Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Rozšíření Sparkle o grafickou tvorbu a vizualizaci dotazů

Plzeň 2019 Bc. Martin Kružej

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2019

Bc. Martin Kružej

Abstract

Extension of Sparkle to graphical building queries

The thesis deals with design and implementation of an extension of Sparkle application, a tool for the creation and evaluation of SPARQL queries. The first section is an introduction, a description of relevant concepts including SPARQL query language. The second section focuses on comparing of existing SPARQL query visualization tools. In the practical section a new solution is designed for Sparkle application, runnable as an extension of Sparkle or independently using a modern browser. Designed and implemented solution addresses use cases of Sparkle users and the most usable designs of other existing editors at the same time, combining their concepts into one solution. The solution offers automated query generation by using gradual choosing of classes and their attributes, their filtration and aggregation and lastly sorting of used variables. The query is visualized to the user in the form of a graph. In the conclusion there is a discussion about the results and a comparison of created application with other existing solutions.

Abstrakt

Diplomová práce se zabývá návrhem a implementací rozšíření aplikace Sparkle, nástroje pro tvorbu a vyhodnocení dotazu v jazyce SPARQL. První část je úvod do problematiky, popis relevantních konceptů včetně dotazovacího jazyka SPARQL. Ve druhé části se nachází srovnání již existujících vizualizačních nástrojů SPARQL. V praktické části je navrženo nové řešení pro aplikaci Sparkle, spustitelné jak prostřednictvím samotné aplikace, tak nezávisle na ní v jakémkoliv prohlížeči. Navržené a implementované řešení adresuje případy užití uživatelů Sparkle a zároveň vychází z nejpoužitelnějších návrhů existujících nástrojů, kombinující jejich jednotlivé koncepty do jednoho nástroje. Řešení nabízí automatické generování dotazu postupným vybíráním tříd či jejich atributů, jejich filtraci a agregaci a následně řazení použitých proměnných. Dotaz je průběžně vizualizován v podobě grafu. V závěru je uvedena diskuse nad dosaženými výsledky a srovnání vytvořeného řešení s existujícími nástroji.

Obsah

1 Úvod

Sparkle je JavaFX aplikací pro vytváření a exekuci SPARQL dotazů. SPARQL jazyk je de facto standardem RDF a sémantických technologií. Sparkle nabízí formulářový a textový editor pro tvorbu dotazů. Oba tyto editory vyžadují od uživatele znalost SPARQL jazyka a jeho syntaxe. Pokud uživatel nezná nebo si není jistý schématem dat, vytvoření SPARQL dotazu mu bude dělat problém.

Vizualizace se ve světě RDF a sémantického webu vyskytuje již dlouhou dobu. Z počátku byla snaha vizualizovat RDF data jako taková, nejčastěji prostřednictvím SPARQL endpointů. S narůstající popularitou jazyka SPARQL a RDF vznikla snaha ulehčit vytváření SPARQL dotazování. Vznikly nástroje, jež si daly za cíl uživatele úplně od jazyka odpoutat a nahradit jej nějakou abstrakcí, kterou by postavily mezi uživatele a samotný jazyk. Tato abstrakce nejčastěji nabrala podobu vizualizace dotazu. Cílem této práce je srovnat dosud vytvořené nástroje a navrhnout rozšíření aplikace Sparkle o grafickou tvorbu dotazu pomocí znalostí získaných z analytického srovnání existujících nástrojů.

V první části práce je popsán RDF, SPARQL a všechny pojmy s tím související. Následuje srovnání nástrojů, jež se zabývaly vizualizací jak RDF dat, tak SPARQL dotazu. V praktické části je nejdříve navrženo rozšíření pro vizuální SPARQL dotazování, následně implementováno a otestováno. V závěrečné části je pak diskuse nad výsledky, srovnání s ostatními nástroji, některé možné nedostatky stávajícího řešení a zhodnocení splnění požadavků.

2 Sémantické technologie

Sémantické technologie a sémantický web je vize o internetu, kde všechna data jsou vzájemně provázaná a zároveň obsahují sémantickou informaci, jež jim dává význam - koncept **Linked data**[1–3]. Taková globální databáze je dobře čitelná hlavně strojově, kteří mohou lépe zkoumat tato data, integrovat je či navigovat v nich [4, 5]. Sémantické technologie si kladou za cíl umožnit lidem vytvářet datová úložiště na Webu, vytvářet slovníky a psát pravidla pro zacházení s daty [1]. K tomu slouží technologie RDF, SPARQL, OWL a SKOS, o kterých bude více řečeno dále. S vizí sémantických technologií přišel poprvé Tim Berners-Lee [2, 4].

2.1 Principy sémantických technologií

K sémantickému webu se váže několik hlavních principů [5]:

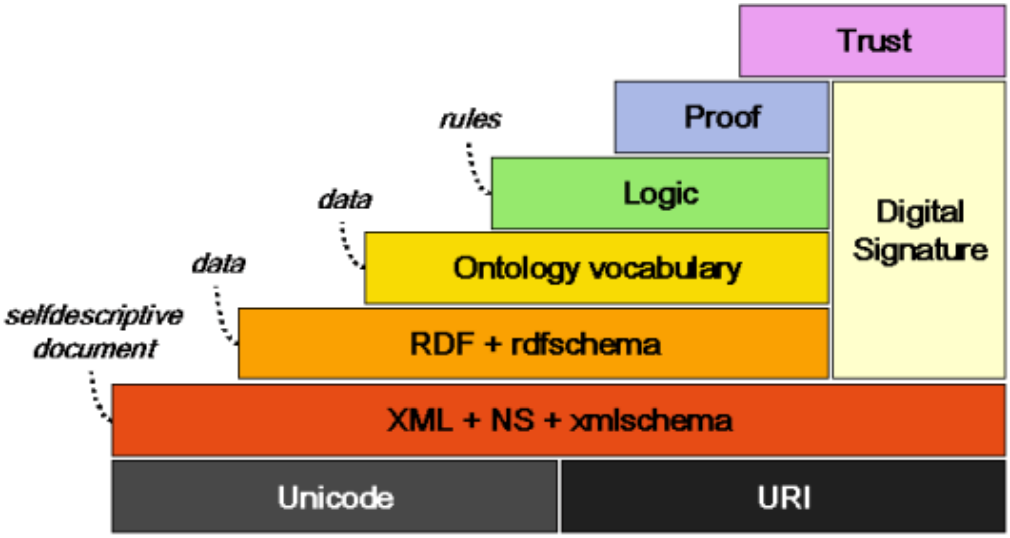
- **Vše je identifikovatelné svým URI** - lidé, místa a věci ve fyzickém světě mají v sémantických technologiích svůj identifikátor.
- **Zdroje a vazby mohou mít typy** - pro lepší interpretaci významu dat počítačem se ke zdrojům a jejich vzájemným vazbám váže typ.
- **Neúplná informace je tolerována** - vazby do neexistujících zdrojů nebo na jinou než původní adresu neznamená nefunkčnost sémantického webu, podobně jako tomu je ve stávajícím webu a jeho chybových stavech 404 a podobně.
- **Není třeba absolutní pravdivosti informací** - ve stávajícím webu není vždy vše pravdivé a v sémantickém je to stejné. Vyhodnocení pravdy je na aplikacích zpracovávajících web.
- **Evoluce je podporována** - definování podobných konceptů různými uskupeními je časté. Sémantické technologie umožňující kombinovat koncepty, řešení nejasností či nekonzistentností.
- **Minimalistický design** - sémantický web se snaží zachovat lehké věci lehkými a komplexní věci možnými. Standardizuje se pouze naprosté minimum a na těchto pevných základech pak staví jednoduché aplikace.

2.2 Vrstvy sémantických technologií

Principy sémantických technologií jsou implementovány do vrstev standardů, zobrazeny na obrázku 2.2. Unicode a URI vrstvy zajišťují použití mezinárodních znakových sad a poskytují způsob identifikace zdrojů na webu. XML vrstva obsahuje jmenný prostor a definice schémat a umožňuje integraci ostatních definic sémantických technologií s XML standardy. S RDF a RDFSchema vrstvami je možno formulovat tvrzení o objektech a definovat slovníky pomocí URI. Vrstva ontologie podporuje evoluci slovníků pomocí definování vazeb mezi různými koncepty. Digitální podpis vrstva detekuje změny v dokumentech.

Vrstvy na vrcholu, tedy logika, důkaz a důvěra, nebyly nikdy patřičně definovány a prozkoumány. Logika umožňuje psát pravidla, jež jsou využity důkazem a důvěrou, kteří společně zkoumají, jestli danému konceptu věřit či nikoliv [5, 6].

Obrázek 2.1: Vrstvy sémantických technologií [5]



O sémantické technologie se stará organizace W3C (World Wide Web Consortium) [7] a staví je na RDF [2, 4, 6].

3 Resource Description Framework

Resource Description Framework (dále RDF) je standardní model pro reprezentaci dat na webu [8–10]. Data jsou popsána definováním vztahů mezi datovými objekty [8]. Jde o velice flexibilní model - umožňuje migrovat data z několika zdrojů odpoutáním dat od jejich schématu [8, 10]. Několik různých schémat je tedy možno aplikovat, provázat a dotazovat jako by byly jedním schématem a nezměnit tím samotné instance dat [8]. Formát je postaven na dvou dalších webových standardech XML a URI [8, 9]. RDF se snaží naplnit vizi sémantického webu ve snaze definovat a provázat veškeré informace na webu do formální strojově čitelné podoby [8, 11].

3.1 Datový model

Datový model RDF se skládá ze dvou částí:

- **RDF graf** - Sada trojic (subjekt-predikát-objekt), vyjadřují popisy zdrojů.
- **RDF dataset** - organizuje kolekce RDF grafů a zahrnuje implicitní graf a nula a více pojmenovaných grafů [12].

Základní stavební jednotkou RDF jsou **trojice** (*triple*), jež tvoří RDF graf. Každá trojice je tvořena **subjektem**, **predikátem** a **objektem**. Trojici lze vyjádřit grafově jako uzel subjektu s orientovanou hranou predikát do uzlu objekt.

Obrázek 3.1: Trojice vyjádřená v podobě grafu

V obecnějším tvaru lze také zapsat jako:

<subjekt> <predikát> <objekt>

Pomocí trojic se potom dají definovat reálně věci ve světě, například:

<Bob>	<je>	<člověk>
<Bob>	<je přítel>	<Alice>
<Bob>	<má rád>	<čokoládu>
<Alice>	<žije v>	<Plzni>

Tento příklad je neformální pseudokód a slouží jako jednoduchá ukázka. Důležité je si všimnout, že zdroj může být referencován v různých trojicích. V tomto příkladu je Bob subjektem tří trojic a Alice objektem jedné trojice a subjektem jiné. Právě tato schopnost umožňuje vyhledávat vztahy mezi jednotlivými trojicemi [11].

RDF trojice ve své podstatě říká, že nějaký vztah, indikovaný predikátem, existuje mezi zdroji označenými jako subjekt a objekt. Toto tvrzení popisující RDF trojici se nazývá RDF tvrzení (RDF statement).

Uzly mohou být jednoho ze tří typů - IRI, literál a anonymní uzly [11, 12].

IRI a literály popisují nějakou věc ve světě. Říká se jim **zdroje** (*resources*). Zdrojem může být cokoliv - fyzické věci, dokumenty, abstraktní koncepty, čísla či řetězce [11].

3.1.1 IRI

Zkratka IRI znamená International Resource Identifier a jednoznačně identifikuje zdroj. Jedna z podob IRI je URL (Uniform Resource Locator), který je používán pro webové adresy. IRI jsou globální identifikátory, pokud na dvou místech je stejné IRI, odkazuje na tu samou věc [11]. IRI byl specifikován v RFC 3987 [13].

Příklad IRI, jež je v RDF často používáno pro vyjádření vztahů známosti mezi lidmi:

```
http://xmlns.com/foaf/0.1/knows
```

Obecně RDF nepřiřazuje IRI význam, ten je získán ze slovníků či konvencí.

IRI mohou být na všech třech místech v trojici [11].

3.1.2 Literál

Literály jsou hodnoty, které nejsou IRI. Příkladem můžou být řetězce jako „Počítač“, datumy jako „13. února 1990“ či prostá čísla jako „3.14159“. K literálům se váží datové typy, které jim umožňují být správně identifikovány a zpracovány. Dále existují řetězcové literály doplněné o jazykový tag, jež označují řetězce v přirozeném jazyce [11, 12].

Množství datových typů je definováno slovníkem XML Schema [14], ale uživatelé mohou definovat i vlastní datové typy.

Literál může být v trojici pouze na místě objektu [11].

3.1.3 Anonymní uzly

Anonymní uzly nepopisují žádný zdroj. Tvrzení obsahující anonymní uzly říkají, že něco s daným vztahem existuje, ale co explicitně nejmenují [12]. Mohou se objevit jako subjekt nebo objekt v trojici [11].

3.2 Slovníky

RDF slovníkem se nazývá kolekce IRI určených pro RDF grafy [12]. Slovníky často definují koncepty a vztahy (také se jim říká termy) popisující a definující danou zájmovou oblast. Slovníky klasifikují termy používané v určité aplikaci, charakterizují možné vztahy a definují limitace použití těchto termů [15].

IRI ve slovnících často začínají se stejným podřetězcem tzv. **IRI jmenné oblasti** (*namespace IRI*). Tyto jmenné oblasti se často konvenčně zkracují na krátké jméno tzv. **prefix jmenné oblasti** (*namespace prefix*) [12].

Příklad jmenných oblastí některých často používaných slovníků a jejich typických prefixů je uvedeno v tabulce 3.1.

RDF slovníky pomáhají s integrací dat ve chvíli, kdy existují nejasnosti mezi termy v datasetech nebo může pomoci s hledáním nových vztahů. Slovníky se často také používají na organizaci vědomostí jako jsou knihovny, muzea či noviny, jež spravují velké kolekce knih, článků apod [15].

W3C poskytuje několik technik pro popis a definici různých forem slovníků ve standardním formátu. Jde o *RDF* a *RDF Schema* [16], *Simple Knowledge Organization System (SKOS)* [17], *Web Ontology Language (OWL)* [18] a *Rule Interchange Format (RIF)* [19].

Tabulka 3.1: Časté slovníky a jejich jmenné prostory a prefixy

Prefix	**Jmenný prostor**	**Slovník**
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	RDF Vocabulary [16]
rdfs	http://www.w3.org/2000/01/rdf-schema#	RDF Schema [16]
skos	http://www.w3.org/2004/02/skos/core#	SKOS [17]
xsd	http://www.w3.org/2001/XMLSchema#	XML Schema [14]
owl	http://www.w3.org/2002/07/owl#	OWL [18]
foaf	http://xmlns.com/foaf/0.1/	FOAF [20]
dc	http://purl.org/dc/elements/1.1/	Dublin Core [21]

3.2.1 RDF Schema

RDF Schema [16] definuje kategorie pro klasifikace zdrojů pomocí zavedení notace **třída** (*class*) a **vlastnost** (*property*).

Třídy jsou skupiny zdrojů, jednotlivým členům tříd se pak říká **instance** těchto tříd. Třídy samotné jsou zdroje a často jsou identifikovány pomocí IRI a popsány za použití jejich vlastností.

Vlastnosti jsou relace mezi subjektovými a objektovými zdroji. Jsou definovány pomocí **domény** (*domain*) a **rozsahu** (*range*), čímž jsou vlastnosti v podstatě definovány pomocí tříd, ke kterým náleží.

Výčet některých konstrukcí slovníku je uveden v tabulkách pro třídy (3.2) a pro vlastnosti (3.3).

Tabulka 3.2: Výčet tříd RDF Schema

Název třídy	**Komentář**
rdfs:Resource	Třída zdrojů, tedy všeho
rdfs:Literal	Třída literálů
rdfs:Class	Třída tříd
rdf:Property	Třída RDF vlastností
rdfs:Datatype	Třída RDF datových typů
rdf:Statement	Třída RDF tvrzení

Tabulka 3.3: Výčet vlastností RDF Schema

Název vlastnosti	**Komentář**	**Doména**	**Rozsah**
rdf:type	Subjekt je instancí třídy	rdfs:Resource	rdfs:Class
rdfs:domain	Doména vlastnosti	rdf:Property	rdfs:Class
rdfs:label	Čitelný název subjektu	rdfs:Resource	rdfs:Literal
rdf:subject	Subjekt RDF tvrzení	rdf:Statement	rdfs:Resource

Pomocí slovníku je možné vytvořit hierarchii tříd, sub-tříd a vlastností a sub-vlastností [11, 16].

3.2.2 SKOS

SKOS [17] poskytuje model pro vyjádření základní struktury a obsahu znalostních organizačních systémů, jako tezaury, taxonomie, folksonomie, věcné katalogy a podobně. SKOS umožňuje vytvoření a publikování konceptů na web, slinkování s jinými daty a integrace s ostatními schématy konceptů [17, 22].

3.2.3 RIF

RIF [19] definuje standard pro výměnu pravidel mezi pravidlovými systémy. Nedefinuje jednotný vždy použitelný pravidlový jazyk v kontrastu s ostatními slovníky, namísto toho navrhuje rodinu jazyků (dialektů) s precizně specifikovanou syntaxí a sémantikou. Důraz je kladen na jednotné a rozšiřitelné dialekty [19, 23].

3.2.4 OWL

OWL [18], také označený od nové verze jako **OWL2**, je sémantický jazyk navržen pro reprezentaci bohatých a komplexních vědomostí o věcech, skupinách věcí a relací mezi věcmi [24]. Jinými slovy OWL vyjadřuje ontologie, v tomto smyslu myšleno jako sada přesných deskriptvních tvrzení o nějaké části světa - **zájmová doména** (*domain of interest*) nebo **sledovaná problematika** (*subject matter*).

OWL je postaven na základních notacích:

- **Axiomy** - základní tvrzení, které OWL vyjadřuje
- **Entity** - elementy referující skutečné objekty
- **Výrazy** - kombinace entit pro vytvoření komplexních popisů z jednoduchých

Pro explicitní vyjádření vědomosti OWL uvažuje, že je tvořeno ze základních částí označovaných jako **tvrzení** (*statements*). Příkladem je něco tak jednoduchého jako "Prší"nebo "Každý člověk je smrtelný". Každá vědomost je potom tvořena kolekcí takových jednoduchých částí. Tvrzením, z kterých se dělají ontologie, jsou potom **axiomy**, a mohou být pravdivé nebo nepravdivé.

OWL napodobuje lidské myšlení v tom, že z vědomostí dedukujeme následky, tedy tvrzení je následek jiného tvrzení, jinými slovy tvrzení je pravdivě pokud ostatní tvrzení jsou pravdivá. V OWL se tomu formálně říká, že sada tvrzení A vyplývá tvrzení a pokud existuje jakákoliv situace kdy jsou všechny tvrzení z A pravdivé, potom a je pravdivé. Sada tvrzení může být konzistentní (je možné dosáhnout pravdivé hodnoty) nebo nekonzistentní (není možné). Formální sémantika OWL potom v podstatě specifikuje pro jaké situace je určitá sada OWL tvrzení pravdivá.

OWL tvrzení často referují skutečné objekty ve světě a popisují je buď přidáváním do kategorií ("Marie je žena") nebo tím, že řeknou něco o jejich vztahu ("Jan a Marie jsou ženatí"). Všechny atomické části tvrzení, ať již objekty (Jan, Marie), kategorie (žena) nebo relace (ženatí), jsou **entity**.

Objekty se dále označují jako **jedinci** (*individuals*), kategorie jako **třídy** (*classes*) a relace jako **vlastnosti** (*properties*). Vlastnosti se dále dělí na **objektové vlastnosti** (*object properties*), spojující objekty s jinými objekty (osoba k jiné osobě), **typové vlastnosti** (*datatype properties*), přiřazující datové hodnoty objektům (věk k osobě) a **anotační vlastnosti** (*annotation properties*), používané na přidání informací o samotné ontologii (autor či datum vytvoření axiomu).

Názvy entit umožňuje OWL kombinovat do **výrazů** použitím tzv. konstruktorů. Například třídy "žena"a "profesor"mohou být zkombinovány a vytvořit popis třídy ženských profesorů. Výsledek by musel být popsán nějakým OWL výrazem pro třídu, jež by mohl být použít v jiných tvrzeních a výrazech. V tomto smyslu jsou výrazy podobné entitám s tím, že jsou definované svojí strukturou [18].

3.3 Formáty

Existuje několik různých serializovatelných formátů pro psaní a uložení RDF grafů. Jde o zápis těch samých dat různými způsoby, formáty jsou tedy ekvivalentní [11].

N-Triples

N-Triples [25] poskytuje řádkový plain-textový způsob serializace RDF grafu.

Turtle

Turtle [26] je rozšířením N-Triples, přinášející řadu syntaktických zkratek jako podpora prefixů jmenných prostorů, seznamy a zkratky pro datové literály.

TriG

TriG [27] je rozšířením Turtle o více grafů.

N-Quads

N-Quads [28] rozšiřuje N-Triples o možnost vzájemné výměny mezi RDF datasety. Formát umožňuje přidat čtvrtý element do řádky a tím k trojici přidává IRI grafu dané trojice.

JSON-LD

JSON-LD [29] poskytuje JSON syntaxi pro RDF, možnost transformovat JSON dokumenty do RDF s minimálními změnami, univerzální identifikátory pro JSON objekty, díky čemuž mohou na sebe JSON dokumenty navzájem odkazovat, a v neposlední řadě poskytuje datové typy a jazyky.

RDFa

RDFa [30] se používá na vnoření RDF dat do HTML a XML dokumentů.

RDF/XML

RDF/XML [31] poskytuje XML syntax pro RDF grafy.

4 SPARQL

Pro práci s datovým modelem RDF bylo nutné navrhnout a specifikovat nový dotazovací jazyk. Několik jazyků bylo W3C zvažováno, až v roce 2008 W3C doporučilo jazyk SPARQL [32] a jeho novější verzi 1.1 potom v roce 2013 [33]. SPARQL se tak v podstatě stal standardizovaným jazykem pro RDF.

Dotaz SPARQL je tvořen sadou šablon, jimž se říká **základní grafové šablony** (*basic graph patterns*). Jsou podobné trojicím RDF s tím rozdílem, že jednotlivé komponenty (subjekt, predikát, objekt) mohou být proměnné. Grafový vzorec se potom porovnává s dotazovanými daty postupným dosazováním do proměnných. Vzorec pasuje na RDF podgraf ve chvíli, kdy RDF termy (IRI, literály nebo prázdné uzly) mohou být dosazeny za proměnné a výsledný RDF graf je ekvivalentní k podgrafu [32, 33].

Výsledek dotazování je obecně seznam výsledků odpovídající způsobu jakým byly v grafu vzorce pasovány. Mohou být žádné, jeden nebo mnoho výsledků k dotazu [33].

4.1 Syntax

Uvedu výtah základní SPARQL syntaxe, dokument [33] uvádí úplnou syntax.

4.1.1 Syntax RDF termů

IRI

IRI mohou být prefixovány. Prefix je deklarován na začátku dotazu klíčovým slovem **PREFIX**. Prefix je tvořen z označení prefixu a lokální části, oddělené dvojtečkou. V rámci prefixu může být jedna z těchto částí prázdná.

Příklad deklarace prefixu:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

Použitím klíčového slova **BASE** můžeme to samé IRI definovat různými způsoby. Pokud v dotazu nebude BASE uveden, je použit implicitní Base URI dle RFC3986 [33, 34].

Následují tři zápasy stejného IRI:

```
<http://example.org/book/book1>
```

```
BASE <http://example.org/book/>
<book1>
```

```
PREFIX book: <http://example.org/book/>
book:book1
```

Literály

Obecná syntax pro literály je uvozování do dvojitých či jednoduchých uvozovek jako řetězce, následováno jazykovým tagem (@) nebo datovým typem (^^).

Z důvodu konvence mohou být čísla (*integery*) psány rovnou a jsou interpretovány automaticky jako literály typu *xsd:integer* nebo v případě použití tečky bez exponentu jako *xsd:decimal*, s exponentem potom *xsd:double*. Logické hodnoty typu *xsd:boolean* mohou být psány jako *true* a *false* [33].

Příklady literálů:

- *„chat“@fr*
- *„rdf“^^<http:example.org/exampleType>*
- *1*, což je stejné jako *„1“^^xsd:integer*
- *1.3*, což je stejné jako *„1.3“^^xsd:decimal*
- *true*, což je stejné jako *„true“^^xsd:boolean*

Proměnné

Proměnné dotazu se označují znakem „?“ nebo „$“, častější je použití „?“. Nepočítají se jako součást názvu proměnné. Příklad proměnné *$abc* a *?abc* identifikují tu samou proměnnou v rámci SPARQL dotazu [33].

Anonymní uzly

SPARQL anonymní uzly bere jako proměnné, nikoliv jako pouhé reference. Jsou indikovány označením „_:abc“ či zkrácenou formou „[]“. Zkrácená forma se používá pro ty anonymní uzly, které jsou použity pouze na jednom místě v dotazu. Daný uzel bude použit pro vytvoření grafového vzoru.

Často se používá konstrukce *[:p :v]*, která automaticky vytvoří anonymní uzel, jež je dále použit jako subjekt všech obsažených dvojic predikát-objekt. Takto vytvořený uzel může být nadále použit jako subjekt a objekt v dalších grafových vzorech, například na pozici subjektu [33]:

```
[ :p "v" ] :q "w" .
```

nebo objektu:

```
:x :q [ :p "v" ] .
```

4.1.2 Vzory trojic

Vzory trojic (*triple patterns*) odpovídají svým RDF protějškům a píší se jako subjekt, predikát a objekt. Příklad [33]:

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { <http://example.org/book/book1> dc:title ?title }
```

Predikát-objekt seznamy

Pokud dotaz obsahuje vzory, které využívají ten samý subjekt, je možno zápis zkrátit použitím notace „;“ [33]:

```
?x  foaf:name  ?name ;
    foaf:mbox  ?mbox .
```

Objekt seznamy

Pokud vzory sdílejí subjekt i predikát, je možno je separovat notací „,“:

```
?x foaf:nick  "Alice" , "Bob" .
```

Predikát-objekt a objekt seznamy mohou být kombinovány [33]:

```
?x  foaf:name ?name ; foaf:nick  "Alice" , "Bob" .
```

RDF Kolekce

RDF kolekce mohou být zapsány v rámci dotazu syntaxí „(element1 element2 ...)“. Používáno ve spojení s prázdnými uzly pro alokaci elementů v kolekci [33]. Příklad:

```
(1 ?x 3 4) :p "w" .
```

rdf:type

Často zkracováno na jednoduché „a“ na místě predikátu. Znak je citlivý na velikost písmen. Příklad:

```
?x  a  :Class .
```

4.2 Grafové vzory

Jak bylo řečeno na začátku kapitoly, SPARQL je založen na pasování grafových vzorů, tzv. *graph patterns*. Těchto vzorů existuje více druhů [33].

Základní

Základní grafový vzor (*basic*) je tvořen množinou RDF trojic, kde všechny musejí odpovídat [33]. Příklad:

```
?x foaf:name ?name .
```

Skupinové

Skupinové grafové vzory (*group*) je tvořen množinou základních grafových vzorů v uzavřených závorkách: . Například samotná klauzule WHERE je skupinovým grafovým vzorem [33].

Příklad:

```
{ { ?x foaf:name ?name . }
  { ?x foaf:mbox ?mbox . }
}
```

Lze použít i prázdný vzor , který souhlasí s jakýmkoliv grafem (včetně prázdného) [33].

Doplňkové

Přidáním klíčového slova **OPTIONAL** je možné přidat do výsledku nějakou část vzoru v případě, že existuje a zároveň neodmítnout jí v případě, že nebude nalezena [33].

Příklad:

```
?x foaf:name  ?name .
OPTIONAL { ?x  foaf:mbox  ?mbox }
```

Doplňkových vzorů může být v dotazu více.

Alternativní

SPARQL umožňuje definovat několik alternativních výsledků, z nichž alespoň jeden by měl souhlasit. Pokud souhlasí více než jeden, jsou vráceny všechny možné řešení. Jednotlivé alternativní vzory se označují klíčovým slovem **UNION** [33].

Příklad:

```
{ { ?book dc10:title  ?title }
UNION
{ ?book dc11:title  ?title } }
```

Jmenné

SPARQL umožňuje při dotazování explicitně deklarovat, na který graf se vztahuje daný grafový vzor. Dotaz umožňuje definovat grafové URI v části proměnných klíčovými slovy **FROM NAMED** [33], například :

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?bob
FROM NAMED <http://example.org/foaf/bob>
```

Vymezení grafu na grafový vzor potom vypadá:

```
1 WHERE
2 {
3   GRAPH <http://example.org/foaf/bob>
4   {
5     ?bob foaf:mbox <mailto:bob@work.example>
6   }
7 }
```

Graf se dá také dotazovat:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?src ?x
FROM NAMED <http://example.org/foaf/bob>
WHERE
{
  GRAPH ?src
  { ?x foaf:mbox <mailto:bob@work.example>
  }
}
```

4.3 Filtry

Filtry jsou další možností jak omezit vrácené výsledky dotazování. Na rozdíl od OPTIONAL nejde jen o pouhé přidání výsledků na základě jejich existence, ale eliminace těch výsledků, pro které je návratová hodnota filtru logická nepravda či produkují chybu. Označují se klíčovým slovem **FILTER** a jsou platné v rámci celé skupiny grafového vzoru [33].

Filtry mohou být použity pro otestování absence nějakého vzoru:

```
FILTER NOT EXISTS { ?person foaf:name ?name }
```

Nebo naopak presence vzoru:

```
FILTER EXISTS { ?person foaf:name ?name }
```

Častěji jsou filtry použity pro omezení hodnot proměnných v dotazu. Příklady:

```
FILTER (?x = ?y) .
FILTER (?price > 100) .
FILTER (?date <= "2005-01-01T00:00:00Z"^^xsd:dateTime) .
FILTER (regex(?name, "Smith"))
```

4.4 Agregace

Agregační funkce ve SPARQL aplikují výrazy pro skupiny řešení. Implicitně sada řešení je tvořena jednou skupinou, jež obsahuje všechny řešení. Grupování se provádí pomocí klíčového slova **GROUP BY** [33].

SPARQL podporuje následující agregační funkce:

- **COUNT** - počet elementů, možnost použít hvězdičku pro počet všech výsledků
- **SUM** - suma elementů
- **AVG** - průměr elementů
- **MIN** - minimální hodnota elementů
- **MAX** - maximální hodnota elementů
- **SAMPLE** - jakýkoli element
- **GROUP_CONCAT** - zřetězení elementů

Agregační funkce se deklarují v části proměnných a je jim zde přiřazena nová proměnná. Použití agregačních skupin znamená, že každá proměnná musí být v dotazu buď agregovaná nebo vyjmenovaná v **GROUP BY** části [33].

Agregované proměnné mohou být filtrovány klíčovým slovem **HAVING**. Komplexní příklad agregací může vypadat následovně:

```
PREFIX : <http://books.example/>
SELECT ?book (SUM(?lprice) AS ?totalPrice)
WHERE {
  ?org :affiliates ?auth .
  ?auth :writesBook ?book .
  ?book :price ?lprice .
}
GROUP BY ?org ?book
HAVING (SUM(?lprice) > 10)
```

4.5 Modifikátory výsledků

SPARQL dotaz generuje neseřazenou kolekci řešení kde každé řešení je částečná funkce z proměnných do RDF termů.Sekvence řešení jsou seřazeny sekvenčními modifikátory a následně vráceny [33].

Modifikátory ve SPARQL existují následující:

Řazení

Klíčové slovo **ORDER BY** určuje řazení výsledků. Po klíčovém slovu následuje výčet proměnných s potenciálním modifikátorem řazení (**ASC** nebo **DESC**). Každá hodnota může být řazena sestupně nebo vzestupně [33].

Příklad (bez udání identifikátoru řazení je implicitně použito ASC):

```
ORDER BY ?name ASC(?price) DESC(?age)
```

Projekce

Omezení proměnných, které se zobrazí uživateli. Jednoduchý příklad:

```
PREFIX foaf:       <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
 { ?x foaf:name ?name }
```

Proměnná x nebude ve výsledcích zobrazena, jelikož není vyjmenovaná v proměnných za SELECT [33].

Distinkce

Omezení na jedinečnost výsledků, tedy aby každý výsledek byl unikátní a zamezilo se duplikátům. Klíčové slovo **DISTINCT** [33].

