncept reprezentuje.
4. Ak hrana IS-A vedie od X ku Y, potom každú vec z reálneho sveta, ktorú označujeme ako X, môžeme označiť aj Y. Teda X je (nejaké) Y, v Angličtine X IS-A Y. Napríklad „A car IS-A vehicle", teda „Auto je (nejaké) vozidlo". Graf je acyklický, po hranách IS-A sa nikdy nedostaneme do toho istého uzla.

5. Uzly môžu mať priradené iné informácie, napríklad vlastnosti, vzťahy a pravidlá.
6. Vlastnosti vždy majú pre konkrétny uzol nejakú hodnotu, napríklad auto má atribút farba s hodnotou červená a podobne.
7. Vzťah je orientovaná hrana medzi dvomi konceptmi, ktorá má svoje meno. Vzťahy môžu vytvárať cykly na rozdiel od hrán IS-A.
8. Hrany IS-A sa môžu využiť na dedenie, prenos vlastností medzi konceptmi v grafe zhora nadol. Dedičnosť sa môže týkať buď iba vlastností, alebo aj ich hodnôt. Zdedené hodnoty sa vždy dajú predefinovať. Je dovolená aj viacnásobná dedičnosť, čo niekedy vedie k protirečeniam. Niektoré ontológie preto majú prostriedky na zablokovanie dedičnosti.
9. Pravidlá vyjadrujú univerzálne poznatky o konceptoch a pomocou nich sa dajú odvodiť nové vzťahy či vlastnosti.
10. Informácie v ontológii musia vystihovať prirodzené ľudské poznatky. Dajú sa využiť na odvodzovanie podobné ľudskému uvažovaniu. To zahŕňa napríklad dedičnosť vlastností, tranzitívnosť a klasifikácie. Klasifikácia znamená, že pre daný koncept so známymi vlastnosťami vieme rozhodnúť, pod ktorý iný koncept v ontológii ho zaradíme.
11. Do ontológie sa obyčajne pridá „umelý“ koreň, aby sme predišli viacnásobným koreňom. Nazýva sa „Thing“ alebo „Entity“.
12. Niektoré ontológie dovoľujú hrany Instance-Of, iné modelujú len triedy.
13. Ontológie obsahujú symbolické znalosti. Ich význam je slovný, nie matematický.

Uvedená definícia v mnohých ohľadoch pripomína definíciu RDF grafu. Najčastejšie naozaj nájdeme ontológie vo forme RDF grafov, ale sú aj iné možnosti. Je vidieť, že pri ontológiách nemáme danú ich presnú formu. Sú to v podstate množiny výrokov, ale ich zmysel spočíva v tom, ako reprezentujú znalosti. Všetky poznatky o svete sa najskôr musia premeniť na entity a vzťahy medzi nimi. Keď tieto entity a vzťahy zapíšeme v nejakom formálnom jazyku, dostaneme ontológiu.

Pojem „ontológia“ má svoju tradíciu vo filozofii, kde znamená vedu o bytí, o existencii. Presnejšie táto veda skúma a vysvetľuje povahu, základné vlastnosti a vzťahy všetkých existujúcich vecí a takisto aj príčiny a zákony bytia. Nie je to celkom bez súvisu s informatickou ontológiou. V informatike všetko, čo „existuje“, čo patrí do našej domény, môžeme popísať. Nadefinujeme si všetky entity, ich vlastnosti a vzájomné vzťahy. Takáto ontológia v informatike tiež svojím spôsobom hovorí o existencii vecí.

Pre potreby ontológií je vytvorených niekoľko jazykov, napríklad OIL, DAML+OIL a predovšetkým OWL, Web Ontology Language (McGuiness, van Harmelen 2004). V OWL je vytvorená aj ontológia, ktorú budeme používať v praktickej časti. OWL pridáva niektoré užitočné funkcie pre prácu s RDF dátami, ktoré nie sú obsiahnuté v RDF Scheme. Napríklad umožňuje zadať kardinalitu vlastností, vylúčiť súčasné použitie niektorých vlastností, definovať inverzné vlastnosti a vzťahy medzi triedami, napríklad ich disjunktnosť. Podporuje aj odvodzovanie nad dátami. Medzi ciele OWL patrí podpora vývojových verzií ontológií, automatické hľadanie chýb, prenosnosť a hromadná použiteľnosť ontológií, zhoda s použitými štandardmi (napríklad RDF a XML).

Pri vytváraní ontológie definujeme triedy a vzťahy medzi nimi (podtriedy), vlastnosti tried, obmedzenia vzťahov a vlastností. Používame abecedu OWL, ktorá je oproti abecede RDF oveľa rozsiahlejšia. Menný priestor

http://www.w3.org/2002/07/owl# skrátene zapisujeme owl:, napríklad v prípade owl:Class, owl:Ontology, owl:Datatype, owl:imports, owl:disjointWith, owl:unionOf a používame aj niektoré predikáty jazyka RDF Schemy, napríklad rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf.

OWL je už natoľko zložitý jazyk, že pre prácu s ním nevyhnutne potrebujeme vhodný editor. Máme na výber niekoľko editorov, väčšinou vytvorených v Jave. Poskytujú grafické rozhranie podobné nástrojom UML, v ktorom jednoducho vytvoríme triedy a vlastnosti na grafickej ploche a program potom vygeneruje súbory RDF/XML, prípadne vlastné pomocné súbory. Ontológia, ktorú používame v tejto práci, bola vytvorená pomocou editora Protégé (1).

Ontológie (s využitím vhodného editora) nám umožnia plne využiť možnosti RDF. Môžeme graficky modelovať entity, vlastnosti a vzťahy z ľubovoľnej domény a túto štruktúru dát potom využiť vo vlastných aplikáciách.

5 Fuzzy logika a RDF

V tejto kapitole prejdeme na úplne inú tému a podáme stručný úvod do fuzzy logiky. Popíšeme predikáty, logické spojky, fuzzy množiny a agregačné funkcie. Potom preskúmame možnosti fuzzyfikácie RDF. Tejto téme sa venuje aj článok (Vaneková, Bella, Gurský, Horváth 2005).

Fuzzy logika je rozšírením Booleovskej logiky. Zatiaľ čo Booleovská logika pozná len dve pravdivostné hodnoty „true“ (1) a „false“ (0), fuzzy logika pracuje aj s čiastočnými pravdivostnými hodnotami, ktoré sú niekde medzi 0 a 1. Presnejšie povedané, fuzzy logika berie do úvahy pravdivostné hodnoty zo spojitého intervalu [0,1]. Existuje množstvo spôsobov, ako spracovať nepresné či približné informácie, jedným z nich je práve fuzzy logické programovanie. Viac informácií o tejto téme je v článku P. Vojtáša (Vojtáš 2001) a v poznámkach z prednášok o fuzzy logickom programovaní (Vojtáš 1998). Uvedieme len niekoľko základných informácií pre lepšie pochopenie problematiky.

5.1 Rekapitulácia fuzzy logiky

Fuzzy výrokový počet pracuje s elementárnymi výrokmi a z nich tvorí zložitejšie výroky pomocou logických spojok. Zhoduje sa s klasickým výrokovým počtom až na to, že pravdivostné hodnoty ľubovoľných výrokov patria do intervalu [0,1]. Podľa toho musíme zmeniť aj logické spojky. V klasickej dvojhodnotovej logike vieme každej spojke jednoznačne priradiť pravdivostnú funkciu. Ale rozšírenie definičného oboru týchto funkcií z $\{0,1\}^2$ na $[0,1]^2$ nie je jednoznačné. Môžeme použiť ľubovoľnú funkciu, ktorá spĺňa isté podmienky. Preto vzniklo viac variantov pravdivostných funkcií:

Lukasiewicz:	$\&_{L\cdot}(x,y)=\max(x+y-1, 0)$	$\rightarrow_{L\cdot}(x,y)=\min(1-x+y, 1)$
Gödel:	$\&_{G\cdot}(x,y)=\min(x,y)$	$\rightarrow_{G\cdot}(x,y)=1$, ak $x\leq y$, inak y
produktové spojky:	$\&_{P\cdot}(x,y)=x*y$	$\rightarrow_{P\cdot}(x,y)=1$, ak $x\leq y$, inak y/x

Definícia. Funkcia $C: [0,1]^2 \rightarrow [0,1]$ nazývame *konjunktor*, ak:

- je rozšírením dvojhodnotovej konjunkcie, teda $C(0,0) = C(0,1) = C(1,0) = 0$, $C(1,1) = 1$,
- je neklesajúca v obidvoch argumentoch a zľava spojitá v druhom argumente,

- platí $(\forall r>0)\ C(1,r)>0$.

Definícia. Funkcia $t: [0,1]^2 \to [0,1]$ sa nazýva *t-norma*, ak:
- je rozšírením dvojhodnotovej konjunkcie, teda $t(0,0) = t(0,1) = t(1,0) = 0$, $t(1,1) = 1$,
- je neklesajúca v obidvoch argumentoch,
- je symetrická , teda $t(x,y) = t(y,x)$ a asociatívna, teda $t(x,t(y,z)) = t(t(x,y),z)$,
- platí $t(x,1)=t(1,x)=x$.

Definícia. Funkcia $I: [0,1]^2 \to [0,1]$ sa nazýva *implikátor*, ak
- je rozšírením dvojhodnotovej implikácie, teda $I(0,0) = I(0,1) = I(1,1) = 1$, $I(1,0) = 0$,
- je nerastúca v prvom argumente, sprava spojitá a neklesajúca v druhom argumente,
- platí $(\forall h<1)\ I(1,h)<1$.

Definícia. *Agregačná funkcia* je ľubovoľná funkcia n premenných neklesajúca vo všetkých argumentoch. Označujeme ju $@\cdot(x_1, ..., x_n)$. Typické agregačné funkcie používané v databázach sú minimum, maximum a priemer.

Uvedené funkcie $\&_{L\cdot}$, $\&_{G\cdot}$, $\&_{P\cdot}$ sú konjunktory, agregačné funkcie aj t-normy. Funkcie $\to_{L\cdot}$, $\to_{G\cdot}$, $\to_{P\cdot}$ sú implikátory.

V predikátovom počte jazyk rozšírime o konštanty, premenné, funkčné symboly a predikátové symboly. Z konštánt, premenných a funkčných symbolov vznikajú termy. Presnú definíciu termu pomocou vytvárajúcej postupnosti alebo stromu nájdete v textoch (Vojtáš 2005) a (Vojtáš 1998). Ako príklady termov uvedieme 2, π, $\sin(3x+1)$, "P12" a pod. Dôležitou súčasťou jazyka sú ďalej predikátové symboly. N-árny predikát aplikujeme na n termov príslušných typov a dostaneme atomárnu formulu, napríklad $\neq(2, \sin(3.5))$. Atomárne formuly tu zohrávajú úlohu elementárnych výrokov. Zložitejšie formuly z nich získame pomocou logických spojok. Atómom priradíme pravdivostné hodnoty z intervalu [0,1]. Zápis $A.x$ znamená, že atóm A má pravdivostnú hodnotu x. Napríklad párne_číslo(4).1 znamená, že predikát párne_číslo má pre term 4 pravdivostnú hodnotu 1.

Definícia. Abeceda jazyka predikátového počtu L je tvorená systémami symbolov Var_a, $a \in$ Atr, $\mathcal{P}$, $\mathcal{F}$, $\mathcal{C}_a$, $a \in$ Atr, S. Pričom S je systém logických spojok, Atr je množina atribútov, Var_a je množina premenných typu a, $\mathcal{P}$ obsahuje predikátové symboly, $\mathcal{F}$ funkčné symboly a $\mathcal{C}_a$ konštanty typu a. Okrem týchto symbolov jazyk obsahuje kvantifikátory $\forall,\exists$, zátvorky a čiarku.

Definícia. *Fuzzy teória* v jazyku L je ľubovoľné zobrazenie $T: FL \to [0,1]$, kde FL je množina formúl jazyka L. Formula $f.x$ je axióma teórie T, ak $T(f) = x$.

Pri fuzzy logickom programovaní (Vojtáš 1998) využívame fakty a pravidlá. Fakty sú formule ohodnotené svojou pravdivostnou hodnotou. Pravidlo je formula v tvare $H \leftarrow B$, kde H je atóm a B pozostáva z atomárnych formúl a fuzzy logických spojok. Logický program je množina pravidiel spolu s faktami. Fakty sú uložené v extenzionálnej databáze vo forme fuzzy predikátov. Logický program začína od faktov a na ne aplikuje pravidlá, čím odvodíme nové fakty.

Definícia. Pravidlo fuzzy logického programu je formula tvaru $H \leftarrow_i B$ kde B je formula jazyka L bez kvantifikátorov a H je atomárna formula. Formula B sa nazýva telo pravidla (body). Formula H sa nazýva hlava pravidla (head). Fakt fuzzy logického programu je ľubovoľná atomárna formula.

Definícia. Fuzzy logický program P je zobrazenie definované na konečnej množine faktov a pravidiel do množiny pravdivostných hodnôt také, že P(f) > 0 pre každú formulu f∈dom(P). Teda fuzzy logický program je teória, ktorej axiómy sú fakty a pravidlá.

Príklad. Máme tieto fakty a pravidlá:

zaujimava(praca1).0.2
zaujimava(praca2).0.45
dobre_platena(praca1).0.98
dobre_platena(praca2).0.6
(dobra_praca(x) $\leftarrow_P$ dobre_platena(x) $\&_G$ zaujimava(x)).0.9

Používame vyššie definované pravdivostné funkcie pre logické spojky.
$\&_G(x,y) = \min(x,y)$, $\&_P(x,y) = x*y$

Pri spracovaní pravidla najskôr vypočítame pravdivostnú hodnotu tela pravidla. Implikátor nahradíme príslušným (reziduovaným) konjunktorom a ten aplikujeme na vypočítanú hodnotu a pravdivostnú hodnotu pravidla. Zložením logických funkcií vlastne dostaneme agregačnú funkciu, ktorá z pravdivostných hodnôt predpokladov vypočíta pravdivostnú hodnotu výsledku.

Uvedený fuzzy logický program odvodí nasledujúce výsledky:

dobra_praca(praca1).ph1 ph1 = $\&_P(\&_G(0.2, 0.98), 0.9) = \min(0.2, 0.98) * 0.9 = 0.18$
dobra_praca(praca2).ph2 ph2 = $\&_P(\&_G(0.45, 0.6), 0.9) = \min(0.45, 0.6) * 0.9 = 0.405$

Fuzzy logické programy vieme reprezentovať napríklad pomocou relačných databáz. Samotná databáza bude obsahovať iba fakty spolu s ich fuzzy pravdivostnými hodnotami. Pravidlá transformujeme na databázové dopyty alebo definície databázových pohľadov (view). Takýto dopyt vytvorí nové dáta z existujúcich dát rovnakým spôsobom, ako by príslušný logický program odvodil nové fakty.

