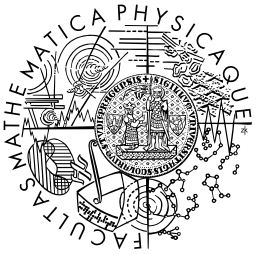


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Daniel Rous

Nástroj pro vizualizaci RDF geodat na mapovém podkladu

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové a datové inženýrství

Praha 2019

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Rád bych poděkoval vedoucímu práce, doc. Mgr. Martinu Nečaskému, Ph.D., za odborné vedení a veškerou poskytnutou pomoc během vypracovávání této bakalářské práce.

Název práce: Nástroj pro vizualizaci RDF geodat na mapovém podkladu

Autor: Daniel Rous

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D., Katedra softwarového inženýrství

Abstrakt: Cílem této práce je navrhnout, implementovat a otestovat webovou aplikaci, která v mapě vizualizuje data nad datovým modelem RDF se zohledněním uživatelem vybrané metriky, která určí míru zabarvení vizualizovaných entit. Aplikace bude uživateli usnadňovat výběr entit pro vizualizaci a bude za určitých podmínek na základě analýzy uživatelem nahraného schématu dat automaticky detekovat numerické a lokační údaje. Při vizualizaci také uživateli umožní filtraci dat v několika vlastnostech. Aplikace bude otestována na dvou různých zdrojích dat, přičemž jedním z nich bude datová sada Centrálního registru dotací (IS CEDR III) a druhým bude datová sada agend z Registru práv a povinností (RPP).

Klíčová slova: otevřená data RDF SPARQL vizualizace heat mapa

Title: Tool for RDF geodata visualization on a map

Author: Daniel Rous

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Department of Software Engineering

Abstract: The goal of this thesis is to design, implement and test a web application, which will visualize RDF data in a heat map. The heat will be defined by the user's metric data selection. The application will simplify user's selection needed for visualization by automatic detection of metric and location data in loaded schema. User will have an opportunity to filter data by some data properties. The application will be tested on two different data sources. One of them will be Central register of grants (IS CEDR III). The other one will be the agenda dataset from Rights and Duties Register (RPP).

Keywords: open data RDF SPARQL visualization heatmap

Obsah

Úvod

Na webu se vyskytuje nepřeberné množství dat, které mimo jiné často obsahují identifikační lokační údaje jistých entit ve formě GPS souřadnic, adresy nebo jiných jednoznačných identifikátorů. Zároveň se s těmito daty majícími identifikátory lokace pojí různá metrická data. Protože je vizualizace dat způsob, jak je možné lidem pomoci lépe porozumět velkému množství dat v krátkém časovém horizontu, nabízí se možnost metriku entit, která značí libovolný numerický údaj, vizualizovat v heat mapě.

Řada dat je na webu dostupná v podobě otevřených nebo v lepším případě i propojených dat. Za příklad uvedeme portály https://www.wikidata.org/ nebo https://dbpedia.org, jež pracují s daty z internetové encyklopedie https://www.wikipedia.org/ a státní portál zabývající se otevřenými daty https://data.gov.cz/, kde lze nalézt nespočet datových sad, které jsou ve formě otevřených (resp. propojených) dat zveřejňovány z nařízení vlády č. 425/2016 Sb., které se opírá o zákon č. 106/1999 Sb., o svobodném přístupu k informacím.

Jedním z takto zveřejňovaných a pravidelně aktualizovaných datových sad je Centrální registr dotací (IS CEDR III), který obsahuje veškeré údaje o přidělování účelových dotací ze státního rozpočtu. Tento datový zdroj splňuje všechny naše požadavky, tedy obsahuje spoustu metrických dat pro možnost vizualizace a zároveň hned v několika formách jednoznačný identifikátor bodu na mapě, a tak byla tato datová sada vyhodnocena jako vhodná pro odprezentování výsledného naimplementovaného nástroje.

Druhým datovým zdrojem určeným k prezentaci je datová sada agend z Registru práv a povinností (RPP).

Pro běžného uživatele je sice snadné zájmová data pro vizualizaci najít, ale je pro něj obtížné s těmito daty pracovat, jelikož to vyžaduje pokročilou znalost datového modelu RDF a dotazovacího jazyka SPARQL.

Cílem této práce je poskytnutí abstrakce uživatelům a možnost jejich odproštění od technické vrstvy jazyka SPARQL a datového modelu RDF a poskytnout jim nástroj pro vizualizaci dat v mapě. Protože je pro finální vizualizaci potřeba zvolit výběr, nástroj má také za cíl usnadnit uživatelům výběr pomocí automatické detekce metrických a lokačních entit na základě analýzy schématu nahraného datového zdroje.

Struktura tohoto textu vypadá následovně: v první kapitole budou stručně popsány využívané technologie. Na toto naváže kapitola druhá, jež se bude zabývat analýzou požadavků a analýzou datových zdrojů. Tam budou představeny jak funkční, tak nefunkční požadavky na vývoj aplikace, společně s typickými příklady užití a bližší analýzou datového zdroje IS CEDR III. Třetí kapitolou bude návrh řešení, kde budou vysvětleny všechny zvolené postupy, popsány zain-

tegrované knihovny a vysvětleny klíčové algoritmy. Po kapitolách s uživatelskou a programátorskou dokumentací již následuje kapitola poslední, která se týká testování práce, kam spadá kromě unit testování i celkové zhodnocení použitelnosti aplikace, které bude měřeno pomocí System Usability Scale.