```
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name WHERE { ?x foaf:name ?name }
```

Redukce

Podobné distinkci, ale místo odstranění duplikátů je pouze povolí k vyřazení, mohou se tedy stále objevit ve výsledcích. Klíčové slovo **REDUCED** [33].

```
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>
SELECT REDUCED ?name WHERE { ?x foaf:name ?name }
```

Offset

Klíčovým slovem **OFFSET** umožňuje výsledné sekvenci začít až od specifikovaného čísla. Offset s hodnotou nula nemá efekt [33].

```
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>

SELECT   ?name
WHERE    { ?x foaf:name ?name }
ORDER BY ?name
OFFSET   10
```

Limit

Klíčové slovo **LIMIT** omezuje sekvenci výsledků na daný počet. Garantuje vrácení nula až daný limit výsledků. Limit s hodnotou nula nevrací žádné výsledky [33].

```
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE { ?x foaf:name ?name }
LIMIT 20
```

4.6 Typy dotazu

SPARQL podporuje čtyři typy dotazu. Tyto formy nakládají vlastním způsobem s nalezenými řešeními dotazu [33].

4.6.1 Dotaz SELECT

Jeden z nejčastějších typů dotazu. Vrací proměnné výsledků přímo. Kombinuje projekci a vytváření nových proměnných ve výsledcích dotazu [33].

Mějme data:

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a     foaf:name    "Alice" .
_:a     foaf:knows   _:b .
_:a     foaf:knows   _:c .

_:b     foaf:name    "Bob" .

_:c     foaf:name    "Clare" .
_:c     foaf:nick    "CT" .
```

Potom dotaz typu SELECT vybere zdroj x, který zná jiný zdroj y, jejich jména uloží do proměnných $nameX$ a $nameY$, dobrovolně může platit, že zdroj y má predikát $nick$ se zdrojem $nickY$.

```
PREFIX foaf:     <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
WHERE
  { ?x foaf:knows ?y ;
       foaf:name ?nameX .
    ?y foaf:name ?nameY .
    OPTIONAL { ?y foaf:nick ?nickY }
  }
```

Výsledkem bude tabulka 4.1.

Tabulka 4.1: Výsledky SELECT dotazu

nameX	**nameY**	**nickY**
"Alice"	"Bob"	
"Alice"	"Claire"	"CT"

4.6.2 Dotaz CONSTRUCT

Tento dotaz vrací jeden RDF graf specifikovaný grafovou šablonou. Graf zniká kombinací výsledných trojic operací union.

Příkladová data:

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a     foaf:name    "Alice" .
_:a     foaf:mbox    <mailto:alice@example.org> .
```

Nad nimi provedeme dotaz typu CONSTRUCT, kterým vytvoříme nový graf (za použití nového jmenného prostoru *vcard*):

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT   { <http://example.org/person#Alice> vcard:FN ?name }
WHERE       { ?x foaf:name ?name }
```

Dotaz jako výsledek vrací nový graf:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

<http://example.org/person#Alice> vcard:FN "Alice" .
```

4.6.3 Dotaz ASK

Tento dotaz testuje, zda-li dotaz má řešení. Dotaz nevrací žádné informace o řešení, pouze jestli nějaké řešení existuje nebo ne [33].

Data:

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

Dotaz:

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

Dotaz vrací odpoveď:

```
true
```

Jelikož v grafu existuje trojice s predikátem *foaf:name* a subjektem **"Alice"**. Pokud by neexistovalo řešení, dotaz by vracel *false*.

4.6.4 Dotaz DESCRIBE

Dotaz typu DESCRIBE vrací RDF podgraf původního grafu. Data není potřeba předepsat do dotazu, což by znamenalo, že by klient strukturu dat musel znát, místo toho jsou determinovány přímo SPARQL procesorem. Dotaz vezme všechny zdroje uvedené ve zdrojích řešení a vytvoří z nich jeden RDF graf, který pak vrací [33].

Jednoduchý dotaz:

```
PREFIX ent:  <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

Vrátí vše, co o daném zdroji ví:

```
_:a      exOrg:employeeId     "1234" ;

         foaf:mbox_sha1sum    "
            bee135d3af1e418104bc42904596fe148e90f033" ;
         vcard:N
           [ vcard:Family         "Smith" ;
             vcard:Given          "John"  ] .

foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty .
```

4.7 SPARQL Endpoint

SPARQL endpoint je veřejně přístupné úložiště přes HTTP protokol, který obsahuje a spravuje RDF data a umožňuje jejich dotazování. Příkladem jsou OpenLink Virtuoso [35], Apache Jena Fuseki [36] a další.

Některé příklady veřejných endpointů jsou uvedeny v tabulce 4.2.

Tabulka 4.2: Příklady různorodých SPARQL endpointů [37, 38]

Název	**Adresa**
DBpedia	http://dbpedia.org/sparql
BBC Programmers and Music	http://lod.openlinksw.com/sparql/
Archives Hub Linked Data	http://data.archiveshub.ac.uk/sparql
Dutch Ships and Sailors	http://dutchshipsandsailors.nl/data/sparql
Bio2RDF:Affymetrix	http://affymetrix.bio2rdf.org/sparql
STW Thesaurus for Economics	http://zbw.eu/beta/sparql/stw/query
Spanish Linguistic Datasets	http://linguistic.linkeddata.es/sparql

5 Analýza vizuálních nástrojů SPARQL

Kapitola pojednává o vizuálních nástrojích, které byly za dobu existence RDF a SPARQL vytvořeny, nebo alespoň teoreticky vymyšleny. Jako první je probrán program Sparkle v kapitole 5.1, do nějž práce vytváří rozšíření. Následují nástroje, které vizualizují RDF data v kapitole 5.2 , přesněji řečeno SPARQL endpointy. Tyto nástroje mají trochu jiné cíle než tato práce, ale poslouží jako dobrý náhled toho, jak je RDF graf vizualizován. Část o nástrojích vizualizace SPARQL dotazu je rozdělena na dvě části - ověřitelné v kapitole 5.3 a neověřitelné v kapitole 5.4. Ověřitelné jsem prakticky vyzkoušel a mohu je tedy posoudit z praktického pohledu, zatímco neověřitelné jsou většinou staré nástroje či pouze nápady, které kdysi vznikly a postupem doby byly opuštěny. Lze se o nich maximálně dočíst v článcích či několika málo ilustračních obrázků.

Nejvíce zajímavé pro tuto práci budou právě ověřitelné nástroje, kterým se povedlo dotáhnout vizualizaci dotazu do použitelné podoby. U každého nástroje je uveden jeho vznik, datum poslední aktivity, použitá technologie a typ vizualizace.

5.1 Sparkle

Sparkle je zmíněn na tomto místě, jelikož nepatří k tradičním vizualizačním nástrojů a není přímo součástí analýzy. Aplikace Sparkle je grafický nástroj pro tvorbu a vyhodnocení SPARQL dotazů [39]. Je napsaný v JavaFX a vyvíjen v rámci projektu MRE (MEdical Research and Education) [40] na katedře Informatiky a informační techniky na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Práci na aplikaci začal Jan Šmucr ve své diplomové práci [41] v roce 2013 pod vedením Ing. Petra Včeláka, pokračovali pod stejným vedením studenti Michal Kazák [42] a Klára Hlaváčová [43].

Funkcionalita aplikace důležitá v kontextu práci je:

- Podpora velkého množství SPARQL 1.0 a 1.1 konstrukcí
- Formulářový a textový editor

- Tři druhy úložiště - lokální (souborový systém, in-memory model), vzdálené (Virtuoso Universal Server), SPARQL endpoint
- Vyhodnocení a zobrazení výsledků dotazu
- Nahrání/uložení dotazu

5.2 Nástroje vizualizace RDF

Tyto nástroje se nezabývají generováním dotazů, ale snaží se nějakým způsobem vizualizovat RDF zdroje. Tabulka 5.1 poskytuje základní přehled o nástrojích.

Tabulka 5.1: Srovnání nástrojů vizualizace RDF

Nástroj	**Vznik**	**Poslední aktivita**	**Technologie**	**Vizualizace**
IsaViz [44]	2002	21. říjen 2007	Java	Grafové uzly
LodLive [45, 46]	2013	7. říjen 2018	Javascript	Grafové uzly
LODmilla [47–49]	2013	5. duben 2017	Javascript, Java	Grafové uzly
RelFinder [50, 51]	2007	25. říjen 2012	Adobe Flex	Grafové uzly

IsaViz

Offline nástroj pro vizualizaci dat. Poskytuje 2.5D uživatelské prostředí se zoomováním a navigací grafu, vytváření/mazání pomocí kreslení elips, obdélníků a oblouků. Nejstarší nástroj a je to znát především na jeho designu. Opuštěn již velice dlouho, nahrazen novějšími nástroji [44].

LodLive

Nástroj vizualizuje RDF zdroje jako kruhy. V tomto kruhu je vepsán název zdroje. Cesty do dalších zdrojů jsou umístěny kolem zdroje a jsou klikatelné. Po kliknutí se načte další zdroj a jeho sada kruhů kolem něj. Cesty mezi zdroji jsou vyjádřeny šipkami, jež jsou označeny názvem dané vlastnosti a směr šipky určuje směr vztahu. Cesty jsou zobrazeny obojího typu – odchozí i příchozí.

Autoři tvrdí, že jeden z případů užití je kontrola vytvářené ontologie, kdy postupným klikáním a vizualizací jsou vidět cesty v RDF grafu. V případě složitých ontologií je vizualizace nepřehledná, kdy se uživatel v záplavě uzlů velice rychle ztratí.

Nástroj načítá obrázky u jednotlivých uzlů, ke kterým potom poskytuje rychlý přístup. Dále rozpoznává geolokační data a umožňuje uživateli jednoduchým kliknutím vidět, kde ve světě se daný zdroj nachází [52].

Postaven na Javascript nevyžaduje žádnou konfiguraci či instalaci, prosté spuštění v prohlížeči [45]. SPARQL endpointy má nástroj předkonfigurované a neposkytuje možnost, jak přidat vlastní endpointy. Poskytuje také možnost vizualizovat RDF zdroj dle jeho URI [46].

LODmilla

Nástroj kombinuje prvky grafové vizualizace a textové. Po vybrání uzlu z předkonfigurovaného RDF zdroje je tento uzel přidán do vizualizace. V případě přidávání dalších uzlů nástroj automaticky mezi nimi hledá cesty a spojuje je v případě, že takové cesty nalezne.

Nástroj v uzlech zobrazuje obrázek z wikiboxu (byl vytvořen primárně pro DBpedia [53]). Po kliknutí na uzel je zobrazen další panel, ve kterém se nachází veškeré informace o vybraném uzlu - vlastnosti, příchozí a odchozí cesty. Všechny položky jsou klikatelné a automaticky přidávají daný uzel do vizualizace.

Umožňuje upravovat data za běhu (delete/insert) a takto upravený graf pak uložit - výsledkem je URL, jenž se dá sdílet. Je zachován také aktuální výběr uzlů [47, 54, 55].

Dále nad uzly umožňuje:

- najít řetězec ve vybraných uzlech – jednoduchým textovým hledáním
- najít řetězec v okolí vybraných uzlů -> využívá BFS, omezuje hloubku nastavitelnou proměnnou
- najít cesty v okolí -> BFS a hledá cesty, jež odpovídají zadanému řetězci, omezeno maximální hloubkou
- najít cestu mezi dvěma uzly -> využívá Dijkstrova algoritmu pro hledání nejkratší cesty [47]

RelFinder

Nástroj se zaměřuje na hledání vztahů mezi dvěma zadanými třídami a jejich následnou vizualizaci. Jsou zde rozlišovány čtyři možné vztahy:

- $a \Rightarrow \cdots \Rightarrow b$ - vztah z a do b
- $a \Leftarrow \cdots \Leftarrow b$ - vztah z b do a
- $a \Rightarrow \cdots \Rightarrow c \Leftarrow \cdots \Leftarrow b$ - vztah, kdy z a a b vede cesta do c
- $a \Leftarrow \cdots \Leftarrow c \Rightarrow \cdots \Rightarrow b$ - vztah, kdy z c vede cesta do a a b

Uzel *c* je nalezen při prohledávání grafu. S pomocí podmínek je možnost hlídat, že každá třída se objeví pouze jednou (zamezení cyklům), ignorování určitých tříd a vztahů či omezení strukturálních vztahů (*rdf:type*, *rdf:subClassOf*). Vizualizace obsahuje filtrovací možnosti a zvýrazňuje aktuální selekci [56].

Na začátku je vždy nutné vybrat ony dvě třídy, mezi kterými se budou hledat vztahy, tedy *a* a *b*. Online demo ilustruje použitelnost nástroje na datech DBpedia. Jiné datasety lze konfigurovat v nástroji [50].

Pro rychlé fungování nástroje byl vytvořen algoritmus, jenž potřebuje předzpracovat RDF data. Při tomto předzpracování jsou nalezeny komponenty grafu, tedy maximální spojité podgrafy pomocí BFS. Pokud dva uzly nejsou ve stejné komponentě, nástroj ihned ví, že mezi nimi neexistuje cesta. Pokud jsou, vypočítá se minimální vzdálenost mezi uzly, následně se s inkrementující vzdáleností formulují SPARQL dotazy pro získání cest, jež nástroj ihned zobrazuje uživateli [57].

I přes předzpracování jde o náročnou výpočetní operaci, kterou je nutno omezit maximální vzdáleností mezi třídami, v RelFinder nastavena na deset cest [56].

5.3 Ověřitelné nástroje

Nyní se nacházíme v části Query Builders - nástrojů jež generují SPARQL dotazy. Nástroje v této kapitole je možné vyzkoušet v rámci online dema. Jejich celkové srovnání v tabulce 5.2.

Tabulka 5.2: Srovnání ověřitelných nástrojů vizualizace SPARQL

Nástroj	**Vznik**	**Poslední aktivita**	**Technologie**	**Vizualizace**
iSPARQL [58]	2005	1. leden 2019	Javascript	Grafové uzly
LinDA [59]	2014	3. říjen 2016	Javascript, Python	Tabulky
QueryVOWL [60]	2015	2015	Javascript	Grafové notace
SemTk: Semantic Toolkit [61]	2016	17. prosinec 2018	Javascript, Java	Grafové uzly
Sparklis [62]	2013	1. únor 2019	Javascript	Přirozený jazyk
SparqlFilterFlow [63]	2014	2014	Javascript	filter/flow
SPARQLGraph [64]	2014	15. říjen 2015	Javascript	Grafové uzly
Visual SPARQL Builder [65]	2010	20. říjen 2015	Javascript	Tabulky
ViziQuer [66]	2016	9. březen 2019	Javascript	UML diagramy

iSPARQL

Dříve distribuováno s OpenLink Virtuoso Open Source Edition [67].

Nástroj obsahuje textové pole pro psaní dotazů (klasický SPARQL editor se syntax zvýrazňováním) a QBE (SPARQL Query By Example) editor. RDF

zdroje jako uzly, šipky mezi nimi potom vztahy. Žádná barevná či geometrická diference mezi tvary. Nastavení uzlů a vlastností přes malý formulář, jež neobsahuje našeptávač.

Lze určit DISTINCT, řazení proměnných včetně změny pořadí (pouze ASC). Není GROUP BY, filtry, prefixy je nutné importovat. Lze importovat a exportovat dotaz mezi QBE a textovým editorem. Nepodporovaná funkcionalita je v dotazu ignorována (např. filtry).

Uživatel musí znát data, SPARQL jazyk a musí vědět co chce. Uživatelská přívětivost je minimální, ovládání neintuitivní a zkoušené demo se zasekává [68].

Mnohem použitelnější je textový editor nástroje.

LinDA

Nástroj LinDA je zkrácené označení použité v této práci pro část produktu LinDA Workbench, což je open-source balíček Enterprise Linked Data nástrojů pro mapování a publikování dat v Linked Data Format, linkování dat s veřejnými či soukromými daty, analýza těchto dat a jejich vizualizace.

Tento balíček se sestává z:

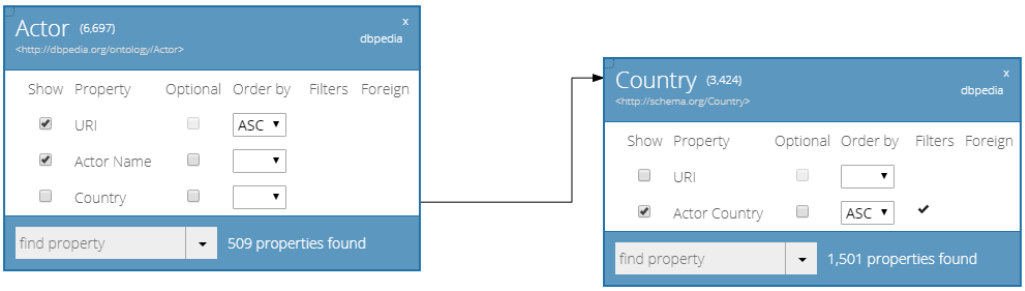
- **LinDA Transformation engine** – transformace dat do linked dat, vybrání dat, vybrání mapování na slovníky a samotná transformace
- **LinDA Vocabulary repository** – repositář a API nejpoužívanějších sémantických slovníků, základ ostatních nástrojů
- **LinDA RDF2Any** – převod RDF dat do ostatních datových formátů získaných SPARQL dotazováním
- **LinDA visualization** – vizualizace RDF dat, grafy, diagramy
- **LinDA analytics package** – konvenční analytické procesy (regrese, shlukování, předvídání) dat získaných SPARQL dotazem
- **LinDA Query Builder** – vizuální tvorba SPARQL dotazu, předmět naší analýzy. Této části balíčku budu nadále říkat jen LinDA [69]

LinDA je inspirována relačními databázemi a jejich nástroji pro vytváření SQL dotazů. Jednotlivé uzly jsou reprezentovány jako tabulky, jejich vlastnosti pak záznamy tabulky.

Na začátku je nutné vybrat endpoint, na který se bude nástroj dotazovat, následně startovací třídu. Po vybrání se vytvoří tabulka této třídy. Tabulka je nadepsaná názvem třídy a dat, kam patří. Řádky tabulky jsou jednotlivé

vlastnosti, na začátku je vybrána automaticky URI vlastnost, která se nedá odebrat či označit jako OPTIONAL. Vizualizace nástroje na obrázku 5.3.

Obrázek 5.1: LinDA vizualizace



Záznamy v tabulce obsahují ovládací prvky, pomocí nichž lze:

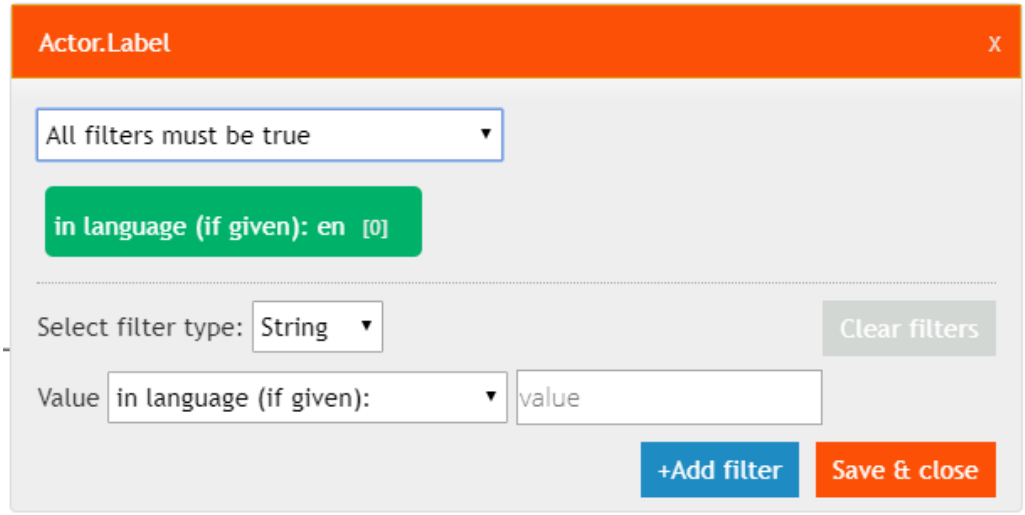
- **Show** - skrytí/zobrazení vlastnosti ve výsledcích
- **Optional** – jestli bude vlastnost obalena OPTIONAL blokem
- **Order by** – můžeme vybrat, že tato hodnota bude přidána do řazení, možnost ASC/DESC
- **Filters** – vyvolá menu filtrů
- **Foreign** – přidá vztah k jiné třídě (připomíná cizí klíč ze SQL). Samotné přidání však musí být přes vyhledání třídy, její přidání do vizualizace a až potom je propojit šipkou.

Nastavení filtrů vyvolá další okno. Zde se mohou jednotlivé filtry vytvářet a přidávat k označené hodnotě. Filtry mohou být nastaveny: všechny pravda, alespoň jeden z nich pravda, všichni musí být nepravda, alespoň jeden z nich nepravda, vlastní booleovský výraz. Následně se vytváří samotný filtr. Uživatel má na výběr:

- **String** - řetězec, jeho hodnoty mohou být rovná se, obsahuje, nerovná se, začíná, končí, regulární výraz. Daná hodnota se píše do připraveného pole.
- **Number** a **Date** - klasické možnosti porovnání pro čísla a data - rovná se, nerovná se, menší, větší, menší či rovno, větší či rovno
- **URL** - doplňuje se ručně

Ukázka filtru na obrázku 5.3.

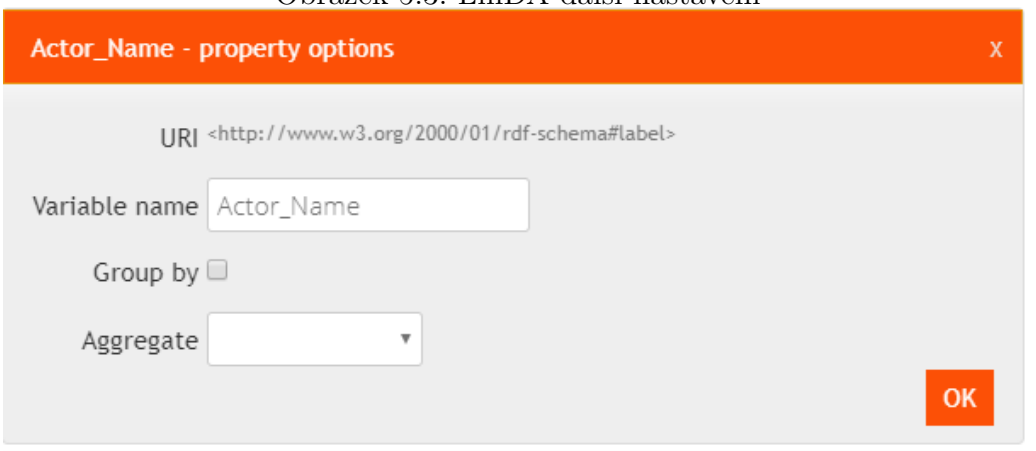
Obrázek 5.2: LinDA výběr filtru



Ve spodní části tabulky je kontrolní panel pro vkládání dalších vlastností. Nástroj nabízí všechny, které našel, seřazené dle počtu, kolikrát byly nalezeny, sestupně. I u třídy je uvedeno, kolikrát byla nástrojem v datech objevena.

Další nastavení vlastností je trochu schováno. Po stisknutí pravého tlačítka myši a vybráním **Options** se zobrazí další formulář (obrázek 5.3) . Zde je vidět URI hodnoty, název proměnné, který se zde dá změnit, zaškrtnutí pro seskupování hodnoty a agregační funkce. Nástroj z agregací podporuje SUM, COUNT, AVERAGE, MINIMUM, MAXIMUM, GROUP_CONCAT a SAMPLE. Na jednu hodnotu lze přidat pouze jednu agregační funkci.

Obrázek 5.3: LinDA další nastavení



Řazení i seskupování se dá ovlivnit pořadím hodnot v tabulce, ale pouze

v té tabulce, do které hodnoty patří. Nelze nastavit DISTINCT. OFFSET a LIMIT řeší nástroj sám. Automaticky vyhledává s limitem na sto záznamů, o další je nutné požádat a nástroj začne přidávat daný offset.

Dotaz se generuje při každé uživatelově akci a na jeho pokyn se až spouští [70, 71].

QueryVOWL

Grafová vizualizace jako iSPARQL s pokusem o lepší definici tvarů – VOWL [72]. Ty nejdůležitější poznatky:

- kruh – třída ontologie
- linky – relace vlastností, často doplněny o šipky pro určení směru vztahu
- šipky – směr vztahu
- obdélníky – značky vlastností a datové typy
- styl linek a okrajů – čárkované linky a okraje mohou nést další význam (např. čárkovaný okraj určuje literály a některé speciální případy tříd)

Nejpoužívanější třídy z používaných slovníků mají přiřazené barvy pro jednoznačné rozeznání [72, 73].

Vytváření uzlů probíhá vybráním třídy, následně možnost vybrat z načtených vlastností, čímž se do grafu přidávají další uzly. Generování dotazů probíhá při klikání na jednotlivé uzly. Vybraný uzel odpovídá výsledku dotazu (tedy co má být výsledkem dotazu to označím). Demo je neúplné a generované dotazy jsou jednoduché [74]. Více proměnných ve výsledku není možné. Chybí zde velké množství SPARQL funkcionality (ORDER BY, GROUP BY, LIMIT, OFFSET, DISTINCT, FILTER) [60, 74, 75].

SemTk: Semantic Toolkit

Pro použití je nutné připojit se na běžící instanci Virtuosa a následně nahrát ontologie a data. Princip vytváření dotazu pak vychází právě z nahrané ontologie.

V levém panelu se nachází nahraná ontologie, její třídy a vlastnosti tříd. Pomocí drag and drop vybereme třídu, jež nás zajímá. Při vybrání druhé třídy se automaticky vytvoří cesta mezi vybranými třídami (pokud vede přes další třídy, automaticky se nahrají do výběru). Výběrem tříd poté určujeme, které z nich budou zařazeny do dotazu. Typ dotazu může být: SELECT DISTINCT,

SELECT COUNT, CONSTRUCT a DELETE. Obsahuje podporu řazení, kdy se zobrazí extra okno, ve kterém se vybírají jednotlivé proměnné a ASC/DESC ražení. Lze také určit LIMIT. Filtry se přidávají ručně [76, 77].