Príklad: Logický program z predošlého príkladu prevedieme na SQL. Fakty z predikátov zaujimava a dobre_platena budú uložené v tabuľkách, kde druhý stĺpec obsahuje pravdivostnú hodnotu faktu:

zaujimava

ID	PH
praca1	0.2
praca2	0.98

dobre_platena

ID	PH
praca1	0.45
praca2	0.6

Tabuľka 6. Databázové tabuľky zaujimava a dobre_platena

Z pravidla potom vytvoríme dopyt:

```
SELECT zaujimava.ID, max(zaujimava.PH, dobre_platena.PH) * 0.9
FROM zaujimava, dobre_platena
WHERE zaujimava.ID = dobre_platena.ID
```

Nie vždy máme k dispozícii fuzzy predikáty, ktoré by sme mohli spracovať, napríklad dobre_platena. Otázkou ostáva, aký vysoký plat má mať práca, aby bola pre nás dobre platená. Takéto pravidlá nie sú jednoznačné, každý si môže kritériá definovať inak. Ak máme tabuľku s presnými informáciami o plate, vieme z nej

podľa daných kritérií vypočítať fuzzy pravdivostné hodnoty. Pre nasledujúcu tabuľku plat a daný dopyt dostaneme presne tabuľku (VIEW) dobre_platena:

plat

ID	plat
praca1	14500
praca2	16000

Tabuľka 7. Databázová tabuľka plat

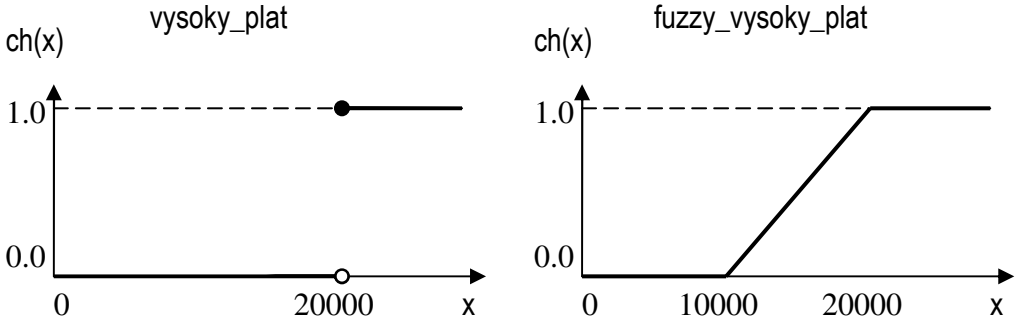
```
CREATE VIEW dobre_platena(ID, PH) AS
(SELECT  ID, max(0, min(1, plat/10000 - 1))
FROM plat)
```

Takéto kritériá vlastne určujú členstvo v množine. Nejde o zvyčajnú množinu v zmysle teórie množín, ale o fuzzy množinu. Pri klasickej množine vieme vždy jednoznačne určiť, či do nej daný prvok patrí alebo nepatrí. Vieme tiež vytvoriť charakteristickú funkciu množiny. Jej obor hodnôt je {0,1}. Ak má charakteristická funkcia pre nejaký prvok hodnotu 1, prvok patrí do množiny, v opačnom prípade nepatrí.

Mohli by sme napríklad definovať množinu vysoky_plat = {x∈R: x ≥ 20000}. Jej charakteristická funkcia bude $ch_{vysoky\_plat}(x) = sgn(sgn(x-20000)+1)$. Problém je, že musíme určiť presnú hranicu. Plat výšky 19999 by už do množiny nepatril, hoci jediná koruna nerobí príliš veľký rozdiel v tom, či plat považujeme za vysoký. V prirodzenom ľudskom uvažovaní málokedy nájdeme presné hranice pojmov ako vysoký či nízky. S klesajúcou výškou platu ho len uznávame za „vysoký“ čoraz menej.

Tak sa dostávame ku fuzzy množine, ktorá pripúšťa aj čiastočné členstvo. Charakteristická funkcia fuzzy množiny má za obor hodnôt celý interval [0,1]. Ak je hodnota charakteristickej funkcie 1, znamená to úplné členstvo prvku v množine, 0 znamená, že prvok do množiny nepatrí a hodnoty medzi 0 a 1 označujú čiastočné členstvo. Definujeme množinu fuzzy_vysoky_plat nasledovne: fuzzy_vysoky_plat = {<x,ch(x)>, x∈R}, kde ch je charakteristická funkcia ch(x) = max(0, min(1, x/10000 - 1)).

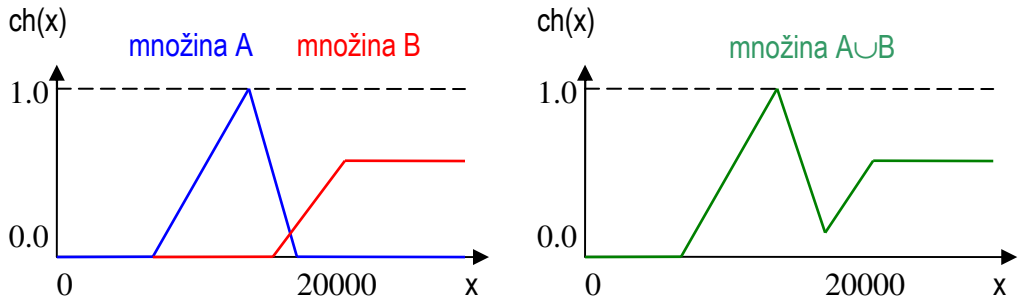
Graficky znázorníme rozdiel medzi klasickou množinou a fuzzy množinou:



Obrázok 13. Klasická množina a fuzzy množina

Pri fuzzy množinách používame podobné množinové operácie ako pri klasických množinách, ale musíme ich charakteristické funkcie rozšíriť na interval [0,1]. Napríklad prienik zostrojíme tak, že na hodnoty charakteristickej funkcie pre

jednotlivé prvky aplikujeme konjunktory definované na začiatku kapitoly. Na obrázku sú dve fuzzy množiny a ich zjednotenie, pričom sme použili funkciu $\vee\cdot(x,y)=\max(x,y)$.



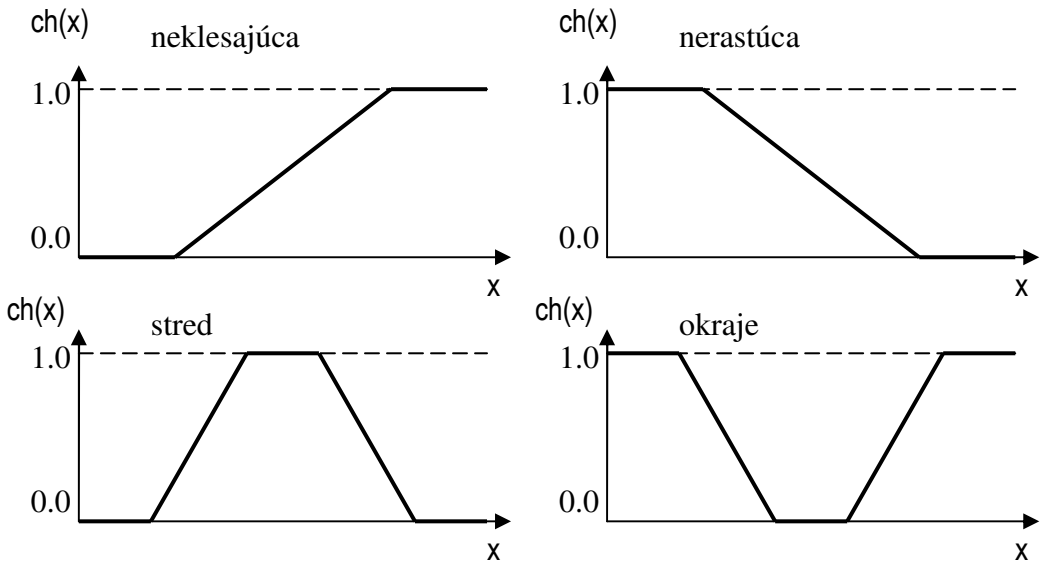
Obrázok 14. Operácie nad fuzzy množinami

Takéto operácie využijeme vtedy, keď chceme z viacerých fuzzy množín dostať jednu výslednú množinu. Napríklad máme vytvorené dve fuzzy množiny blizko a fuzzy_vysoky_plat. Chceme však prácu, ktorá by spĺňala obidve podmienky, musí byť zároveň dobre platená aj blízko. Použijeme niektorý konjunktor a dostaneme prienik, napríklad $ch_{dobra\_praca}(x) = ch_{blizko}(x)\ \&_P\ ch_{fuzzy\_vysoky\_plat}(x) = ch_{blizko}(x) * ch_{fuzzy\_vysoky\_plat}(x)$.

Namiesto logických spojok sa na vypočítanie výslednej pravdivostnej hodnoty často používa vážený priemer. Váhy potom vyjadrujú dôležitosť množín a príslušných vlastností, ktoré množiny definujú. Napríklad je pre nás plat dvakrát dôležitejší než to, aby práca bola blízko. Potom $ch_{dobra\_praca}(x) = (ch_{blizko}(x) + 2*ch_{fuzzy\_vysoky\_plat}(x))/3$. Vážený priemer je agregačná funkcia, pretože je neklesajúca vo všetkých argumentoch. Samozrejme váhy musia byť kladné čísla.

Fuzzy množiny a agregačné funkcie vyjadrujú požiadavky užívateľa. Dáta budú mať výslednú pravdivostnú hodnotu podľa toho, nakoľko týmto požiadavkám vyhovujú. Pravdivostnú hodnotu tu nazývame ohodnotenie (v angličtine rating, ranking). Hľadáme ten najlepší výsledok, k najlepších výsledkov (Gurský 2006) alebo výsledky lepšie ako prahová hodnota. Pre každú možnosť už existujú rôzne algoritmy a heuristiky.

Agregačné funkcie vyjadrujú spôsob, ako skombinujeme čiastočné výsledky, zatiaľ čo fuzzy množiny určujú usporiadanie dát. Napríklad množina fuzzy_vysoky_plat priradí vyšším hodnotám platu vyššie ohodnotenie, teda zaradí ich vyššie podľa užívateľských preferencií. Okrem možnosti „čím viac, tým lepšie“ môže nastať aj možnosť „čím menej, tým lepšie“ (napríklad dane, ceny tovaru). Niekedy hľadáme hodnoty niekde v strede možného intervalu alebo na okrajoch intervalu. Toto sú základné typy fuzzy množín, ktoré budeme ďalej používať:



Obrázok 15. Základné typy fuzzy množín

5.2 Fuzzyfikácia

Fuzzyfikácia znamená zmenu prístupu od dvojhodnotovej logiky na fuzzy logiku. V prípade RDF musíme rozhodnúť, kam vložíme pravdivostnú hodnotu. Špecifikácie RDF nenavrhujú nijaké riešenie tohto problému, lebo sa pri návrhu nerátalo s nijakou fuzzyfikáciou. Všetky trojice v RDF sa jednoducho považujú za pravdivé.

Spôsob ukladania a spracovania informácií o pravdivostných hodnotách závisí od toho, na ktorej úrovni sa vlastne vyskytnú približné údaje. Musíme zodpovedať nasledujúce otázky: Sú zdrojové dáta presné, alebo už pri zdrojových dátach rozlišujeme ich pravdivostné hodnoty? Je užívateľský dopyt presný alebo približný? Očakávame približné výsledky? Budú sa fuzzy výsledky dopytov ďalej spracovávať alebo to nie je potrebné? Z odpovedí vyplynie, nakoľko vlastne musíme meniť existujúcu dátovú štruktúru. V odbornej literatúre, napríklad v článku (Priebe, Schlager, Pernul 2003), sa uvádzajú nasledujúce stupne viachodnotovosti:

1) presné dáta, presný dopyt, presný výsledok (PPP)
2) presné dáta, fuzzy dopyt, fuzzy výsledok (PFF)
3) fuzzy dáta, fuzzy dopyt, fuzzy výsledok (FFF)
4) presné dáta, presný dopyt, fuzzy výsledok (PPF)

Fuzzy logické programovanie predstavuje možnosť (FFF), pretože všetky predikáty, či už vopred dané alebo odvodené, musia byť fuzzy, a rovnako aj odvodzovacie pravidlá majú svoje pravdivostné hodnoty. Klasické logické programovanie a databázy zase predstavujú možnosť (PPP). Tá je zároveň obsiahnutá aj v RDF, či už v odvodzovacích pravidlách alebo dopytovacích jazykoch. Možnosti (PFF) a (PPF) sa líšia podľa toho, čo považujeme za fuzzy dopyt. Článok (Priebe, Schlager, Pernul 2003) uvádza ako fuzzy dopyt taký, v ktorom použijeme operátor LIKE a divoké karty *, _. Tak získame namiesto presnej hodnoty reťazca podobné hodnoty. Fuzzy množiny by podľa tohto rozdelenia prekvapujúco patrili pod možnosť (PPF), hoci presne vystihujú intuitívnu predstavu fuzzy dopytu.

V tejto práci vychádzame z presných RDF dát a chceme ich ohodnotiť podľa požiadaviek užívateľa, čo je možnosť (PPF). Našou úlohou je priradiť nejaké ohodnotenie rôznym subjektom s použitím fuzzy množín transformovaných na databázové dopyty. Toto ohodnotenie môže slúžiť aj na ďalšie spracovanie.

Ľahko sa vidí, že RDF ako výrokový jazyk pozostáva z binárnych predikátov. Niekedy je možné transformovať n binárnych predikátov na jeden n-árny, prípadne (n+1)-árny. Napríklad predikáty minAmount(Salary_1, 29.7), isInCurrency(Salary_1, USD), perPeriod(Salary_1, tuHour), majú rovnakú prvú položku Salary_1. Vytvoríme z nich predikát salary(29.7, USD, tuHOUR) alebo salary(Salary_1, 29.7, USD, tuHOUR). RDF trojice môžeme takto zlepiť na n-árny predikát, ak majú rovnaký subjekt, v našom prípade Salary_1. Existujúce nástroje na spracovanie RDF však neumožňujú nijaké integritné obmedzenia. Nemôžeme zaručiť, že v grafe bude vždy všetkých n trojíc, ktoré potrebujeme na zostavenie n-árneho predikátu.

Podobne sa tiež dajú n-árne alebo (n+1)-árne predikáty rozdeliť na n binárnych. Platí to aj pre databázové tabuľky. Ak by sme mali tabuľku plat so stĺpcami id, hodnota, mena a obdobie, zobrali by sme hodnoty v stĺpci id ako subjekty, názvy stĺpcov ako predikáty a príslušné hodnoty v stĺpcoch ako objekty RDF trojíc. Ak chýba stĺpec id, dosadíme vlastné identifikátory na miesto subjektov. Postup ilustruje nasledujúci obrázok.

id	minAmount	isInCurrency	perPeriod
Salary_1	29.7	USD	tuHour
...			

→

Salary_1 minAmount "29.7" .
Salary_1 isInCurrency "USD".
Salary_1 perPeriod tuHour .

Obrázok 16. Transformácia tabuliek na RDF trojice

V RDF sa teda dajú zapísať ľubovoľné dáta. Ak chceme pracovať s fuzzy dátami, potrebujeme do nich vložiť pravdivostné hodnoty z intervalu [0,1]. Na ktorej úrovni však pravdivostné hodnoty vložíme, to závisí od účelu, na ktorý chceme fuzzy RDF dáta použiť. Analyzovali sme niekoľko možností, ktoré popisujeme v nasledujúcich podkapitolách.