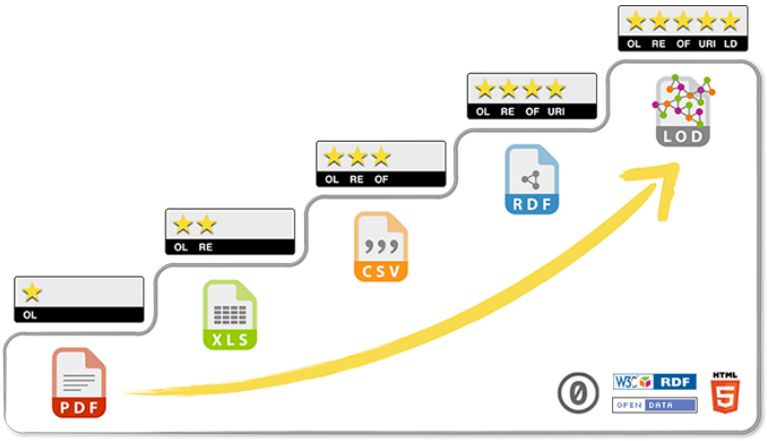
1. Technologie

V této kapitole budou stručně představeny hlavní technologie, které byly extenzivně využívány pří vypracovávání této bakalářské práce.

1.1 Otevřená a propojená data

Termín otevřená data označuje veřejně přístupné data na internetu, které mohou uživatelé dále využívat pro svou potřebu. Ku příkladu je možné nad těmito daty vyvíjet aplikace integrující data z jednoho či několika takovýchto zdrojů. Definice otevřených dat je poměrně volná, a tak bylo zavedeno 5 hvězdičkové hodnocení otevřených dat pro jejich lepší kategorizaci. Nejlepší data jsou pětihvězdičková. Pro data N-té úrovně platí vše, co pro data (N−1)-ní úrovně a k tomu něco navíc. V tabulce 1.1 proto uvádíme pouze nově vzniklé rozdíly.

Obrázek 1.1[1] znázorňuje kategorizaci otevřených dat s často využívanými formáty spadajícími do těchto kategorií.



Obrázek 1.1: Kategorie otevřených dat

[1] https://5stardata.info/en/

Hvězdy	Popis
1	data jsou vůbec zveřejněna na webu pod otevřenou licencí
2	data jsou strukturovaná (tj. ve strojově zpracovatelné podobě)
3	formát dat je neproprietární, otevřený
4	jako identifikátory entit jsou použity URI
5	data jsou napojená na další otevřená (propojená) data

Tabulka 1.1: Tabulka kategorizace otevřených dat

1.1.1 Propojená data

Narazili jsme na termín propojených dat. Jako propojená data se označují taková data, které mají 5 hvězd dle výše zmíněné stupnice, jak uvádí Berners-Lee (2006). Jinými slovy splňují tyto podmínky:

1. identifikátory entit jsou URI
2. k těmto URI se dá přistoupit přes protokol HTTP, aby mohl uživatel zjistit něco více o těchto datech
3. na těchto HTTP URI se nachází užitečné informace popisující danou entitu v RDF reprezentaci
4. data se odkazují na další URI souvisejících entit

1.2 Resource Description Framework

Resource Description Framework (zkráceně RDF) je datovým modelem pro výměnu dat na webu. Zásadním pro něj je koncept trojic. Každá trojice se skládá ze subjektu, predikátu a objektu a je nějakým tvrzením o subjektu. Predikáty pojednávají o nějaké vlastností subjektu a jsou také označeny pomocí URI, které pochází ze slovníků nebo ontologií.

Specifikace RDF 1.1 [2] Je souborem několika dalších otevřených specifikací konsorcia W3C. Ty například definují standardy pro různé serializace RDF dat.

1.2.1 Slovníky a ontologie

Slovníky jsou kolekcemi predikátů, které definují vztahy. Jak je uvedeno na stránce World Wide Web Consorcium (W3C), ontologie značí – podobně jako slovníky – kolekce predikátů, nicméně se tento termín používá spíše pro komplexnější a formálnější definice vztahů.

Známé a námi využívané slovníky a ontologie Známé a pro nás důležité slovníky a ontologie jsou vypsány v seznamu níže. U každého slovníku, resp. ontologie, je v závorkách uveden standardní prefix.

- Web Ontology Language (owl) je ontologie jazyka pro tvorbu tříd, vlastností, slovníků a dalších ontologií. Jedná se o specifikaci konsorcia W3C.
- RDF Schema (rdfs) slouží také pro tvorbu dalších slovníků a ontologií, ale předchozí ontologie nabízí mnohem širší paletu nástrojů. Též se jedná o specifikaci konsorcia W3C.
- XML Schema (xsd) je známý slovník všem, kdo se v životě setkali se značkovacím jazykem XML. V RDF světě je využíván pro jeho širokou definici datových typů.

[2] https://www.w3.org/TR/rdf11-concepts/

- CEDR (cedr) je slovník datové sady IS CEDR III, ve kterém je definováno 21 tříd. Podrobná analýza IS CEDR III a mimo jiné i tohoto slovníku je k vidění v kapitole 2.4

1.2.2 Typy RDF dat

Neboli „RDF terms“. V RDF existuje několik typů dat. Jsou to následující.

- **URI odkaz:** Reprezentuje validní a absolutní URI adresu.
- **Literál:** Reprezentují skutečné hodnoty atributů. Ze základních typů sem spadají datum, číslo, řetězec, boolean apod. Literály mohou mít přiřazen datový typ (např. “1”^^xsd:integer) nebo jazykovou značku (např. “Popisek”@cs)
- **Prázdný uzel:** Reprezentuje zdroj, který nemá přiřazenou žádnou hodnotu.

1.2.3 Serializace RDF dat

RDF data lze serializovat různými způsoby a do různých formátů. Mezi hlavní patří Turtle, N-Triples, JSON-LD nebo RDF/XML. Na příkladu si předvedeme jednotlivé serializace.
Tvrzení: „Jan Novák vystudoval Univerzitu Karlovu.“
Daným částem trojice přiřaďme tyto URI:

- *Subjekt*: Jan Novák (http://example.cz/osoby/10000000)
- *Predikát*: vystudovat (http://dbpedia.org/ontology/almaMater)
- *Objekt*: Univerzita Karlova (http://example.cz/instituce/1348)

Turtle serializace[3]

```
@prefix dbo:  <http://dbpedia.org/ontology/> .
<http://example.cz/osoby/10000000> dbo:almaMater
<http://example.cz/instituce/1348> .
```