Sparklis

Tento nástroj je velice odlišný od ostatních – generování SPARQL dotazu je postavena na přirozeném jazyku. Uživatel vybíráním návrhů postupně sestavuje dotaz a se SPARQL syntaxí nepřijde do styku.

Autoři popisují tři aspekty architektury Sparklis:

- **Faceted search (FS)** – step-by-step tvorba dotazu, Sparklis dává návrhy při každém kroku
- **Query Builders (QB)** – vytvořením přirozené věty je vytvořen SPARQL dotaz
- **Natural Language Interface (NLI)** – návrhy, dotaz a výsledky verbalizovány do přirozeného jazyka

Kvůli škálování nástroj nikdy nevrací všechny výsledky, tedy i našeptávač zobrazuje pouze určitou množinu. Je tedy nutné poskytnout počáteční znaky hledané třídy či vlastnosti než se Sparklis chytí.

Nástroj je rozdělen do tří oken - Třídy a Vztahy, Instance a Hodnoty a Agregace a Operace. Architektura pak staví na *query* - aktuální podobě SPARQL dotazu a *focus* - aktuální zaměření uživatele v přirozené větě. Dle zaměření Sparklis nabízí návrhy ve zmíněných třech oknech. Uživatel pak jen postupným výběrem a posouváním zaměření vytváří SPARQL dotaz, jež působí jako pomalé a metodické psaní přirozené věty.

Nástroj nabízí filtry, ale je problém se zaměřením a následné generování toho správného filtru. SPARQL dotazy automaticky obsahují DISTINCT. Filtry podporují řetězce, čísla, datumy a časy, jazykové tagy a datové typy. Dle autorů dále podporuje ASK, UNION, OPTIONAL, NOT EXISTS, ORDER BY, GROUP BY a HAVING, ale žádné z těchto funkcionalit nebyly vyzkoušeny (a v návodu se nepíše, jak jich dosáhnout) [62, 78].

SparqlFilterFlow

Založeno na filter/flow modelu, jenž byl poprvé navržen pro SQL dotazování v relačních databázích. Nabízí intuitivní reprezentaci Booleanovských výrazů, které mohou být využity pro filtrování dat. Výrazy jsou vizualizovány jako orientované acyklické grafy, jejichž uzly definují filtrovací podmínky a

hrany popisují tok dat. Šířka hran indikuje hustotu dat v toku. Konjunkce se modelují jako sekvenční cesty zatímco disjunktní jako paralelní cesty [79]. Autoři mluví o zlepšeném modelu (extended filter/flow), kdy přidali možnost více cest mezi uzly a tím umožnili více toků být zpracováno různými způsoby. Dále také přidali konečný uzel, jenž zobrazuje výsledky [80].

Nástroj umožňuje vybrat třídu, z níž získáváme informace pomocí filtrů. Dostupné demo je velice strohé a z popisovaných vlastností obsahuje minimum. Nahraná data jsou zjednodušená a je jich velice málo, demo podporuje pouze vybrání jedné třídy a následně vztahů, které ale nelze nijak definovat [63, 81].

Jedna z potenciálních silných stránek návrhu je, že je možno sledovat šířku filtrů a tím zjistit, jak velký vliv má určitý uzel na velikost výsledků [82].

SPARQLGraph

Vytvořeno pro biologické RDF databáze [83], velice jednoduchý grafový design, kdy třídy jsou zeleně zbarvené obdélníky, vlastnosti šedé obdélníky a šipky mezi nimi určují směr cesty. V uzlech vlastností se vypisuje hodnota, kterou má SPARQL pro daný uzel hledat (jde tedy o filtr regulárního výrazu). Následně se jen spouští vytvořený dotaz na přednastavená biologická data. Další vlastnosti SPARQL nejsou implementovány, včetně řazení a filtrace [83, 84].

Znalost RDF datasetů umožňuje nástroji kontrolovat směr vztahů (nedovolí šipku mezi uzly, které spolu nijak nesouvisí) [84].

Visual SPARQL Builder

Připomíná SQL tabulky jako LinDA. Jednotlivé tabulky představují uzly a jejich řádky potom vlastnosti.

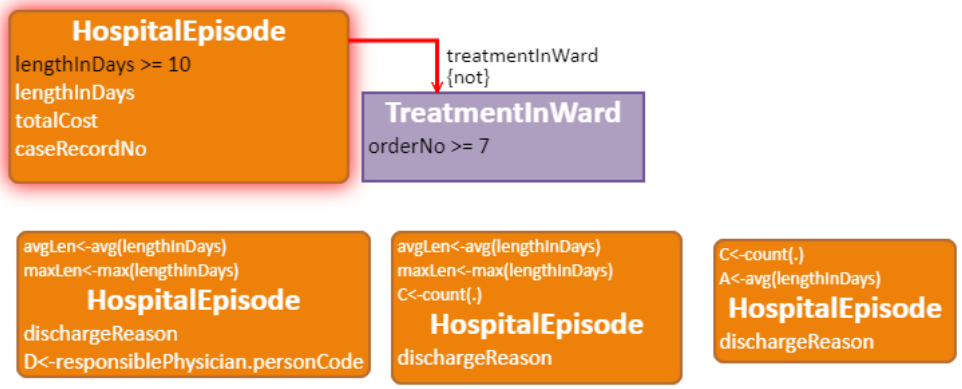
Nástroj neumí řazení ani agregace. Obsahuje OPTIONAL, zobrazení proměnné, filtry. Jazykové filtry chybně generuje dvakrát. Omezeno pouze jeden filtr na jednu vlastnost. Relace obsahují chyby a nefungují. Velice rychle zaplňuje obrazovku i při malém počtu tříd [85].

ViziQuer

V nástroji je nutno založit projekt, v něm poté importovat ontologie a nastavit SPARQL endpoint. Následně je možno vytvářet diagramy, jež obsahují vytvořené SPARQL dotazy. K dispozici je demo, které dobře prezentuje funkčnost nástroje [86].

Vizualizace je založená na UML diagramu tříd (obrázek 5.3). Obdélníky představují uzly, šipky mezi nimi potom vztahy. Počáteční uzel má oranžovou barvu, ostatní šedou. Šipky pro relace mají různý tvar (plné pro normální vztah, tečkované pro OPTIONAL a červené pro negaci).

Obrázek 5.4: ViziQuer vizualizace



Tvoření dotazu začne vytvořením počátečního uzlu. Nyní je třeba vybrat třídu – našeptávač zobrazuje všechny možnosti. Pro další nastavení uzlu je třeba mít uzel vybraný, potom se zobrazí formulářový panel (kvůli velikost pouze na CD ve složce `img`). V této části uživatel určuje další nastavení zvoleného uzlu, jmenovitě:

- **Agregations** - agregační funkce – určení alias a výrazu – našeptávač zobrazuje nabídky (SUM, COUNT, MIN, MAX s předvyplněnými hodnotami - jsou zde všechny možné kombinace vlastností dané třídy)
- **Class** - název třídy – určení třídy (dělá se pro startovní třídu)
- **GROUP BY THIS** - seskupování pro celou třídu (včetně samotné třídy, atributů a dalších linkovaných tříd)
- **DISTINCT** - možnost zaškrtnutí
- **Conditions** - určení filtrů (píšou se jako výrazy)
- **Attributes** - píšou se alias a výraz, lze zaškrtnout **Require** (automaticky vše OPTIONAL) a show/hide. Našeptávač radí
- **OrderBy** - vypisují se jednotlivé alias (názvy proměnných), lze určit DESC jinak ASC – není našeptávač

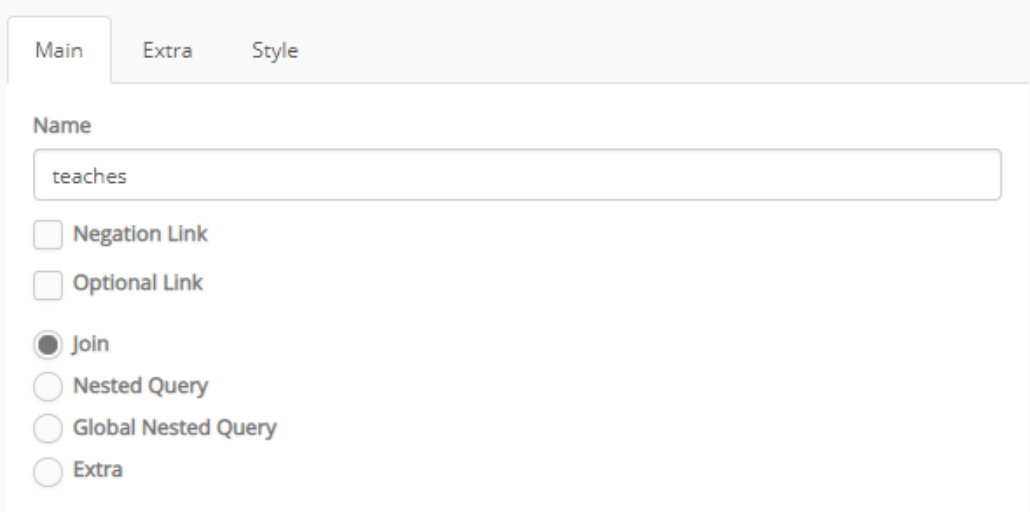
- **GroupBy** - platí to samé co pro **OderBy**
- **Show rows (LIMIT)** - lze určit
- **Skip rows (OFFSET)** - lze určit

Při úpravách se dotaz negeneruje, jen na pokyn uživatele.

Nástroj je rozdělen na vizuální a formulářovou část, kdy v grafové vizualizaci jsou jen naprosto minimální věci (název třídy, filtry, vlastnosti, řazení a limity dotazu a to jen jako názvy, ukázka – oznamují, že byly použity), zbytek je poté ve větším detailu uveden ve strukturovaném formuláři, kde je možnost tyto věci měnit a přidávat. Lze generovat dotaz nad celým propojením nebo jen jedním uzlem (částečný dotaz).

Relace na další třídy se přidávají přes extra formulář, kde nástroj nabízí možnosti (obrázek 5.3). Jde o inverzní i externí cesty. Po přidání se automaticky objeví nový uzel spojený šipkou s původním uzlem, vždy šedé barvy. S uzly se dá hýbat po plátně.

Obrázek 5.5: ViziQuer relace



Spojení se dá dále nastavovat:

- **Negation Link** - určení, že jde o negaci (FILTER NOT EXISTS)
- **Optional Link** - určení, že jde o OPTIONAL
- výběr typu relace - základní nastavení je **Join**, klasické spojení mezi třídami, další možnosti jsou **Nested Query**, **Global Nested Query** (vnořené dotazy) a **Extra** (což potom automaticky nezahrnuje třídu do dotazu)

5.4 Neověřitelné nástroje

O těchto nástrojích je možno se dočíst v článcích a lze najít i obrázky či ilustrační videa, ale nikde není ke stažení funkční verze nebo alespoň dostupné demo online. Často jde o zapomenuté a nedodělané projekty.

Konduit VQB

O tomto nástroji je zmiňováno i mimo jeho vlastní publikaci [87], ale bez dalších informacích. Původně vytvořeno pro KDE Nepomuk, poslední verze již zřejmě neobsahují [88]. Vizualizace tohoto nástroje by měla vycházet z formuláře, podobně jako SPARQLViz. Nástroj by měl fungovat na bázi schématu, tzv. *schema-based*, kdy se po vybrání třídy vytváří dotaz postupným vybíráním vlastností třídy. Kromě pár ilustračních obrázků nebylo nic dalšího objeveno [89].

Nitelight

Nástroj vizualizuje v podobě grafových notací, jež se označují jako vSPARQL. Dle autorů není nástroj vhodný pro uživatele nezkušené se SPARQL. K dispozici jsou pouze ilustrační obrázky [87, 90].

OptiqueVQS

O tomto nástroji existuje velice málo informací a i jeho stránky jsou velice staré a neaktualizované [91]. Autoři se snažili vytvořit nástroj pro neznalé SPARQL a rozdělili GUI na tři části - část konceptů, kde jsou třídy vybírány, část atributů zvoleného konceptu a část diagramu vytvářeného dotazu. Mělo by jít o nástroj založený na ontologiích. Příliš více informací se najít nedá [92].

Quatro2

O tomto nástroji nebyl nalezen žádný článek. Jeho webové stránky jsou několik let staré a opuštěné a často vypadávají [93]. Quatro2 je na těchto stránkách ke stažení, ale po pomalém několikahodinovém stahování nelze spustit [94]. Nebyly nalezeny ani žádné ilustrační obrázky.

RDF-GL

RDF-GL není nástrojem ve smyslu ostatních zmíněných nástrojů - jde spíše o teoretický pokus navrhující grafický jazyk založený na SPARQL.

Veškeré elementy SPARQL jsou definovány pomocí šipek, obdélníků a kruhů. Některé základní prvky jsou:

- Obdélníky:
  - Oranžové – informace o výsledku dotazu
  - Růžový – reprezentuje subjekt/objekt trojice
  - Zelený – filtrovaný subjekt/objekt trojice
- Kruhy:
  - Modré – vztah OR (UNION)
  - Fialové – OPTIONAL
- Šipky
  - Černé – vlastnost (property)
  - Šedé – OPTIONAL vlastnost
  - Žluté a červené – vztahy patřící do UNION, začátek a konec bloku

Následuje detailnější popis elementů včetně různých tvarů. Článek dále zavádí EBNF pro SPARQL dotaz, tedy způsob převodu mezi grafickým a normálním SPARQL a naopak.

Více informací než tento článek k nalezení není, jde tedy o jednu z teorií vizuální reprezentace SPARQL dotazu, jež se nikdy neuchytila [95].

SPARQL Views

Modul pro Drupal – open-source CMS (systém pro správu obsahu – Content Management system) [96]. Ilustrační videa ani obrázky nejsou dostupné. Vytvořeno jako rozšíření modulu Views pro Drupal umožňuje přistupovat k SPARQL enndpointům [97].

Dle dostupného článku nástroj na počátku nabídne seznam predikátů, jenž v endpointu objevil. Toto se bere jako startovní bod. Seznam těchto predikátů je získán pomocí SPARQL dotazování. Prefixy nástroj generuje automaticky. Další popis generování dotazu chybí [98].

SPARQLViz

Nástroj vznikl jako plugin pro IsaViz. Obsahuje tzv. Graphical Query Composer, který pomocí formulářů sestavuje dotaz bez nutnosti znát SPARQL [99]. Vzhledem ke svému stáří často citováno v dalších publikacích [87]. Autoři tvrdí, že lze automaticky generovat dotazy SELECT, DESCRIBE, ASK a CONSTRUCT. Neexistuje našeptávač [99]. Zdrojové kódy nejdou přeložit [100].

5.5 Shrnutí nástrojů

Následuje shrnutí všech probraných nástrojů v rámci několika odstavců.

Vizualizace RDF zdrojů je první věcí, kterou jsem zkoumal. Jde o velice starou snahu, v některých případech dokonce starší než samotný jazyk SPARQL. Nástroj podporovaný W3C **IsaViz** vznikl na začátku století, ale v následujících letech byl postupně opuštěn. Šlo o jednoduché zobrazení RDF založené na grafu. Pomyslnou štafetu převzaly modernější nástroje **Lodmilla** a **LodLive**, kteří se snaží o grafové zobrazení RDF zdrojů s tím, že poskytují přístup k celému zdroji, nebo alespoň k některým jeho nejdůležitějším částem. Trochu stranou stojí **RelFinder**, jenž se zaobíral hledáním cest mezi uzly. Všechny tyto nástroje se snaží vizuálně zobrazit RDF data tak, že jsou čitelná člověkem a že je možno pouhým pohledem získat nějaké informace. Užitečnost těchto nástrojů se těžko měří. Jsou dobré pro ověření správnosti vytvářené ontologie, například, ale při hledání informací v datech nemohou obstát proti SPARQL dotazovacímu jazyku.

Teoretické SPARQL editory je možno vyhledat, dočíst se o nich ve článcích a někdy i najít obrázky či ukázková videa, ale jejich vývoj je dávno opuštěn, zdrojové kódy zastaralé a často nepřeložitelné. **KonduitVQB**, **OptiqueVQS**, **Quatro2**, **SPARQL Views** jsou zástupci těch, o kterých víme naprosté minimum. **SPARQLViz** rozšiřoval **IsaViz**, jde tedy o velice starý nástroj, který byl založen na formulářové tvorbě dotazu. **Nitelight** byl mnohem více grafický a snažil se i o určitou definici svých notací. Kromě obrázků ale více nevíme. Další grafové notace se pokoušel definovat RDF-GL, který šel o krok dále a přišel s jazykem, jenž tuto notaci popisuje. Pokusil se definovat gramaticky zcela nový jazyk, jenž by bylo možné vzájemně převést s vizualizací a následně se SPARQL dotazem. Žádné praktické využití této teorie se nepodařilo vyhledat.

Nejzajímavější části analýzy jsou ty nástroje, které se dají spustit a vyzkoušet. Začnu tím nejodlišnějším, **Sparklis**. Na rozdíl od ostatních se zabývá přirozeným jazykem a dotaz vytváří v podobě věty. Verbalizováno

je vše, SPARQL SELECT trojice, filtry, agregace i výsledky. Jde o stále vyvíjený projekt a vzhledem k popularitě přirozeného jazyka bude tento přístup jistě nadále zajímavým.

iSPARQL je nástroj, jenž se převážně používá kvůli svému textovému editoru SPARQLu, jelikož jeho grafická verze je omezená a špatně se používá. Mnohem rychleji se dá dotaz napsat než se postupným klikáním v tomto nástroji člověk kamkoli dopracuje. Vyžaduje znalost jak SPARQL tak dat, vzhledem k tomu že je nutné psát vše ručně. Jde tedy jen o trochu jiný přístup k psaní dotazu, jenž je těžší používat než klasický textový editor.

QueryVOWL se inspiroval nástroji, které se snažily o definici grafických notací a stavět na tom. Dostupné demo poskytuje jen základní SPARQL generování bez většiny funkcionality a je to uznatelné pouze jako prototyp. Jeho bratrský projekt **SparqlFitlerFlow** vychází z filter/flow modelu a snaží se jej implementovat pro SPARQL, ale výsledek je pouze omezený a spíše nefungující prototyp. Je třeba říci, že tyto nástroje se nebály přijít s něčím novým pro RDF a SPARQL, ale nebyly dotaženy do funkčního konce.

SPARQLGraph obsahuje jednoduchou vizualizaci a generované dotazy pouze nejzákladnější funkcionalitu. Jde o nedokončený prototyp. **Visual SPARQL Builder** se snaží o tabulkový design, trpí ale opět na nedokončenou implementaci, spoustu chyb a omezenou SPARQL funkčnost. **SemTk** je komplexní nástroj, jenž vyžaduje značnou konfiguraci před svým použitím. Generování dotazů pak probíhá vybíráním tříd z nahraných ontologií, řazení a filtrace se poté dělá na jednotlivých uzlech. Problém u tohoto nástroje je chaos, jenž nastane při použití velkých ontologiích.

ViziQeur nabízí jiný přístup k vizualizaci, jenž je založen na UML diagramech tříd. Jednotlivé třídy jsou obdélníky obsahující jen ty nejnutnější informace o uzlu, vše ostatní je umístěno ve formulářové části nástroje. Toto rozdělení jsme mohli vidět i u **Lodmilla**, pokus o rozdělení vizualizace a SPARQL funkčnosti, což je nutné při velkém počtu tříd. Nástroj nabízí celou řadu SPARQL funkcionality, jediná jeho nevýhoda jsou ručně psané filtry.

LinDA je posledním probíraným nástrojem a její přístup je přiblížit SPARQL k SQL, jazyku mnohem známějšímu běžnému uživateli. Vizualizace je založená na tabulkách, které velice připomínají relační tabulky. Tabulka odpovídá vybrané třídě a jednotlivé řádky potom vlastnostem. Funkčnostmi je na tom nástroj velice dobře a především ve filtrech je napřed před ostatními. Velkými problémy je nutná konfigurace nástroje a při velkém počtu tříd je vizualizace nepřehledná. Agregace jsou omezeny jedna na vlastnost. Některá funkcionalita je také schována nepříliš intuitivně.

Pomyslným vítězem srovnání je **LinDA** a **ViziQuer**, kteří nabízejí nejvíce funkcionality, jsou celkem snadní na používání a jsou hezky vizualizovány.

6 Návrh

V následující kapitole bude probrán návrh rozšíření aplikace Sparkle o vizuální tvorbu dotazu typu SELECT. Většina nástrojů se právě tímto dotazem zaobírala nejvíce a jde také o nejpoužívanější typ dotazu. Jako inspirace pro nové rozšíření budou nástroje LinDA a ViziQuer, které svou funkcionalitou dominují nad ostatními nástroji. Nástroj **Sparklis** je vyřazen kvůli své odlišnosti v budování dotazu, která je odlišná od způsobu, kterým se práce snaží řešení dosáhnout.

6.1 Funkční požadavky

Následující body vystihují základní požadavky na novou funkcionalitu:

- **Vizuální dotazování** - uživatel by neměl psát SPARQL dotaz, ten se musí generovat z vizualizace.
- **Vybírání proměnných** - uživatel na začátku vybere to, co ho zajímá, aplikace z toho připraví dotaz.
- **Čitelný design vizualizace** - vizualizace by měla být přehledná a dbát na možnost přetečení obrazovky při velkém počtu proměnných.
- **Intuitivní pro laiky** - uživatel, který nezná nebo si není jistý schématem, bude moci vytvořit dotaz tak, aby z dat získal to, co ho zajímá.
- **Podmínky** - možnost nastavení, jestli dané atributy budou OPTIONAL či jestli se budou zobrazovat ve výsledcích
- **Filtry** - možnost nastavení filtrace atributů - řetězec, číslo a data
- **Agregace** - možnost nastavení agregace atributů - základní SPARQL funkce
- **Načítání dat z endpointu** - aplikace data musí stahovat z endpointu určeného uživatelem
- **Řazení** - řazení proměnných včetně přetypování
- **Výsledky** - běh dotazu a získání výsledků v základních formátech tabulky, JSON a CSV

6.1.1 Třídy a atributy

Aplikace získá třídy a jejich jednotlivé atributy ze SPARQL endpointu. Tyto dva seznamy poté nabídne uživateli k výběru. Uživatel přidává danou třídu a atribut do dotazu jejím výběrem. Pro uživatelův komfort je nutné myslet na dva zásadní problémy:

- Automatický výběr třídy atributu - uživatel vybral atribut, jehož třída nebyla předtím vybrána. Je tedy nutné vybrat danou třídu do vizualizace automaticky.
- Spojitost vybraných tříd - při vybírání tříd a atributů uživatel nemá povinnost přemýšlet nad schématem. To znamená, že toto musí řešit aplikace. Při vybrání nové třídy je nutné vyhledat potenciální cesty mezi již vybranými třídami a novým přírůstkem. Pokud taková cesta existuje, je nutné jí zahrnout do dotazu společně se všemi třídami, kterými prochází.

Třídy je možno vybírat vícekrát, atributy jednou.

6.1.2 Další informace o třídách

Při kliknutí na třídu aplikace zobrazí další informace o dané třídě, jako je *label* či *comment*. Jde o vlastnosti anotačního typu a často se k třídám váží jako doplňující informace. Protože v datech často k dispozici nejsou, je nutné je získat z daných ontologií. Aplikace načte ontologie, vyhledá v nich třídu a získá její anotační vlastnosti.

6.1.3 Filtry

Filtry nebo také podmínky je nutné specifikovat pro jednotlivé atributy. Typ filtru je dán datovým typem atributu. Časté typy jsou řetězce, čísla a data. Aplikace podporovat filtry (inspirováno zejména LinDA):

- **Řetězec**
  - *contains* - řetězec obsahuje podřetězec
  - *is equal to* - řetězec odpovídá řetězci
  - *not equal to* - řetězec neodpovídá řetězci
  - *starts with* - řetězec začíná podřetězcem
  - *ends with* - řetězec končí podřetězcem

  - *regex* - obecný regulární výraz

- **Číslo a Datum**
  - = - rovno hodnotě
  - < - menší než hodnota
  - <= - menší nebo rovno hodnotě
  - > - větší než hodnota
  - >= - větší nebo rovno hodnotě

- **URI** - bude naloženo jako s řetězci

Atribut bude mít na výběr všechny možnosti s tím, že při výběru nekompatibilního typu dojde k přetypování v dotazu.

6.1.4 Agregace

Aplikace bude podporovat základní agregace COUNT, AVG, SUM, MIN a MAX.

6.1.5 Řazení

Řazení se obyčejně provádí ve chvíli, kdy již uživatel vybral všechny požadované atributy jako poslední věc před samotným dotazováním. Uživatel by měl mít možnost vybrat z vygenerovaných proměnných ty, které ho zajímají, a zařadit je do řazení. U každé proměnné určí, jestli je chce řadit sestupně či vzestupně včetně přetypování a nakonec je bude moci seřadit.

6.1.6 Ostatní

Aplikace umožní nastavit dotazu LIMIT a OFFSET s tím, že je implicitně nepoužívá. Dotaz by měl být generován při samotném vytváření pro okamžitou zpětnou vazbu. Zároveň by měla aplikace oznamovat, pokud dotaz vrací nějaké výsledky, nebo poslední změna znamenala, že žádná data danému dotazu již neodpovídají. Tohoto lze snadno dosáhnout limitem dotazu řekněme **LIMIT 5** a zkouška, jestli se vrací nějaký počet záznamů. Tato operace je nenáročná a nenaruší plynulý chod aplikace.

6.2 Technologie

Rozšíření aplikace v Javě je možné dvěma způsoby - Javou anebo také Javascriptem.

6.2.1 Java

Celkem logickou úvahou by bylo rozšířit aplikaci v Javě rozšířením napsané také v Javě. V tomto případě by šlo o přidání dalšího modulu JavaFX [101]. Za dobu napsání aplikace se Java posunula dále a v době psaní této diplomové práce je ke stažení Java verze 12 [102]. Tato verze již neobsahuje automaticky modul JavaFX - je nutné stáhnout jeho aktuální verzi (v současné době 12) z jejich stránek, nebo je přidat jako Maven moduly [103]. V době psaní této práce tato verze právě vyšla. Opravila nemalé množství chyb s tím, že spousta jich je na GitHub stránkách projektu otevřených. Jde o velice čerstvou a turbulentní věc, která se bude ještě často měnit. Teoretická migrace na novou verzi vyžaduje nemalé úsilí vzhledem k nekompatibilitě knihoven třetích stran a nutnosti zkontrolovat všechny použité knihovny.

Samotná vizualizace se specifickými nároky, které na ní klademe, není v JavaFX snadná věc. JavaFX nabízí několik možností pro zobrazení diagramů [104], ty se ale zabývají spíše vizualizací dat než interaktivní dynamickou komponentou, která je potřeba. V rámci třetích knihoven pro JavaFx není přílišný výběr:

- **Prefuse** - sada nástrojů pro data modelování, vizualizaci a interakci. Vytvořeno pro Swing, předchůdce JavaFX a velice dlouho nevyvíjeno [105]. Jejich domovské stránky byly dokonce zabrány jinou firmou (`https://prefuse.org/`).
- **JGraphX** - zabývá se vizualizací dat, podpora pro Swing, není již aktivně vyvíjeno. Vhodné pro grafy založené na uzlech a hranách. Poskytuje také Javascript verzi [106].
- **yFiles** - používá JavaFX třídy pro interaktivitu a vizualizaci různých dat. Placené [107].

Nejpoužitelnější z těchto nástrojů se jeví JGraphX, jeho nevýhodou je zastavený vývoj a minimální podpora.

6.2.2 Javascript

Další možností je odpoutat se od Javy a využít další moderní technologii - Javascript. Rozšíření aplikace by pak bylo možno spustit nezávisle na Sparkle v moderním prohlížeči. Javascript by také umožnil využít knihovnu D3.js [108].

D3.js je knihovna umožňuje navázat data na Document Object Model (DOM) a pak aplikovat transformace na tento dokument založených na datech.

Pomocí D3 se dá vytvořit HTML tabulka z pole čísel nebo interaktivní sloupcový diagram z těch samých dat. Veškerá manipulace dokumentů je založená na datech, D3 tedy naplno využívá webové standardy HTML, SVG a CSS [108].

Při vyhledávání vizualizace v Javascriptu člověk dříve nebo později na D3 narazí. Kolem knihovny se vytvořila velká komunita, jak dokazuje galerie D3 [109].

6.2.3 Propojení Java a Javascript

Od Javy 8 je součástí každé verze i Javascript engine, Nashorn [110]. Díky tomuto enginu není problém k Java aplikaci připojit skripty v Javascriptu a nechat je běžet společně s Java kódem. Nashorn podporuje ECMAScript páté edice [111], přičemž poslední verze je již devátá [112], což může znamenat absenci některých funkcionalit jako například anonymní funkce, deklarace proměnných `let` či některé funkce jako `includes` nad řetězci či `assign` nad objekty. Tyto nedostatky se však dají obejít.