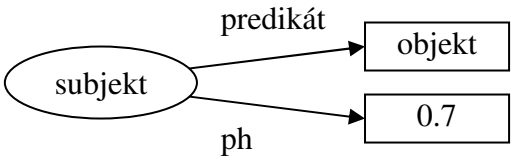
5.2.1 Implementácia fuzzy logického programovania v RDF

Fuzzy logické programovanie (FLP) znamená odvodzovanie nových fuzzy n-árnych predikátov zo zadaných predikátov pomocou logických programov, pravidiel. Chceme teda ľubovoľný predikát typu $p(a_1, a_2, ..., a_n).PH$ transformovať na trojice a tie ďalej spracovať. Ak berieme pravdivostnú hodnotu ako ďalší atribút, máme predikát $p(a_1, a_2, ..., a_n, PH)$. Z neho dostaneme trojice $<id_1, p_1, a_1>$, $<id_1, p_2, a_2>$, ..., $<id_1, p_n, a_n>$, $<id_1, ph, PH>$. Trojíc je n+1. Nemali sme daný nijaký identifikátor v rámci predikátu, tak sme dodali id_1. Táto možnosť využitia RDF je však trochu neopodstatnená (pre fuzzy logické programovanie existujú vhodnejšie nástroje) a naráža na množstvo problémov. RDF poskytuje príliš veľkú voľnosť pri tvorení predikátov. Je možné vytvoriť trojice s tým istým subjektom a predikátom a s rôznymi objektmi. V dátach sa teda môžu vyskytnúť dve alebo ešte viac rôznych pravdivostných hodnôt toho istého predikátu. Takisto nemôžeme zabrániť vymazaniu niektorých trojíc, ktoré sú časťou predikátu. Toto všetko je prípustné v rámci RDF, ale nie vo fuzzy logickom programovaní. Ak by sme chceli korektne implementovať fuzzy logické programovanie teda možnosť (FFF), potrebovali by sme nástroj na zabezpečenie

integrity dát. To by znamenalo naprogramovať nový databázový systém alebo nadstavbu existujúceho.

5.2.2 Implementácia FLP pre binárne predikáty

RDF trojice prirodzene reprezentujú binárne predikáty. Väčšina problémov implementácie FLP v predchádzajúcej možnosti vzniká pri rozdeľovaní a skladaní n-árnych predikátov. Je teda možné obmedziť sa na binárne predikáty. Problémom ostáva, kam vložiť pravdivostnú hodnotu. V RDF sú predikáty typu p(s,o), teda trojice <s,p,o>, ale my potrebujeme p(s,o).PH. Ak by sme vložili pravdivostnú hodnotu ako objekt novej trojice, dostali by sme asi takýto graf:



Obrázok 17. Pravdivostná hodnota ako objekt

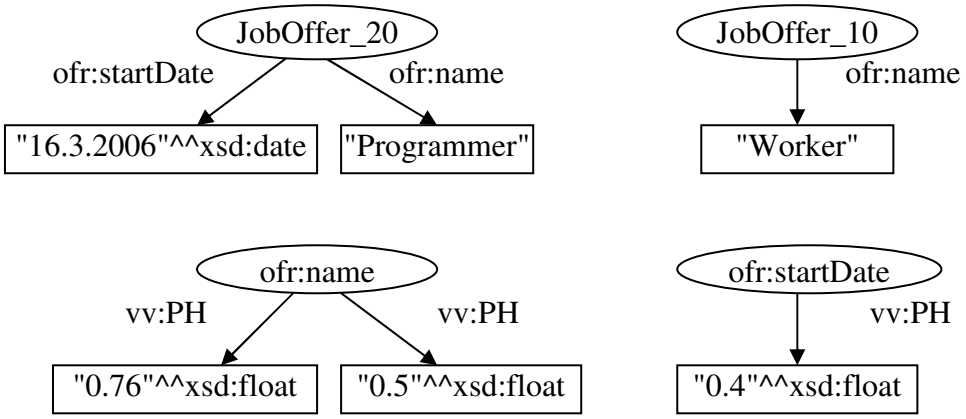
Problém vznikne, ak bude mať subjekt priradených viac predikátových hrán. Potom bude mať aj viac hodnôt ph a nevieme určiť, ktorá pravdivostná hodnota patrí ktorému predikátu. Zrejme nie je správne priradiť pravdivostnú hodnotu subjektu, lebo sa skôr viaže na predikát. Predikáty majú svoje identifikátory URI a tie môžeme dať na miesto subjektu a vytvárať o nich trojice. Nech RDF databáza obsahuje tieto tri trojice vo formáte Turtle:

```
JobOffer_20  ofr:name  "Programmer" .
JobOffer_20  ofr:startDate  "16.3.2006"^^xsd:date .
JobOffer_10  ofr:name  "Worker" .
```

Teraz chceme predikátom priradiť pravdivostné hodnoty. Do databázy pridáme ďalšie tri trojice:

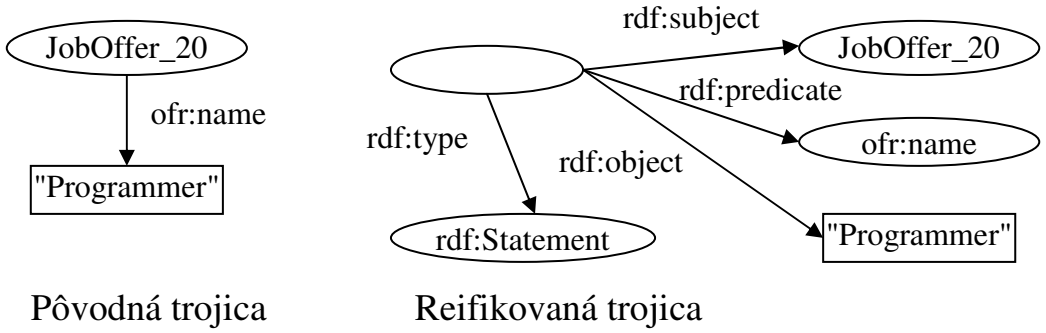
```
ofr:name  vv:PH  "0.76"^^xsd:float .
ofr:startDate  vv:PH  "0.4"^^xsd:float .
ofr:name  vv:PH  "0.5"^^xsd:float .
```

Pozrime sa na výsledný graf. Subjekt ofr:name má teraz dve rôzne hodnoty vlastnosti vv:PH a opäť ich nevieme jednoznačne priradiť predikátom ofr:name.



Obrázok 18. Pravdivostná hodnota ako vlastnosť predikátu

Zdá sa, že ani subjekt, ani predikát nie sú správnymi kandidátmi na pridelenie pravdivostnej hodnoty. Najsprávnejšie by bolo priradiť ju celej trojici. RDF umožňuje tvoriť „tvrdenia o tvrdeniach“ pomocou reifikácie. Nazývajú sa tiež tvrdenia vyššieho rádu (Priebe, Schlager, Pernul 2003). Znamená to, že celú RDF trojicu vezmeme ako subjekt a tvoríme o nej ďalšie tvrdenia. Pre tento účel musíme zostrojiť model pôvodnej trojice. Postup znázorníme graficky kvôli jednoduchosti.

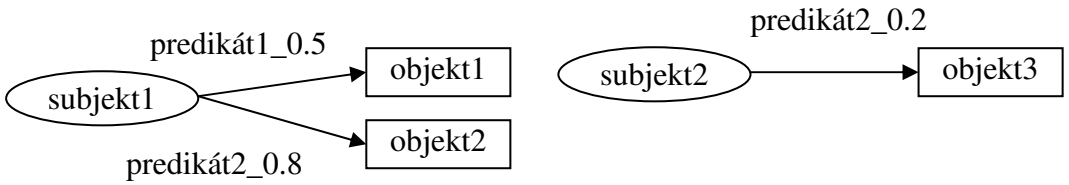


Obrázok 19. Príklad reifikácie

Ako subjekt vznikne prázdny uzol, ktorý má vlastnosti rdf:subject, rdf:predicate a rdf:object. Navyše je nutné uviesť, že nový uzol je typu rdf:Statement, teda ide o tvrdenie. Tomuto uzlu potom môžeme priradiť ďalšie vlastnosti, napríklad aj pravdivostnú hodnotu.

Reifikácia je súčasťou špecifikácií RDF, ale v niektorých publikáciách, napríklad (Powers 2003), sa jej použitie neodporúča. Príčinou sú pravdepodobne technické komplikácie. Reifikácia jedinej trojice má za následok pridanie štyroch trojíc, zložitú dátovú štruktúru a sťažené dopytovanie. Väčšina dopytovacích jazykov síce podporuje aj reifikáciu, ale dopytovanie prestáva byť triviálne. Keby sme teda chceli reifikovať všetky trojice, podstatne by to sťažilo celkové spracovanie dát. Priradenie ohodnotení cez reifikáciu je možné, ale značne neefektívne.

To, čo sa nedá urobiť dostatočne jednoducho na úrovni abstraktnej syntaxe, je možné na sémantickej úrovni. Do názvu každého predikátu pridáme vhodný oddeľovač a pravdivostnú hodnotu. Dostaneme napríklad takéto grafy:



Obrázok 20. Fuzzyfikácia na sémantickej úrovni

Sťaží to dopytovanie, lebo RDF bude považovať predikát2_0.8 a predikát2_0.2 za rôzne, ale to sa dá obísť napríklad použitím divokých kariet v dopytoch. Potom musíme zabezpečiť oddelenie pravdivostnej hodnoty a jej spracovanie pomocou vlastného programu.

5.2.3 Použitie fuzzy množín

Pokiaľ nechceme zasahovať do existujúcej štruktúry RDF, môžeme pracovať iba s unárnymi fuzzy predikátmi. Jedna hodnota v trojici je potrebná na uloženie pravdivostnej hodnoty fuzzy predikátu. Príkladom takého predikátu je dobre_platená(JobOffer_20).0.7 a zodpovedajúca RDF trojica <JobOffer_20, dobre_platena, 0.7>. Unárne fuzzy predikáty sú výsledkom aplikovania fuzzy množín či dopytov na dáta. Obsahujú informáciu o tom, nakoľko dáta vyhovujú pravidlu alebo dopytu. Fuzzy množiny a ich databázovú reprezentáciu sme popísali v kapitole 5.1. Použijeme tu rovnaký príklad.
fuzzy_vysoky_plat = {<x,ch(x)>, x∈ R}, kde ch(x) = max(0, min(1, x/10000 - 1)).
Ak máme tabuľku plat so stĺpcami ID a plat, potom použijeme dopyt:

```
SELECT  ID, max(0, min(1, plat/10000 - 1))
FROM plat
```

Naše dáta nie sú uložené v relačnej SQL databáze, ale v RDF databáze. Namiesto tabuľky plat s dvomi stĺpcami máme RDF trojice, napríklad:

```
plat1  ofr:minAmount  "20000"^^xsd:float .
plat2  ofr:minAmount  "10000"^^xsd:float .
```

Dopyt musíme modifikovať pre dopytovací jazyk SeRQL.

```
SELECT id, max(0, min(1, plat/10000 - 1))
FROM {id} ofr:minAmount {plat}
USING NAMESPACE ofr = <http://www.fiit.stuba.sk/nazou/ontologies/offer#>
```

Tento dopyt však databáza nedokáže spracovať. Klauzula SELECT pripúšťa iba názvy premenných, ktoré sa vyskytujú ďalej za klauzulou FROM. Nie je to iba nedostatok jazyka SeRQL, ale všetkých doteraz vytvorených RDF dopytovacích jazykov. Či už budeme chcieť spracovať fuzzy dáta alebo presné dáta pomocou fuzzy množín, v každom prípade potrebujeme používať agregačné funkcie v dopytoch.

Problém vyriešime nasledovne: použijeme databázu Sesame a vyberieme z nej údaje, ktoré chceme ohodnotiť fuzzy množinami a spracovať agregačnou funkciou. Operácie, ktoré RDF databáza nepodporuje, doplníme pomocou Java programu. Výsledky uložíme späť do databázy Sesame. To je úlohou praktickej časti tejto diplomovej práce.

5.2.4 Zhrnutie

Fuzzyfikácia RDF nie je jednoduchá, pretože tento jazyk je príliš voľný a neposkytuje možnosť vytvárať nové integritné obmedzenia tak ako relačné databázy. Ak obmedzenia nevyhnutne potrebujeme, musí ich kontrolovať iný program, ktorý sa spustí napríklad pri každom pridávaní a odoberaní dát.

RDF trojiciam môžeme priradiť pravdivostné hodnoty najmenej tromi spôsobmi. Prvým je reifikácia, kde vytvoríme model pôvodnej trojice a môžeme mu pridávať nové vlastnosti. Tento spôsob odporúčame vtedy, ak pravdivostnú hodnotu má iba malý počet trojíc. Jeho nevýhodou je zložitá dátová štruktúra. Druhý spôsob je „prilepenie“ pravdivostnej hodnoty za každý predikát. Je to jednoduchšie a prehľadnejšie. Nevýhodou je, že ďalšie spracovanie pravdivostných hodnôt musí riešiť opäť nejaký program.

V praktickej časti sa budeme venovať tretej možnosti fuzzyfikácie. Nebudeme ohodnocovať každú RDF trojicu, ale ohodnotíme zdroje (pracovné ponuky) podľa ich vlastností s použitím fuzzy množín a agregačných funkcií. Výsledkom budú unárne fuzzy predikáty, napríklad dobrá_práca(JobOffer_1).0.9, ktoré vieme transformovať na RDF trojice JobOffer_1 dobra_praca "0.9"^^xsd:float .

6 Praktická časť

Praktická časť je rozdelená na podkapitoly: Analýza štruktúry dát, Prípady použitia, Diagramy tried a Možnosti rozšírenia programu. Časť analýzy sa nachádza aj v kapitole 5.2, kde sa zaoberáme možnosťami fuzzyfikácie RDF. Tu sa analýza bude týkať RDF databázy pracovných ponúk a vlastností, ktoré budeme skúmať, ďalej vytvorených tried v Jave a spracovania dát.

V kapitole 6.1 sa podrobne venujeme dátam, v kapitole 6.2 popíšeme použitie programu a jednotlivé fázy spracovania dát. Kapitola 6.3 obsahuje diagramy tried rozdelené do niekoľkých skupín kvôli prehľadnosti. Navyše uvádzame sekvenčné diagramy na ilustráciu behu programu. V kapitole 6.4 hodnotíme dosiahnuté výsledky a časti programu, ktoré sa dajú ďalej rozvíjať a zdokonaľovať.