N-Triples serializace[4]

```
<http://example.cz/osoby/10000000>
<http://dbpedia.org/ontology/almaMater>
<http://example.cz/instituce/1348> .
```

JSON-LD serializace[5]

```
{ "@graph": [
    { "@id": "http://example.cz/osoby/10000000",
      "http://dbpedia.org/ontology/almaMater": { "@id":
      "http://example.cz/instituce/1348"} }
] }
```

[3] https://www.w3.org/TR/turtle/
[4] https://www.w3.org/TR/n-triples/
[5] https://www.w3.org/TR/json-ld11/

RDF/XML serializace[6]

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:dbo="http://dbpedia.org/ontology/">
<rdf:Description rdf:about="http://example.cz/osoby/10000000">
<dbo:almaMater rdf:resource="http://example.cz/instituce/1348"/>
</rdf:Description>
</rdf:RDF>
```

1.3 SPARQL

Jak je uvedeno na World Wide Web Consorcium (W3C), SPARQL je standardním dotazovacím jazykem pro manipulaci s RDF daty. SPARQL dotazy můžeme spouštět nad RDF databázemi, které se také nazývají jako „triple stores“.

1.3.1 Úložiště

RDF databáze často fungují jako služba serveru a umí pracovat s dotazy přenášenými přes protokol HTTP. Existuje i SPARQL protokol, který je nadstavbou HTTP protokolu a definuje přesné události, které se mají po různých typech HTTP dotazů a s jakými parametry stát.

Některé vybrané SPARQL endpointy Data čerpána z World Wide Web Consorcium (W3C) a z dokumentace IS CEDR III.

- http://dbpedia.org/sparql
- https://query.wikidata.org/
- http://cedropendata.mfcr.cz/c3lod/cedr/sparql

1.3.2 Typy SPARQL dotazů

Existují 4 typy SPARQL dotazů, které si v následujících odstavcích popíšeme a uvedeme ke každému typu i příklad takového dotazu s vysvětlením a SPARQL endpoint, na kterém jde takový dotaz spustit.

Typ SELECT

Typ SELECT se typicky využívá pro standardní dotazování na data se do datového zdroje, je obdobný SQL dotazu SELECT.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?vysokaSkola ?pocetStudentu ?pocetObyvatel ?pomer
WHERE {
```

[6] https://www.w3.org/TR/rdf-syntax-grammar/

```
  ?vysokaSkola a dbo:University .
  ?vysokaSkola dbo:numberOfStudents ?pocetStudentu .
  ?vysokaSkola dbo:country
  <http://dbpedia.org/resource/Czech_Republic> .
  ?vysokaSkola dbo:city/dbo:populationTotal ?pocetObyvatel .
  BIND((xsd:double(?pocetStudentu) / xsd:double(?pocetObyvatel))
  as ?pomer) .
}
ORDER BY DESC(?pomer)
```

SPARQL Endpoint: http://dbpedia.org/sparql

Vysvětlení: Vybíráme vysoké školy, jejich počty studentů a počet obyvatel města, ve kterém se daná škola nachází. Řadíme sestupně podle vypočítaného poměru těchto dvou počtů.

Typ CONSTRUCT

Typ CONSTRUCT se používá pro sestrojení trojic do modelu RDF.

```
CONSTRUCT
{
  ?s ?p "1" .
}
WHERE {
  ?s ?p "one" .
}
LIMIT 10
```

SPARQL Endpoint: http://dbpedia.org/sparql

Vysvětlení: Najdeme prvních deset trojic, které mají pro libovolný subjekt (?s) a libovolný predikát (?p) hodnotu objektu (?o) „one“. Pro tyto vybraná data sestrojíme trojice mající stejné hodnoty subjektu a predikátu, ale řetězec “one” bude změněn v nově sestrojených trojicích na RDF literál „1“.

Typ ASK

Typ ASK vrací True/False odpověď.

```
ASK
WHERE
{
  ?uni a <http://dbpedia.org/ontology/University> .
  ?uni <http://dbpedia.org/ontology/city>
  <http://dbpedia.org/resource/Český_Krumlov> .
}
```

SPARQL Endpoint: http://dbpedia.org/sparql

Vysvětlení: Ptáme se, zda existuje vysoká škola v Českém Krumlově.

Typ DESCRIBE

Typ DESCRIBE[7] vrací RDF graf obsahující trojice, které popisují data, na které se dotazujeme

```
DESCRIBE <http://dbpedia.org/resource/Český_Krumlov>
LIMIT 10
```

SPARQL Endpoint: http://dbpedia.org/sparql

Vysvětlení: Necháváme si popsat jednu konkrétní entitu s uvedenou URI, tedy Český Krumlov. Vybíráme prvních deset trojic, ve kterých se jako subjekt nebo objekt tato entita vyskytuje.

1.4 OpenStreetMap

Pro provedení vizualizace v naší aplikaci je potřeba mít k dispozici mapová data. OpenStreetMap (OSM) je poskytuje pod otevřenou licencí Open Database Licence (ODbL). Tímto tedy integrujeme další otevřená data do naší aplikace. OpenStreetMap poskytuje i několik API, která můžeme využívat. Pro nás je důležité hlavně Nominatim API.

1.4.1 Nominatim API

Slouží ke **geokódování** a také pro reverzní geokódování. Neběží pouze na serverech OSM, ale toto API je poskytováno i mnoha dalšími servery. Dokonce si můžeme rozchodit své vlastní Nominatim API na našem serveru. Výhodou by bylo snížení režie přenosu dat po síti. Nevýhodou je časově náročná inicializace služby. Data OSM zabírají přes 700 GB. Import do databáze je několikadenní záležitost[8].