Nashorn byl označen za `deprecated` od verze JDK 11 [113], ale zatím do dalších verzi Javy nebyl přidán plnohodnotný nástupce. Nejpravděpodobnějším kandidátem se jeví GraalJS [114], který je součástí univerzálního virtuálního stroje GraalVM [115], jež podporuje další jazyky jako Python, Ruby, R, Scala, samotnou Javu či C++. GraalVM je dostupný pouze pro Linux a Mac [115]. Kromě zmíněn nedostupnosti na Windows je další nevýhodou nulová podpora prohlížečů a tedy i JavaFX komponenty `WebView`.

Samotné propojení obou jazyků je jednoduché. V případě JavaFX je možnost využít WebView komponentu, která je na to vybavena třídou `WebEngine`. Této třídě stačí předat HTML a Javascript soubory a ona se již postará o zbytek. Tímto způsobem je možné simulovat webový prohlížeč v JavaFX okně.

Třídě WebEngine se předávají další třídy přes tzv. `JSObject` třídu. Javascript pak může volat metody těchto tříd ze svého kódu, je však třeba dát pozor na přetížené metody - Nashorn nebude vědět, kterou z nich má zavolat a tak bude volání mezi nimi střídat, což může vést k neočekávaným chybám.

Renderování HTML a především vizualizace bude také pomalejší než v prohlížeči. Nashorn neimplementuje XMLHttpRequest specifikaci a HTML 5 specifikaci (funkce jako `setInterval` a `setTimeout`).

6.3 Napojení na SPARQL endpoint

Aplikace bude dostávat data ze SPARQL endpointu a proto je nutné se na něj napojit. Pro tento účely je možné využít sílu Javascriptových knihoven, v tomto případě jQuery, konkrétně její funkci `ajax` [116]. Nyní stačí jen specifikovat **url** endpointu, přidat dotaz a specifikovat formát výsledků a knihovna se již postará o zbytek.

6.4 Vizualizace

Vizualizace aplikace bude následovat stěžejní body:

- **Zřetelná** - na první pohled bude zřejmé, co vizualizace zobrazuje.
- **Minimalistická** - zobrazování příliš mnoho věcí zaplňuje obrazovku. Je třeba ukázat co nejméně.
- **Interaktivní** - s uzly bude možno manipulovat a vybírat je.

Nástroj se vyhýbá chybám ostatních vizuálních nástrojů - většina se snažila zobrazit příliš mnoho věcí na jednom místě a vizualizaci tak zaplnila velkým množstvím informací a učinila vizualizaci nepřehlednou. Poučení nabízí i nástroje vizualizace RDF, které jednotlivé třídy a atributy zobrazují jako jeden jednoduchý geometrický tvar - většinou kruh. Po kliknutí na tento tvar je místo a prostor pro zobrazení dalších detailů.

Vizualizace bude obsahovat pouze kruhy pro jednotlivé třídy. Vzájemně propojené třídy budou ve vizualizaci propojeny hranami. Hrany i uzly budou popsány názvem dané třídy či relace. Veškeré další detaily - další informace, atributy, agregace, filtry - budou zobrazeny po vybrání daného uzlu. Právě vybraný uzel by měl být nějakým způsobem zvýrazněn.

Na další designové aspekty vizualizace nejsou kladeny další nároky.

6.5 Obsluha

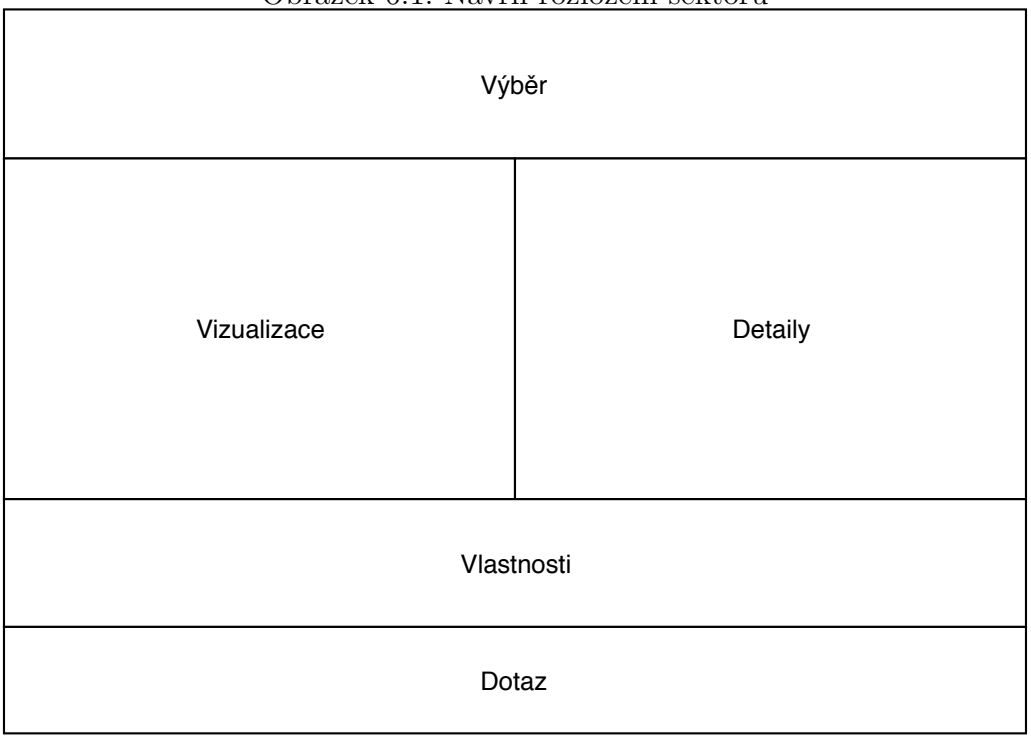
Nástroj bude rozdělen do několika sektorů:

- **Výběr** - tento sektor bude nabízet k vybrání třídy a atributy získané ze SPARQL endpointu. Pokud není žádný endpoint načten, uživatel by jej zde měl specifikovat, čímž otevře zbytek aplikace.
- **Vizualizace** - zde se budou zobrazovat vybrané třídy a jejich relace.

- **Detaily** - zde budou další informace o vybraném uzlu, jeho atributy, agregace a filtry. Z praktického důvodu zde budou také možnost výběru atributů dané třídy.
- **Vlastnosti** - poslední funkcionální sektor, zde budou další vlastnosti dotazu typu řazení či limitu a offsetu SPARQL dotazu.
- **Dotaz** - zde bude zobrazen vygenerovaný dotaz a bude nabídnuta možnost nechat jej běžet proti endpointu.

Rozmístění jednotlivých sektorů po obrazovce je vyobrazeno na obrázku 6.5. Uživatel začne v horní části, kde vybírá jednotlivé třídy a atributy. Dotaz je mu postupně vizualizován v sektoru vizualizace. Po kliknutí na některou ze tříd jsou mu zobrazeny detaily. Po vybrání všeho, co ho zajímá, uživatel se přesune do dolní části a má možnost nastavit některé vlastnosti dotazu, než se dostane k samotnému dotazu, jež nyní má možnost vidět a spustit.

Obrázek 6.1: Návrh rozložení sektorů



Uživatel je nucen napsat na začátku daný SPARQL endpoint, na který se aplikace bude nadále dotazovat, a pak by si již měl vystačit pouze s klikáním na nabízené třídy a atributy. Další výjimku potom tvoří filtry a agregace, které je nutno částečně doplnit ručně. Data často obsahují velké množství trojic, je dobré poskytnout našeptávač, který daný výběr zkrátí, tedy např. je

hledán atribut *Patient_s_name*, uživatel napíše *Patient* a výběr se mu zúží na ty hodnoty, které tomuto hledání odpovídají. Toto bude mít za následek zvýšení použitelnosti aplikace i při velkém množství tříd/atributů.

7 Implementace

Aplikace rozšiřuje původní aplikaci v Javě za použití Javascriptu, přesněji řečeno Nashorn engine [110], a musí se tedy pohybovat v jeho vymezených omezeních, které byly probrány v návrhu. Pro implementaci to znamená o něco obtížnější debug a spoléhání na konzoli moderních prohlížečů.

Aplikace bude rozšířením původní Java aplikace, zároveň ale její zdrojové soubory lze spustit natažením do prohlížeče. Závislost na Sparkle je tedy minimální a spíše podpůrného charakteru, kdy aplikaci poskytuje jen několik základních funkcí a o většinu se stará sama. Tomu odpovídá i struktura aplikace.

7.1 Knihovny

Aplikace využívá několika málo knihoven, které vypomáhají s určitou částí funkcionalit aplikace. Jde o:

- **Bootstrap** [117] - CSS knihovna pro layout a styl stránky a jednotlivých HTML komponent. Použití této knihovny znamená minimální ruční psaní CSS stylů.
- **jQuery** [118] - Javascript knihovna pro zjednodušení přístupu k HTML dokumentu a Ajax asynchronní volání. Využívá také Bootstrap pro některé své komponenty.
- **d3** [108] - Javascript knihovna pro vizualizaci dat, v případě aplikace tedy grafové reprezentace dotazu.
- **select2** [119] - Javascript knihovna, jež rozšiřuje možnosti HTML select komponenty o možnost textového vyhledávání. Využito pro většinu rozevíracích nabídek v aplikaci

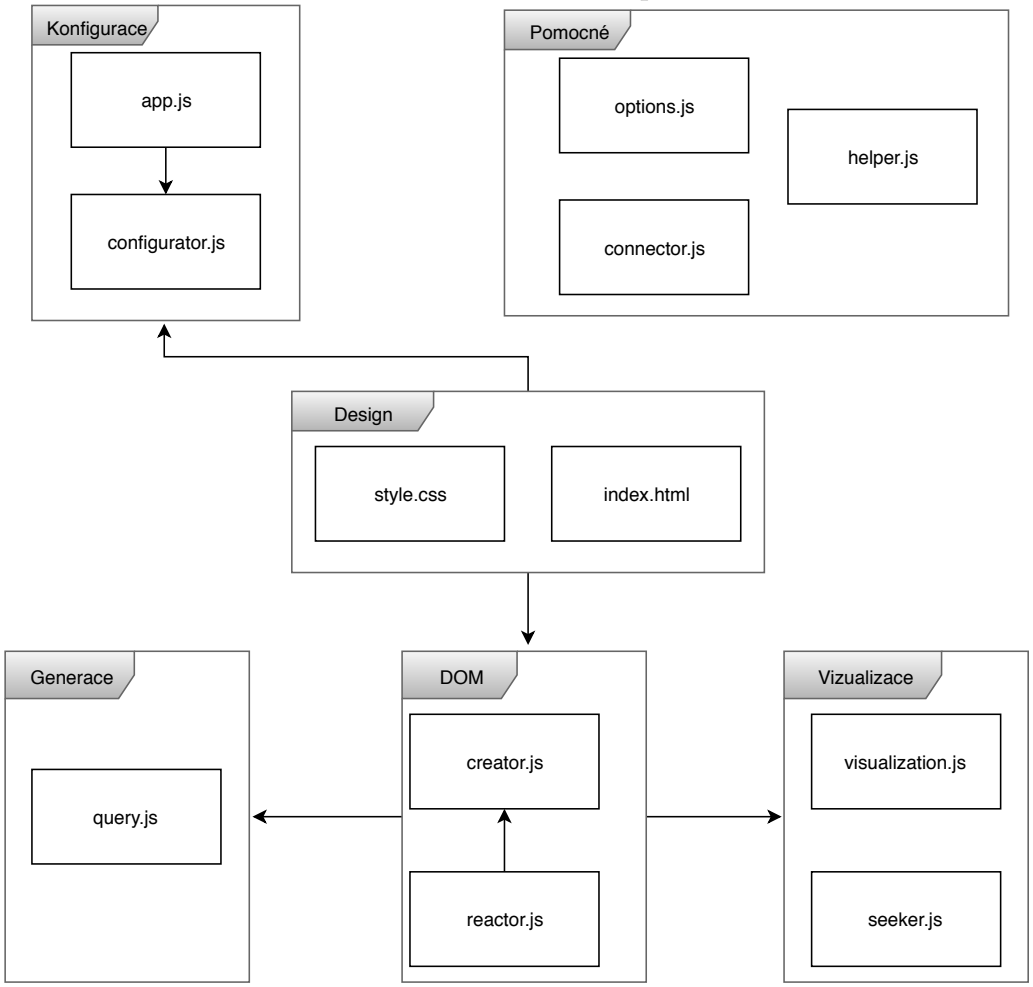
Veškerá další funkcionalita je již jen HTML, CSS a Javascript psaný autorem.

7.2 Struktura aplikace

Nashorn nedává příliš mnoho výběru jak aplikaci strukturovat. Nejjednodušší způsob je použití čistého Javascriptu, HTML stránky a případné

knihovny třetích stran pro nějakou další funkcionalitu. Diagram struktury aplikace je uveden na obrázku 7.1.

Obrázek 7.1: Struktura aplikace



Aplikace se logicky dělí na několik segmentů dle funkcionality, kterou jednotlivé skripty obstarávají. Segmenty nejsou totožné se sektory rozložení stránky probrané v předchozí kapitole. Na nejvyšší abstraktní úrovni by se dalo mluvit pouze o dvou segmentech - Design (HTML a CSS) a Logika (Javascript). Logika se dá rozdělit na více segmentů dle funkcionality a to bude probráno v další části.

7.2.1 Design

Design aplikace je tvořen pouze dvěma soubory - HTML stránkou `index.html` a souborem s CSS styly `style.css`. Layout HTML stránky odpovídá navrženému rozdělení designu na sektory, čehož je dosaženo za použití

knihovny Bootstrap pomocí předdefinovaných HTML `class` atributů. Tato část neobsahuje žádnou logiku. HTML elementy nutné pro logiku jsou vždy označeny pomocí atributu `id`. Veškeré Javascript skripty a knihovny jsou načítány v dolní části `body` elementu.

Jednotlivé sektory se dají zhruba rozdělit dle Bootstrap panelů. Vizualizace na rozdíl od ostatních využívá HTML element `svg`. Všechny panely jsou dle potřeby obaleny kontejnerem. Aplikační layout je tedy:

- Panelová skupina výběru
  - Panel zpráv (zobrazuje některé důležité zprávy pro uživatele)
  - Panel endpointu (umožňuje vybrat SPARQL endpoint k napojení)
  - Panel výběru tříd a atributů
- SVG vizualizace
- Panelová skupina detailu
  - Panel informací
  - Panel objektových vlastností
  - Panel atributů
  - Panel filtrů
  - Panel agregací
- Panelová skupina vlastností
  - Panel řazení
  - Panel ostatních
- Panel dotazu

Jednotlivé panely budou více vysvětleny v části o sektorech, ke kterým patří.

7.2.2 Pomocné

Tvořeno třemi Javascript skripty - `options.js`, `connector.js` a `helper.js`. Jak název napovídá, jde o pomocné skripty, jež se používají v rámci celé aplikace.

Skript `connector.js` zajišťuje spojení se SPARQL endpointem a načítání ontologií a také obsahuje funkce na převedení výsledků dotazů do vnitřních objektů aplikace.

Skript `options.js` obsahuje některé předdefinované věci, jako je seznam často používaných prefixů, ontologií k nahrání a další.

Skript `helper.js` obsahuje pomocné funkce. Často jde o hledání v rámci polí či kontrola některých vlastností objektů.

7.2.3 Konfigurace

Základní páteř aplikace, jež je spuštěna při nahrání HTML stránky. Tvořeno dvěma Javascript skripty - `app.js` a `configurator.js`.

Skript `app.js` inicializuje stránku a připravuje sektor Výběru pro `configurator.js`.

Skript `configurator.js` obstarává načtení tříd a jejich vlastností ze SPARQL endpointu do aplikace.

7.2.4 DOM

Skripty v tomto segmentu pracují hodně s HTML stránkou a její reprezentací - DOMem (Document Object Model).

Skript `reactor.js` přidává logiku do HTML stránky pomocí funkcí `onclick`, `onchange` a podobně. Často musí zavolat další skript v segmentu, `creator.js`. Ten vytváří nové DOM objekty a přidává je do HTML stránky. Více bude vysvětleno v dalších kapitolách.

7.2.5 Vizualizace

Segment vizualizace obstarává vizualizaci dotazu - za použití knihovny D3 tedy vykresluje postupně vytvářený dotaz. O to se stará Javascript skript `visualization.js`. Druhý skript `seeker.js` obsahuje funkce pro prohledávání a hledání cest v grafu vizualizace, o čemž bude více řeč v další části práce.

7.2.6 Generování

Obsahuje jediný skript `query`. Tento skript obstarává generování SPARQL dotazu.

7.3 Propojení se Sparkle

Protože rozšíření aplikace nezávisí na Sparkle, zásahy do původní aplikace mohou být minimální, ale je třeba určité změny provést. Vedle formulářového

a textového editoru byla přidána záložka `Visual`, která ukáže tuto aplikaci. Zobrazení další záložky byla záležitost relativně malé změny FXML souboru, samotné spuštění aplikace vyžaduje použití třídy `WebView`, což je komponenta JavaFX schopná renderovat HTML, CSS a Javascript. Do aplikace se přidává jako ostatní komponenty.

Komponenta obsahuje `WebEngine`, což je interní třída, jež umožňuje renderovat webové stránky (podobné jádrům prohlížečů). Této třídě stačí předat `index.html` z aplikace a ta ho již spustí jako klasické prohlížeče přímo v JavaFX okně. Pro vzájemnou komunikaci mezi Javascript a Java částí je třeba ještě nutno přidat `JSObject`. Přes tento objekt je pak možné předávat Java třídy do `WebEngine` a jejich metody mohou pak být volány z Javascript kódu. Konkrétně jde o třídy `DataAgent`, `QuerySaver`, `OntologyLoader`, `ClassConsumer` a `ClassList`.

Třída `DataAgent` slouží v aplikaci Sparkle pro komunikaci s datovým úložištěm a využívá knihovnu Apache Jena [120]. Pokud je rozšíření puštěno jako součást Sparkle, využívá tuto třídu také. Pro uložení výsledků dotazů je používán `ClassConsumer`, jenž výsledky v Javě parsuje a ukládá je do `ClassList` seznamu, odkud je pak Javascript může načíst.

Třída `QuerySaver` slouží pro uchovávání dotazu v podobě řetězce, aby šlo předávat vygenerovaný dotaz z vizualizační části do formulářového a textového editoru. Třída obsahuje jednoduchý getter a setter, jejichž voláním pak může v aplikaci dojít k předání dotazu od nové vizualizační části do ostatních editorů.

Třída `OntologyLoader` načítá ontologie přidané k aplikaci a následně je vrací v podobě řetězce. V Javascript části potom stačí tento řetězec parsovat zabudovaným `DOMParser`. Sama Java XML neparsuje, protože jde o dva různé XML parsery, které nejsou kompatibilní, a v aplikaci by pak ontologie nešly nijak využít. Třída načítá ontologie do mapy, kde klíčem je URL dané ontologie.

7.4 Segment výběru

V následující části budou postupně probrány jednotlivé sektory vzhledem k návrhu aplikace v kapitole 6.5 a funkcionalita, kterou obstarávají.

Segment výběru slouží pro tři základní věci - výběr SPARQL endpointu, výběr tříd/atributů a případný reset výběru.

7.4.1 Vybrání SPARQL endpointu

Na začátku je aplikace inicializovaná v rámci skriptu `app.js`. V podstatě jde o ukrytí všech ostatních panelů kromě panelu výběru SPARQL endpoint. Pokud je aplikace spuštěná přes Sparkle, do textového pole pro endpoint je automaticky vložena hodnota SPARQL endpointu, se kterou byl Sparkle spuštěn. Nezávisle spuštěná aplikace je nutné tento endpoint ručně doplnit.

Panel dále obsahuje tlačítko **Connect**. Po stisknutí se aplikace pokusí spojit s vybraným SPARQL endpointem. Zároveň se odkryje zbytek aplikace.

7.4.2 Napojení na SPARQL endpoint

Spojení s endpointem obstarává funkce `ajax` [116] knihovny jQuery [118]. Jde o asynchronní požadavek na vzdálený server, proto je třeba počítat s čekáním na výsledky pomocí promisy. URL požadavku je vytvořeno dle vzorce:

```
endpointUrl + "?query=" + encodeURIComponent(query) + "&format=
    json";
```

Kde `query` je daný SPARQL dotaz, `endpointUrl` je URL adresa SPARQL endpointu a `encodeURIComponent` je funkce pro escapování znaků v URL. Takto formované URL je předáno `ajax` funkci. Výsledek je očekáván v JSON formátu, jenž se v Javascriptu dobře parsuje. Požadavek může skončit chybou nebo timeoutem, v tom případě je informace o chybě vypsána uživateli a do konzole.

7.4.3 Načtení ontologií

Po zadání endpointu je kromě tříd a vlastností potřeba načíst i ontologie, kde je třeba získat další informace o třídách a vlastností. Využito je toho, že ontologie MRE projektu jsou dostupné přes HTTP svými URI jmennými prostory (s malou změnou protokolu HTTP za HTTPS). Ve skriptu `options` je připraveno pole `ontologies`, jež obsahuje objekty s parametry uvedených v tabulce 7.1.

Tabulka 7.1: Objekt ontologie

Vlastnost	**Popis**
nameSpace	Jmenný prostor dané ontologie
url	URL adresa ontologie
xml	Nahraná ontologie v XML formátu

Vlastnosti objektu `nameSpace` a `url` jsou nastavené v aplikaci ve skriptu `options.js`. Přes skript `connector.js` pak stačí pro jednotlivé ontologie načíst jejich XML dokument a uložit jej do objektu ontologií. Jde o asynchronní operaci, proto je zde využito Javascript promis a zbytek aplikace čeká na dokončení načítání ontologií. Parsování XML dokumentu provádí samotná knihovna pro spojení `ajax`.

V případě napojení na Sparkle je situace o něco komplikovanější. Prohlížeč dokáže stáhnout i větší množství souborů ze vzdáleného serveru přiměřeně rychle, v Javě je však tato operace velice náročná a místo několika sekund trvá desítky sekund. Proto byly ontologie přidány jako zdroje k aplikaci, která je může rychle nahrát z disku. K tomu je využita Java třída `OntologyLoader`, kdy v Javascript části stačí jen předat URL ontologie, na kterou třída vrací její XML dokument v podobě řetězce. Tento řetězec pak stačí parsovat zabudovaným `DOMParser` parserem.

Aplikace je nakonfigurována načíst ontologie *mre*, *dasta*, *dsm*, *dscl*, *form*, *image*, *image-mapping*, *nihss*, *pop*, *sits* a *stroke*. Všechny jsou z projektu MRE.

7.4.4 Načtení tříd

Načtení všech tříd vyskytujících se v datech je možno za pomoci SPARQL dotazu:

```
SELECT DISTINCT ?class ((count(?x)) AS ?cnt)
WHERE {
  ?x a ?class .
}
GROUP BY ?class
ORDER BY DESC(?cnt)
```

Dotazem je získán seznam tříd, přesněji řečeno jejich URI, seřazené sestupně dle počtu jejich výskytu v aplikaci. Řazení odpovídá pořadí, v jakém jsou pak nabídnuta uživateli. Řazení dle počtu výskytu je určitou metrikou důležitosti tříd.

Vytvoření vnitřního objektu třídy bude ukázáno na příkladu. Budeme mít data, která obsahují jednu třídu, např. `LaboratoryReport`, potom výsledkem tohoto dotazu bude JSON řetězec:

```
{
 "head": { "vars": [ "class" , "cnt" ] } ,
 "results": {
   "bindings": [
  {
```

```
  "class": { "type": "uri" , "value": "http://mre.zcu.cz/
     ontology/dasta.owl#LaboratoryReport" } ,
  "cnt": { "type": "literal" , "datatype": "http://www.w3.org
     /2001/XMLSchema#integer" , "value": "1003" }    }
   ]
 }
}
```

Vybrané proměnné jsou známé, proto se nemusíme starat o část `head` a jedinou zajímavou částí je `results.bindings`, kde jsou v poli dodané výsledky, v tomto případě jedna třída. Jsou vyjmenované proměnné a informace k nim - jejich typ, hodnota a v případě literálu datový typ. Z výsledků se vytvářejí vnitřní třídy. Objekty tříd se ukládají do pole `classes`, se kterou potom jednotlivé skripty často pracují. Podoba tohoto objektu je v tabulce 7.2.

Atribut `localName` je čitelný název třídy a jde o poslední část řetězce URI, v našem případě tedy `LaboratoryReport`. Atribut `nameSpace` je část URI bez lokálního názvu, tedy vše před `LaboratoryReport`. Atribut `uri` je celé URI, které jsme získali z dotazu, to samé `count`. Pole `attributes` a `objectProperties` jsou prozatím prázdné a budou později naplněny. Logický atribut `selected` slouží pro přehled, která třída je aktuálně vybrána a bude důležitá pro vizualizační část. Identifikátor třídy je číselný index a odpovídá pozici dané třídy v poli tříd `classes`. Slouží pro pomocné metody. Atribut `show` pak určuje, jestli je URI třídy zařazeno do dotazu. Atribut `aggrCount` přepíná agregační funkci COUNT pro třídu a je důležitý pro generování dotazu.

Tabulka 7.2: Objekt třídy

Vlastnost	**Popis**
localName	Lokální název třídy, čitelný člověkem
nameSpace	Jmenný prostor třídy
uri	URI třídy
count	Počet výskytu třídy v datech
attributes	Vlastnosti
objectProperties	Objektové vlastnosti
selected	Jestli je třída vybrána pro dotaz
id	Identifikátor třídy
show	Zobrazení třídy v dotazu
aggrCount	Zobrazení agregační funkce COUNT třídy

K atributům se později přidávají další, ale objekt je na začátku uložen do

polí tříd. Příklad naplněného objektu nalezenou třídou `LaboratoryReport` v tabulce 7.3.

Tabulka 7.3: Příklad objektu třídy

Vlastnost	**Hodnota**
localName	LaboratoryReport
nameSpace	http://mre.zcu.cz/ontology/dasta.owl
uri	http://mre.zcu.cz/ontology/dasta.owl#LaboratoryReport
count	1003
attributes	[]
objectProperties	[]
selected	false
id	0
show	false
aggrCount	false

7.4.5 Prefixy

Před načtením vlastností je třeba připravit prefixy. Není vyloučeno, že mohou existovat dvě třídy se stejným lokálním jménem a jediný jejich rozdíl bude v jmenném prostoru. Protože jmenný prostor je velice dlouhý řetězec a vypisovat jej celý jen zahltí obrazovku a uživatele, budou tyto jmenné prostory zkracovány na prefix.

Objekty prefixů jsou uloženy v poli `prefixes` ve skriptu `options.js`, jejich vlastnosti zobrazeny v tabulce 7.4. Atribut `prefix` je samotný prefix, který se používá pro zkrácení jmenného prostoru ontologie (`uri`). Atribut `uriSPARQL` zjednodušuje pozdější volání prefixů při generování dotazu. Jmenné prostory tříd při konfiguraci neobsahují separátní znak, proto je atribut `uriClass`.

Tabulka 7.4: Objekt prefixu

Vlastnost	**Popis**
prefix	Prefix dané ontologie
uri	URI ontologie, jmenný prostor
uriClass	URI ontologie bez separátního znaku
uriSPARQL	URI ontologie jak by se použila v dotazu SPARQL

Nejlépe bude ukázat na příkladu. V aplikaci je prefix velice často používaného RDF slovníku v tabulce 7.5.

Tabulka 7.5: Příklad objektu prefixu

Vlastnost	**Hodnota**
prefix	rdf
uri	http://www.w3.org/1999/02/22-rdf-syntax-ns#
uriClass	http://www.w3.org/1999/02/22-rdf-syntax-ns
uriSPARQL	<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

Na začátku aplikace jsou připraveny objekty tříd pro často používané slovníky, jmenovitě *rdf*, *rdfs*, *skos*, *xsd*, *foaf* a *owl*.

Další prefixy je nutné doplnit v průběhu aplikace. Po načtení ontologií a tříd aplikace otevře XML reprezentace ontologií a dynamicky přidá prefixy, které najde. Na začátku ontologií jsou často, ale ne vždy, definovány prefixy používané v rámci ontologie. Tyto prefixy aplikace načítá a tak dostává stejnou podobu prefixů, na kterou je uživatel dané ontologie zvyklý. Pokud v průběhu programu aplikace narazí na jmenný prostor ontologie, ke které nemá připravený prefix, jednoduše vytvoří vlastní ve formátu `prx` + `index`, který začíná na nule. Tímto se zajistí, že nikdy nedojde k tomu, aby nějaký jmenný prostor neměl přidělený prefix.

Po přípravě prefixů je prostor přidělit je jednotlivým třídám. Každá třída má připravený atribut `nameSpace`, podle čehož poznáme daný prefix. Pak ke třídě přidělíme dva nové atributy, zobrazeno v tabulce 7.6.

Tabulka 7.6: Přidání prefixových atributů k třídě

Vlastnost	**Popis**
prefix	Prefix
prefixedLocalName	Prefixované lokální jméno třidy

Atribut `prefix` odpovídá atributu `prefix` v objektu Prefix, prefixované lokální jméno je prefix + : + lokální jméno. V příkladě `LaboratoryReport` ukázka v tabulce 7.7.