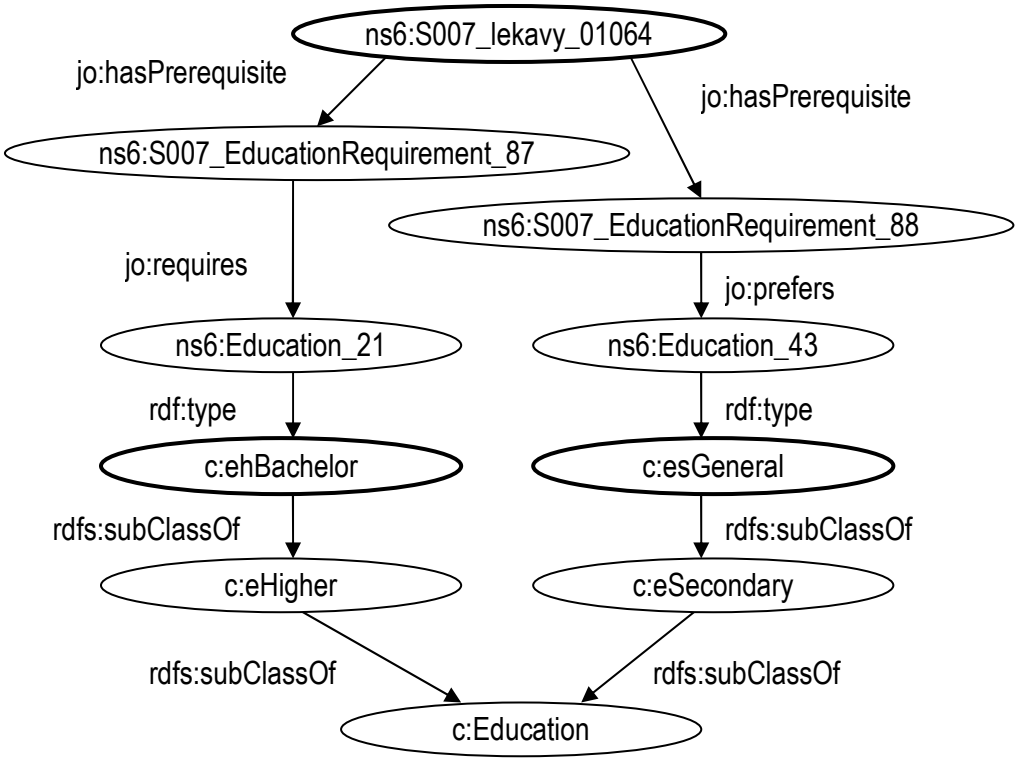
6.1 Analýza štruktúry dát

Táto diplomová práca aj software sú súčasťou projektu „Nástroje pre získavanie, organizovanie a udržovanie znalostí v prostredí heterogénnych informačných zdrojov“. Z projektu pochádza aj ontológia pracovných ponúk, ktorú tu používame ako dáta. Pozrieme sa bližšie na štruktúru týchto dát a ich vlastnosti.

Ontológia je vytvorená programom Protégé (1), ale RDF dáta z nej sa dajú štandardným spôsobom načítať do databázy Sesame. V Sesame môžu byť dáta fyzicky uložené v súboroch na disku, v operačnej pamäti alebo v databáze MySQL. Logicky však ukladáme dáta do *schránky* (*repository*). Každá operácia v Sesame sa deje v schránke, či už ide o pridávanie dát alebo dopyty. Pre každú schránku je daný názov, miesto, kde sa dáta fyzicky nachádzajú, ako sú indexované, či je povolené odvodzovanie a ďalšie vlastnosti. Kvôli jednoduchosti používame schránku s názvom „native-rdf-db“, čiže implicitne definovanú schránku, ktorá ukladá dáta na disk.

Keď máme pripravené dáta, môžeme písať dopyty na získanie zaujímavých vlastností, ktoré ohodnotí užívateľ. V príkladoch v tejto práci sa niekoľkokrát

vyskytli vlastnosti ako napríklad plat, ktorý je dôležitou vlastnosťou každej pracovnej ponuky. Teraz uvedieme ďalšie vlastnosti. Z objektívnych príčin nasledujúce obrázky neukazujú úplný graf, ktorý by obsahoval desiatky uzlov a hrán. Vyberáme len vzorové časti grafu na ilustráciu vybraných vlastností. Uzol, ktorý sa v nasledujúcich grafoch nachádza hore (alebo vľavo), v koreni grafu, reprezentuje identifikátor pracovnej ponuky. Obsahuje poradové číslo ponuky, podčiarovníky a meno osoby, ktorá spracovala danú ponuku do ontológie. Niektoré uzly v grafe sú tiež označené poradovými číslami a reťazcom „Education“, „Salary“ alebo iným podľa toho, akú entitu uzol označuje. Čísla sa vzťahujú k poradiu vytvárania uzlov v ontológii.



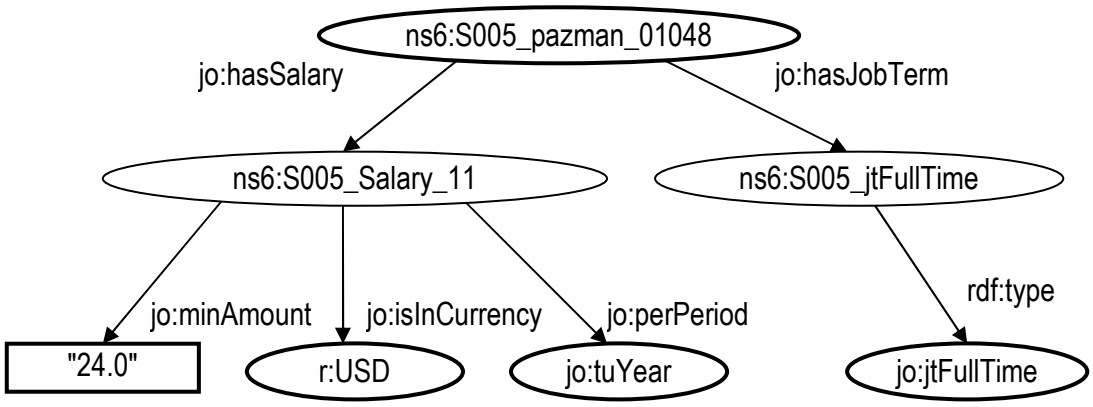
Obrázok 21. Požadované vzdelanie

Pracovná ponuka v tomto grafe má vlastnosti hasPrerequisite, teda požiadavky pre uchádzača. Na tretej úrovni v grafe sú uzly pre požadované vzdelanie, ktoré môže byť rôznych typov. Na štvrtej úrovni je potom typ vzdelania. Ak chceme z grafu vybrať len dvojice identifikátor ponuky a typ vzdelania, na obrázku naznačené hrubým orámovaním, použijeme dopyt:

```
select distinct ponuka,localName(uroven)
from {ponuka} jo:hasPrerequisite {podmienka} p {vzdelanie} isInstanceOf {uroven}
isSubclassOf {tr1} isSubclassOf {tr2}
where (tr1 = c:Education) or (tr2 = c:Education)
```

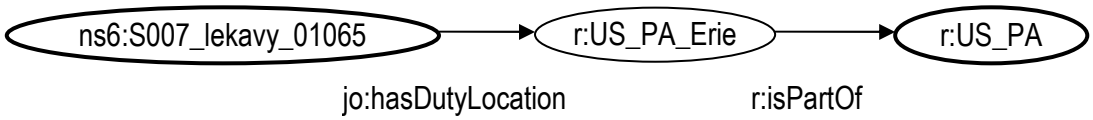
Už sme spomenuli plat, pri ktorom máme údaje ako výška platu, obdobie za ktoré plat dostávame a mena, v ktorej ho dostaneme. Ďalšie dôležité vlastnosti sú miesto práce, požadovaná prax v odbore, typ pracovnej pozície a to, či práca zahŕňa časté cestovanie. Všetky vlastnosti uvádzame na príkladoch RDF grafov a pod každým grafom uvádzame dopyt v jazyku SeRQL, ktorý tieto údaje vyberie vo forme tabuľky. Teda výsledkom dopytu nie je daný podgraf z obrázku, ale iba hodnoty

vlastností, ktoré nás zaujímajú. Na obrázkoch sú zvýraznené príslušné uzly, ktoré sú výsledkom dopytov. Z dopytov vynechávame klauzuly USING NAMESPACE, ktoré by sa zbytočne opakovali.



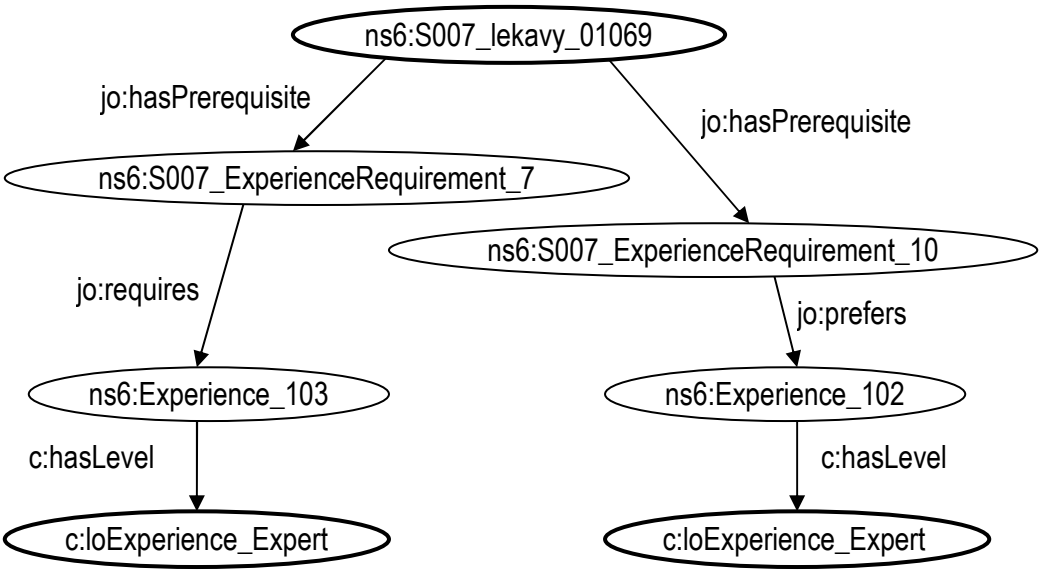
Obrázok 22. Plat, mena a obdobie a typ úväzku

```
select distinct Ponuka, label(Min), localName(Mena), localName(Interval), localName(Type)
from {Ponuka} jo:hasSalary {Plat} jo:minAmount {Min}; jo:isInCurrency {Mena},
[{Plat} jo:perPeriod {Interval}],
[{Ponuka} jo:hasJobTerm {x} rdf:type {Type}]
```



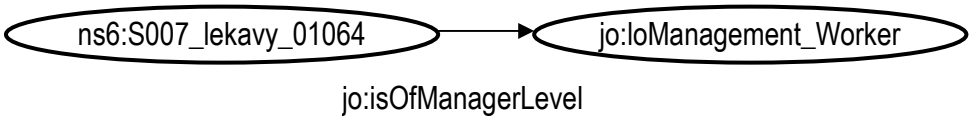
Obrázok 23. Krajina a štát

```
select distinct Ponuka, label(krajina)
from {Ponuka} jo:hasDutyLocation {miesto} r:isPartOf {k} rdfs:label {krajina}
```



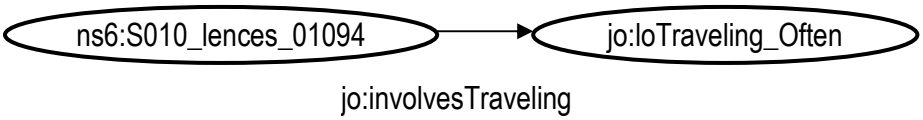
Obrázok 24. Požadovaná prax a úroveň praxe

```
select distinct ponuka, localName(uroven)
from {ponuka} jo:hasPrerequisite {podmienka} p {prax} c:hasLevel {uroven}
```



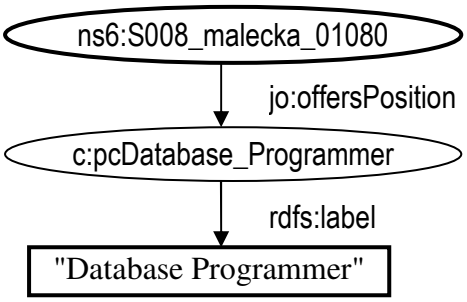
Obrázok 25. Pracovná pozícia

```
select distinct Ponuka, localName(Level)
from {Ponuka} jo:isOfManagementLevel {Level}
```



Obrázok 26. Cestovanie v rámci práce

```
select distinct Ponuka, localName(Travel)
from {Ponuka} jo:involvesTraveling {Travel}
```



Obrázok 27. Názov pracovnej pozície

```
select ponuka, label(meno)
from {ponuka}jo:offersPosition {pozicia} rdfs:label {meno}
```

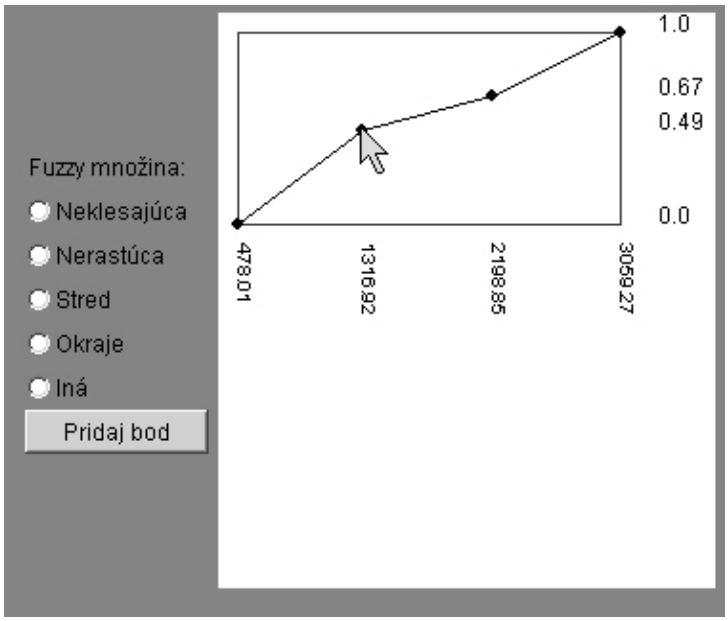
Je vidieť, že niektoré údaje sú v ontológii zadané celkom jednoducho v jednej RDF trojici. Ku iným sa musíme dostať cez viac úrovní a vetvenie grafu. Podľa toho sa mení aj zložitosť dopytov. Výsledkom uvedených dopytov sú tabuľky hodnôt, väčšinou obsahujú iba dva stĺpce, identifikátor ponuky a hľadanú vlastnosť. Výnimkou je dopyt, ktorý sa týka platu, tam potrebujeme aj údaje o mene, o období, pre ktoré platí daná suma a o tom, či je práca na plný alebo polovičný úväzok. Na výsledky dopytov (okrem údajov o mieste práce a názve, ktoré nevieme porovnať) chceme aplikovať fuzzy množiny a tak definovať užívateľské preferencie. V kapitole 5.2.3 sme však dospeli k záveru, že fuzzy množiny nie je možné vytvoriť v dopytovacom jazyku SeRQL. Preto bude s fuzzy množinami pracovať program. Jeho funkčnosť popíšeme v ďalšej kapitole.

6.2 Prípady použitia

Úlohou programu je ohodnotiť pracovné ponuky z RDF databázy Sesame podľa užívateľských požiadaviek. Požiadavky sú zadané ako fuzzy množiny pre vybrané vlastnosti spolu s agregačnou funkciou. Program vypíše výsledky alebo ich uloží do RDF databázy. Na zadávanie fuzzy množín poskytne grafické rozhranie.

Prvý krok je načítať dáta z databázy Sesame do programu. Na to použijeme knižnice príkazov Sesame org.openrdf.sesame.*, ktoré zabezpečia prihlásenie do databázy, odoslanie dopytu a načítanie výsledkov. Samozrejme, databáza musí byť vytvorená a naplnená dátami a potrebujeme poznať jej URL adresu. Po načítaní transformujeme dáta na vhodný tvar. Podľa fuzzy množím zadaných užívateľom dáta zotriedime a na výsledky aplikujeme agregačnú funkciu, ktorú tiež zadá užívateľ. Pre každú pracovnú ponuku dostaneme ohodnotenie z intervalu [0,1], ktoré vyjadruje, nakoľko táto ponuka vyhovuje užívateľovi podľa zadaných kritérií.