[7] Jedná se o jediný typ, který nemusí obsahovat povinně klauzuli WHERE
[8] http://nominatim.org/release-docs/latest/admin/Installation/

2. Analýza požadavků a dat

2.1 Funkční požadavky

- Aplikace by měla být schopna vykreslit heat mapu na základě selekce uživatele. Heat mapa by měla být interaktivní, mělo by být možné s ní pohybovat, přibližovat a oddalovat se v ní. Data, které bude aplikace vizualizovat v mapě, budou v datovém modelu RDF. Vstupní data pro aplikaci se skládají ze schématu datového zdroje, což je podmnožina RDF slovníků či ontologií, a ze samotných dat určených pro vizualizaci v mapě. Nahrání dat může uživatel provést ze souboru nebo SPARQL endpointu.
- Aplikace by měla umět vizualizovat nahrané vstupní schéma a automaticky v něm detekovat metrické data a některé data určující lokaci entit. Tato detekce bude usnadňovat uživateli provedení selekce dat potřebných pro vizualizaci v mapě.
- Aplikace by měla uživateli umožnit vizualizaci řádově tisíce bodů v heat mapě a umožnit jejich filtraci.

2.2 Nefunkční požadavky

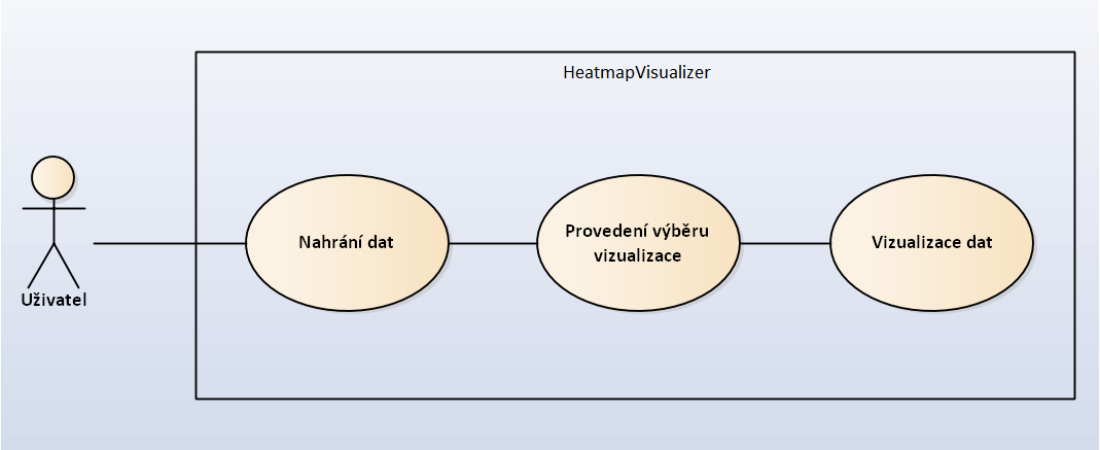
- Aplikace bude vyvíjena nad frameworkem .NET.
- Aplikace bude využívat otevřených mapových dat z OpenStreetMap.
- Aplikace bude využívat pro vizualizaci schématu knihovnu D3.js.
- Aplikace bude využívat pro vizualizaci heat mapy knihovnu Leaflet.js.
- Používání aplikace by mělo být pro uživatele návodné a intuitivní.
- Aplikace by měla být snadno rozšiřitelná a snadno modifikovatelná.

2.3 Příklady užití aplikace

- **Vizualizace.** Uživatelé – například datoví novináři (analytická činnost) nebo proaktivní občané (veřejný zájem) – naší aplikace si budou nyní mít možnost vizualizovat data v datovém modelu RDF, ve kterém jsou často zveřejňována data různých státních institucí aj.

Na obrázku 2.1 je typický příklad užití znázorněn. Nejprve musí uživatel nahrát do aplikace data se schématem, které se k nim vztahuje. V dalším kroku po vizualizaci schématu uživatel vybere, která data se mají do mapy vizualizovat a ve třetím kroku již uživatel pracuje se samotnou vizualizací, kde může filtrovat data, manipulovat s mapou a s vykreslenými objekty. Na obrázcích 2.2 a 2.3, které znázorňují diagramy aktivit, jsou dva kroky detailněji rozebrány.

Na obrázku 2.2 je k vidění diagram aktivit při nahrávání datového zdroje do úložiště na našem serveru. Pro úplnost uvedeme, že krok *Nahrání dat* neznamená překopírovávání dat na naše úložiště v případě zvolení SPARQL endpointu. Není to totiž potřeba, protože je účelem tohoto kroku je pouze zajištění toho, abychom měli kam posílat naše SPARQL dotazy. Obrázek 2.3 pojednává o tom, které všechny scénáře mohou nastat při provádění výběru vizualizace.