Tabulka 7.7: Příklad přidání prefixovaných atributů k třídě

Vlastnost	**Popis**
prefix	ds
prefixedLocalName	ds:LaboratoryReport

7.4.6 Načtení vlastností

Po načtení tříd jsou načteny jejich vlastnosti. Je využit SPARQL dotaz, kde `class uri` je vyplněn URI dané třídy:

```
SELECT DISTINCT ?property (count(?x) AS ?cnt)
WHERE {
  ?x a <uri class> .
  ?x ?property ?o .
}
GROUP BY ?property
ORDER BY DESC(?cnt)
```

JVlastnosti jsou řazeny sestupně dle jejich četnosti. Uvedu příklad vlastnosti. Představme si, že načtená třída `LaboratoryReport` má jednu vlastnost, kterou nám vrátí SPARQL endpoint:

```
{
  "head": { "vars": [ "property" , "cnt" ] } ,
  "results": {
    "bindings": [
    {
  "property": { "type": "uri" , "value": "http://mre.zcu.cz/
      ontology/dasta.owl#datetimeEvent" } ,
  "cnt": { "type": "literal" , "datatype": "http://www.w3.org
      /2001/XMLSchema#integer" , "value": "1003" }
    }
   ]
 }
}
```

Opět je třeba si všímat pouze `results.bindings`. Výsledek je v podstatě totožný s dotazováním na třídy. Výsledný objekt je sestaven s atributy uvedenými v tabulce 7.8.

Tabulka 7.8: Objekt vlastnosti

Vlastnost	**Popis**
localName	Lokální jméno vlastnosti
nameSpace	Jmenný prostor vlastnosti
uri	URI vlastnosti
count	Počet výskytů vlastnosti
aggregations	Agregace
filters	Filtry
selected	Jestli je vlastnost vybrána pro dotaz
optional	Jestli je vlastnost OPTIONAL
show	Jestli je vlastnost ukázána v dotazu
clicked	Jestli je vlastnost vybrána, důležité pro filtry a agregace

Pro atributy `localName`, `nameSpace`, `uri`, `count` a `selected` znamenají

to samé co u tříd. Navíc jsou zde `aggregations` a `filters` pro agregace a filtry, na začátku prázdné. Atribut `optional` určuje, jestli je atribut označen jako OPTIONAL. Atribut `show` určuje, jestli se ve výsledcích atribut zobrazí. Atribut `clicked` bude více popsán v části pro filtry a agregace.

Protože už jsou v této chvíli načtené prefixy, jsou okamžitě přidány k vlastnostem ve stejném duchu jako třídy (tedy atributy `prefix` a `prefixedLocalName`). Ukázka objektu vlastnosti v tabulce 7.9.

Tabulka 7.9: Objekt vlastnosti příklad

Vlastnost	**Hodnota**
localName	datetimeEvent
nameSpace	http://mre.zcu.cz/ontology/dasta.owl
uri	http://mre.zcu.cz/ontology/dasta.owl#datetimeEvent
count	1003
prefix	ds
prefixedLocalName	ds:datetimeEvent
aggregations	[]
filters	[]
selected	false
optional	true
show	true
clicked	false

7.4.7 Další informace

Pro doplnění dalších informací k třídám a vlastnostem je nutno podívat se do relevantních ontologií, tyto informace často nezjistíme přímo u dat.

Pro každou třídu je uložen její jmenný prostor, jenž odpovídá jmennému prostoru ontologie, které se v aplikaci načetly již předtím. Stačí projít ontologie a najít relevantní údaje k nahraným třídám a vlastnostem. Uvedu příklad definice třídy `LaboratoryReport` (zkráceno na zajímavé atributy):

```
1 <owl:Class rdf:about="http://mre.zcu.cz/ontology/dasta.owl#
      LaboratoryReport">
2   <rdfs:subClassOf rdf:resource="http://mre.zcu.cz/ontology/
        dasta.owl#ClinicalEvent"/>
3   <rdfs:comment xml:lang="cs">Sdělované laboratorní vyšetření pacienta vázané
        k jedné klinické události.</rdfs:comment>
4   <rdfs:label xml:lang="cs">Laboratorní vyšetření</rdfs:label>
5   <rdfs:label>Laboratory report</rdfs:label>
6   <rdfs:label xml:lang="en">Laboratory report</rdfs:label>
7 </owl:Class>
```

Atributy `rdfs:label` a `rdfs:comment` jsou anotační vlastnosti a blíže třídu popisují. V datech se mohou a nemusí vyskytovat a proto je třeba je najít zde. Dále je vidět, že třída je podtřídou třídy `ClinicalEvent`. Toto bude důležité později. Z těchto definic pro třídy si aplikace vezme anotační vlastnosti, které bude zobrazovat v segmentu detailů.

U vlastností je to podobné, například vlastnost `datetimeEvent`:

```
1  <owl:DatatypeProperty rdf:about="http://mre.zcu.cz/ontology/
       dasta.owl#datetimeEvent">
2    <rdfs:domain rdf:resource="http://mre.zcu.cz/ontology/dasta.
       owl#Anamnesis"/>
3    <rdfs:domain rdf:resource="http://mre.zcu.cz/ontology/dasta.
       owl#ClinicalEvent"/>
4    <rdfs:domain rdf:resource="http://mre.zcu.cz/ontology/dasta.
       owl#Diagnosis"/>
5    <rdfs:domain rdf:resource="http://mre.zcu.cz/ontology/dasta.
       owl#LaboratoryReport"/>
6    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
       date"/>
7    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
       dateTime"/>
8    <rdfs:label xml:lang="cs">datum a čas události</rdfs:label>
9    <rdfs:label xml:lang="en">event date and time</rdfs:label>
10 </owl:DatatypeProperty>
```

Opět je zde možno nalézt `rdfs:label`. Mnohem zajímavější jsou ostatní elementy. Atribut `rdfs:domain` určuje do jaké domény může atribut patřit. V tomto případě kromě `LaboratoryReport`, kde jsme proměnnou našli, je možno jí vázat i s třídami `Anamnesis`, `ClinicalEvent` a `Diagnosis`. Protože už víme, že je atribut vázán v našem případě s `LaboratoryReport`, toto již více řešit nemusíme. Atribut `rdfs:range` určuje obor hodnot daného atributu. Určuje jestli jde o řetězec, číslo, datum či vlastní typ proměnné, v tomto případě datum. Tato informace se bude později hodit filtrům.

Podle názvu kořenového elementu je možno zjistit, o jaký typ vlastnosti jde. Na výběr je z `AnnotationProperty`, `DatatypeProperty`, `ObjectProperty` a `Property`. Atributy s `ObjectProperty` se zařadí do pole `objectProperties` v rámci třídy, ostatní atributy do pole `attributes`.

7.4.8 Výběr tříd a atributů

V panelu výběru jsou připraveny dva výběrové elementy a za použití knihovny *select2* jsou do nich nahrány jednotlivé třídy a atributy. Třídy jsou zobrazeny se svým prefixovaným lokálním jménem a dalšími anotačními jmény, pokud byly načteny, a jejich četnosti. Atributy jsou zobrazeny se svým

prefixovaným lokálním jménem a četností. Oba seznamy se řadí dle svého prefixovaného jména. U atributů jsou skupiny atributů nadepsány třídou, ke které atributy patří. Při přidání nové třídy je automaticky vybrána její anotační vlastnost, pokud existuje.

Díky knihovně *select2* se v seznamech dá textově vyhledávat pro rychlejší nalezení hledané třídy či atributu.

7.4.9 Reset

Pod výběrem tříd a atributů se nachází tlačítko `Reset`. Toto tlačítko resetuje aplikaci do podoby po načtení ontologií, tříd a atributů. Veškeré části dotazu, které mohly být uživatelem vybrány, jsou resetovány do původní podoby. V případě potřeby slouží jako rychlý bod pro návrat, který nevyžaduje znovu konfigurovat aplikaci.

7.5 Segment vizualizace

Vizualizační část poskytuje vizualizaci SPARQL dotazu za použití knihovny *D3*, konkrétně její část *force*, která umožňuje simulovat fyzickou sílu na jednotlivé elementy. Zajišťuje , že vizualizace působila přirozeně. Knihovna se postará o veškeré matematické výpočty, které se s vizualizací pojí. Aplikace jen aktualizuje průběžně vizualizační graf.

7.5.1 Data grafu

Vizualizace má připravený vlastní graf, který má za úkol vykreslovat. Jde o objekt `dataset`, jež obsahuje dvě pole - `nodes` a `edges` pro reprezentaci uzlů a hran. Objekty do těchto polí dodává aplikace, na začátku jsou prázdné. U uzlu je ukládáno:

- `id` - odpovídá *id* třídy
- `name` - název zobrazený nad uzlem

Hrana je pak tvořena identifikátory počátečního a koncového uzlu (`source` a `target`) a jméno pro zobrazení nad hranou.

Při přidání třídy do vizualizace je objekt třídy rozšířen o atribut `d3Id`, který odpovídá identifikátoru jeho uzlu.

Vizualizace generuje elementy do `svg`. Tyto elementy jsou:

- `nodes` - Uzly grafu.

- `edges` - Hrany grafu.
- `nodelabels` - Textové popisky uzlů
- `edgelabels` - Textové popisky hran
- `edgepaths` - Také hrany, ale umožňují být nadepsány textem

Data jednotlivých elementů tvoří zmíněný `dateset`. Veškeré nastavení je nutno provést při jakékoli úpravě původních dat.

Uzly i hrany jsou popsány. Použité geometrické tvary jsou kruh pro uzly a šipka pro hrany. Uzly jsou obarvovány dle svého `id` a pokud je nějaký uzel vybrán, je zvětšen o 20 % pro zdůraznění výběru.

7.5.2 Přidání tříd a atributů do grafu

Základní myšlenka je, že každá vybraná třída odpovídá jednomu uzlu a všechny objektové vlastnosti odpovídají jedné hraně. Atributy ve vizualizaci zobrazovány nejsou.

Uživatel může kdykoliv přidat jakýkoli atribut i třídu.

Při vybrání atributu je vždy nutné přidat i jeho třídu automaticky, pokud není vybrána. Následně je nutné se podívat na stávající zobrazení dotazu. Pokud jde o první třídu, která se do vizualizace přidává, není třeba nic řešit. V případě dalších tříd nastávají komplikace.

Pro každou již vizualizovanou třídu je třeba zjistit a najít cestu, která by vedla z již vizualizovaných tříd do nové třídy. Pokud cesta neexistuje, třída je přidána bez jakýchkoliv vztahů s ostatními třídami. Pokud existuje, je nutné cestu vybrat a přidat do grafu.

Hledání cesty probíhá pomocí algoritmu BFS a ukládání informací o aktuální cestě. Tento algoritmus vždy vrátí první nalezenou cestu, pokud existuje. Hledání cesty je postupně spuštěno ze a do všech tříd již vizualizovaných. Konečný seznam cest je seřazen dle délky a bere se nejkratší cesta, jejíž třídy a objektové vlastnosti se přidají do grafu.

Veškerou funkcionalitu kolem tohoto algoritmu obstarává skript `seeker.js`. Ten si kvůli tomu inicializuje vlastní reprezentaci grafu vytvořením polí uzlů a hran, kde každý uzel je označen svým identifikátorem, jenž odpovídá identifikátoru třídy, a každá hrana obsahuje identifikátor zdrojového a koncového uzlu.

Při vytváření grafu je k objektovým vlastnostem přidán atribut `target`, jenž představuje identifikátor třídy, do které relace vede. Při konstrukci grafu je možno narazit na problém u podtříd. Každá objektová vlastnost obsahuje URI třídy, ke které daná relace patří. Z tohoto URI je pak možné

zjistit identifikátor třídy. U podtříd však jejich URI k nalezení není mezi URI tříd. Proto při přidávání informací o třídách se přidává také informace `subClassOf`. Algoritmus v případě podtříd připraví seznam tříd, které jsou podtřídami dané třídy, a je nucen duplikovat objektové vlastnosti s atributem `target` na danou třídu. Kdyby se graf takto neaktualizoval, nebylo by možno s podtřídami vůbec pracovat.

Cokoliv se přidává do grafu je vždy označeno atributem `selected` na *true*.

7.5.3 Duplikace tříd

Aplikace nabízí možnost přidat třídy do vizualizace víckrát, čemuž se v kontextu aplikace říká duplikace tříd. Probíhá v krocích:

- Vytvoří se kopie třídy, k jejímu lokálnímu jménu se přiřadí číslo pořadí duplikátu a vloží se na konec pole tříd.
- Veškeré již označené relace z a od původní třídy se aplikují do nové a přidávají se do vizualizace.
- Neduplikují se atributy, filtry ani agregace.

Do původní třídy se doplní atribut pole `duplicates`, který obsahuje identifikátory jednotlivých duplikací. Duplikátu je přiřazen `duplicateId`, což je jeho pořadové číslo a označení `duplicate` na hodnotu *true*. Každý duplikát obsahuje identifikátor svého rodiče v atributu `parent`.

Všem duplikátům se dá separátně od původní třídy nastavovat jejich atributy, filtry i agregace. V dotazu i ve vizualizaci jsou duplikáty rozpoznatelné svým indexem doplněný do jejich lokálního jména.

Při resetování jsou všechny duplikáty smazány.

7.5.4 Interakce

S uzly lze hýbat, což zajišťuje knihovna *D3*. Vizualizace se vždy snaží udržovat uzly uprostřed vykreslovací plochy a uzly vždy postupně přesune. Další interakce spočívá v klikání na jednotlivé uzly, čímž se uzly vybírají.

Aplikace si udržuje aktuálně vybraný uzel. Pokud uživatel klikne na jiný, tento uzel je zvýrazněn a vybrán, což znamená, že se změní panel detailů.

7.6 Segment detailu

Segment se zobrazí po kliknutí na některý z vizualizovaných uzlů, které představují jednotlivé třídy. Segment se dělí na část pro informace, objektové vlastnosti, atributy, filtry a agregace, kdy každá z nich je ve vlastním panelu.

7.6.1 Informace

První část segmentu zobrazuje informace o právě vybrané třídě. Implementováno ve formě tabulky, která se vždy naplní aktuálními daty. Tabulka vždy obsahuje název třídy (`localName`), prefixovaný název třídy (`prefixedLocalName`), jmenný prostor (`nameSpace`), URI (`uri`) a četnost `count`). Navíc může obsahovat různé atributy nahrané z ontologií, například `label` a `comment` v různých jazycích.

Navíc obsahuje dvě tlačítka pro smazání a duplikaci. Tlačítko `Delete` smaže vybranou třídu, všechny relace do a od ní, atributy, filtry a agregace. Pokud jde o duplikát, je odstraněn z pole tříd. Tlačítko `Duplicate` duplikuje vybranou třídu bez jejích relací (platí stejná pravidla jako i při duplikaci přes výběr). Tlačítko `Duplicate with relations` duplikuje třídu i s relacemi. Kopírují se relace rodiče. Při pokusu o duplikaci duplikátu je místo toho duplikován jeho rodič.

7.6.2 Objektové vlastnosti

Panel umožňuje ručně vybrat objektové vlastnosti dané třídy. Obsahuje výběr s možnými relacemi. Pouze nevybrané relace, tedy ty které mají `selected` nastaven na *false*, mohou být vybrány. Po výběru některé z relací je daná hrana a uzel přidány do grafu. V této chvíli nedochází k žádným kontrolám cest.

Všechny vybrané objektové vlastnosti se zobrazují v tabulce. V tabulce je tlačítko pro odstranění relace, jež jí odstraní z vizualizace. Toto tlačítko neodstraňuje třídu, ke které relace vede.

7.6.3 Atributy

Panel atributů umožňuje vybírat atributy dané třídy. Obsahuje HTML select s možnými nevybranými atributy. Tento seznam je obrazem seznamu ze segmentu výběru, jen zde jde pouze o ty atributy náležící vybrané třídě.

Vybrané atributy jsou zobrazeny v témže panelu v tabulce. Tato tabulka obsahuje:

- Název atributu - proměnná `prefixedLocalName`
- Četnost atributu - proměnná `count`
- Přepnutí zobrazení atributu - checkbox pro přepnutí proměnné `show`
- Přepnutí doplňkového atributu - checkbox pro přepnutí proměnné `optional`
- Tlačítko pro vybrání atributu - důležité pro filtry a agregace, toto tlačítko vybere daný atribut (`clicked`)
- Tlačítko pro odstranění atributu - odstraní atribut z výběru - změní `selected` na *false* a smaže případně vytvořené filtry, agregace a vypustí atribut z řazení

Řádky tabulky jsou zbarveny podle doména atributů:

- **Bílá** - anotační vlastnost nebo rdf:type.
- **Zelená** - doména atributu zahrnuje nahranou třídu.
- **Žlutá** - doména atributu nebyla nahrána, aplikace si není jista.
- **Červená** - doména atributu nezahrnuje nahranou třídu.

Aplikace nabízí všechny atributy bez ohledu na jejich doménu v rámci tříd, které byly nahrány z endpointu, ale varuje barvami uživatele o možných problémech.

7.6.4 Filtry

Panel filtrů umožňuje přidávání filtrů nad atributem. Aby se panel zobrazil, je nutné nějaký atribut vybrat, což se provede kliknutím na tlačítko `Select` v tabulkové části atributů (předchozí sekce). Právě zvolený atribut je označen `clicked` na hodnotu *true*.

Ve chvíli vybrání se načte tlačítko pro přidání filtru. Který filtr je možno přidat se řeší podle atributu `flterType`, které je vlastnosti aplikací přiřazeno na základě oboru hodnot atributu, pokud je rozeznaný. Aplikace rozeznává číselné, datumové a řetězcové. Pokud nerozezná typ, bere se atribut jako řetězec.

Kliknutí na vytvořené tlačítko přidá další filtr do tabulky, kde jsou jednotlivé filtry zobrazeny se sloupci:

- **Operator** - daný operátor filtru, může být *contains*, *is equal to*, *not equal to*, *starts with*, *ends with*, *regex*, =, =!, <, <=, >, >=
- **Value** - textové pole, do které se píše hodnota filtru k porovnání
- **Remove** - tlačítko pro smazání filtru

Samotný objekt filtru je vždy vytvořen po kliknutí na tlačítko a přiřazen k vybranému atributu. Vlastnosti tohoto objektu jsou v tabulce 7.10. Na začátku je index zvolené možnosti nastaven na nulu, uživatel toto mění vybíráním jedné z možností. Hodnota filtru se píše do zmíněného textového pole a aplikace jí aktualizuje při každém napsaném znaku. Pokud dojde k smazání filtru, filtr je smazán z pole filtrů, není zde žádná hodnota `selected`.

Tabulka 7.10: Objekt filtru

Vlastnost	**Popis**
value	Hodnota filtru
options	Pole možností pro filtr
selectedOption	Index zvolené možnosti

7.6.5 Agregace

Agregační funkce se nachází v panelu agregací a jde o výběr agregačních funkcí a tabulku již vybraných funkcí. Možnostmi v aplikaci jsou funkce *SUM*, *COUNT*, *AVG*, *MIN* a *MAX*. Tyto možnosti jsou načtené při vybrání atributu a přidány do pole `aggregations` v rámci atributu. Objekt agregace je uveden v tabulce 7.11.

Atribut `aggr` je hodnota agregační funkce, tedy *SUM*, *COUNT* a podobně. Atribut `attribute` je název atributu, ke kterému agregace patří, hodí se při pozdější generování dotazu. Atribut `variable` je potom název proměnné, kterou agregační funkce vytváří. Její tvar je název agregační funkce v malých písmenech + podtržítko + název atributu. Příklad je možno uvést agregační funkci *SUM* pro proměnnou `datetimeEvent`, která vypadá: `sum_datetimeEvent`.

Tabulka 7.11: Objekt agregace

Vlastnost	**Popis**
selected	Zda je agregace vybrána
attribute	Lokální jméno atributu
aggr	Agregační funkce
variable	Název proměnné funkce

Aktivní agregace se vypisují do tabulky. Ta má dva sloupce:

- **Aggregate function** - název agregační funkce
- **Remove** - tlačítko na smazání agregační funkce

V případě smazání agregace je její atribut `selected` označen jako *false* a již se nebude zobrazovat v tabulce.

7.7 Segment vlastností

Poslední segment, na který by se měl uživatel při tvorbě dotazu dostat. Je zde nastavení řazení a několik dalších omezení na dotaz.

7.7.1 Řazení

Řazení se drží základního principu, že uživatel nechce měnit pořadí všech proměnných použitých v dotazu, ale jen některých, které ho zajímají. Proto aplikace poskytuje výběr proměnných a tabulku řazení, ve které si uživatel může řazení více zosobnit.

Výběr proměnných se provádí při každé aktualizaci dotazu a obsahuje jak proměnné tříd a atributů, tak také agregačních funkcí. Pokud dojde k smazání atributu či agregační funkce, je nutné zkontrolovat, zda nebyla zařazena k řazení a případně jí vymazat.

Při vybrání proměnné je vytvořen objekt řazení a přiřazen do pole `orderBy`. Podoba objektu je uvedena v tabulce 7.12. Možnosti pro řazení jsou dvě *ASC* a *DESC*, kde implicitní je *ASC*. Pro přetypování je možnost nechat proměnnou v rámci řazení přetypovat na `int`, `date`, `datetime`, `time` nebo `float`. Dle názvu proměnné se potom lehce pozná, která proměnná je přiřazena k řazení.

Tabulka 7.12: Objekt řazení

Vlastnost	**Popis**
variable	Název proměnné
orderOptions	Možnosti pro řazení
selectedOrderOption	Vybraná možnost řazení

Tabulka pro řazení obsahuje sloupce:

- **Variable** - název řazené proměnné

- **Ordering** - výběr typu řazení, *ASC* nebo *DESC*
- **Sort** - tlačítka se šipkami nahoru a dolů, manipulace s nimi se mění pořadí proměnných v tabulce
- **Remove** - tlačítko pro vymazání řazení

Pokud uživatel chce nějakou proměnnou řadit jako první, pomocí tlačítek řazení je možno přesouvat řádky tabulky. Proměnná v prvním řádku tabulky je řazena jako první. Tlačítko pro smazání smaže objekt řazení z jeho pole.

7.7.2 Ostatní

Zde jsou tři menší možnosti, jak se dá ovlivnit SPARQL dotaz.

- **DISTINCT** - checkbox pro použití klíčového slova *DISTINCT*. Při načtení aplikace použito.
- **LIMIT** - textové pole, které přijímá pouze čísla. Při vepsání čísla většího než nula je na konci dotazu přidáno klíčové slovo *LIMIT* s danou hodnotou.
- **OFFSET** - to samé jako *LIMIT* pro klíčové slovo *OFFSET*.

7.8 Segment dotazu

Panel dotazu zobrazuje generovaný dotaz, obsahuje tlačítko na běh dotazu, informace o úspěšnosti dotazu a výsledky dotazu ve formátu tabulky a JSON.

7.8.1 Generování dotazu

Dotaz je generován automaticky a pravidelně po jakékoli akci, která by vedla k úpravě dotazu. Konkrétně se dotaz generuje pokaždé po:

- Přidání třídy či atributů do vizualizace
- Přidání filtru
- Přidání agregace
- Přidání řazení proměnné
- Odstranění řazení, agregace, filtru, atributu, relace nebo třídy

- Úprava hodnoty filtru
- Úprava řazení
- Úprava DISTINCT, LIMIT a OFFSET
- Úprava SHOW a COUNT u třídy

Dotaz je reprezentován objektem `query`, jehož vlastnosti jsou vidět v tabulce 7.13. Seznamy prefixů a proměnných jsou pole řetězců, ostatní pole budou vysvětleny později. Na začátku generování se nejdříve připraví pole těch uzlů, které jsou vybrány, tedy vizualizovány. Následně se vyhledá počáteční uzel.

Tabulka 7.13: Objekt dotazu

Vlastnost	**Počáteční hodnota**	**Popis**
prefixes	[]	Seznam použitých prefixů
variables	[]	Seznam použitých proměnných
where	[]	Seznam podmínek
filters	[]	Seznam filtrů
aggregations	[]	Seznam agregací
DISTINCT	true	Přepnutí DISTINCT
LIMIT	0	Hodnota LIMIT
OFFSET	0	Hodnota OFFSET

Počáteční uzel je uzel, do něhož nevedou žádné hrany a všechny hranou vedou z něj. Pokud v grafu nejsou pouze izolované uzly a neexistují cykly, potom takový uzel existuje. V případě jeho neexistence se jako startovní uzel bere první označený.

Procházení grafem se provádí algoritmem DFS s počátkem ve startovním uzlu. Pokud je nastaven u třídy hodnota `aggrCount` na *true*, potom je vytvořena agregace COUNT pro tuto třídu, jinak se v každém cyklu algoritmu řeší nejdříve atributy dané třídy.

U každého vybraného atributu se řeší:

- Pokud prefix atributu není v prefixech, přidat.
- Pokud má atribut vlastnost `show` na hodnotu *true*, přidat jeho název do proměnných.

Každopádně je vždy vytvořen objekt podmínky, neboli `where`, zobrazen v tabulce 7.14. Cílem je vytvořit podmínku v podobě trojice, kde subjekt je

proměnná třídy, ke které atribut patří, predikát je prefixovaný název atributu a subjekt je proměnná atributu, tedy jeho název. Atribut `OPTIONAL` říká, jestli podmínku obalit blokem OPTIONAL.

Tabulka 7.14: Objekt podmínky

Vlastnost	**Hodnota**	**Popis**
subject	Název třídy	Subjekt trojice
predicate	Prefixovaný název atributu	Predikát trojice
object	Název atributu	Objekt trojice
optional	Hodnota atributu OPTIONAL	Označení OPTIONAL

U každého atributu je nutné řešit jeho filtry a agregace, pokud byly uživatelem nastaveny.

Filtry se nachází v poli filtrů atributu pouze, pokud byly nastaveny, stačí je tedy projít a vytvořit z nich objekt filtrů, tabulka 7.15. Datový typ se získává z oboru hodnot daného atributu a je důležité jej doplnit pro správně fungující porovnávání.

Tabulka 7.15: Objekt filtru v dotazu

Vlastnost	**Popis**
type	Typ filtru
operator	Vybraný operátor
value	Hodnota filtru
attribute	Název atributu
dataType	datový typ hodnoty podmínky

U agregací je nutno kontrolovat, jestli jsou vybrány, atribut `selected`. V případě jejich vybrání se přidává do dotazu objekt agregace, tabulka 7.16.

Tabulka 7.16: Objekt agregace v dotazu

Vlastnost	**Popis**
aggr	Agregační funkce
attribute	Název atributu
variable	Název proměnné

Po vyřešení všech vybraných atributů a jejich filtrů a agregací algoritmus vyhledá všechny sousedy stávajícího uzlu přes jeho objektové vlastnosti. Pokud koncový uzel nebyl ještě prohledán, je přidán další objekt podmínky,

kde subjekt je název stávající třídy, predikát prefixovaný název objektové vlastnosti a subjekt lokální název koncového uzlu.

Nyní se provede ještě několik kontrol:

- Pokud proměnné neobsahují název koncového uzlu a má být zobrazen, přidat.
- Pokud proměnné neobsahují název startovního uzlu a má být zařazen, přidat.
- Pokud prefix neobsahují prefix koncového uzlu, přidat.

Algoritmem DFS pak program projde od startovního uzlu až po všechny, do kterých se dostane. Po skončení tohoto kola je zkontrolováno, zda se prošlo všemi vybranými uzly. Pokud ne, je proveden pokus o alokaci nového startovního uzlu z těch nevybraných. DFS se spouští dokud neprojde všemi uzly.

7.8.2 Tisk dotazu

Z objektu dotazu je postupně vytvářena jeho řetězcová reprezentace. Při tomto tisku je dbáno na čitelnost dotazu, jsou tedy přidávány mezery a konce řádek.

Pokud vygenerovaný dotaz neobsahuje žádné proměnné, dotaz se vůbec netiskne.

Nejdříve řeší prefixy. Pole `prefixes` v objektu dotazu obsahuje seznam prefixů, který teď stačí srovnat s aplikačním seznamem prefixů, od kterých je možno získat celé URI daného prefixu. Vytisknutý prefix pak nabývá podoby:

```
PREFIX prefix: <prefix uri>
```

Za každým prefixem se umisťuje konec řádky. Následuje SELECT, který je v případě hodnoty doplněn DISTINCT.

Nyní je třeba vyjmenovat jednotlivé proměnné, nad kterými je provedena selekce. Všechny proměnné z pole `variables` se doplní otazníkem a vytisknou za sebou.

Pokud dotaz využívá agregace, tisknou se na další řádku. Tisknou se ve formě:

```
(agregacni funkce(?atribut) AS ?proměnná
```

Kvůli tomuto tisku bylo v objektech agregací ukládány atributy `attribute` a `variable`.

Následuje vyjmenování podmínek, tedy část WHERE. Nejdříve se tisknou podmínky ve formě trojic:

```
?subjekt predikát*?objekt .
```

V případě potřeby jsou obaleny blokem OPTIONAL:

```
OPTIONAL { ?subjekt predikát*?objekt
```

Další na řadě jsou filtry. Pokud jde o řetězce, tisknou se za pomoci SPARQL funkcí:

```
FILTER ((regex(str(?atribut), hodnota, 'i'))) .
```

Operátor pak ovlivňuje, s jakými symboly regulárních výrazů je hodnota zapsána.