Ako agregačnú funkciu použijeme vážený priemer, užívateľ teda musí zadať váhy. Prípad zadávania fuzzy množín však nie je taký jednoduchý. Preto použijeme grafické rozhranie. Fuzzy množiny tam budú zadané tak ako na nasledujúcom obrázku. Bude možné vybrať typ množiny (nerastúca, neklesajúca, stred, okraje) a body sa budú dať posúvať myšou. Takéto ovládanie je intuitívne, užívateľ nemusí vedieť, čo vlastne sú fuzzy množiny. Stačí si uvedomiť, že čím bližšie k jednotke je ohodnotenie, tým je hodnota vlastnosti lepšia.

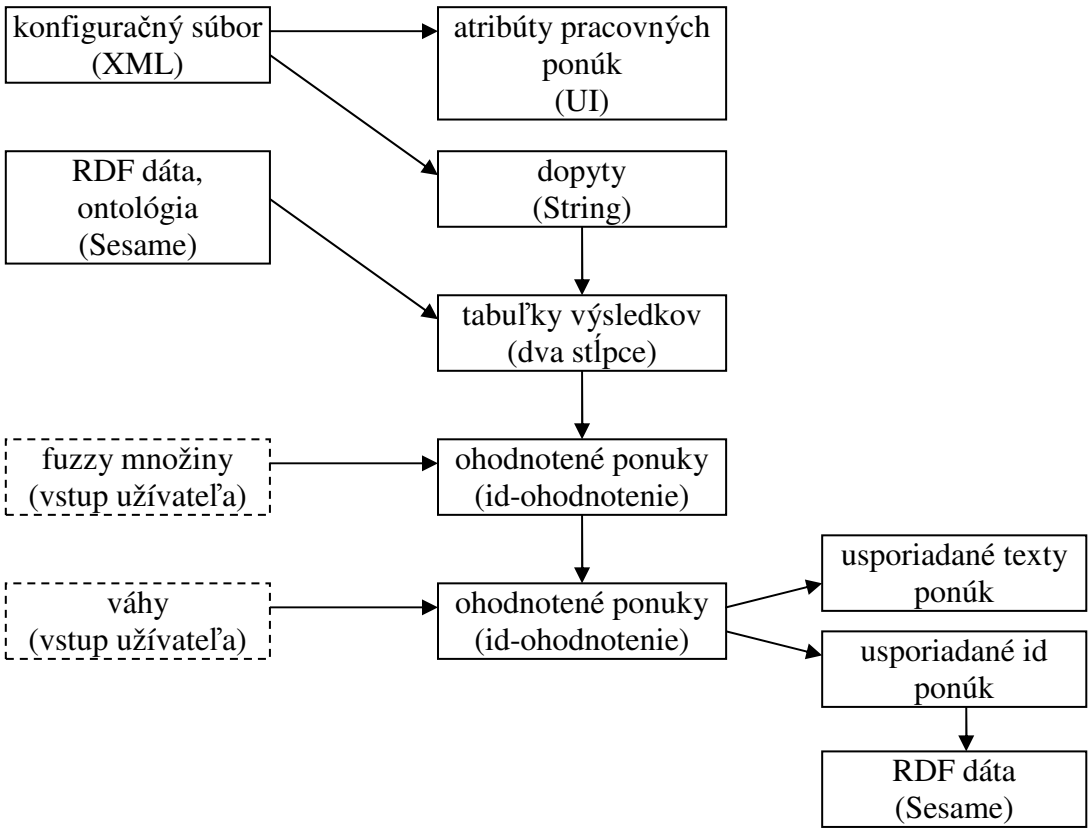


Obrázok 28. Rozhranie pre zadávanie fuzzy množín

Program by tiež mal navrhnúť pre každú vlastnosť implicitnú fuzzy množinu a nechať užívateľovi možnosť ju zmeniť. Štyri spomenuté typy fuzzy množín (popísané na konci kapitoly 5.1) zachovávajú usporiadanie hodnôt charakteristickej funkcie. Napríklad typ „nerastúca“ sa nesmie dať zmeniť tak, aby hodnoty rástli. Pre prípad, že by žiadna z týchto štyroch fuzzy množín nevyhovovala, program bude obsahovať možnosť vytvoriť vlastnú fuzzy množinu bez obmedzení na monotónnosť.

Užívateľ môže ohodnotiť sedem vybraných vlastností, ktoré sme popísali v predchádzajúcej kapitole. Aby sme zachovali všeobecnosť programu, vlastnosti nie sú dané pevne. Nachádzajú sa v konfiguračnom súbore vo formáte XML spolu s dopytmi a ďalšími informáciami. Tento súbor teda obsahuje akúsi schému, štruktúru dopytov a dát, ktoré používame. Ak by sme chceli použiť inú ontológiu alebo by sa zmenila jej štruktúra (čo sa v priebehu písania tejto práce stalo dvakrát), stačí zmeniť konfiguračný súbor. Program sa dá použiť bez zmeny. Je teda značne nezávislý od použitých dát. Podobne by sme postupovali vtedy, ak by sme chceli z RDF dát využiť iné vlastnosti, atribúty. V tejto práci skúmame tie najbežnejšie, ale existuje oveľa viac možných atribútov, ktoré ovplyvňujú rozhodovanie užívateľa. Mohli by sme napríklad brať do úvahy vzdialenosť miesta práce od bydliska, fyzickú náročnosť, rôzne výhody poskytované zamestnávateľom a podobne.

Beh programu schematicky znázorňuje nasledujúci obrázok, na ktorom vidieť, ako sa spracúvajú RDF dáta a vstupy užívateľa. Vstupy, dáta a konfiguračný súbor sú na ľavej strane obrázku. Stredný stĺpec obsahuje objekty, ktoré počas behu programu vytvoríme a pracujeme s nimi. Atribúty pracovných ponúk ovplyvnia aj grafické rozhranie programu. Napríklad užívateľ má možnosť definovať fuzzy množiny len pre tie atribúty, ktoré sú v konfiguračnom súbore a len tie atribúty sa zobrazia v programe. Vpravo sú výstupy programu, ktoré sa buď zobrazia užívateľovi, alebo vložia do databázy.



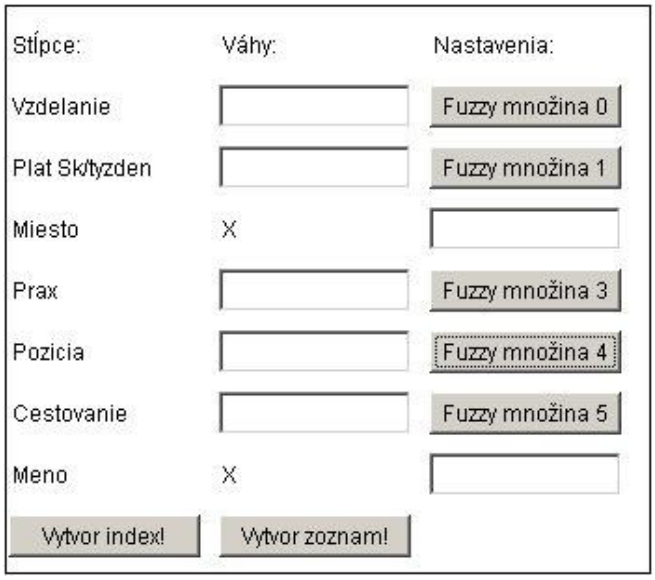
Obrázok 29. Spracovanie dát a užívateľských vstupov

Ako vidieť, program má jediný hlavný prípad použitia, a to ohodnotenie pracovných ponúk podľa zadaných požiadaviek. Program by mal byť prístupný ako applet na web stránke. Po spustení vykoná nasledujúce kroky:

1. Spracuje konfiguračný súbor, z neho získa zoznam atribútov a príslušné dopyty.
2. Pripojí sa na databázu Sesame a pomocou dopytov získa hodnoty atribútov.
3. Pre každý atribút (napríklad plat, požadované vzdelanie) zobrazí v okne appletu v jednom riadku názov atribútu, textové pole pre zadanie váhy a tlačidlo pre zadanie fuzzy množiny. Ak sa hodnoty atribútu nedajú vzájomne porovnať, zobrazí sa miesto tlačidla textové pole, kde môže užívateľ zadať hľadanú hodnotu atribútu.
4. V okne appletu sa navyše zobrazí grafické rozhranie na zadávanie fuzzy množín a textové okno na výsledky.
5. Po stlačení niektorého tlačidla označeného „Fuzzy množina“ program zobrazí na grafickej ploche príslušnú fuzzy množinu. Plocha umožní zmeniť typ množiny, presúvať a pridávať aktívne body, prípadne vytvoriť vlastný typ fuzzy množiny.
6. Po stlačení tlačidla označeného „Vytvor index!“ alebo „Vytvor zoznam!“ program vypočíta agregačnú funkciu pre všetky ponuky. Do úvahy pritom berie tie atribúty, ktoré majú zadané váhy v textových poliach v okne appletu. Ak pre takýto atribút nezadal užívateľ fuzzy množinu, použije sa implicitná.
7. Vypíše výsledky. Ak užívateľ klikol na tlačidlo „Vytvor index!“, applet zobrazí iba identifikátory pracovných ponúk zoradené podľa výsledného

ohodnotenia. Ak klikol na tlačidlo „Vytvor zoznam!“, applet zobrazí zoradené texty pracovných ponúk.

8. Ak program vytvoril index, zapíše identifikátory ponúk a ohodnotenia do databázy Sesame ako nové trojice.



Obrázok 30. Rozhranie pre zadávanie váh

Obrázok ukazuje, ako vyzerá okno appletu z bodu 3. V prvom stĺpci sú názvy atribútov z konfiguračného súboru. Hodnoty všetkých atribútov okrem atribútu Miesto a Meno vieme usporiadať (napríklad „ProjectManager“ je vyššia pozícia ako „TeamLeader“, 5 rokov praxe je viac ako 2 roky a podobne). Preto pre všetky atribúty okrem miesta máme k dispozícii tlačidlá, ktoré zobrazia rozhranie z obrázku 28 na vstup fuzzy množiny. V prostrednom stĺpci sú textové okná pre váhy. Pre atribút Miesto alebo Meno máme jediné textové pole, kde môžeme zadať požadovanú vlastnosť. Váha by tu nemala zmysel, lebo pracovná ponuka danú hodnotu buď má, alebo nemá. Ak teda do textového okna pre Miesto zadáme napríklad „California“, obmedzíme množinu výsledných ponúk. Či už máme ďalšie požiadavky na iné atribúty alebo nie, do výsledkov sa dostanú len tie ponuky, ktoré majú pre Miesto hodnotu „California“. Samozrejme, pokiaľ nezadáme do textového okna pre Miesto alebo Meno nič, program ohodnotí všetky pracovné ponuky. Nie je potrebné vždy definovať požiadavky pre všetky atribúty.

6.3 Diagramy tried

Diagram tried rozdelíme na viac častí podľa účelu a súvislostí jednotlivých tried. Poradie sa približne riadi obrázkom 29. Najskôr popíšeme triedy, ktoré súvisia s konfiguráciou programu, potom triedy, ktoré vyberajú alebo uchovávajú dáta z databázy Sesame. Nakoniec špecifikujeme triedy grafického rozhrania – fuzzy množiny dvoch typov, plochu pre zadávanie fuzzy množín, listenery a samotný applet, ktorý toto všetko spája dohromady. Triedy uvedené v diagramoch sú vytvorené v rámci tejto práce. Ak obsahujú diagramy iné triedy či rozhrania, uvádzame pri nich aj balík, napríklad java.io.Serializable.

6.3.1 Atribúty pracovných ponúk

Pozrieme sa na XML konfiguračný súbor a na triedy, ktoré s ním súvisia. Súbor má meno config.xml. Vonkajším elementom je <attributes>, ktorý obsahuje ľubovoľný počet elementov <attribute>. Tie reprezentujú atribúty pracovných ponúk. Uvedieme schému DTD tohto súboru:

```
<!ELEMENT attributes (attribute*)>
<!ELEMENT attribute (select, column*)>
<!ATTLIST attribute
        id CDATA #REQUIRED
        type ("computable"|"crisp") #REQUIRED
        name CDATA #REQUIRED
        value CDATA #REQUIRED>
<!ELEMENT select (#PCDATA)>
<!ELEMENT column (assignment+)>
<!ATTLIST column
        num CDATA #REQUIRED
        variable CDATA #REQUIRED
        type ("fuzzy"|"number") #REQUIRED
        null CDATA #REQUIRED>
<!ELEMENT assignment (#PCDATA)>
<!ATTLIST assignment
        level CDATA #REQUIRED
        evaluation CDATA #REQUIRED>
```

Je vidieť, že každý element <assignment> obsahuje jeden <select> a niekoľko <column> podľa toho, koľko stĺpcov je výsledkom dopytu uvedeného pod <select>. Hodnota value je vzorec, pomocou ktorého vypočítame výslednú hodnotu. Väčšinou je to iba „x“, ak nepotrebujeme nič počítať. Hodnota type je „computable“, ak sa dajú výsledné hodnoty porovnať, inak je „crisp“. Napríklad ak použijeme dopyt:

```
select distinct Ponuka,localName(Travel)
from {Ponuka} jo:involvesTraveling {Travel}
```

dostaneme dva stĺpce, ale URI ponuky tu nerátame, teda ostáva jeden. Ale pri nasledujúcom dopyte máme štyri stĺpce okrem URI identifikátora ponuky a musíme z nich vyrátať výslednú hodnotu platu:

```
select distinct Ponuka, label(Min), localName(Mena), localName(Interval), localName(Type)
from {Ponuka} jo:hasSalary {Plat} jo:minAmount {Min}; jo:isInCurrency {Mena},
[{Plat} jo:perPeriod {Interval}],
[{Ponuka} jo:hasJobTerm {x} rdf:type {Type}]
```

Každý stĺpec potom popíšeme jeho poradovým číslom, premennou, ktorú použijeme vo vzorci, typom hodnôt a implicitnou hodnotou, ktorú nahradíme za hodnoty NULL. Typ hodnôt je „number“, ak stĺpec obsahuje čísla, alebo „fuzzy“, ak obsahuje reťazce, ktoré vieme nejako usporiadať. Napríklad stĺpec Interval z predchádzajúceho dopytu môže obsahovať hodnoty „tuYear“, „tuMonth“, „tuWeek“, „tuDay“, „tuHour“, teda obdobie rok, mesiac, týždeň, deň a hodina, usporiadané podľa časovej dĺžky. Veľa stĺpcov z dopytov obsahuje takéto fuzzy hodnoty. Ich usporiadanie definujeme pomocou elementov <assignment>, kde