Obrázek 2.1: Příklad užití aplikace

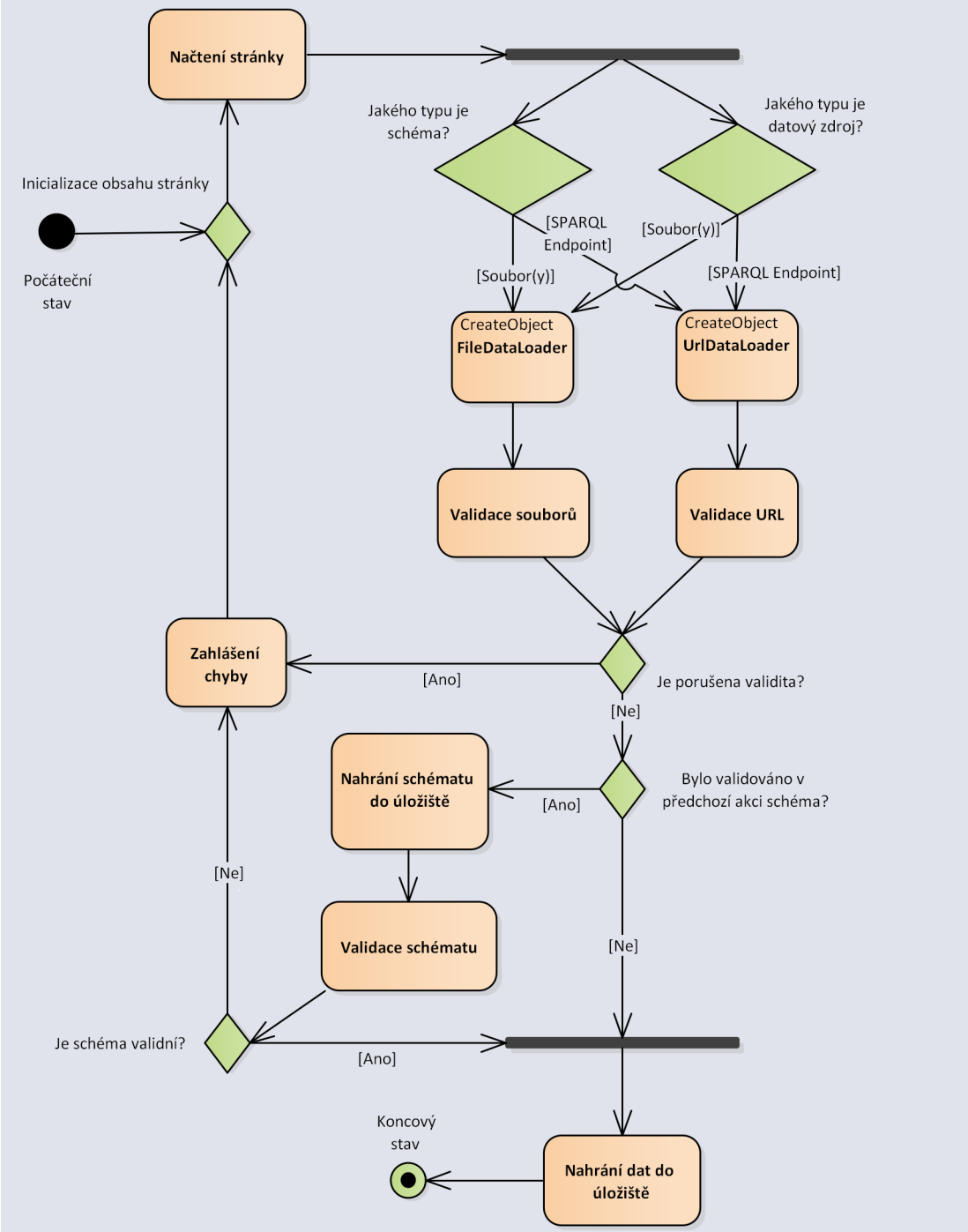
2.4 Datová sada IS CEDR III

Centrální registr dotací neboli IS CEDR III je datovou sadou, která integruje data z několika datových zdrojů a systémů. Datová sada obsahuje veškeré údaje o poskytnutých účelových dotacích od roku 1999.

Mezi integrované datové sady patří tyto:

- datová sada IS CEDR III
- datová sada ČSÚ
- datová sada SZR
- datová sada MMR
- datová sada RUIAN
- datová sada EDS/SMSV
- datová sada ARES
- datová sada ROB
- datová sada ISDP

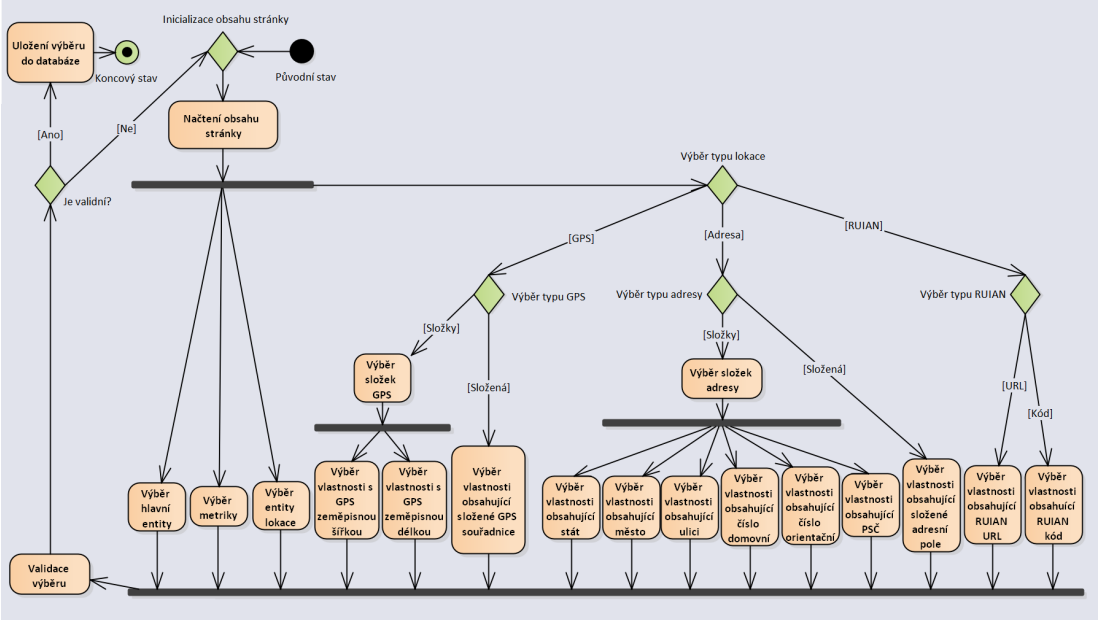
Datová sada IS CEDR III obsahuje definici 21 tříd. Pro náš zjednodušený demonstrativní příklad jsou nejdůležitější třídy *cedr:PravnickaOsoba*, *cedr:AdresaSidlo*, *cedr:Dotace* a *cedr:Rozhodnuti*. Následuje stručný popis sémantiky těchto tříd a jejich vlastností.



Obrázek 2.2: Diagram aktivit pro nahrání vstupu uživatele

2.4.1 Třída cedr:PravnickaOsoba

- Reprezentuje právnickou osobu.
- Právnické osoby obsahují i informace z datové sady ARES.
- Dále obsahuje údaj o tom, na jaké adrese právnická osoba sídlí a jaké dotace obdržela.



Obrázek 2.3: Diagram aktivit provedení výběru vizualizace

2.4.2 Třída cedr:AdresaSidlo

- Reprezentuje adresu sídla – tedy adresu právnických osob, fyzických osob podnikajících a cizinců.
- Obsahuje pole s nestrukturovanou adresou (adresa jako jeden řetězec), jednotlivé složky adresy (stát, město, ulice, ...) a kód RUIAN, který se používá pro identifikaci adres v rámci ČR.