V případě číselných typů se pracuje s operátorem a jeho datovým typem:

```
FILTER (?atribut operátor hodnota)^^datový typ*.
```

Pokud operátor neodpovídá typu atributu, například řetězcový typ má přiřazen číselný operátor, potom je přetypován do číselného typu.

Po podmínkách následuje seskupování (GROUP BY) a to pouze tehdy, pokud byly v dotazu použity agregační funkce. Pokud ano, všechny proměnné z pole `variables` jsou zde vytisknuty.

Pokud bylo použito řazení, dotaz zde vytiskne ORDER BY a vytiskne jednotlivé objekty řazení:

```
modifikace řazení*(?
```

V závislosti na zvolené modifikaci mohou být proměnné přetypovány.

V případě nastavení limitu a offsetu jsou zde na konci dotazu také vytisknuty jako:

```
OFFSET hodnota
```

```
LIMIT hodnota
```

Tisknou se pouze, pokud je jejich hodnota větší než nula.

7.8.3 Zkouška dotazu

Po každém generování dotazu je dotaz zkoušen proti SPARQL endpointu, s tím, že se nastavuje LIMIT na pět a testuje se, zda dotaz něco vrací. Uživatel je o tomto informován na panelu dotazu, kde je vždy napsáno, jestli aktuální dotaz stále ještě vrací alespoň nějaké výsledky, anebo jestli už nevrací žádné.

7.8.4 Výsledky

Tento segment není zobrazen při spouštění se Sparkle, jelikož tu samou funkcionalitu poskytuje právě Sparkle.

Kromě ověření dotazu je možné dotaz odeslat takový jaký je proti endpointu a aplikace sama zobrazí výsledky. Podporuje výsledky ve formátu tabulky anebo ve formátu JSON tak, jak je dostala od endpointu. Panel obsahuje dvě tlačítka na přepínání mezi zobrazeními. Tabulka je rozdělena do sloupců dle proměnných použitých v selekci, jednotlivé hodnoty jsou potom záznamy v řádcích.

Další možností je stáhnout výsledky ve formátu CSV.

Aplikace umožňuje URI ve výsledcích přepisovat do čitelnější podoby anotačních proměnných. Jsou zde zadefinovány názvy vlastností a jejich jazyky seřazeny prioritně. Na začátku je aplikace nastavena na:

- **Vlastnosti** - form:code, rdfs:label, URI. Aplikace v načtených ontologií hledá nejdříve form:code, poté rdfs:label a nakonec vrací URI, pokud ostatní nenajde.
- **Jazyky** - cs,en,C. Aplikace zkouší, jestli jsou k dispozici vlastnosti v daném jazyce. Nejdříve zkouší cs, potom en. Proměnná C značí vrátit vlastnost bez jazykového tagu.

8 Testování a validace

Testování aplikace probíhalo jak v rámci Sparkle tak v samostatné verzi.

Aplikace byla nahrána na server MRE a je dostupná z rozsahu IP ZČU na adrese `https://mre.zcu.cz/sparkle-webview/`. Implicitní adresa předvyplněná pro endpoint je `https://mre.zcu.cz/sparql`, ale je možné jí přepsat na jakýchkoliv endpoint.

K testování byla použita data na serveru MRE a cvičná data dostupné na CD. Cvičná data byla nahrána na *localhost*, zatímco MRE je přístupné přes adresu endpointu.

Na CD se nachází přednastavený server Fuseki, který cvičná data obsahuje. Jeho adresa je po spuštění `http://localhost:3030/test/sparql`.

Prohlížeče použité k testování byly:

- Google Chrome 73.0.3683.103
- Firefox Quantum 66.0.3

Rychlost konfigurace byla změřena v závislosti na velikosti dat. Konfigurace zahrnuje načtení ontologií, tříd a atributů. Výsledky zobrazeny v tabulce 8.1. V případě malých datasetů o desítkách tisíc trojic trvá konfigurace v průměru jednu sekundu. Skutečně použitelná data jsou v případě MRE, které obsahuje přes osm set milionů trojic, a konfigurace v průměru trvá v průměru jednu minutu.

Tabulka 8.1: Tabulka měření času konfigurace

Endpoint	**MRE**	**localhost**
Počet trojic	801 833 670	33 158
Měření [ms]		
1	58 886	1 033
2	61 285	1 087
3	59 285	1 189
4	58 716	1 204
5	60 184	955
Průměr	59 671	1 093

Endpoint s velkým množstvím dat jako je DBpedia [121] není možné bez zapracování škálování v této chvíli vyzkoušet. DBpedia obsahuje následující omezení [122] (výčet těch, které se týkají rozšíření aplikace):

- Maximální počet výsledků na dotaz - 10 000. Nutno dotaz poslat vícekrát s LIMIT a OFFSET parametry (stránkování)
- Počet dotazů za sekundu na připojení - 120. Nutno dotazy posílat s intervalem, aby nedošlo k překročení limitu.

Funkčnost aplikace byla validována vytvořením a odzkoušením dotazů. Dotazy jsou umístěny ve složce `test`, kde se nachází i testovací data. Ve složkách `query1` až `query6` jsou jednotlivé dotazy k validaci. Na příkladu prvního dotazu ukáži, co jednotlivé složky obsahují:

- `popis.txt` - postup jak naklikat dotaz
- `query1.sqf` - uložený dotaz, který je nahratelný do Sparkle aplikace
- `query1.txt` - uložený dotaz
- `query1_generated.png` - obrázek generovaného dotazu v aplikaci
- `query1_graph.png` - obrázek vytvořeného grafu dotazu v aplikaci
- `query1_results.png` - obrázek výsledků dotazu v aplikaci

Postupným výběrem tříd a atributů by měl každý uživatel dojít ke stejnému vygenerovanému dotazu a stejným výsledkům. Příklad dotazu je uveden v příloze C.

Byli vybráni čtyři uživatelé s minimální znalostí SPARQL a měli za úkol projít všechny dotazy a vyzkoušet tak funkcionalitu aplikace. Před testováním jim byla krátce aplikace popsána s ilustračními obrázky, aby věděli kam mohou klikat pro přidání tříd, relací, atributů, filtrů a agregací. V rámci každého dotazu měli k dispozici soubory zmíněné výše. Scénář testování například pro první dotaz:

1. Vyberte třídu ds:ActualDiagnosis.
2. Vyberte atributy této třídy ds:diagOrder, ds:diagCode a ds:diagDetail.
3. Vyberte třídu ds:Patient.
4. Vyberte atributy této třídy ds:datetimeBirth, ds:lastName, ds:patientID, ds:sex.
5. Filtrujte ds:diagDetail, které obsahují slovo „infarkt“.
6. Seřaďte vzestupně dle ds:diagOrder a sestupně dle ds:patientID.

Zpětná vazba od uživatelů:

1. Jeden z uživatelů často místo atributů třídy ds:Patient přidával atributy třídy dcm:Patient (mají stejné atributy).

2. Uživatelé se při první filtraci zdrželi, protože jim nešla přidat. Zapomínali, že je nejdříve nutné kliknout na atribut, ke kterému se filtr přidává.

3. Odstranění vazby a přidání jiné působilo problémy dvěma uživatelům. Snažili se kliknout na hranu a smazat i mačkáním klávesy Delete. Až po dvaceti sekundách si všimli, že seznam relací je u každé třídy, ze které relace vychází.

4. Uživatel věděl o seznamu relací, ale hledal je ve třídě, kam vazba vedla.

5. Uživatel místo smazání třídy stiskl tlačítko Reset.

6. Uživatel se pokusil o řazení atributů pomocí drag and drop proměnných v tabulce.

7. Všichni uživatelé nemohli v průměru patnácti sekund najít možnost přidat agregaci COUNT pro třídu.

8. Uživatelé u složitějších dotazů, kde vyřízení dotazu trvá více jak sekundu, si stěžovali, že nevěděli jestli se dotaz poslal nebo je aplikace zablokovaná.

Vyhodnocení jednotlivých bodů:

1. V seznamu všech atributů aplikace zobrazuje, ke které třídě náleží. Uživatel často klikal na první, kterou viděl, což v tomto případě je dcm:Patient, jelikož jsou atributy a třídy seřazeny abecedně. Další možností je řadit podle četností, kdy dc:Patient by byl umístěn nad dcm:Patient v případě testovacích dat.

2. Přidávání filtrů a agregací je vždy vázáno na určitý atribut. V případě, že by stačilo označit jen třídu, výběr pro atributy by musel specifikovat ke kterému atributu se váže. Řešením by mohlo být více zdůraznit nutnost výběru, například hláškou nebo zvýrazněním tlačítka Select.

3. Interaktivita grafové vizualizace láká uživatele k tomu, že relace je klikatelná, což není. Pro komfort uživatele by bylo možné implementovat nový informační panel pro relace a umožňovat je zde mazat či jinak upravovat (negace, vnořený dotaz ...).

4. Graf je orientovaný a hledat relace v koncovém uzlu vazby je více nepozornost uživatele.

5. Obě dvě tlačítka jsou červená a ve stejném stylu. Pro budoucí vývoj je možné zkusit přesunout tlačítko Reset na méně dostupné místo, aby nedošlo k jeho chybovému stisknutí.

6. Řazení proměnných v tabulce pomocí drag and drop je intuitivní a velice rychlé. Pro budoucí vývoj existují Javascript knihovny, například jQuery Sortable [123].

7. Přidání agregace pro třídu je na jiném místě, než agregace pro atributy, ale na místě dalších tlačítek pro úpravu třídy. Ve stávajícím layoutu stránky není další místo, kam by se tato vlastnost mohla přesunout.

8. Aplikace žádným způsobem neinformuje uživatele, že je dotaz prováděn a že má počkat. V rámci budoucích rozšíření přidat signál nebo zprávu.

Uživatelé nakonec dostali scénář, podle kterého postupovali. Účelem scénáře bylo vyzkoušet stabilitu a konzistenci aplikace při úpravách grafu vizualizace a výběru. Scénář je k dispozici na CD ve složce `test`.

9 Diskuze

9.1 Nová funkcionalita

Do aplikace Sparkle byl přidán vizuální editor SPARQL. Editor umožňuje vybrat třídy a atributy, jež budou dotazovány. Dotaz je vizualizován jako orientovaný graf s popsanými uzly a hranami. Uzly představují třídy a hrany relace mezi nimi. Vizualizace je interaktivní, uživatel kliknutím na uzel zobrazí o třídě další informace. Třída může být zobrazena nebo přidána její agregační funkce COUNT. Třída poskytuje výběr relací a atributů. Atributy umožňují určit jejich zobrazení v dotazu a OPTIONAL parametr. Každý atribut nabízí výběr filtrů a agregace. Filtry podporují řetězce, data a čísla. Umožňují určit jejich hodnotu. Agregační funkce jsou podporovány COUNT, MIN, MAX, AVG a SUM. Vybrané proměnné jsou na výběr u řazení, kde se určuje typ řazení a pořadí. Další vlastnosti dotazu k nastavení je DISTINCT, LIMIT a OFFSET. Dotaz je generován po každé změně, jenž ho ovlivňuje a zároveň testován proti endpointu. Dotaz je možné spustit a zobrazit výsledky.

Další informace a relace mezi třídami získává aplikace z ontologií. Bez jejich načtení je aplikace funkční, ale neobsahuje tuto funkcionalitu. Aplikace spravuje prefixy a generuje vlastní pokud je nezíská z ontologií.

Hlavním cílem aplikace je generovat dotaz, jež bude obsahovat zadané třídy/atributy. Daný dotaz nemusí být předmětem zájmu uživatele, kterého spíše více zajímají výsledky, proto aplikace sama řeší přidávání tříd a relace mezi nimi. Celý dotaz se dá naklikat pouze za použití atributů a člověk ani nemusí vědět, ke kterým třídám patří. Příkladem může být, když v projektu MRE bude uživatele zajímat jméno pacienta a datum jeho vyšetření.

Nahrávání ontologií slouží k několika věcem:

- Získání anotačních vlastností
- Získání datového typu literálů
- Získání prefixů
- Získání objektových vlastností
- Získání domén atributů

Bez ontologií jsou přiřazeny prefixy za jejich jmenné prostory automaticky, ale ty samozřejmě nepůsobí tak dobře, jako klasické zavedené prefixy. Datový

typ literálu je důležitý pro určení správného typu filtru. Anotační vlastnosti jako jsou `label`, `comment`, `prefLabel` a `title` jsou často používané anotační vlastnosti, které se často váží k třídám a jejich vlastnostem a v čitelné podobě pro člověka popisují daný element. Tyto anotační vlastnosti bývají označeny jazykovým tagem, ve světě projektu MRE často anglickým a českým jazykem. Člověk, jenž se nevyzná ve schématu dat, je to další zjednodušení, jelikož se podle těchto anotacích dá vyhledávat při výběru tříd a atributů.

9.2 Srovnání s existujícími řešeními

Funkcionalita bude srovnávána s nástroji LinDA [70] a ViziQuer [124].

Konfigurace

Vytvořené řešení nevyžaduje žádnou konfiguraci od uživatele. Aplikace se spustí, načte ontologie/třídy/atributy ze zadaného endpointu a je připravena. LinDA funguje podobně, ale nelze zadat vlastní endpoint, pouze si vybírat ze seznamu nakonfigurovaných endpointů. ViziQuer umožňuje připojit se na kterýkoliv endpoint, pro využití našeptávače je ještě nutné načíst ontologie.

Samotná konfigurace aplikace je značná. LinDA server běží pouze na Linuxových systémech a jeho instalace je netriviální. ViziQuer je napsán za použití Meteor frameworku, který podporuje jak Linux tak Windows. Ve srovnání jde o jednodušší instalaci.

Řešení vyžaduje adresu SPARQL endpointu a nastavení ontologií pokud se používají jiné než ty z MRE.

Vybírání tříd a atributů

LinDA umožňuje vybírání tříd ze seznamu načtených tříd. ViziQuer žádný výběr nemá a uživatel musí znát název třídy, ve které chce začínat. Jak bylo řečeno v předchozí sekci, bez načtených ontologií ViziQuer neposkytuje žádnou radu. U vybraných tříd je potom nutné vybírat jednotlivé atributy - nelze je vybírat bez vybrání třídy.

Řešení práce poskytuje seznam tříd k vybrání včetně jejich atributů. Pokud daný atribut náleží ke třídě, jež nebyla ještě vybrána, aplikace jí sama vybere. Relace mezi třídami také řeší sama aplikace a uživatel tedy jen vybírá to, co ho zajímá bez ohledu na vnitřní vztahy v datech. Tato volnost je nesrovnatelná s ostatními nástroji.

Vizualizace

LinDA vizualizuje RDF data jako SQL tabulky, kdy třída odpovídá jedné tabulce a jednotlivé položky jsou atributy. Při větším počtu tříd a atributů rychle zaplňuje obrazovku a stává se lehce nepřehledným. Kvůli nedostatku místa další funkcionalita - jako filtry, agregace - je skryta v dalších oknech, které se rozevřou po dané akci na daném atributu.

ViziQuer vizualizuje RDF data jako UML diagramy, kdy třída odpovídá jednomu diagramu. Ve zbytku prostoru diagramu jsou vyjmenovány atributy, filtry, agregace a limit a offset. Veškeré nastavení je nutno provádět v jiné části aplikace, ve vlastní vizualizaci je pouze zobrazení dotazu. Barevně je rozlišen startovní uzel a ostatní uzly.

Řešení v této práci se vydalo více cestou ViziQuer v jednoduchosti vizualizace, která sama pouze zobrazuje absolutně nejnutnější informaci o dané třídě a zbytek je přesunut do jiného okna. Na rozdíl od ViziQuer je zobrazen název třídy, prefixován pro jasné rozdělení do jmenného prostoru, a relace mezi třídami. Barvy jsou přiřazovány náhodně. Vizualizace odpovídá zavedeným konvencím o RDF jako o orientovaném grafu.

Zobrazení dalších informací

LinDA u všech tříd a atributů zobrazuje jejich četnosti. ViziQuer i přes načtenou ontologii z ní nic nenabízí.

Rozšíření zobrazuje u každé třídy a atributu jejich četnosti, jako LinDA. U každé třídy je panel věnován dalším informacím, kde vždy je zobrazen název třídy, prefixovaný název třídy, jmenný prostor, URI a počet výskytů. Z načtených ontologií jsou potom zobrazeny anotační vlastnosti jako `label` či `comment`. Aplikace se snaží poskytnout co nejvíce informací ať už pro ty, kteří více datům rozumějí, ale i ty, kteří potřebují více lidsky čitelné informace.

Přidání relací

U nástroje LinDA je nutné při přidání další třídy ručně vytvořit relaci mezi nimi pomocí šipky. Tato šipka neobsahuje žádné další nastavení. Aplikace sama toto nerozezná. ViziQuer poskytuje seznam možných relací, kdy si uživatel jednu z nich vybere. Toto lze samozřejmě jen pokud jsou načteny ontologie. Relace jsou dále vybratelné a lze jim nastavit některé vlastnosti jako negaci, OPTIONAL nebo vnitřní dotaz. Tvar šipky ve vizualizaci je potom změněn dle nastavení.

Rozšíření vychází více z ViziQuer a poskytuje seznam možných relací. Zde opět zobrazuje vedle názvu tříd také prefixy a další lidsky čitelné informace,

pokud byly nahrány. Další nastavení relací aplikace nepodporuje.

Nastavení třídy

LinDA žádné nastavení pro samotnou třídu neposkytuje. ViziQuer nabízí změnu názvu proměnné a agregační funkci COUNT pro danou třídu. Nelze použít třídu jako jednu z proměnných ve výsledcích.

V aplikaci je podpora pro agregační funkci jako ve ViziQuer, navíc je i možnost nechat třídu zobrazit v selekci.

Nastavení atributů

LinDA umožňuje nechat atribut zobrazit v selekci (implicitně ano), jestli je atribut OPTIONAL (implicitně ne), jeho řazení (ASC nebo DESC) a pokud používá filtry (otevře nové okno). ViziQuer také nechává možnost pro zobrazení (implicitně ano) a OPTIONAL (implicitně ano), kromě toho navíc umožňuje změnit název proměnné. U obou nástrojů lze vybraný atribut smazat.

Aplikace se neliší od těchto nástrojů a poskytuje stejnou nabídku s tím, že atribut je implicitně zobrazován a implicitně nastaven na OPTIONAL. Přejmenování proměnné není možné.

Přidání filtrů

LinDA u každého atributu poskytuje tlačítko pro přidání filtru. Toto tlačítko otevře nové okno, kde lze určit filtry pro daný atribut. Kromě relace mezi jednotlivými filtry, každý typ filtr je nutno určit ručně. LinDA nabízí filtry pro řetězce, datumy, čísla a URL, podle nichž vygeneruje daný operátor a uživatel doplňuje hodnotu.

ViziQuer nerozděluje filtry dle atributů. Daný filtr je nutno napsat ve formě výrazu, nástroj poskytuje vyčerpávající seznam SPARQL funkcí, ale uživatel je nucen výraz napsat.

Rozšíření se drželo více LinDA filtrace. Každý atribut je možno vybrat a pak pro něj přidávat filtry. Jejich typ je získán z ontologií a aplikace sama poskytuje operátory a uživatel jen píše hodnotu.

Přidání agregací

Nástroj LinDA u každého atributu poskytuje možnost vybrat agregaci. Jsou zde na výběr všechny, tedy SUM, COUNT, AVG, MIN, MAX, GROUP CONCAT a SAMPLE. Lze vybrat pouze jednu z nich. Nástroj ViziQuer nepodporuje SAMPLE. Agregace jsou podobné filtrům v tom, že je třeba je

napsat ručně jako výraz. Je možno určit název výsledné proměnné. Výběr je doplněn aplikací, která připraví všechny možné kombinace agregační funkce a vybraných atributů a dá je uživateli k nabídnutí.

Rozšíření rozděluje agregace pro jednotlivé atributy a poskytuje základní funkce kromě GROUP CONCAT a SAMPLE. Pro jeden atribut lze vybrat více agregací.

Řazení

LinDA řazení řeší již v části výběru atributů, kdy pořadí atributů v tabulce určuje jejich pořadí v řazení. Modifikace řazení je zde nastavitelná, výběr mezi ASC a DESC. U ViziQuer je řazení uděláno přidáváním jednotlivých proměnných. Tyto proměnné je nutné ručně vyjmenovat a aplikace je neradí. Jejich pořadí určuje potom pořadí řazení. Toto řazení je součástí panelu s ostatní funkcionalitou jako filtry, agregace a GROUP BY, které je dělané ve stejném duchu.

Řešení v této práci má pro řazení extra panel v dolní části okna tak, aby to byla poslední věc ke které se uživatel dostane. Z výběru použitých proměnných je možno vybrat daný atribut k řazení, modifikaci řazení ASC a DESC a pořadí ve vytvořené tabulce určuje pořadí řazení. Funkcionálně se toto neliší od existujících nástrojů s tím, že LinDA si neporadí s proměnnými z více tříd a ViziQuer neposkytuje seznam atributů, ale vyžaduje je ručně napsat.

Další nastavení

LinDA žádné další nastavení pro dotaz neposkytuje. ViziQuer poskytuje možnost nastavit LIMIT, OFFSET a DISTINCT, který je implicitně vypnutý.

Řešení poskytuje stejnou funkcionalitu jako ViziQuer s tím, že DISTINCT je automaticky zapnutý.

Generovaný dotaz

Nástroj LinDA generuje dotaz po každé změně, stejně jako vytvořená aplikace, na rozdíl od ViziQuer, kterému se toto musí říct. Všechny nástroje samy doplňují prefixy a syntax dotazu je ve všech případech správná. Největší rozdíl je, že v této práci třídy nejsou v dotazu určeny dle svého typu (`rdf:type`). LinDA i ViziQuer dotaz generují do textového pole, takže ho lze ručně měnit.

Ve všech nástrojích lze generovaný dotaz spustit a nechat si zobrazit výsledky.

Výsledky

Aplikace zobrazuje výsledky v tabulce, JSON a ke stažení v CSV. Nejvíce se od ostatních nástrojů liší v tom, že umožňuje nastavení přepisování výsledných URI do popisků anotačních vlastností. Výsledky se tím stávají lépe čitelné pro laiky.

Shrnutí

Všechny nástroje nabízí podobnou funkcionalitu a všechny jsou funkční ve smyslu vygenerování dotazu a jeho spuštění, liší se v některých implementačních vlastnostech a některých přístupů k designu.

LinDA se snaží využít podobnosti se SQL, ale omezuje tím výslednou funkcionalitu. Řazení v rámci více tříd je problematické, vizualizace velkých tabulek je problematická a neexistence postranního panelu znamená, že na první pohled je viditelný pouze základní výběr a vše ostatní je skryto na dalších oknech. Vybírání tříd selhává v případě, kdy uživatel nezná schéma dat a potřeboval by spíše atributy. Ruční spojování tříd tomuto příliš nepomáhá. Nástroj neumožňuje kompletní reset výběru, ale lze mazat atributy i třídy.

ViziQuer vizualizuje dotaz jako UML diagramy s minimem informací ve vizualizaci a veškeré nastavení přesunuté do postranního panelu. V tomto panelu je potom veškerá funkcionalita, kterou ViziQuer podporuje včetně takových věcí, které se dělají až na konci dotazu po vybrání všech atributů, jako řazení. Nízká úroveň našeptávače nástroj dále sráží, kdy psaní agregací a hlavně filtrů je nelehká záležitost. Dobré je nastavení relací a možnost vybrat z čeho se SPARQL dotaz generuje. ViziQuer je jediný z tří diskutovaných nástrojů, který toto umožňuje, u LinDA a řešení této práce se vždy jedná o reprezentaci jednoho dotazu. Kompletní reset v tomto nástroji není, lze mazat agregace, filtry, řazení, atributy, relace i třídy.

Rozšíření umožňuje kromě tříd vybírat i atributy a aplikace sama již řeší relace mezi uzly. Od ostatních nástrojů se liší zkouškou dotazu, kdy vypisuje, jestli generovaný dotaz stále vrací alespoň nějaké výsledky. Aplikace obsahuje kompletní reset, jelikož neobsahuje možnost mazat třídy a relace.

9.3 Známá omezení a limity

9.3.1 CORS

Aplikace oddělená od Sparkle a pouštěná přes prohlížeč funguje pouze na lokálním endpointu a pro načítání ontologií je nutno použít CORS (Cross-Origin Resource Sharing) [125]. CORS se uplatňuje ve chvíli, kdy jsou po-

žadavky posílány na jinou doménu než tu, ve které je aplikace puštěna. Tato politika zabraňuje všem dotazům a aplikace nemůže získat přístup k ontologiím. Bez načtených ontologií jsou použity implicitní prefixy, nejsou zobrazeny další informace a mezi třídami není možné dělat relace.

Veřejné endpointy většinou CORS přístup povolují, ale požadavky aplikace jsou zde odmítnuty z důvodu chybného atributu v hlavičce - `origin`, který je nastaven na `null` v případě pouštění z disku. Atribut `origin` je nemožné změnit.

Pro vyřešení problému CORS a nastavení hlavičky byla aplikace umístěna na server `https://mre.zcu.cz/sparkle-webview`, čímž byly tyto problémy vyřešeny.

9.3.2 Nashorn

Protože je Nashorn implementován dle ECMA edice 5 [111], chybí mu některé vlastnosti, jež jsou implementovány v rámci moderních prohlížečů a ostatních Javascript enginů. Omezení výrazně neovlivňují funkcionalitu či implementaci a v případě budoucí aktualizace jsou lehce nahraditelné:

- `let` → `var` - novější deklaraci `let` není možné použít, stačí místo ní psát starší `var`
- anonymní funkce - nutno přepsat na klasické funkce: $(x) \Rightarrow \{\cdots kód \cdots\} \rightarrow function(x)\{\cdots kód \cdots\}$
- chybí `Set`, `Map` - obejít za použití polí
- `Array.includes()` - napsat vlastní `for` cyklus
- `String.includes()` - použít funkci `String.indexOf()` > -1
- funkce `finally` u promis - nutno obejít dalšími promisy
- `setTimeout` a `setInterval` - v aplikaci nejsou použity, v případě nutnosti použití funkcí lze obejít za využití Java metod
- `XMLHttpRequest` - využití knihovny Ajax od jQuery

Tento seznam omezení není vyčerpávající a dají se lehce nahradit. Větší omezení je spíše neexistence debugovacího nástroje, ale i to se dá obejít použitím prohlížeče.

9.3.3 Škálování

Škálování je problém pro SPARQL endpointy, které obsahují data s velkým množštvím trojic. Například DBpedia obsahuje tři biliony trojic. Množství tříd je v tomto datasetu tak obrovské, že nelze vrátit v jednom dotazu. DBpedia má nastaven vnitřní limit na počet vrácených záznamů na 10 000 a těchto dotazů je zapotřebí pět. Vrácení všech tříd navíc trvá dlouho a to ještě aplikace poté bude načítat pro všechny třídy jejich vlastnosti. V případě hojně početných tříd nejde o rychlou záležitost. Pro získání všech tříd by bylo nutné stránkovat výsledky a posílat jednotlivé dotazy s offsetem a limitem.

Další problém škálování je při vytváření vizualizace grafu. Aplikace k tomu, aby mohla nacházet cesty mezi třídami a přidávat je automaticky do vizualizace, potřebuje znát graf dat a ten může být velice komplexní a velký. Aplikace využívá BFS a vrací první nalezenou cestu, ale hledá jí pro všechny stávající třídy vizualizace a to se větším grafu může podepsat na výpočetním času.

Ontologie jsou v tuto chvíli stahovány přes HTTP protokol a server třetí strany kvůli CORS. To je problém při problémech se sítí a také v případě, kdy server třetí strany vypadne. Aplikace naštěstí na ontologiích nestojí, ale zároveň přináší několik vylepšení pro každodenního uživatele.

Škálování je oblast, která by se v budoucnu dala v aplikaci vylepšit.

9.4 Možnosti zlepšení

Hlavní slabina nástroje je škálování. U škálování je nutné, aby se potenciálně náročné dotazy, jež výběr všech tříd a jejich atributů je, rozdělily do více dotazů (stránkování) za použití limitů a offsetů. Zároveň je třeba uživateli poskytovat částečné výsledky v mezičase načítání.

Aplikace je v této chvíli na půli cesty mezi Javascript a Java aplikací. To má určitou výhodu, že je spustitelná se Sparkle i bez ní a funguje stejně. Do budoucna by bylo lepší si vybrat jednu z těchto možností a tu naplnit zcela.

V návrhu již bylo zmíněno, že Nashorn je opouštěn a místo něho bude do dalších verzí Javy dodán jiný Javascript engine, nejpravděpodobněji GraalJS. Nevýhodou tohoto enginu v této chvíli je, že není dostpuný na Windows a nepodporuje JavaFX WebView, který renderuje HTML stránky. V případě vyřešení podpory Windows se předpokládá jeho přidání do JDK. Pokud nebude stále podporovat JavaFX WebView, lze toto vyřešit přepsáním HTML layoutu do FXML, s čímž si již GraalJS poradí. Tam samozřejmě vyvstane problém vizualizace, na kterou nepůjde použít knihovna D3. Nejlepším řešením bude, pokud kromě podpory Windows bude GraalJS integrován

s WebView, což by bylo implementačně nejméně náročně.