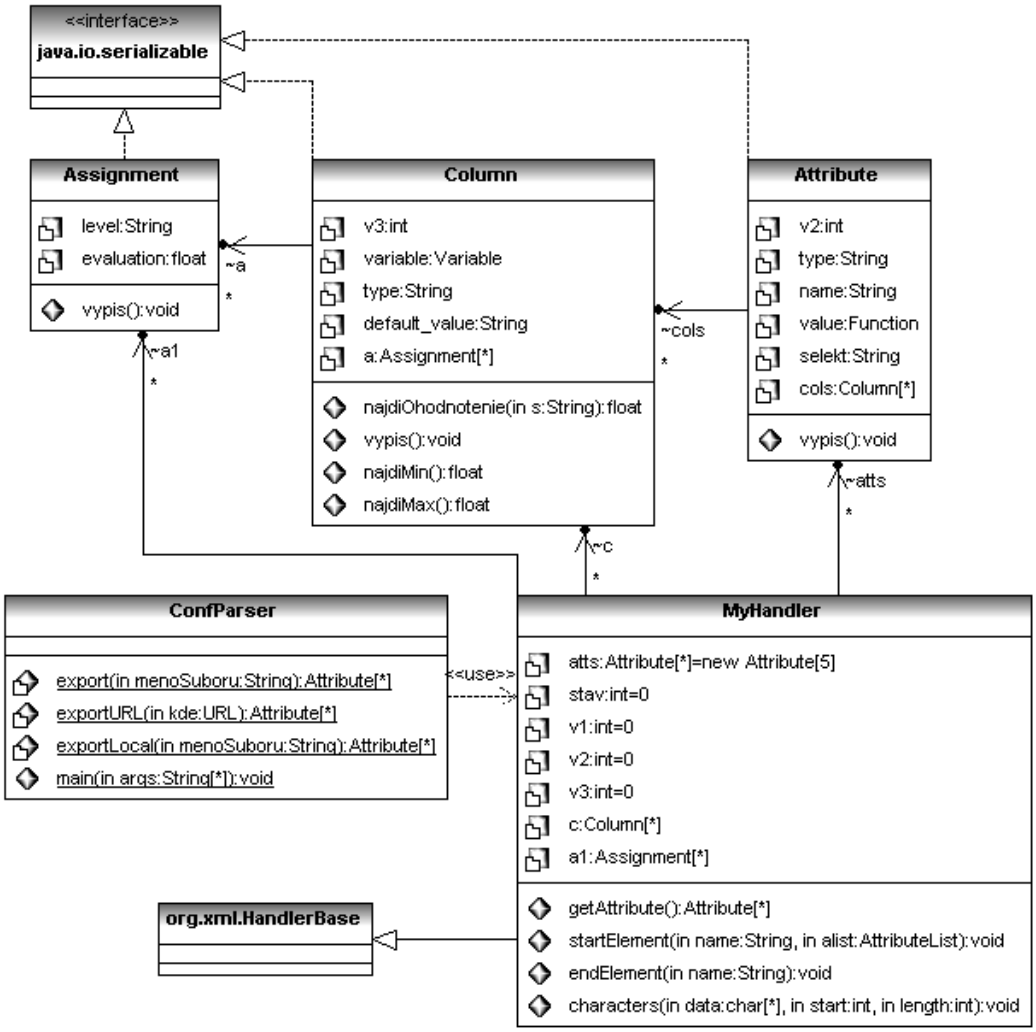
vlastnosť level obsahuje reťazec a evaluation jeho ohodnotenie. Uvedieme tu časť konfiguračného súboru ako príklad:

```
<attribute id="2" type="computable" name="MinSalary" value="(a*b*d)/c">
        <select><![CDATA[select distinct localName(Ponuka), label(Min), localName(Mena), localName(Interval), localName(Type) from {Ponuka} jo:hasSalary {Plat} jo:minAmount {Min}; jo:isInCurrency {Mena}, [{Plat} jo:perPeriod {Interval}], [{Ponuka} jo:hasJobTerm {x} rdf:type {Type}] using namespace jo = <http://nazou.fiit.stuba.sk/nazou/ontologies/v0.6.15/offer-job#>]]>
        </select>
        <column num="1" variable="a" type="number" null="1" />
        <column num="2" variable="b" type="fuzzy" null="1">
                <assignment level="USD" evaluation="32" />
                <assignment level="GBP" evaluation="56" />
                <assignment level="EUR" evaluation="39" />
        </column>
        <column num="3" variable="c" type="fuzzy" null="52.3">
                <assignment level="tuYear" evaluation="52.3" />
                <assignment level="tuMonth" evaluation="4.28" />
                <assignment level="tuWeek" evaluation="1" />
                <assignment level="tuDay" evaluation="0.2" />
                <assignment level="tuHour" evaluation="0.025" />
        </column>
        <column num="4" variable="d" type="fuzzy" null="1">
                <assignment level="jtFullTime" evaluation="1" />
                <assignment level="jtPartTime" evaluation="0.5" />
                <assignment level="jtJobShare" evaluation="0.5" />
                <assignment level="jtOccasional" evaluation="1" />
        </column>
</attribute>
```

Konfiguračný súbor teda obsahuje dopyty, informáciu o počte stĺpcov, ktoré získame ako výsledok dopytu, ich možné hodnoty a usporiadanie týchto hodnôt. Podľa usporiadania potom vieme vytvoriť fuzzy množiny. XML súbor potrebujeme prečítať iba raz, pri spustení programu. Všetky informácie program transformuje na objekty, ktoré budeme ďalej využívať. Zvyšuje to nároky programu na pamäť, ale bolo by menej výhodné spracovať súbor vždy znovu, pretože ovplyvňuje detaily grafického rozhrania aj spracovania dát. Pokiaľ súbor obsahuje atribút s názvom „text“, tento atribút neuvidíme na rozhraní (uvedenom na obrázku 30), ale použije sa pri vypisovaní výsledkov na získanie textov ponúk.

Stromovú štruktúru XML zachováme v štruktúre vytvorených objektov. Trieda Attribute bude obsahovať pole objektov Column a tie zase pole objektov Assignment. Všetky vlastnosti XML elementov sa tiež nachádzajú v týchto triedach.

O spracovanie XML súboru sa stará trieda MyHandler, ktorú potom využíva trieda ConfParser.



Obrázok 31. Triedy, ktoré pracujú s konfiguračným súborom

6.3.2 Tabuľky výsledkov

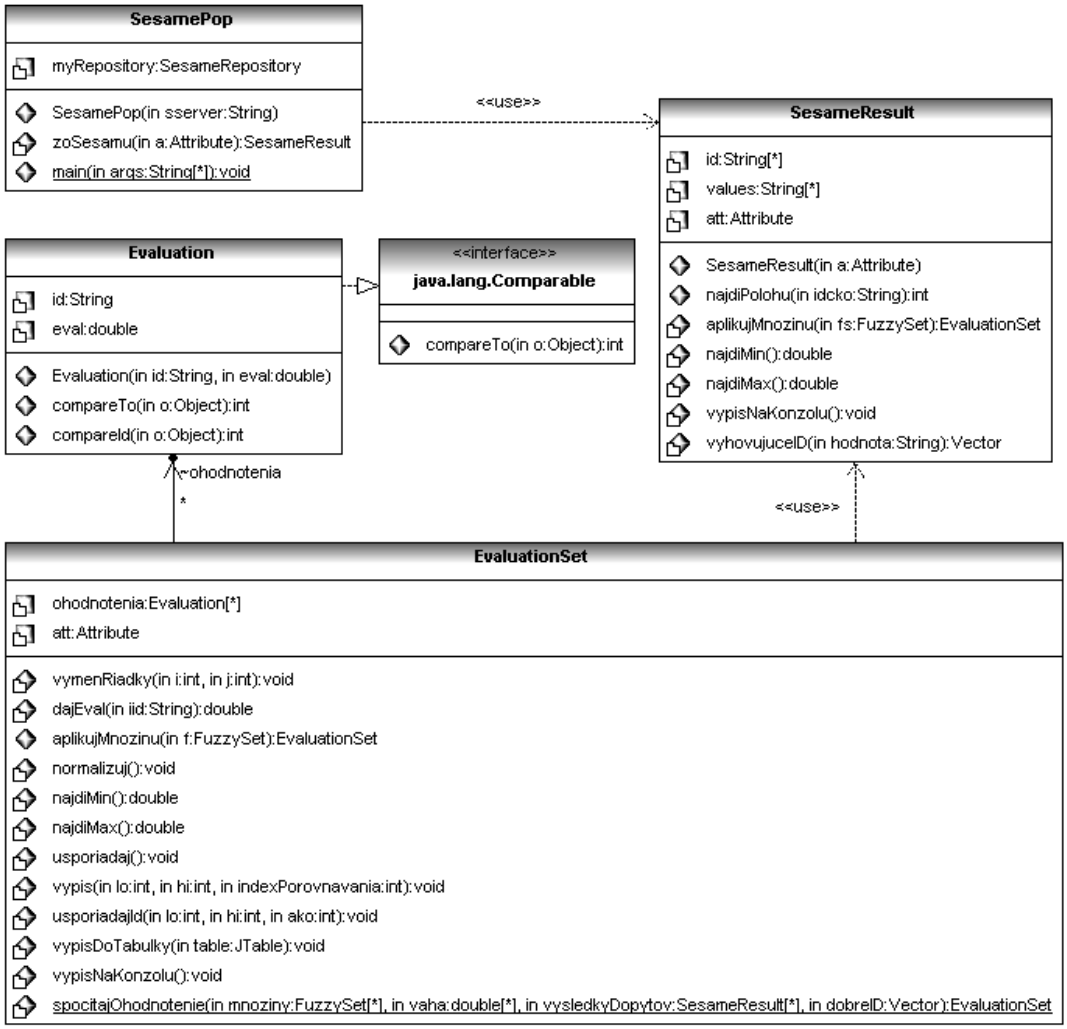
Každý získaný dopyt program odošle databáze pomocou triedy SesamePop a výsledky spracuje na vhodný formát. SesamePop využíva triedy balíka org.openrdf.sesame, ktorý slúži pre Java programy na prácu s databázou Sesame. Z tohto balíka používame len triedy SesameService, SesameRepository, QueryLanguage a QueryResultsTable. Najdôležitejšia je metóda SesameRepository.performTableQuery(QueryLanguage ql, String select). Pomocou nej získame výsledky dopytu ako objekt QueryResultsTable a spracujeme ich.

Výsledkom každého dopytu po spracovaní budú dva stĺpce, identifikátor ponuky a hodnota vlastnosti. Ak by dopyt vrátil viac stĺpcov, dostaneme z nich dva stĺpce po aplikovaní vzorca z objektu Attribute. Hodnota vlastnosti je číslo, reťazec alebo fuzzy hodnota. Napríklad:

S005_pazman_01048	ehBachelor
S005_pazman_01049	ehMaster

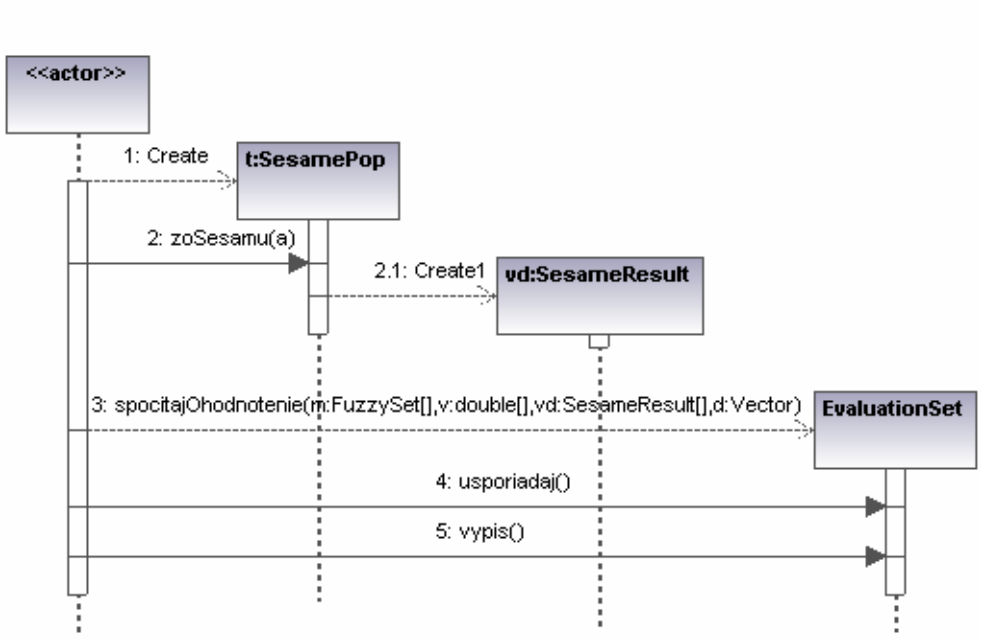
Tieto dva stĺpce vložíme do polí id a values triedy SesameResult. Najdôležitejšie metódy tejto triedy sú vyhovujuceID(String hodnota) a aplikujMnozinu(FuzzySet fs). Prvá

metóda vráti vektor identifikátorov id, ktoré majú danú hodnotu values. Druhá metóda vytvorí z každého riadku objekt Evaluation, a to tak, že na hodnotu poľa values aplikuje fuzzy množinu. O fuzzy množinách napíšeme ďalej. Výsledkom metódy bude objekt triedy EvaluationSet, ktorý okrem poľa typu Evaluation obsahuje odkaz na atribút, ktorého sa ohodnotenie týka. Implementujeme tu aj usporiadanie výsledkov, ktoré v Sesame chýba. Konečné spracovanie výsledkov zabezpečí metóda spocitajOhodnotenie(...).



Obrázok 32. Triedy, ktoré pracujú s databázou Sesame a uchovávajú výsledky

Nasledujúci obrázok ukazuje, ako používame uvedené triedy a metódy. Najskôr vytvoríme inštanciu triedy SesamePop, pomocou nej dostaneme objekty triedy SesameResult. Výsledkom metódy spocitajOhodnotenie(...) je objekt EvaluationSet, na ktorý môžeme použiť metódy usporiadaj() a vypis(). Metóda spocitajOhodnotenie() má jeden parameter, o ktorom sme sa zatiaľ nezmienili, a to mnoziny, pole objektov FuzzySet. Ide o fuzzy množiny, ktoré použijeme pri spracovaní výsledkov.