Poznámka: v současné době ve verzi datové sady k 6. červnu 2019 nejsou žádné adresy evidovány v nestrukturované formě. Jako důkaz poslouží následující SPARQL dotaz, který si kdokoliv může spustit nad SPARQL endpointem IS CEDR III.

```
SELECT DISTINCT ?adresa
WHERE {
  ?subjekt
  <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#adresaText>
  ?adresa .
}
```

2.4.3 Třída cedr:Dotace

- Reprezentuje dotaci.
- Obsahuje odkazy na jednotlivá rozhodnutí spadající pod jednu konkrétní dotaci a různé informace o tom, do jakého programu a podprogramu dotace spadala apod. Pro náš demonstrativní příklad budou nejdůležitější odkazy vedoucí na rozhodnutí o dotacích.

2.4.4 Třída cedr:Rozhodnuti

- Reprezentuje rozhodnutí o přidělení dotace.
- Obsahuje informace příslušící tomuto rozhodnutí jako jsou například částky požadované a rozhodnuté na danou dotaci, jaký poskytovatel dotace ji poskytl a měnu, ve které byla dotace vyplacena a plno dalších informací.

Poznámka: v současné době jsou všechny dotace vypláceny v českých korunách, což zjednodušuje selekci dat vykreslovaných do mapy. Nemusíme se starat o nějaké sjednocování částek a podobné záležitosti. Že je měna skutečně evidována pouze jedna ve verzi datové sady k 6. červnu 2019, se každý může přesvědčit spuštěním následujícího dotazu na SPARQL endpointu IS CEDR III.

```
SELECT DISTINCT ?mena
WHERE {
  ?rozhodnuti <http://cedropendata.mfcr.cz/c3lod/isdp/vocabIsdp/c
  ommon/v1#menaKod> ?mena
  .
}
```

Níže je k vidění na obrázku 2.4 schéma datové sady z oficiální dokumentace IS CEDR III.

2.4.5 Přístup k datům

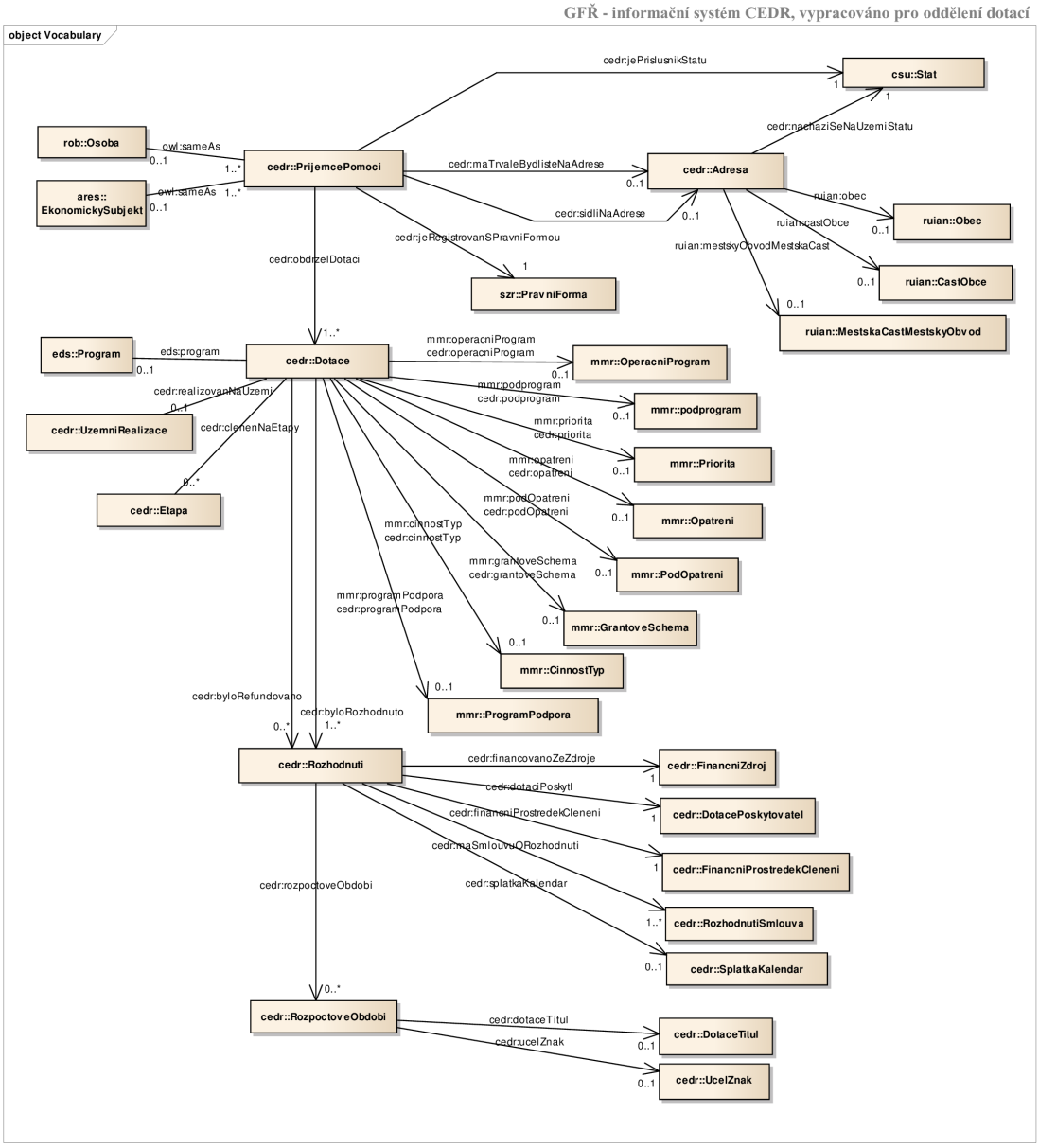
K samotným datům datové sady IS CEDR III se dá dostat několika způsoby.