Při opuštění Sparkle a vytvoření samostatné aplikace není od věci přidat do implementace některé Javascript frameworky jako Node.js či React. Načítání ontologií by pak mohl řešit webový server. Moderní Javascript frameworky dodají do projektu udržitelnost kódu. Toto řešení by zachovalo implementaci vizualizační části v podobě knihovny D3.

V obou případech by došlo k odstranění limitů Nashorn enginu.

10 Závěr

V práci byly vysvětleny pojmy RDF, slovníky a SPARQL jazyk. Následně byly analyzovány a srovnány existující teorie a řešení vizualizace SPARQL dotazu. Inspiroval jsem se těmi nejpoužitelnějšími, jmenovitě LinDA a ViziQuer, a navrhl a realizoval rozšíření aplikace Sparkle, které umožňuje vytvořit a vizualizovat SPARQL dotaz. Vytvořená aplikace se připojí na datový endpoint, z něhož získá seznam tříd a jejich atributů, které pak nabízí uživateli. Ten si jejich postupným výběrem určuje, která část dat ho zajímá a aplikace z tohoto výběru vytváří vizualizaci dotazu a dotaz samotný. Další funkcionalitou aplikace jsou filtry, agregace, řazení, limity a offsety.

Od ostatních nástrojů se liší především stylem vytváření dotazu, tedy samotné vybírání nejen tříd, ale i jejich atributů. Další funkcionalitou navíc oproti ostatním je získávání dalších informací z ontologií, datový typ literálů pro filtry a průběžné zkoušení dotazu proti endpointu a informování uživatele o tom, zda dotaz stále vrací nějaké výsledky.

Zadáním práce bylo rozšíření stávající aplikace Sparkle, čehož bylo dosaženo způsobem takovým, že rozšíření je zároveň samostatně fungující jednotka. Zdrojové soubory rozšíření lze z aplikace oddělit a spustit v prohlížeči nezávisle na ní. Toto nabízí dvě cesty dalšího vývoje - stále jako součást aplikace Sparkle nebo jako samostatná Javascript aplikace, kde by bylo možno rozdělit stávající implementaci mezi klient a server a využít některých moderních frameworků jako Node.js či React.

Implementace byla testována a validována proti lokálnímu SPARQL endpointu naplněnými MRE daty. Funkcionalita byla dále validována na používaném endpointu s miliony trojic a byl srovnán čas nutný ke konfiguraci v obou případech. Validace proběhla vytvořením pěti dotazům.

Seznam zkratek

BFS Breath First Search

CSS Cascading Style Sheets

DFS Depth First Search

DOM Document Object Model

FOAF Friend of a Friend

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IRI International Resource Identifier

JSON Javascript Object Notation

MRE Medical Research and Education

OWL Web Ontology Language

RDF Resource Description Framework

RDFS RDF Schema

RIF Rule Interchange Format

SKOS Simple Knowledge Organization System

SVG Scalable Vector Graphics

URI Uniform Resource Identifier

URL Uniform Resource Locator

W3C World Wide Web Consortium

XML eXtensible Markup Language

Seznam obrázků

Seznam tabulek

Literatura

[1] *SEMANTIC WEB* [online]. 2015. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/standards/semanticweb/.

[2] *Semantic Web* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.techopedia.com/definition/27961/semantic-web.

[3] *Linked Data* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/standards/semanticweb/data.

[4] *What Is the Semantic Web?* [online]. [cit. 2019/10/3]. Dostupné z: https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/.

[5] Koivunen, M.-R. – Miller, E. *W3C Semantic Web Activity* [online]. 2001. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/2001/12/semweb-fin/w3csw.

[6] *The Semantic Web Made Easy* [online]. 2004. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/RDF/Metalog/docs/sw-easy.

[7] *W3C* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/.

[8] *What is RDF?* [online]. [cit. 2019/10/3]. Dostupné z: https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf/.

[9] *Resource Description Framework (RDF)* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.techopedia.com/definition/859/resource-description-framework-rdf.

[10] *Resource Description Framework (RDF)* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/RDF/.

[11] *RDF 1.1 Primer* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/rdf11-primer/.

[12] *RDF 1.1 Concepts and Abstract Syntax* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/rdf11-concepts/.

[13] *Internationalized Resource Identifiers (IRIs)* [online]. 2005. [cit. 2019/10/3]. Dostupné z: https://tools.ietf.org/html/rfc3987.

[14] *XML Schema Part 0: Primer Second Edition* [online]. 2004. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[15] *VOCABULARIES* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/standards/semanticweb/ontology.

[16] *RDF Schema 1.1* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[17] *SKOS Simple Knowledge Organization System Primer* [online]. 2009. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2009/NOTE-skos-primer-20090818/.

[18] *OWL 2 Web Ontology Language Primer (Second Edition)* [online]. 2012. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[19] *RIF Overview (Second Edition)* [online]. 2013. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/rif-overview/.

[20] *FOAF Vocabulary Specification 0.99* [online]. 2014. [cit. 2019/10/3]. Dostupné z: http://xmlns.com/foaf/spec/.

[21] *DCMI Metadata Terms* [online]. 2012. [cit. 2019/10/3]. Dostupné z: http://www.dublincore.org/specifications/dublin-core/dcmi-terms/.

[22] *Simple Knowledge Organization System (SKOS)* [online]. 2009. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/2001/sw/wiki/SKOS.

[23] *Rule Interchange Format (RIF)* [online]. 2010. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/2001/sw/wiki/RIF.

[24] *Web Ontology Language (OWL)* [online]. 2012. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/2001/sw/wiki/OWL.

[25] *RDF 1.1 N-Triples* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2014/REC-n-triples-20140225/.

[26] *RDF 1.1 Turtle* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2014/REC-turtle-20140225/.

[27] *RDF 1.1 TriG* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2014/REC-trig-20140225/.

[28] *RDF 1.1 N-Quads* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/2014/REC-n-quads-20140225/.

[29] *JSON-LD 1.0* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/json-ld/.

[30] *RDFa 1.1 Primer - Third Edition* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/rdfa-primer/.

[31] *RDF 1.1 XML Syntax* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/rdf-syntax-grammar/.

[32] *SPARQL Query Language for RDF* [online]. 2008. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/rdf-sparql-query/.

[33] *SPARQL 1.1 Query Language* [online]. 2013. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/TR/sparql11-query/.

[34] *Uniform Resource Identifier (URI): Generic Syntax* [online]. 2005. [cit. 2019/10/3]. Dostupné z: https://tools.ietf.org/html/rfc3986.

[35] *About OpenLink Virtuoso* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://virtuoso.openlinksw.com/.

[36] *Apache Jena Fuseki* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://jena.apache.org/documentation/fuseki2/.

[37] *SparqlEndpoints* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/wiki/SparqlEndpoints.

[38] *AVAILABILITY* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://sparqles.ai.wu.ac.at/availability.

[39] *Sparkle* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://mre.zcu.cz/sparkle/.

[40] *Medical Information Systems* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://medical.zcu.cz/.

[41] Šmucr, J. Grafická tvorba SPARQL. Master's thesis, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2014.

[42] Kazák, J. Sparkle - rozšíření nástroje pro tvorbu SPARQL dotazů. Master's thesis, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2017.

[43] Hlaváčová, K. Rozšíření Sparkle o podporu SPARQL Endpointu a využití titulků při tvorbě dotazu. Master's thesis, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2018.

[44] Pietriga, E. *IsaViz: A Visual Authoring Tool for RDF* [online]. 2007. [cit. 2019/10/3]. Dostupné z: https://www.w3.org/2001/11/IsaViz/.

[45] *LodLive - GitHub* [online]. 2013. [cit. 2019/10/3]. Dostupné z: https://github.com/dvcama/LodLive.

[46] *LodLive* [online]. 2012. [cit. 2019/10/3]. Dostupné z: http://en.lodlive.it/.

[47] *LODmilla* [online]. 2016. [cit. 2019/10/3]. Dostupné z: http://lodmilla.sztaki.hu/lodmilla/.

[48] *LODmilla frontend GitHub* [online]. 2013. [cit. 2019/10/3]. Dostupné z: https://github.com/dsd-sztaki-hu/LODmilla-frontend/.

[49] *LODmilla backend GitHub* [online]. 2013. [cit. 2019/10/3]. Dostupné z: https://github.com/dsd-sztaki-hu/LODmilla-backend.

[50] *RelFinder Relationship Discovery in RDF Data* [online]. 2011. [cit. 2019/10/3]. Dostupné z: http://www.visualdataweb.org/relfinder.php.

[51] *RelFinder - Google Code* [online]. 2011. [cit. 2019/10/3]. Dostupné z: https://code.google.com/archive/p/relfinder/.

[52] Camarda, D. V. – Mazzini, S. – Antonuccio, A. LodLive, exploring the web of data. In *Proceedings of the 8th International Conference on Semantic Systems - I-SEMANTICS 12*. ACM Press, 2012. doi: 10.1145/2362499.2362532. Dostupné z: https://doi.org/10.1145/2362499.2362532.

[53] Micsik, A. *DBpedia LODmilla project* [online]. 2016. [cit. 2019/10/3]. Dostupné z: https://wiki.dbpedia.org/projects/lodmilla.

[54] Micsik, A. – Turbucz, S. – Tóth, Z. Exploring publication metadata graphs with the LODmilla browser and editor. *International Journal on Digital Libraries*. oct 2014, 16, 1, s. 15–24. doi: 10.1007/s00799-014-0130-2. Dostupné z: https://doi.org/10.1007/s00799-014-0130-2.

[55] Micsik, A. – Turbucz, S. – Györök, A. LODmilla: a Linked Data Browser for All. oct 2014.

[56] Heim, P. et al. RelFinder: Revealing Relationships in RDF Knowledge Bases. 2009, s. 182–187. doi: 10.1007/978-3-642-10543-2_21. Dostupné z: https://doi.org/10.1007/978-3-642-10543-2_21.

[57] Lehmann, J. – Schüppel, J. – Auer, S. Discovering Unknown Connections - the DBpedia Relationship Finder. s. 99–110, jan 2007.

[58] *iSPARQL - GitHub* [online]. 2011. [cit. 2019/10/3]. Dostupné z: https://github.com/openlink/iSPARQL.

[59] *GitHub - LinDA Workbench* [online]. 2016. [cit. 2019/10/3]. Dostupné z: https://github.com/LinDA-tools/LindaWorkbench/.

[60] *QueryVOWL* [online]. 2015. [cit. 2019/10/3]. Dostupné z: http://vowl.visualdataweb.org/queryvowl/.

[61] *SemTk - GitHub* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://github.com/ge-semtk/semtk.

[62] Ferré, S. *Sparklis* [online]. 2013. [cit. 2019/10/3]. Dostupné z: http://www.irisa.fr/LIS/ferre/sparklis/.

[63] *SparqlFilterFlow* [online]. 2014. [cit. 2019/10/3]. Dostupné z: http://sparql.visualdataweb.org/L.

[64] *SPARQLGraph - GitHub* [online]. 2015. [cit. 2019/10/3]. Dostupné z: https://github.com/tadKeys/sparqlgraph.

[65] *Visual SPARQL builder - GitHub* [online]. 2010. [cit. 2019/10/3]. Dostupné z: https://github.com/leipert/vsb.

[66] *ViziQuer - GitHub* [online]. 2016. [cit. 2019/10/3]. Dostupné z: https://github.com/LUMII-Syslab/viziquer.

[67] *Virtuoso Open-Source Edition* [online]. [cit. 2019/10/3]. Dostupné z: http://vos.openlinksw.com/owiki/wiki/VOS.

[68] *OpenLink iSPARQL* [online]. 2011. [cit. 2019/10/3]. Dostupné z: https://www.openlinksw.com/isparql/.

[69] *LinDA Workbench* [online]. [cit. 2019/10/3]. Dostupné z: http://linda-project.eu/tools/.

[70] *LinDA Query Designer* [online]. 2016. [cit. 2019/10/3]. Dostupné z: https://2016.semantics.cc/sites/2016.semantics.cc/files/files/LinDAQueryDesigner.pdf.

[71] *LinDA Query Designer Demo* [online]. [cit. 2019/10/3]. Dostupné z: http://linda.epu.ntua.gr/query-designer/.

[72] *VOWL: Visual Notation for OWL Ontologies* [online]. 2014. [cit. 2019/10/3]. Dostupné z: http://vowl.visualdataweb.org/v2/.

[73] Lohmann, S. et al. VOWL2: User-Oriented Visualization of Ontologies. In *Lecture Notes in Computer Science*. : Springer International Publishing, 2014. s. 266–281. doi: 10.1007/978-3-319-13704-9_21. Dostupné z: https://doi.org/10.1007/978-3-319-13704-9_21.

[74] *QueryVOWL Demo* [online]. 2017. [cit. 2019/10/3]. Dostupné z: http://vowl.visualdataweb.org/queryvowl/queryvowl.html.

[75] Haag, F. et al. QueryVOWL: Visual Composition of SPARQL Queries. 2015, s. 62–66. doi: 10.1007/978-3-319-25639-9_12. Dostupné z: https://doi.org/10.1007/978-3-319-25639-9_12.

[76] *SemTk* [online]. 2019. [cit. 2019/10/3]. Dostupné z: http://semtk.research.ge.com/.

[77] *SemTk - SPARQLGraph* [online]. 2019. [cit. 2019/10/3]. Dostupné z: http://semtk.research.ge.com/sparqlGraph/main-oss/sparqlGraph.html.

[78] Ferré, S. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web*. Dec 2016, 8, 3, s. 405–418. ISSN 2210-4968. doi: 10.3233/SW-150208. Dostupné z: http://doi.org/10.3233/SW-150208.

[79] Young, D. – Shneiderman, B. A graphical filter/flow representation of Boolean queries: A prototype implementation and evaluation. *Journal of the American Society for Information Science*. jul 1993, 44, 6, s. 327–339. doi: 10.1002/(sici)1097-4571(199307)44:6<327::aid-asi3>3.0.co;2-j. Dostupné z: https://doi.org/10.1002/(sici)1097-4571(199307)44:6<327::aid-asi3>3.0.co;2-j.

[80] Haag, F. – Lohmann, S. – Ertl, T. Simplifying filter/flow graphs by subgraph substitution. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, sep 2012. doi: 10.1109/vlhcc.2012.6344501. Dostupné z: https://doi.org/10.1109/vlhcc.2012.6344501.

[81] *SparqlFilterFlow Demo* [online]. [cit. 2019/10/3]. Dostupné z: http://sparql.visualdataweb.org/sparqlfilterflow.php.

[82] Haag, F. – Lohmann, S. – Ertl, T. SparqlFilterFlow: SPARQL Query Composition for Everyone. In *Lecture Notes in Computer Science*. : Springer International Publishing, 2014. s. 362–367. doi: 10.1007/978-3-319-11955-7_49. Dostupné z: https://doi.org/10.1007/978-3-319-11955-7_49.

[83] Schweiger, D. – Trajanoski, Z. – Pabinger, S. SPARQLGraph: a web-based platform for graphically querying biological Semantic Web databases. *BMC Bioinformatics*. 2014, 15, 1, s. 279. doi: 10.1186/1471-2105-15-279. Dostupné z: https://doi.org/10.1186/1471-2105-15-279.

[84] *SPARQLGraph* [online]. [cit. 2019/10/3]. Dostupné z: http://sparqlgraph.i-med.ac.at/.

[85] Eipert, L. *Visual SPARQL builder* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://leipert.github.io/vsb/.

[86] *ViziQuer/web Tool for RDF Data Analysis Queries* [online]. 2019. [cit. 2019/10/3]. Dostupné z: http://viziquer.lumii.lv/.

[87] Grafkin, P. et al. SPARQL Query Builders: Overview and Comparison. In *BIR Workshops*, 2016.

[88] *Nepomuk* [online]. 2018. [cit. 2019/10/3]. Dostupné z: https://userbase.kde.org/Nepomuk.

[89] Ambrus, O. – Möller, K. – Handschuh, S. Konduit VQB: A visual query builder for SPARQL on the social semantic desktop. 01 2010, 565.

[90] Russell, A. et al. NITELIGHT: A graphical tool for semantic query construction. jan 2009, 543.

[91] *Visual Query System (VQS)* [online]. [cit. 2019/10/3]. Dostupné z: http://optique-project.eu/training-programme/module-vqs/.

[92] Soylu, A. et al. OptiqueVQS: Ontology-based Visual Querying. 10 2015.

[93] AGH, A. C. *Visual query construction over RDF data - the QUaTRO2 tool* [online]. 2011. [cit. 2019/10/3]. Dostupné z: http://dice.cyfronet.pl/products/quatro.

[94] AGH, A. C. *QUaTRO releases* [online]. 2011. [cit. 2019/10/3]. Dostupné z: http://dice.cyfronet.pl/products/quatro.

[95] Hogenboom, F. et al. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. 2010, s. 87–116. doi: 10.1007/978-1-84996-074-8_4. Dostupné z: https://doi.org/10.1007/978-1-84996-074-8_4.

[96] *Drupal* [online]. [cit. 2019/10/3]. Dostupné z: https://www.drupal.org/.

[97] Clark, L. *SPARQL Views* [online]. 2010. [cit. 2019/10/3]. Dostupné z: https://www.drupal.org/project/sparql_views.

[98] Clark, L. SPARQL Views: A Visual SPARQL Query Builder for Drupal. In *Proceedings of the 2010 International Conference on Posters and Demonstrations Track - Volume 658*, ISWC-PD'10, s. 161–164, Aachen, Germany, Germany, 2010. CEUR-WS.org. Dostupné z: http://dl.acm.org/citation.cfm?id=2878399.2878440.

[99] Borsje, J. Graphical Query Composition and Natural Language Processing in an RDF Visualization Interface. jul 2006.

[100] *SPARQLViz* [online]. 2006. [cit. 2019/10/3]. Dostupné z: https://sourceforge.net/projects/sparqlviz/.

[101] *JavaFX Overview* [online]. 2014. [cit. 2019/10/3]. Dostupné z: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm.

[102] *Java SE Downloads* [online]. [cit. 2019/10/3]. Dostupné z: https://www.oracle.com/technetwork/java/javase/downloads/index.html.

[103] *JavaFX 12* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://openjfx.io/.

[104] *Using JavaFX Charts* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm.

[105] *Prefuse* [online]. 2011. [cit. 2019/10/3]. Dostupné z: https://github.com/prefuse/Prefuse.

[106] *JGraphX* [online]. 2012. [cit. 2019/10/3]. Dostupné z: https://github.com/jgraph/jgraphx.

[107] *yFiles for JavaFX* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.yworks.com/products/yfiles-for-javafx.

[108] *D3 Data-Driven Documents* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://d3js.org/.

[109] *Gallery* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://github.com/d3/d3/wiki/Gallery.

[110] *Project Nashorn* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://openjdk.java.net/projects/nashorn/.

[111] Ponge, J. *Oracle Nashorn: A Next-Generation JavaScript Engine for the JVM* [online]. [cit. 2019/10/3]. Dostupné z: https://www.oracle.com/technetwork/articles/java/jf14-nashorn-2126515.html.

[112] *Standard ECMA-262* [online]. 2018. [cit. 2019/10/3]. Dostupné z: https://www.ecma-international.org/publications/standards/Ecma-262.htm.

[113] *JEP 335: Deprecate the Nashorn JavaScript Engine* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://openjdk.java.net/jeps/335.

[114] *GraalJS* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://github.com/graalvm/graaljs.

[115] *GraalVM* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://www.graalvm.org/.

[116] *jQuery.ajax()* [online]. 2019. [cit. 2019/10/3]. Dostupné z: http://api.jquery.com/jquery.ajax/.

[117] *Bootstrap* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://getbootstrap.com/.

[118] *jQuery* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://jquery.com/.

[119] *Select2* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://select2.org/.

[120] *Apache Jena* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://jena.apache.org/.

[121] *About DBpedia* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://wiki.dbpedia.org/about.

[122] *Public SPARQL Endpoint* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://wiki.dbpedia.org/public-sparql-endpoint.

[123] *jQuery Sortable* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://johnny.github.io/jquery-sortable/.

[124] Cerans, K. – Ovcinnikova, J. – Zviedris, M. SPARQL Aggregate Queries Made Easy with Diagrammatic Query Language ViziQuer. In *International Semantic Web Conference*, 2015.

[125] *Cross-Origin Resource Sharing (CORS)* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[126] *Apache Jena Releases* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://jena.apache.org/download/.

[127] *Apache Maven Project* [online]. 2019. [cit. 2019/10/3]. Dostupné z: https://maven.apache.org/.

A Přílohy na CD

Přiložené CD obsahuje několik složek:

- `dp` - obsahuje elektronickou verzi diplomové práce.
- `img` - obrázky relevantních analyzovaných nástrojů.
- `test` - testovací data a dotazy
- `sparkle` - zdrojové soubory aplikace Sparkle.
- `sparkle-webview` - zdrojové soubory rozšíření Sparkle, spustitelné samostatně

Soubor `apache-jena-fuseki-3.10.0.zip` obsahuje přednastavenou instanci Fuseki, ve které jsou již nahrána testovací data.

B Uživatelská dokumentace

B.1 Požadavky

Při spuštění aplikace v rámci Sparkle platí stejné požadavky jako v předchozích pracích [41–43], tedy správně nainstalovaná Java verze alespoň 8. Pro správné fungování rozšíření je doporučeno používat nejnovější verzi, v době psaní práce `8.0.191`. Na místě je upozornit na chybu, která se v průběhu JRE vyskytla od verze `8.0.42` a byla opravena až v pozdějších verzích. Rozšíření by v případě této chyby nefungovalo a je nutné aktualizovat verzi JRE. Další verze Javy by neměly působit problémy. V případě spouštění přes prohlížeč se očekává nová verze Google Chrome nebo Mozilla Firefox, na kterých byla aplikace otestována. Není třeba žádných rozšíření těchto prohlížečů.

B.2 Lokální endpoint

Pro otestování aplikace je dobré vytvořit lokální SPARQL endpoint. Doporučuji Apache Fuseki [36], open-source řešení, jenž nevyžaduje složitou konfiguraci. Je možné použít instanci přiloženou na CD, stačí jí rozbalit a spustit `fuseki-server.bat`. Tato instance již obsahuje testovací data. Druhou možností je Fuseki stáhnout ze stránek projektu Apache Jena [126], V době psaní této práce je poslední verzi 3.10.0. Uživatel nechť stáhne ve formátu zip nebo tar dle svého systému a staženou aplikaci rozbalí. Fuseki server se pak spouští přes zmíněný `bat` soubor a je následně dostupný na adrese `localhost:3030`. Na této webové adrese je nyní nutné připravit testovací databázi. Přes záložky `Manage datasets` a `add new dataset` vybere uživatel název nové databáze, například `test`. Uživatel může vybrat `In-memory` nebo `Persistent` dle svého výběru. Vytvořením nové databáze se přidá do seznamu existujících databázích. Nyní kliknutím na `upload data` je nutné načíst testovací data, která se nachází ve složce `data`. Mělo by zde být:

- `patient7-imaging.ttl`
- `patient7-medical.ttl`

Data jsou z projektu MRE a dobře poslouží k ukázce funkcionality. Po načtení všech dat by měla databáze obsahovat 58 161 trojic.

Adresa tohoto testovacího endpointu potom bude:

```
http://localhost:3030/test/sparql
```

Adresa přednastaveného Fuseki je stejná.

B.3 Sestavení a spuštění aplikace

Spuštění aplikace se liší v závislosti na tom, jestli uživatel spouští aplikaci v rámci Sparkle nebo přes prohlížeč.

B.3.1 Sparkle

Sparkle se nachází ve složce `sparkle`. Pro překlad aplikace je nejlepší využít nástroje Maven [127]. Pokud uživatel nemá nainstalovaný a nakonfigurovaný Maven, stačí tento nástroj stáhnout i bez potřeby instalovat. Příkaz ke kompilaci aplikace je:

```
mvn clean install
```

Pokud uživatel nemá Maven nakonfigurovaný, stačí před příkaz `mvn` doplnit cestu do staženého Maven adresáře, následně podsložku `bin`, kde se nachází `mvn.cmd`.

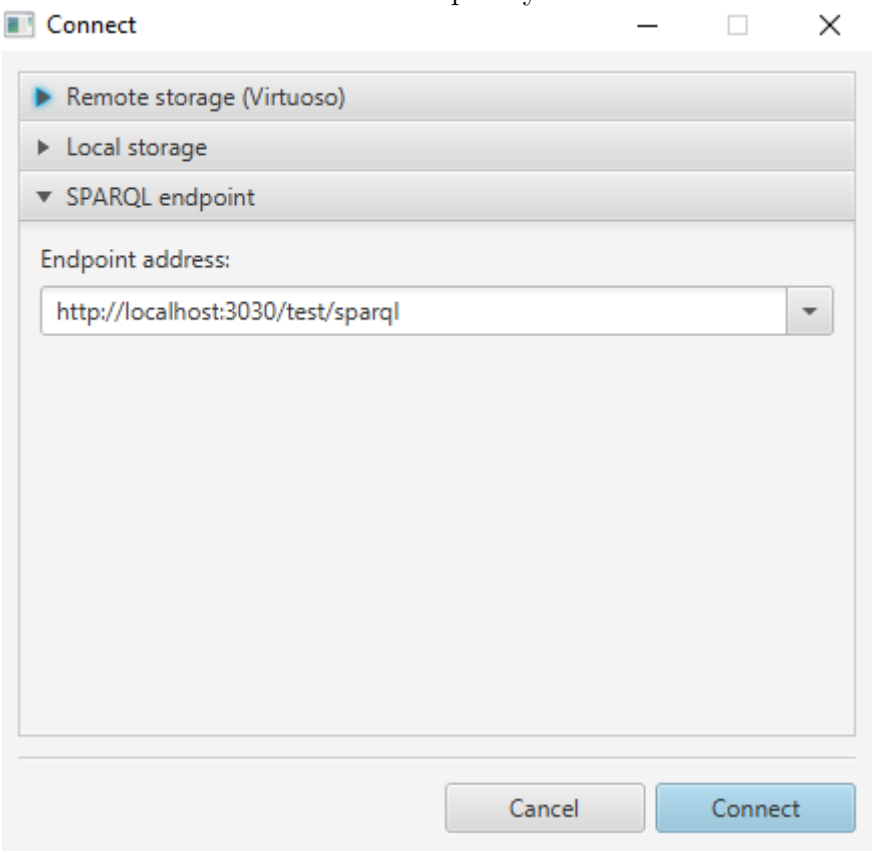
Maven následně stáhne všechny potřebné knihovny a zkompiluje aplikaci. Tato operace může trvat i delší dobu. Po skončení procesu by se měla ve stávajícím adresáři objevit složka `target`. Zde se nachází přeložená aplikace.

V závislosti na verzi Sparkle může mít různé názvy, bude to ale jediný soubor typu `jar` v adresáři. V době psaní této práce by se měl soubor jmenovat `sparkle-2019-01.jar`. Spouští se klasicky jako všechny ostatní `jar` soubory:

```
java -jar sparkle-2019-01.jar
```

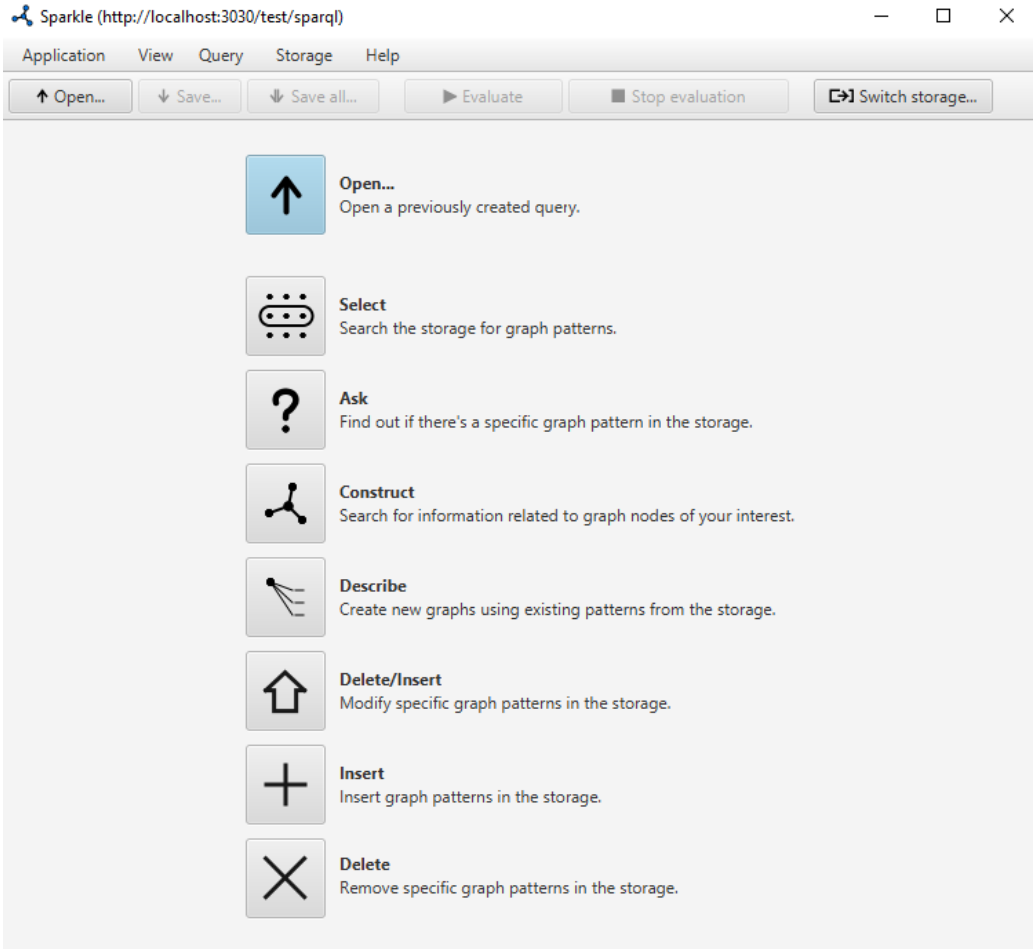
Název aplikace nechť uživatel nahradí za aktuální název souboru aplikace. Aplikace by se nyní měla spustit a uživatel by měl vidět obrázek B.1, okno pro výběr úložiště. Rozšíření Sparkle pracuje pouze se SPARQL endpointem, proto je dobré zde zvolit a doplnit nějaký endpoint. Vybrání jiného uložiště neovlivní chod vizuálního editoru, ale není to podporováno.