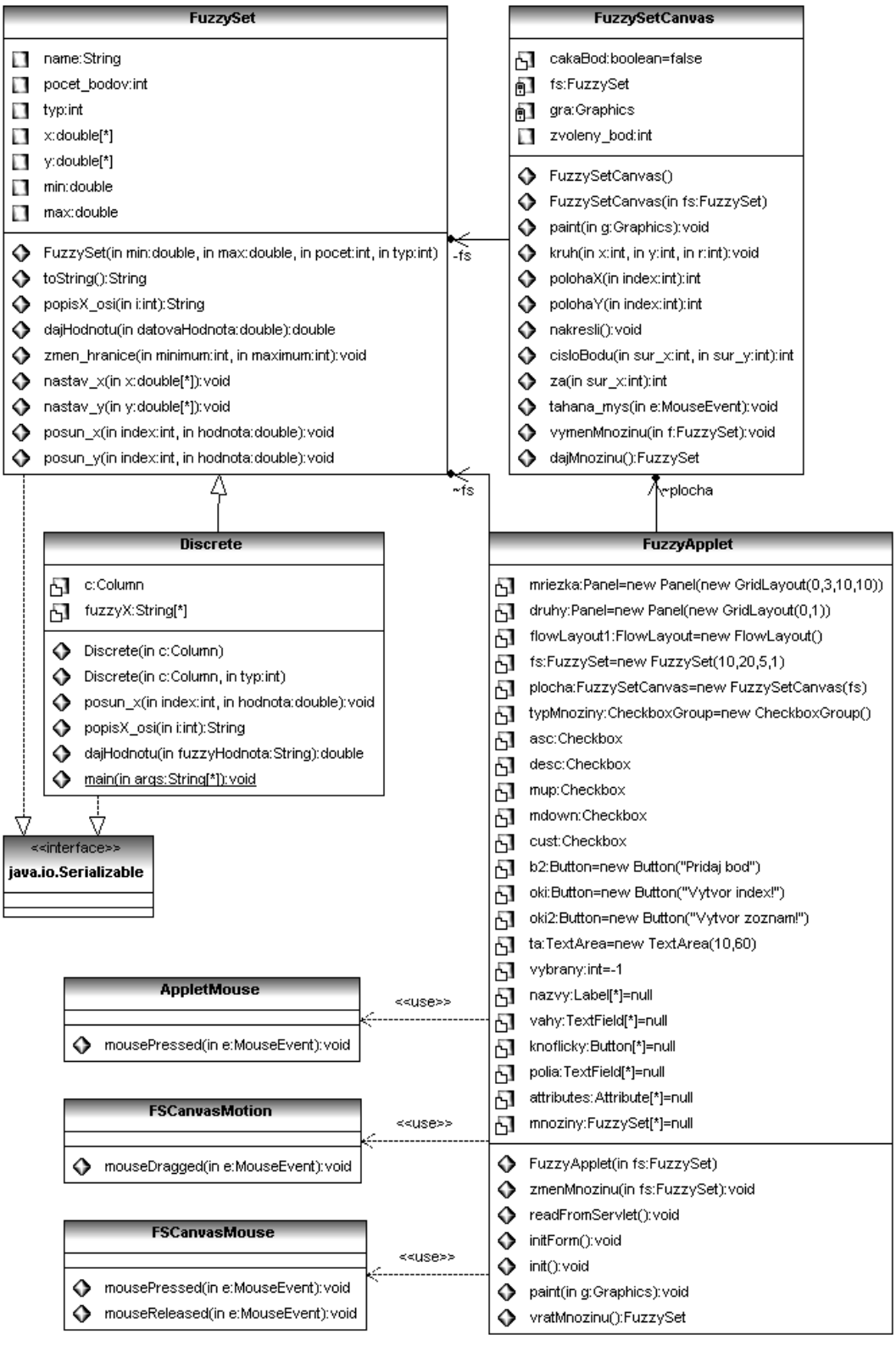
Obrázok 33. Spracovanie dát v programe

6.3.3 Spojité a diskrétne fuzzy množiny

Fuzzy množiny sú zadané ako objekty triedy FuzzySet. Obsahujú pole hodnôt x a y, názov, počet bodov, minimálnu a maximálnu hodnotu a typ. Číslo typu môže byť 0 pre neklesajúcu charakteristickú funkciu, 1 pre nerastúcu, 2 pre stred a 3 pre okraje. Iné číslo znamená ľubovoľný typ. Ak má množina niektorý z typov 0-3, musí zachovávať daný tvar charakteristickej funkcie, napríklad typ 0 nesmie klesať.

Trieda Discrete sa odlišuje od FuzzySet v niektorých podrobnostiach. Obsahuje premennú typu Column a definuje usporiadanie fuzzy hodnôt. Teda prvkami množiny tu nie sú čísla, ale reťazce ako „ehBachelor“, „ehMaster“, „ehPhD“, a pod.

FuzzySetCanvas je grafický komponent určený na zobrazovanie fuzzy množín. Triedy FSCanvasMouse a FSCanvasMotion zachytávajú udalosti ako kliknutie a ťahanie myši. Vďaka nim komponent reaguje na ťahanie aktívnych bodov množiny. Vzhľad tohto komponentu ukazuje obrázok 28. Trieda FuzzyApplet zobrazí túto grafickú plochu, textové okná na zadanie váh a tlačidlá na zadanie fuzzy množín pre každý atribút.



Obrázok 34. Triedy reprezentujúce fuzzy množiny a applet, ktorý ich využíva

6.3.4 Applet a servlety

Pôvodne mal celé spracovanie dát zabezpečovať FuzzyApplet. To však nebolo možné kvôli bezpečnostným obmedzeniam, aké platia pre všetky applety. Nemôžu sa pripájať k databáze ani čítať zo súborov, ale to všetko nevyhnutne potrebujeme. Program musí spracovať konfiguračný súbor aj načítať údaje z databázy. Potrebujeme teda nejako nahradiť obmedzenú funkčnosť appletov. Riešení je viac, hoci ani jedno nie je celkom triviálne.

Samotná databáza Sesame funguje ako JSP stránka pod Tomcatom. Pre JSP (Java Server Pages) a servlety neplatia bezpečnostné obmedzenia appletov. Nevýhodou je, že sú určené a prispôsobené predovšetkým na dynamické generovanie stránok a spracovanie údajov z HTML formulárov. V rámci servletu alebo JSP stránky nemôžeme priamo použiť komponent FuzzySetCanvas ani reagovať na stlačenie formulárového tlačidla vykreslením množiny. Všetky funkcie servletov sa musia vložiť do metód doGet(...) a doPost(...).

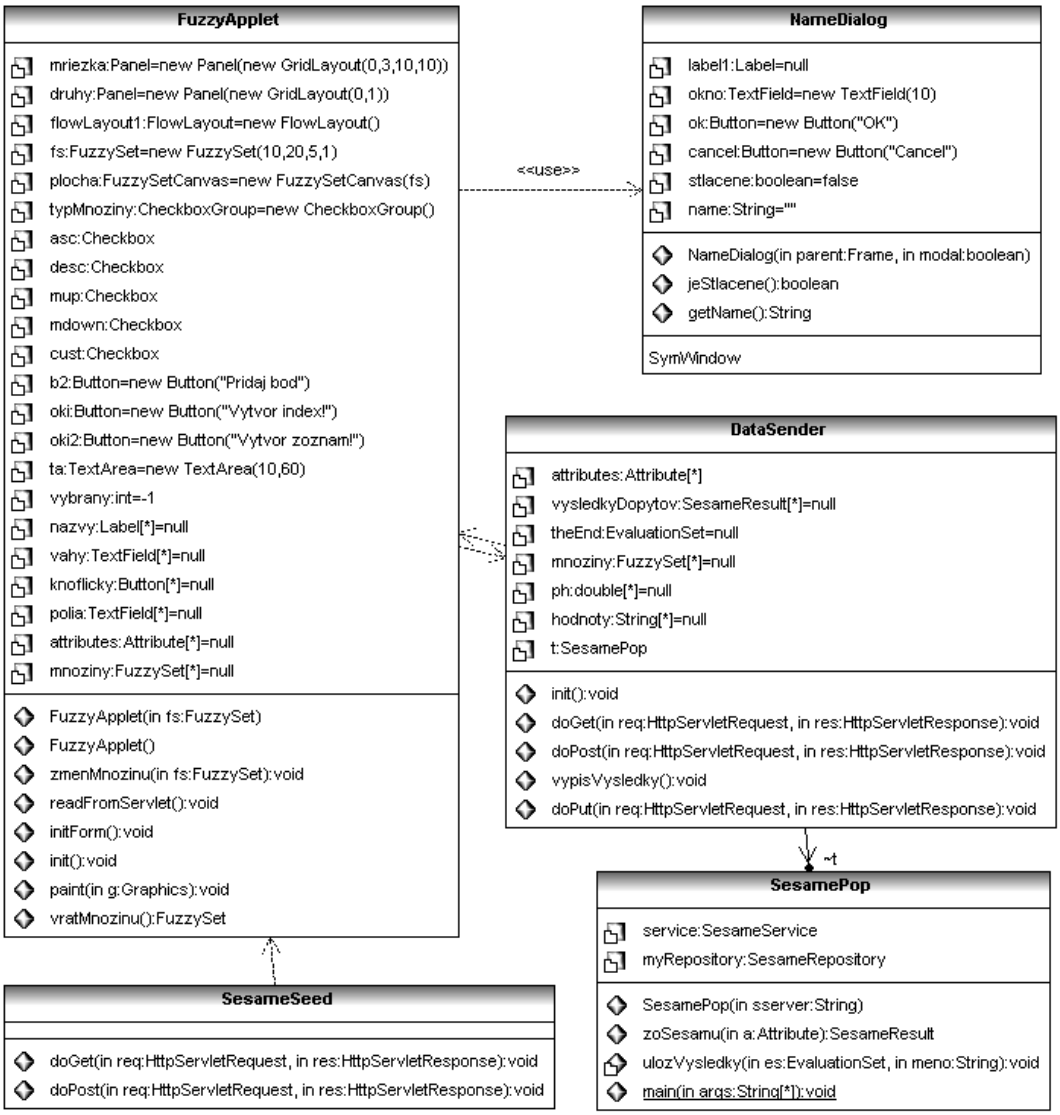
Problém sme vyriešili rozdelením zodpovednosti medzi applet a dva servlety. Servlet SesameSeed má na starosti grafické rozhranie. Pri načítaní príslušnej stránky v prehliadači tento servlet jednoducho zobrazí okno appletu FuzzyApplet. Ten pri svojom spustení volá servlet DataSender. Všetko, čo applet nemôže vykonať kvôli bezpečnostným obmedzeniam, prenecháme servletu DataSender. Ako naznačuje jeho názov, DataSender posiela appletu všetky potrebné dáta. Najskôr spracuje konfiguračný súbor, potom odošle dopyty databáze a spracuje výsledky a nakoniec pošle objekty Attribute a FuzzySet appletu. Na odosielanie a prijímanie objektov využívame triedu URLConnection z balíka java.net. Všetky objekty, ktoré chceme takto odosielať, musia implementovať rozhranie java.io.Serializable. Po zozbieraní všetkých užívateľských vstupov (váh, fuzzy množín alebo hodnôt atribútov) a po stlačení tlačidla „Vytvor index!“ alebo „Vytvor zoznam!“ applet zase odošle tieto vstupy servletu, ktorý ich spracuje a vráti appletu index ponúk alebo usporiadaný zoznam textov ponúk. Applet výsledky vypíše do textového okna. Ak chcel užívateľ vytvoriť index, tak sa ohodnotené ponuky uložia späť do databázy ako RDF trojice tvaru:

```
id_ponuky  http://s.ics.upjs.sk/~vanekova/evaluation#meno  ohodnotenie .
```

Meno, ktoré je súčasťou predikátu, si program vyžiada od užívateľa. Zobrazí jednoduché dialógové okno, ktoré obsahuje textové pole a tlačidlo. Ak užívateľ meno nezadá, výsledky sa neuložia. Subjektom v trojici je identifikátor ponuky a objektom ohodnotenie, ktoré program vypočítal. Tieto nové RDF trojice môže využiť iný program, prípadne sa môžu použiť ako indexy na zrýchlenie budúceho hľadania, čo však v programe (zatiaľ) nie je implementované. Trojice sa uložia do rovnakej schránky ako pôvodné dáta, ale vieme ich v prípade potreby ľahko oddeliť, pretože predikát má iný prefix ako všetky ostatné použité predikáty v dátach. Použili by sme dopyt:

```
select ponuka, localName(meno), label(ph)
from {ponuka} meno {ph}
where namespace(meno) like "http://s.ics.upjs.sk/~vanekova/evaluation#"
```

Nasledujúci obrázok ukazuje triedy (applety a servlety), ktoré v programe zabezpečujú vizuálnu stránku a využívajú funkcie ostatných popísaných tried.

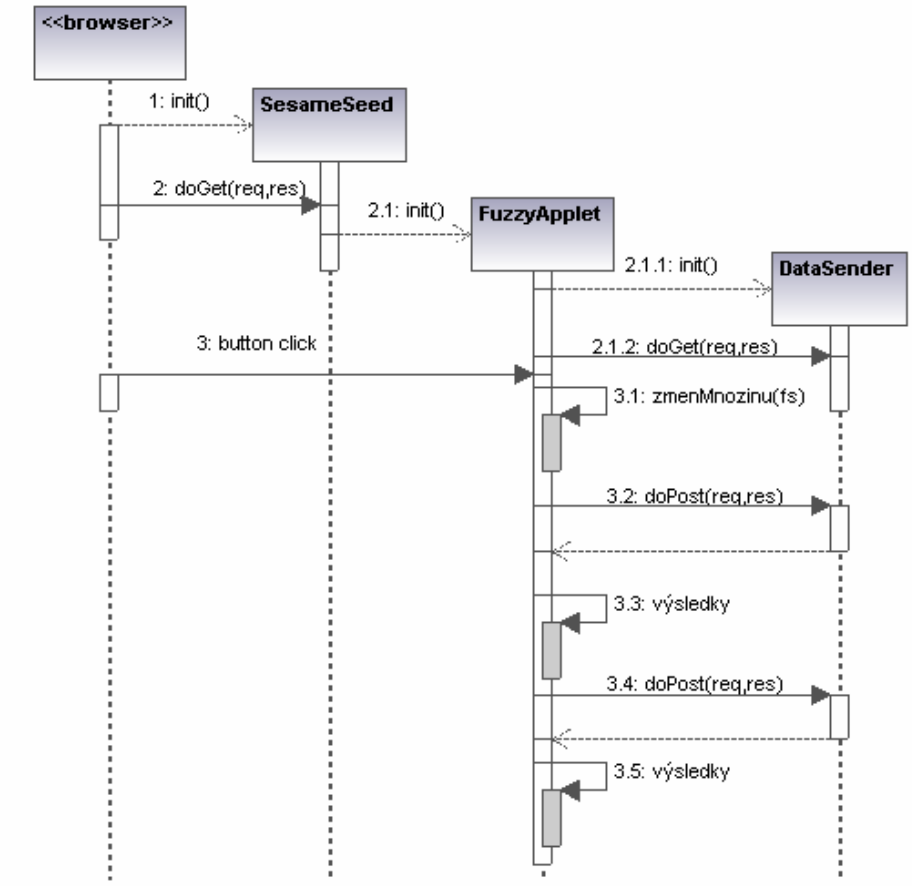


Obrázok 35. Servlety SesameSeed a DataSender

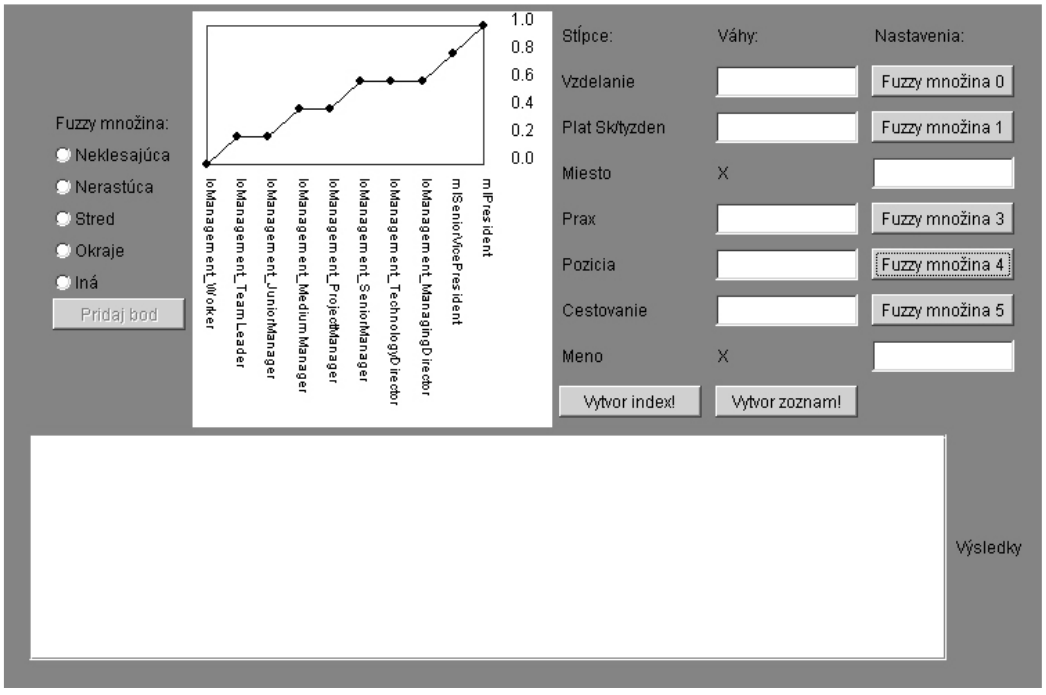
V prípade servletov sú ich najdôležitejšie metódy doGet(...) a doPost(...). Vysvetlíme ich základný význam. Metóda doGet(...) je volaná vždy, keď servlet odosiela dáta, teda aj vtedy, keď chceme servlet spustiť v okne prehliadača alebo keď s ním iný servlet či applet nadviaže spojenie a chce od neho čítať vstup. Ak servlet prijíma dáta od iného servletu alebo appletu, zabezpečuje to jeho metóda doPost(...). Tieto metódy majú dva argumenty, HttpServletRequest req a HttpServletResponse res, čo sú odkazy na volajúci servlet a volaný servlet.