Prvním je veřejný SPARQL endpoint na adrese http://cedropendata.mfcr.cz/c3lod/cedr/sparql.

Druhou možností je stažení souborů datové sady ve formátu N-Triples nebo CSV, které jsou dohledatelné na stránce http://cedr.mfcr.cz/Cedr3InternetV419/OpenData/OpenDataDumpPage.aspx.

Celkové velikosti vystavených dat jsou circa 3 GB pro formát CSV a circa 34 GB pro formát N-Triples. SPARQL endpoint funguje jako REST API a data by z něj dle dokumentace měly být dostupné ve formátech: HTML, CSV, TSV, Turtle, N-Triples, RDF/XML, XML, JSON, RDF+JSON a JSON-LD. V průběhu vypracovávání bakalářské práce jsem se ale setkal s tím, že spolehlivě fungují pouze formáty HTML, CSV, TSV, XML, JSON a RDF+JSON. Také k 6. červnu 2019 nefunguje export výsledku SPARQL dotazu ve zbylých v předchozí větě nezmíněných formátech.

Slovník datové sady, jež v naší terminologii odpovídá schématu datového zdroje, je dostupný na odkazu http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#.



Obrázek 2.4: Schéma IS CEDR III

2.4.6 Lokace entit

Lokace je v IS CEDR III dána vlastnostmi ve třech třídách. Jedná se o třídy *cedr:Adresa*, *cedr:AdresaSidlo* a *cedr:AdresaTrvaleBydliste*. Z názvů vyplývá, co každá třída reprezentuje. Třídy *cedr:AdresaSidlo* a *cedr:AdresaTrvaleBydliste* jsou podtřídami třídy *cedr:Adresa*. Ta obsahuje společné vlastnosti adresy trvalého bydliště a adresy sídla.

Adresa sídla se přiřazuje právnickým osobám, fyzickým osobám podnikajícím a cizincům. Adresa trvalého bydliště se přiřazuje fyzickým osobám, fyzickým osobám podnikajícím a cizincům. Rozdíly mezi těmito dvěma třídami jsou v tom, že u adresy trvalého bydliště není určena přesná lokalizace místa a jsou neúplné a tudíž není možné z těchto adres získat GPS souřadnice, a tedy není možné

tyto entity vizualizovat v mapě. Neboli můžeme vizualizovat pouze entity mající adresu typu *cedr:AdresaSidlo*. Adresa sídla obsahuje buď adresu ve složkách (jak jsme již zmiňovali), adresu v textu – která však v datech není obsažena – a RUIAN identifikátor.

Co je to RUIAN a co je RUIAN identifikátor bude zmíněno v následující kapitole. Níže je seznam, ze kterých složek se adresa skládá a které vlastnosti třídy *cedr:AdresaSidlo* tomu odpovídají. V závorkách za vlastností je RDF typ dané vlastnosti.

Složená adresa

- **adresa:** cedr:adresaText *(RDF literál)*

Složky adresy

- **stát:** cedr:nachaziSeNaUzemiStatu *(URL)*
- **obec:** spa:obecNazev *(RDF literál)*
- **ulice:** spa:uliceNazev *(RDF literál)*
- **číslo domovní:** spa:objektCisloDomovni *(RDF literál)*
- **číslo orientační:** spa:objektCisloOrientacni *(RDF literál)*
- **PSČ:** spa:psc *(RDF literál)*

RUIAN

- **identifikátor:** spa:adresniMistoKod *(RDF literál)*

2.4.7 Registr územní idenfitikace, adres a nemovitostí (RÚIAN)

RUIAN, resp. přesněji RÚIAN, jak stojí v Český úřad zeměměřičský a katastrální spadá pod správu Českého úřadu zeměměřického a katastrálního (ČÚZK). V registru je všem adresám v ČR přiřazeno jednoznačné identifikační celé číslo, přes které se dá dostat k celé adrese, která je pod tímto identifikátorem uchována. Toto identifikační číslo budeme v naší terminologii nazývat jako „RUIAN identifikátor“ či „RUIAN kód“.

2.4.8 Data demonstrativního příkladu

Zde shrneme, které vlastnosti budeme pro vizualizaci potřebovat a co budou značit.

- **Hlavní entita vizualizace** je entita, se kterou se pojí některá metrická hodnota a dá se do ní přes libovolný (i nulový) počet vlastností dostat do lokační entity vizualizace. Musí být RDF typu URI odkaz a mít typ třídy. Formálně řečeno musí v datech existovat trojice

(<`hlavní_entita`>, rdf:type, owl:Class).

Výběr: třída *cedr:PravnickaOsoba*

- **Metrické hodnoty entit** určují míru zabarvení jednotlivých entit. Musí být RDF literál.

  Výběr: vlastnost *cedr:castkaPozadovana* ve třídě *cedr:Rozhodnuti*

- **Lokační entita vizualizace** je entita, se kterou se pojí nějaký identifikátor lokace. Musí být RDF typu URI odkaz a mít typ třídy. Formálně řečeno musí v datech existovat trojice (<`lokační_entita`>, rdf:type, owl:Class).

  Výběr: třída *cedr:AdresaSidlo*

- **Identifikátor lokace** je identifikátor, kterým lze jednoznačně určit lokaci dané entity v mapě. Například se jedná o adresu v různých formách, o GPS souřadnice v různých formách a jiné identifikátory, mezi které patří například kód RUIAN.

  Výběr:

  – Složená adresa. Vlastnosti byly popsány v kapitole 2.4.6. V současné době není možné použít tento typ vizualizace, jelikož taková data v datové sadě chybí.

  – Složky adresy. Vlastnosti byly popsány v kapitole 2.4.6.

  – RUIAN kód. Vlastnosti byly popsány v kapitole 2.4.6.

3. Návrh řešení

Tato kapitola obsahuje popis všech provedených rozhodnutí, které jsme učinili a také problémů, na které jsme při vypracovávání narazili.