Obrázek B.1: Okno pro výběr úložiště



Po vybrání úložiště se uživatel dostane na hlavní okno aplikace na obrázku B.2. Rozšíření aplikace bylo děláno pro dotaz typu SELECT, je dobré jej tedy vybrat. Na následujícím okně jsou k dispozici tři záložky editorů - formulářový, textový a vizuální s tím, že při příchodu do okna je uživatel umístěn do formulářového. Po kliknutí na záložku vizuálního editoru je uživatel přesunut do okna s vizuálním editorem a od této chvíle jsou obě možnosti spuštění aplikace stejné.

Obrázek B.2: Hlavní okno aplikace Sparkle



B.3.2 Prohlížeč

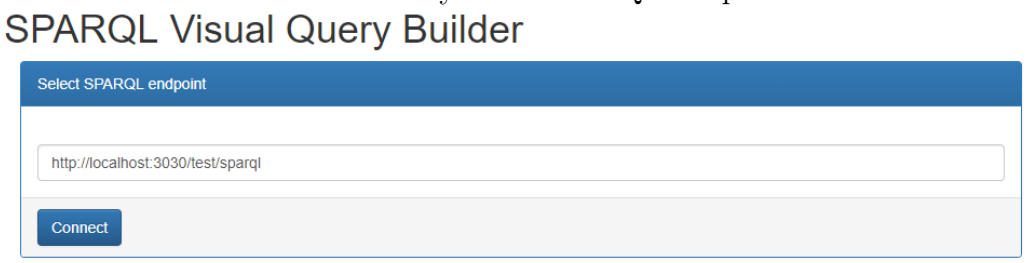
Verze bez Sparkle se nachází ve složce `ext`. Zde stačí otevřít soubor `index.html` přes prohlížeč Chrome nebo Mozilla podporovaných verzí. Aplikace je tímto spuštěna. Pro takto spuštěnou aplikaci je nutné nastavit CORS, což se udělá odkomentováním proměnné `cors` na začátku skriptu `options.js`. Endpoint bude fungovat pouze `localhost`. Pro vyzkoušení jiných veřejných endpointů, uživatel nechť zadá jako adresu `https://mre.zcu.cz/sparkle-webview/`, kde se nachází již spuštěná aplikace.

B.4 Sektor výběru

Při spuštění aplikace je k dispozici pouze sektor Výběru, vyobrazen na obrázku B.3. Do připraveného textové pole je nutno doplnit adresu SPARQL endpointu, na který se bude aplikace dotazovat. Uživatelé, kteří aplikaci

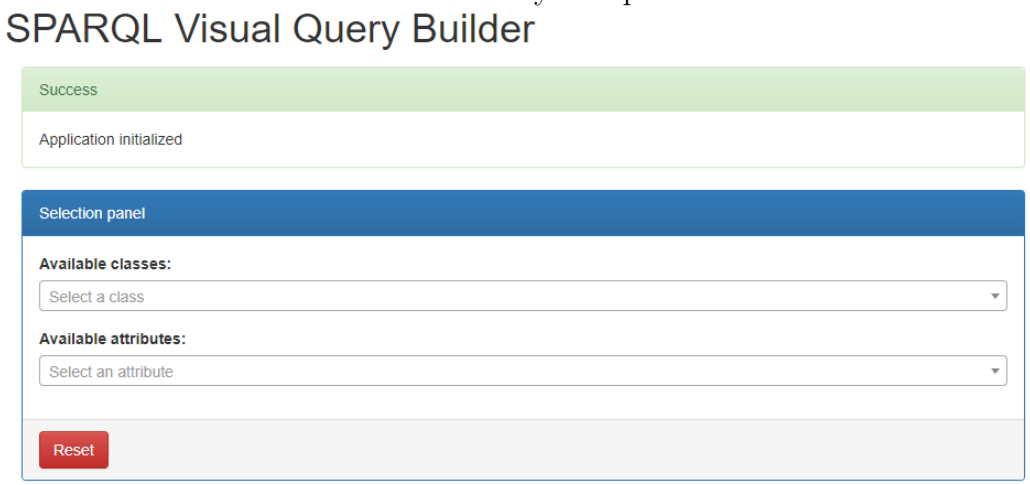
spustili přes Sparkle, budou mít toto pole předvyplněno endpointem, jež zadali při spuštění Sparkle.

Obrázek B.3: Vybrání SPARQL endpointu



Po vyplnění endpointu je nutné kliknout na tlačítko `Connect`. Aplikace v této chvíli bude nějakou dobu pracovat v rozmezí sekundy až zhruba pět sekund. Při dokončení inicializace aplikace vypíše na obrazovku `Application initialized` a nyní je možné s aplikací nerušeně pracovat (obrázek B.4).

Obrázek B.4: Sektor výběru po inicializaci



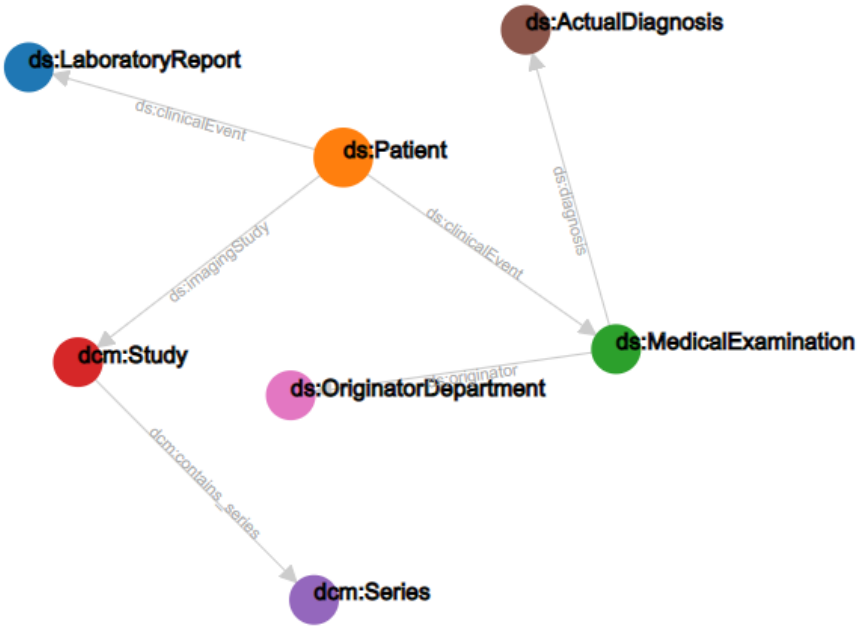
Na výběr je nyní seznam načtených tříd, seznam načtených atributů a tlačítko `Reset`. Toto tlačítko resetuje aplikaci do stavu po inicializaci. V seznamu tříd se nacházejí všechny načtené třídy z endpointu. Kliknutím na jednu ze tříd je třída zařazena do dotazu. Třídy mohou být zařazeny víckrát a pak jsou v dotazu zrcadleny. Atributy lze přidat v rámci každé třídy jen jednou. Jejich seznam je rozdělen podle třídy, ke které patří. Přidáním atributu se automaticky přidává i jeho třída. Aplikace sama hledá cestu v grafu a automaticky doplňuje náležité třídy a relace, uživatel tedy toto nemusí řešit.

Třídy i atributy jsou zobrazeny v seznamu nejen se svým daným jménem, ale také titulky načtenými z ontologií. V seznamech lze textově vyhledávat a usnadnit tak hledání dané třídy či atributu.

B.5 Sektor vizualizace

Uživatel nechť vybere ty atributy a třídy, jež ho zajímají. Ukázka možné vizualizace dotazu na obrázku B.5. Vizualizace zobrazuje vybrané třídy a relace mezi nimi ve formě grafu, uzly i hrany jsou popsány pro lepší čitelnost. Všechny ostatní informace jsou ukryty v sektoru informací a je nutné nejdříve vybrat nějakou třídu předtím, než se o ní zobrazí další informace.

Obrázek B.5: Ukázka vizualizovaného dotazu



Kliknutím na uzel je uzel zvýrazněn (zvětší se) a sektor informací se zaplní informacemi o dané třídě.

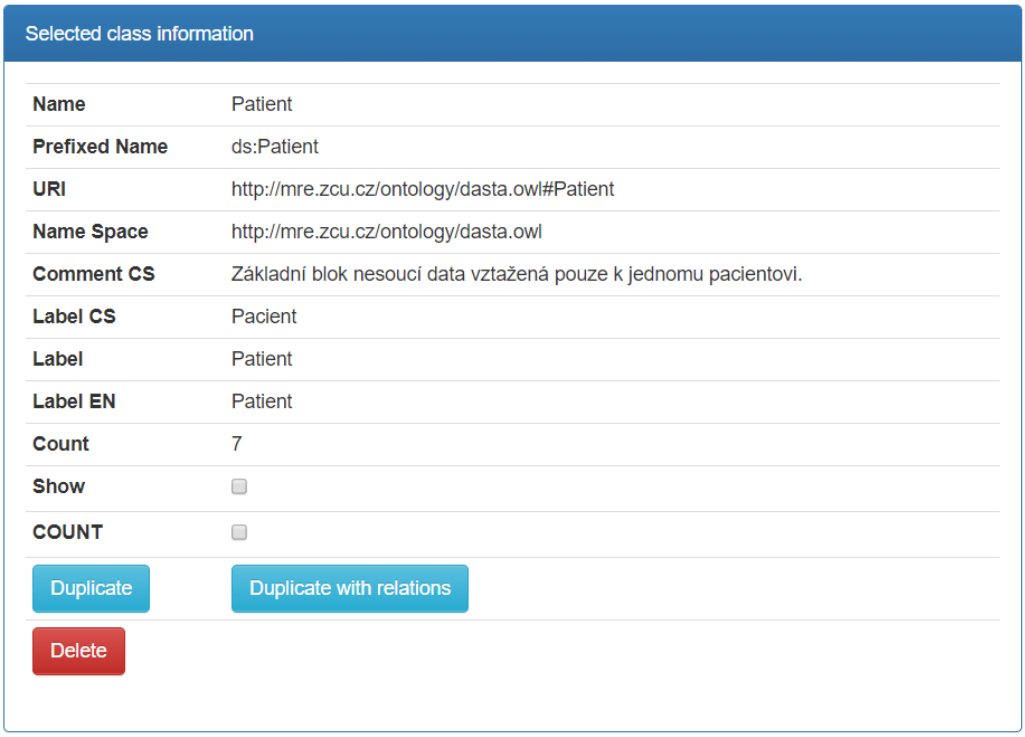
B.6 Sektor informací

Po kliknutí na informací je možné ihned vidět informace o této třídě (obrázek B.6). V této tabulce informací je zobrazen název třídy, její prefixovaný

název (ten odpovídá názvu zobrazeném v grafu), URI, jmenný prostor a počet výskytů v datech. Další informace může obsahovat z načtených ontologií, jde o anotační vlastnosti, které se zobrazují společně s jejich jazykovými tagy. V dolní části tabulky je pak možnost přepnout, jestli bude URI třídy zobrazeno v dotazu (`show`) a jestli se do dotazu přidá agregační funkce COUNT této třídy (`COUNT`) (obojí implicitně ne). Přepnutí těchto věcí je okamžitě viditelné na generovaném dotazu.

Tlačítko *Delete* smaže vybranou třídu z dotazu a všechny její označené relace, atributy, filtry, agregace a řazení. Tlačítko *Duplicate* udělá kopii třídy bez relací, tlačítko `Duplicate with relations` s relacemi.

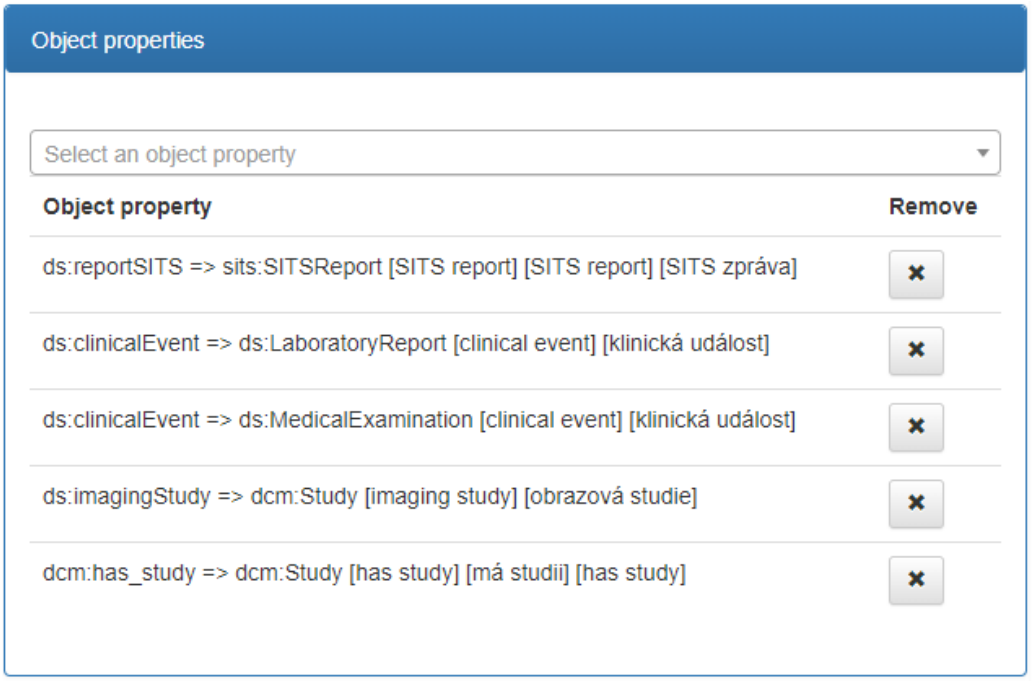
Obrázek B.6: Informace o třídě



V sektoru se dále nachází výběr objektových vlastností, atributů, filtrů a agregačních funkcí.

Výběr relací dané třídy je seznam dosud nevybraných relací a uživatel si je může libovolně vybírat. Vybrání relace jí spolu s koncovou třídou přidá do vizualizace a zobrazí v tabulce (obrázek .

Obrázek B.7: Výběr relací



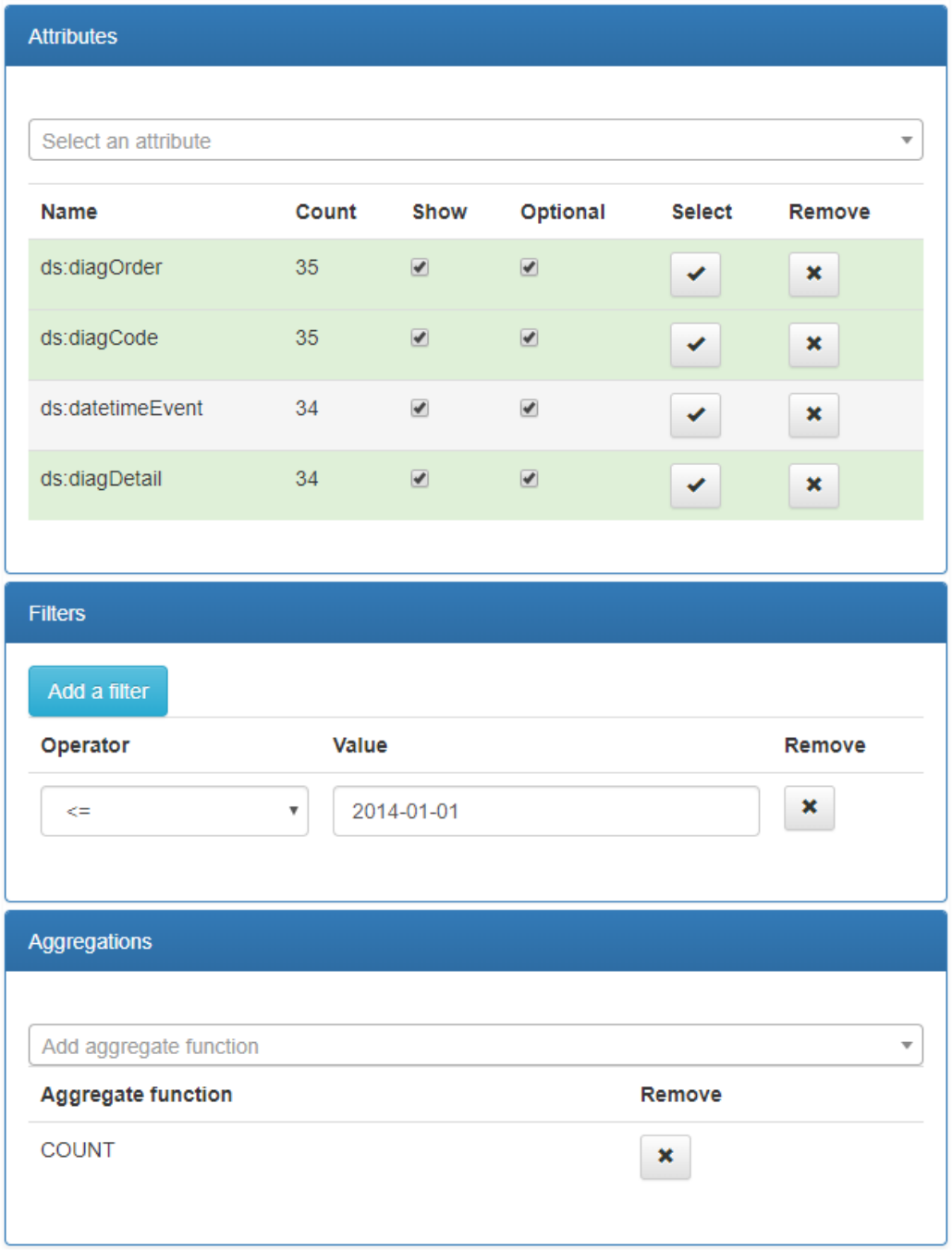
Výběr atributů je stejný jako v sektoru Výběru, zde jsou však jen atributy vázané k dané třídě. Vybrané atributy jsou zobrazeny v tabulce, kde jim lze nastavit, jestli budou zobrazovány v dotazu (implicitně ano), jestli jsou dobrovolné (obojí implicitně ano) anebo se dají odstranit z dotazu. Dotaz lze vybrat kliknutím na tlačítko pro výběr čímž se odemkne možnost nastavení filtrů a agregace daného atributu. Vybraný atribut je zvýrazněn zelenou barvou.

Pro přidání filtru je nutné kliknout na tlačítko `Add a filter`, jehož typ je dán atributem. Následně se vybírá operátor filtru, jeho hodnota a dá se zde i vymazat.

Agregace je možno vybírat z výběru, vybranou agregační funkci je opět možnost odstranit.

Odstraněním atributu jsou odstraněny i jeho filtry a agregace.

Obrázek B.8: Atributy, filtry a agregace



B.7 Sektor vlastností

Všechny proměnné použité v dotazu je možno řadit. V tomto sektoru se nachází výběr všech proměnných, které je možno zařadit do řazení. Všechny řazené proměnné jsou uvedeny v tabulce, kde je možno jim nastavit modi-

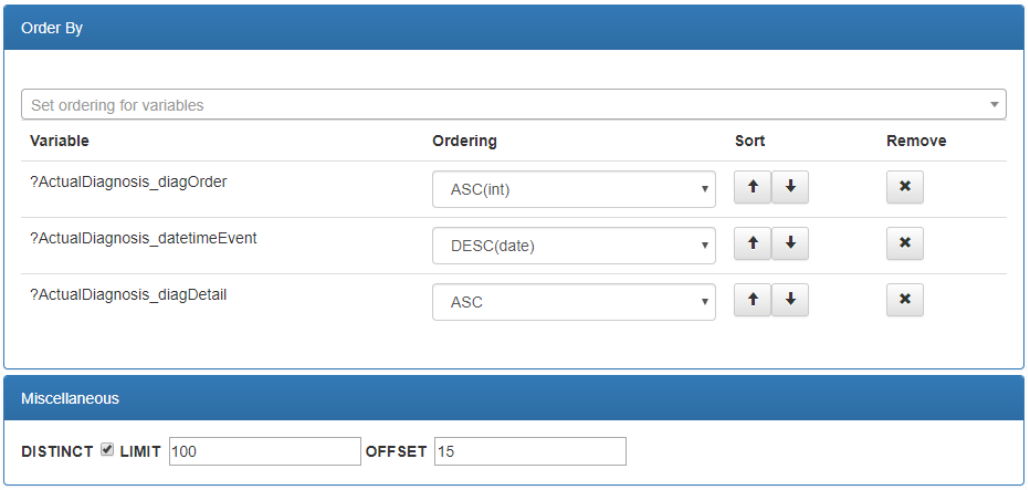
fikaci řazení, tedy vzestupné nebo sestupné řazení, pořadí řazení a jejich odstranění. Proměnné na vrcholu tabulky jsou řazeny jako první, jejich pořadí lze ovlivňovat šipkami nahoru a dolů. Veškeré změny jsou ihned promítány do dotazu.

V případě, že je řazen atribut nebo agregační funkce a jsou odstraněny z dotazu, aplikace je automaticky odstraní i z řazení.

Další možností nastavení je jestli dotaz bude obsahovat klíčové slovo DISTINCT. Omezení počtu výsledných záznamů a případný offset lze zde také nastavit. Aplikace přijímá pouze čísla pro tyto dvě funkce a do dotazu se promítnou pouze tehdy pokud jsou větší než nula.

Ukázka na obrázku B.9.

Obrázek B.9: Sektor vlastností



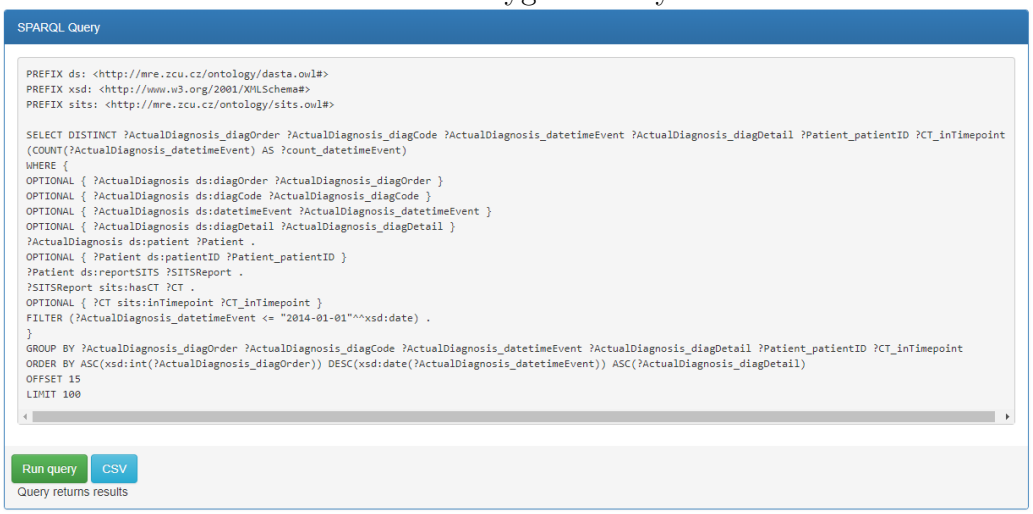
B.8 Sektor dotazu

V poslední části je zobrazován generovaný dotaz. Je zde pouze zobrazen, nelze upravovat. Aplikace vždy informuje, jestli daný dotaz vrací nějaká data, což je vypisováno pod tlačítkem pro běh dotazu. Toto tlačítko je označeno textem `Run query` a je možno jím nechat dotaz běžet proti endpointu. Tlačítko `CSV` stáhne výsledky ve formátu CSV.

Aplikace vždy vytváří syntakticky správný dotaz. Pokud na dotaz nejsou vráceny žádné výsledky, je to dáno neexistencí podgrafu trojic, jenž by na dotaz odpovídal.

Ukázka generovaného dotazu na obrázku B.10.

Obrázek B.10: Vygenerovaný dotaz



Výsledky dotazu jsou pak zobrazeny pod tímto panelem v podobě tabulky nebo JSON, mezi nimiž se dá libovolně přepínat.

Tabulka je rozdělena na sloupce dle proměnných, které dotaz vybíral. Jednotlivé řádky pak představují výsledné hodnoty. Hodnoty sloupců mohou být prázdné, pokud se používá OPTIONAL.

JSON forma dotazu odpovídá odpovědi endpointu, jež aplikace na dotaz obdržela.

Nastavení přepisování URI je možno využít, pokud uživatel místo URI ve výsledcích chce vidět textovou reprezentaci daných URI. Lze nastavit název anotační vlastnosti pro přepis a její jazyk. Aplikace potom prioritně URI nahrazuje textem. Hodnota URI naznačuje vrátit URI bez přepisu, hodnota C u jazyků naznačuje hodnotu bez jazykového tagu.

Pokud je spuštěna aplikace v rámci Sparkle, tento sektor obsahuje pouze generovaný dotaz. Pro běh dotazu je možné se přepnout do jednoho z ostatních editorů - formulářového či textového - a nechat dotaz běžet pomocí zabudované funkcionality `Evaluate` ve Sparkle. Aplikace pak sama výsledky zobrazuje. Vygenerovaný dotaz je automaticky importován do těchto editorů, jde pouze o jednostrannou funkcionalitu.

Ukázka tabulky výsledků vygenerovaného dotazu na obrázku B.11.

Obrázek B.11: Výsledky dotazu

Results

Output attributes: form:code,rdfs:label,URI

Output languages: cs,en,C

Table JSON

ActualDiagnosis_diagOrder	ActualDiagnosis_datetimeEvent	ActualDiagnosis_diagDetail	Patient_patientID	CT_inTimepoint
1	2013-09-20	Mozkový infarkt způs.embolií mozkových tepen	123	po 24 hodinách
1	2013-09-20	Mozkový infarkt způs.embolií mozkových tepen	123	příjem
1	2013-09-19	Diabetes mellitus 2. typu bez komplikací na PAD, těžká obezita	123	po 24 hodinách
1	2013-09-19	Diabetes mellitus 2. typu bez komplikací na PAD, těžká obezita	123	příjem
1	2013-08-28	Smíšená hyperlipidemie	892	příjem

C Příklad dotazu

```
1  PREFIX ds: <http://mre.zcu.cz/ontology/dasta.owl#>
2  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3
4  SELECT DISTINCT ?Patient_lastName ?Patient_patientID ?
       MedicalExamination_reportTitle ?ActualDiagnosis_diagOrder ?
       ActualDiagnosis_diagCode ?ActualDiagnosis_datetimeEvent
5  WHERE {
6  OPTIONAL { ?Patient ds:lastName ?Patient_lastName }
7  OPTIONAL { ?Patient ds:patientID ?Patient_patientID }
8  ?Patient ds:clinicalEvent ?MedicalExamination .
9  OPTIONAL { ?MedicalExamination ds:reportTitle ?
       MedicalExamination_reportTitle }
10 ?MedicalExamination ds:diagnosis ?ActualDiagnosis .
11 OPTIONAL { ?ActualDiagnosis ds:diagOrder ?
       ActualDiagnosis_diagOrder }
12 OPTIONAL { ?ActualDiagnosis ds:diagCode ?
       ActualDiagnosis_diagCode }
13 OPTIONAL { ?ActualDiagnosis ds:datetimeEvent ?
       ActualDiagnosis_datetimeEvent }
14 FILTER (xsd:int(?Patient_patientID) >= 200) .
15 FILTER ((regex(str(?MedicalExamination_reportTitle), 'CT', 'i'))
       ) .
16 FILTER (?ActualDiagnosis_datetimeEvent >= "2013-01-01"^^xsd:date
       ) .
17 }
18 ORDER BY ASC(xsd:int(?Patient_patientID)) ASC(?
       ActualDiagnosis_diagOrder) DESC(xsd:date(?
       ActualDiagnosis_datetimeEvent))
19 OFFSET 5
20 LIMIT 10
```