Servlet SesameSeed má metódu doPost(...) prázdnu a nepotrebujeme ju nikdy volať, zatiaľ čo doGet(...) sa zavolá iba raz, keď v prehliadači napíšeme príslušnú adresu. Zobrazí stránku s appletom FuzzyApplet. Počas inicializácie potrebuje applet objekty tried FuzzySet a Attribute, ktoré dostane od servletu DataSender, teda musí zavolať doGet(...) iba raz. Metódu doPost(...) môže volať viackrát, kedykoľvek užívateľ stlačí jedno z tlačidiel slúžiace na zobrazenie výsledkov. Komunikáciu

medzi servletmi a appletom vysvetľuje obrázok 36. Na ďalšom obrázku je užívateľské rozhranie programu, ktorého jednotlivé časti sme už opísali.

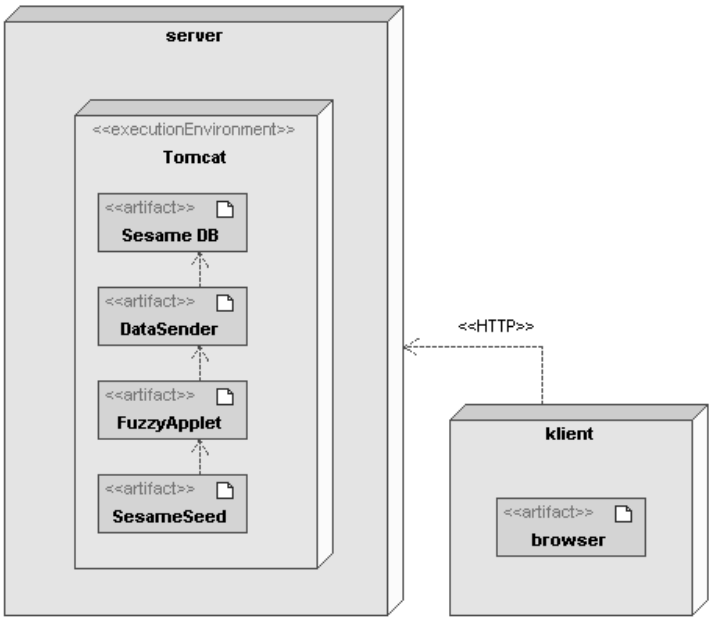


Obrázok 36. Postupnosť volaní servletov a appletu



Obrázok 37. Užívateľské rozhranie programu

Ako sme už spomenuli, servlety potrebujú web server, ktorý sa stará o ich spúšťanie. V tejto práci používame Tomcat, ktorý je nutný aj pre databázu Sesame. Náš výsledný softvér je rozdelený do niekoľkých adresárov, ktoré obsahujú skompilované triedy v Jave, zdrojové kódy, konfiguračný súbor a súbor web.xml, v ktorom sú názvy servletov pre Tomcat. Pomocou prehliadača vieme program spustiť lokálne alebo z iného počítača, ako ukazuje tento obrázok:



Obrázok 38. Diagram nasadenia

6.4 Možnosti rozšírenia programu

Program implementuje všetky plánované vlastnosti, ale obsahuje aj viacero možností rozšírenia. Napríklad sa dá využiť na indexovanie RDF dát. V tom prípade potrebujeme zadať na vstup fuzzy množiny a agregačné funkcie a na výstupe dostaneme zoradené (indexované) pracovné ponuky. Fuzzy množiny nemusia pochádzať priamo od užívateľa cez rozhranie, ktoré tu používame, ale môže sa ich naučiť nejaký systém induktívneho logického programovania (ILP) z testovacích príkladov. Tak sa dostávame od databázovej aplikácie k širšiemu využitiu vytvorených tried pri spracovaní dát.

Pre účely indexovania je potrebné zabezpečiť správu vytvorených indexov, napríklad ich zmenu alebo vymazanie pri zmene dát.

Tiež je možné vylepšiť užívateľské rozhranie podľa toho, aké skúsenosti by prinieslo testovanie v praxi. Napríklad pri vypisovaní výsledkov by mohli byť v tabuľke všetky vlastnosti ponuky, ktoré sme brali do úvahy, nie iba text ponuky. Dali by sa tiež pridať linky na pôvodné internetové stránky pracovných ponúk, aby si ich užívateľ mohol pozrieť v originálnom tvare, kde sú lepšie čitateľné ako v textovom okne appletu. Textové polia, ktoré slúžia na zadanie presnej hodnoty (napríklad pri atribútoch Miesto a Meno) je možné zmeniť na otváracie zoznamy, v ktorých budú všetky rôzne hodnoty daného atribútu z databázy.

Podobný program je možné vytvoriť nad databázou RDF Gateway (2). V tom prípade by sme použili dopytovací jazyk RDFQL, ktorý obsahuje viac funkcií oproti

jazyku SeRQL a tiež poskytuje knižnice v Jave pre spracovanie výsledkov dopytov. Môžeme využiť aj jazyk SPARQL. Zo všetkých vytvorených tried by stačilo zmeniť triedu SesamePop a mohli by sme využiť inú databázu a iný dopytovací jazyk.

7 Záver

V tejto práci sme skúmali možnosti spracovania RDF dát, konkrétne dopytovanie a ohodnocovanie pomocou užívateľských preferencií. V teoretickej časti sme podrobnejšie prepracovali označenia a dôkazy v oblasti sémantiky RDF. Dôležitým výsledkom je napríklad dôkaz vety o zmiešaní v kapitole 3.3. Zaoberali sme sa aj možnosťami reprezentácie fuzzy predikátov v RDF. Zo všetkých uvažovaných možností v kapitole 5.2 sme navrhli tri spôsoby fuzzyfikácie: pridanie pravdivostnej hodnoty na sémantickej úrovni v rámci označenia predikátu, využitie „tvrdenia o tvrdení“ (reifikácie) alebo obmedzenie sa výlučne na unárne fuzzy predikáty. Poslednú možnosť sme preniesli aj do praktickej časti. Ďalej sme skúmali ohodnotenie dát pomocou fuzzy množín a agregačných funkcií. Pri analýze súčasných RDF dopytovacích jazykov v kapitole 4.1 sme zistili, že majú viacero vážnych nedostatkov, predovšetkým absenciu triedenia dát a podpory matematických a iných funkcií v dopytoch. Tieto nedostatky sme obišli tak, že výsledky dopytov z databázy Sesame ďalej spracujeme v programe a ten implementuje chýbajúce vlastnosti.

V praktickej časti sme použili ontológiu pracovných ponúk, ale výsledný program SesameSeed sa dá použiť aj na iné RDF dáta. Nie je závislý od štruktúry dát, ktoré spracúva. Ak chceme použiť iné dáta, musíme zmeniť aj konfiguračný súbor vo formáte XML. Vstupom od užívateľa sú fuzzy množiny a váhy, pomocou ktorých program spočíta agregačnú funkciu a dostane výsledné ohodnotenie pracovných ponúk. Výstupom sú texty alebo len identifikátory pracovných ponúk zoradené podľa toho, ako vyhovujú užívateľským požiadavkám.

Program sa skladá zo servletov, appletu a ďalších tried, ktoré uchovávajú čiastočné výsledky dopytov, fuzzy množiny a atribúty, alebo počítajú ohodnotenia a pracujú s dátami. Cez internetový prehliadač sa dá spustiť hlavný servlet SesameSeed, ktorý pracuje pod serverom Tomcat. Predstavuje nadstavbu nad databázou Sesame a jeho hlavnou funkciou je vyhľadávanie v RDF dátach podľa užívateľských preferencií.

Literatúra

[Beckett 2004] Beckett, D.: *RDF/XML Syntax Specification (Revised).* W3C Recommendation. 10.2.2004.
URL <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>.

[Beckett 2006] Beckett, D.: *Turtle - Terse RDF Triple Language.* 4.4.2006.
URL <http://www.ilrt.bris.ac.uk/discovery/2004/01/turtle/>.

[Beckett, Barstow 2001] Beckett, D. – Barstow, A.: *N-Triples.* W3C RDF Core Working Group Internal Working Draft. 6.9.2001.
URL <http://www.w3.org/2001/sw/RDFCore/ntriples/>.

[Berners-Lee 2001] Berners-Lee, T.: *Notation3: An RDF language for the Semantic Web.* 27.11.2001.
URL <http://www.w3.org/DesignIssues/Notation3>.

[Berners-Lee, Fielding, Masinter 1998] Berners-Lee, T. – Fielding, R. – Masinter, L.: *RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax.* IETF, August 1998.
URL <http://www.isi.edu/in-notes/rfc2396.txt>.

[Biron, Malhotra 2001] Biron, P. V. - Malhotra, A.: *XML Schema Part 2: Datatypes.* W3C Recommendation. 2.5.2001.
URL <http://www.w3.org/TR/xmlschema-2/>.

[Broekstra, Kampman 2005] Broekstra, J. – Kampman, A.: *User Guide for Sesame.* 2005. URL <http://www.openrdf.org/doc/sesame/users/userguide.html>.

[Broekstra, Kampman, van Harmelen] Broekstra, J. - Kampman, A. - van Harmelen, F.: *Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema.* Springer Verlag, July 2002, s. 54-68. ISBN 3-540-43760-6.
URL <http://www.cs.vu.nl/~frankh/postscript/ISWC02.pdf>.

[Carroll 2003] Carroll, J. J.: *Syntax NP Complete – Hamiltonian Paths.* 2. April 2003. <www-webont-wg@w3.org>.
URL <http://lists.w3.org/Archives/Public/www-webont-wg/2003Apr/0003.html>.

[Geller] Geller, J.: *What is an ontology.*
URL <http://web.njit.edu/~geller/what_is_an_ontology.html>.

[Gruber 2003] Gruber, T.: *What is an ontology.* 2003.
URL <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>.

[Gurský 2006] Gurský, P.: *Algoritmy na vyhľadávanie najlepších k objektov bez priameho prístupu*. Proceedings of Znalosti 2006, s. 95-105. ISBN 80-248-1001-8.
URL <http://klud.ics.upjs.sk/~gursky/papers/2006znalosti.pdf>.

[Haarslev, Möller 2003] Haarslev, V. - Möller, R.: *Racer: An OWL Reasoning Agent for the SemanticWeb.* Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with 2003

IEEE/WIC, s. 91-95. 13. October 2003.
URL <http://www.sts.tu-harburg.de/papers/2003/HaMo03d.pdf>.

[Haase, Broekstra, Eberhart, Volz 2004] Haase, P. - Broekstra, J. – Eberhart, A. - Volz, R.: *A Comparison of RDF Query Languages.* Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004. Springer-Verlag, November 2004. ISBN 3-540-23798-4.
URL <http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/>.

[Hayes 2004] Hayes, P.: *RDF Semantics.* W3C Recommendation. 10.2.2004.
URL <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>.

[Klyne, Carroll 2004] Klyne, G. – Carroll, J. J.: *Resource Description Framework (RDF): Concepts and Abstract Syntax.* W3C Recommendation. 10.2.2004.
URL <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.

[McGuiness, van Harmelen 2004] McGuiness, D. L. - van Harmelen, F.: *OWL: Web Ontology Language Overview.* W3C Recommendation. 10.2.2004.
URL <http://www.w3.org/TR/owl-features/>.

[Powers 2003] Powers, S.: *Practical RDF.* O'Reilly & Associates, USA, Júl 2003. ISBN 0-596-00263-7.

[Priebe, Schlager, Pernul 2003] Priebe, T. - Schlager, C. - Pernul, G.: *A Search Engine for RDF Metadata*. Proceedings of the DEXA 2004 Workshop on Web Semantics, September 2004.
URL <http://www-ifs.uni-regensburg.de/PDF_Publikationen/PrSP04.pdf>.

[Prud'hommeaux, Seaborne 2006] Prud'hommeaux, E. – Seaborne, A.: *SPARQL Query Language for RDF.* W3C Candidate Recommendation. 6. April 2006.
URL <http://www.w3.org/TR/rdf-sparql-query/>.

[Vaneková, Bella, Gurský, Horváth 2005] Vaneková, V. – Bella, J. - Gurský, P. – Horváth, T.: *Fuzzy RDF in the Semantic Web: Deduction and Induction.* Proceedings of the WDA 2005, s. 16-29. Elfa Academic Press, Košice. Jún 2005. ISBN 80-8086-015-7.

[Vojtáš 1998] Vojtáš, P.: *Fuzzy logické programovanie*, 1998.
URL
<http://s.ics.upjs.sk/~novotnyr/home/skola/deduktivne_a_znalostne_systemy/fuzzy.pdf>.

[Vojtáš 2001] Vojtáš, P.: Fuzzy logic programming. *Fuzzy sets and systems*, 124/2001, s. 361–370.

[Vojtáš 2005] Vojtáš, P.: *Funkcionálne a logické programovanie: Poznámky z prednášok.* 27.2.2003.
URL <http://klud.ics.upjs.sk/~gursky/vyuka/vyl/vyl.pdf>.

[1] URL <http://protege.stanford.edu/>.

[2] URL <http://www.intellidimension.com/>.

[3] URL <http://www.mozilla.org/rdf/doc/>.

[4] URL <http://www.dublincore.org/>.

Prílohy

Príloha A (CD)

CD obsahuje štyri adresáre s názvami Diplomová práca, Použitá literatúra, Inštalácia a Software. Prvý obsahuje text diplomovej práce, druhý tie citované publikácie a stránky, ktoré sú dostupné v elektronickej forme. Označujeme ich rovnako ako v texte diplomovej práce. Adresár Inštalácia obsahuje inštalačné súbory pre Javu a Tomcat, ktoré je potrebné nainštalovať v tomto poradí ešte pred skopírovaním programu. Postup inštalácie je podrobne uvedený v súbore „instalacia.pdf“. V adresári Software sa nachádza program vytvorený v rámci praktickej časti tejto práce.

Príloha B (Užívateľská príručka)

V príručke sa nachádza popis ovládania programu a vysvetlenie funkcie jednotlivých komponentov. Uvádzame tiež chyby, ktoré môžu nastať a ako sa dajú vyriešiť.