3.1 Manipulace s RDF daty v .NET

Pro práci s RDF daty jsme vybrali knihovnu dotNetRDF (http://www.dotnetrdf.org/). Výběr ovlivnilo to, že dotNetRDF je již zavedenou open source knihovnou, nabízí pro nás užitečné API, přímo z kódu můžeme operovat s různými triple stores a je mezi nimi i Apache Jena Fuseki, jehož použití je naším nefunkčním požadavkem. Na závěr má obsáhlou a přehlednou dokumentaci na portálu GitHub (https://github.com/dotnetrdf/dotnetrdf/wiki).

3.2 Nástroj pro automatizované extrahování dat z IS CEDR III

3.2.1 Úvod

Při vývoji práce jsme navíc naimplementovali i nástroj pro automatizované extrahování dat z IS CEDR III. Důvodem bylo to, že celá databáze Centrálního registru dotací je poměrně velká (circa 34 GB) a my jsme potřebovali z tohoto množství dat vybrat určitou jejich podmnožinu pro testování aplikace. To není tak snadné, jelikož pro extrakci úplné podmnožiny dat potřebujeme provést buď poměrně netriviální CONSTRUCT dotaz, anebo data iterovaně stahovat po částech. Rozhodli jsme se pro iterované stahování, jelikož tento algoritmus bude jistě fungovat i případně v budoucnosti na dalších SPARQL endpointech v případě rozšiřování naší aplikace a zobecňování naší implementace.

Současný nástroj je velice jednoduchý a je prozatím pouze konzolovou aplikací s natvrdo nastavenými parametry v kódu, nicméně by nastavování těchto parametrů šlo velice snadno upravit tak, aby se parametry braly z konfiguračního souboru nebo z argumentů příkazové řádky. Mohli bychom dokonce do budoucna tento nástroj poskytovat uživatelům naší aplikace pro to, aby si do souborů mohli vyextrahovat jimi požadovaná data. Museli by definovat pouze několik parametrů, mezi něž patří:

- URL SPARQL endpointu, kam se máme dotazovat.
- Hlavní entitu, která je typu owl:Class.[1]. Například: http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#PravnickaOsoba.
- Volitelně predikát, který má hlavní entita povinně obsahovat. Například: http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#sidliNaAdrese nebo http://cedropendata.mfcr.cz/c3lod/ruian/vocabRUIAN#

[1] Vztah mezi hlavní entitou vizualizace a hlavní entitou stahování je 1:1

adresniMistoKod. Hodí se pro to, aby se při vizualizaci nezjistilo, že v extrahovaných datech hlavní entita nemá požadovanou vlastnost a nelze ji tedy lokalizovat.

- Prefixy dalších entit, které nějak souvisí s hlavní entitou a chceme získat jejich detail. Například: http://cedropendata.mfcr.cz/c3lod/cedr/resource/Dotace/. Při zadávání je potřeba i vybrat, ke kterému dříve definovanému prefixu se právě zadávaný prefix vztahuje anebo vybrat, že se vztahuje k hlavní entitě.

- Počet, který udává maximální počet hlavních entit, které budou soubory obsahovat.

3.2.2 Popis funkčnosti

Aplikace funguje tak, že nejprve stáhne ze zadaného SPARQL endpointu soubor s identifikátory hlavních entit. Ve staženém souboru aplikace projde všechny záznamy a pro každý identifikátor hlavní entity stáhne detail této entity. Tyto detaily se stahují po několika identifikátorech najednou (chunk). Po dokončení tohoto stahování se stažené soubory sloučí do jednoho, který se následně prochází a hledají se v něm zadané prefixy, které se vážou na hlavní entity. Celý identifikátor si uložíme do souboru identifikátorů (např. soubor s identifikátory dotací pro IS CEDR III) a tento soubor procházíme stejně jako při procházení identifikátorů hlavních entit a stahujeme jejich detaily. Stažené detaily sloučíme do jednoho souboru, kde opět hledáme zadané prefixy, u kterých chce uživatel stáhnout jejich detail. Pozorný čtenář si všiml, že algoritmus extrakce dat právě provedl druhé opakování, a tak jej nyní sepišme.

Algoritmus

1. Extrakce identifikátorů entit.[2]
2. Stažení detailu entit po částech (chunks).
3. Sloučení detailů do jednoho souboru.
4. Pokud jsme doposud nestáhli všechny prefixy, opakujeme algoritmus od kroku 1.

Na závěr vrátíme uživateli všechny sloučené soubory, které jsme vytvořili.

Poznámka: Vzhledem k již zmiňovanému problematickému stahování dat ze SPARQL endpointu IS CEDR III se aktuálně stahují výsledky dotazů ve formátu CSV, protože stahování v RDF formátech nefungovalo a z ostatních formátů je CSV nejpřívětivější, protože se data snadno parsují. Má to ale i tu nevýhodu, že následně musíme CSV výstup převést do nějakého RDF formátu. Zvolen byl formát N-Triples a tato konzolová aplikace obsahuje třídu *CsvToNtriplesConverter*, která převádí obsah z CSV do N-Triples. Předpoklady pro správné fungování

[2] V prvním kroku algoritmu data extrahujeme ze souboru staženého ze SPARQL endpointu, poté již pouze z našich souborů.

jsou ty, že URI subjektů a predikátů neobsahují čárky, což je pro IS CEDR III dostačující a funkční, alespoň v jeho zkoumané podmnožině dat potřebné pro demonstraci funkčnosti aplikace. V moment, kdy by s tímto nastaly problémy a nástroj by měli používat i další uživatelé, je potřeba tuto část implementace upravit.

3.2.3 Závěr

Pokud by tento nástroj fungoval jako jedna ze služeb naší webové aplikace, by byl vysoce užitečný pro uživatele, kteří neovládají jazyk SPARQL nebo neumí programovat či nechtějí psaním podobného programu ztrácet drahocenný čas.

3.3 Architektura webové aplikace

3.3.1 Úvod

S ohledem na nefunkční požadavky a požadavku na vývoj na platformě .NET jsme pro implementaci zvolili webový framework **ASP.NET MVC**. Jak již název frameworku napovídá, webová aplikace používá architekturální vzor Model-View-Controller (MVC). Výhody použití vzoru MVC jsou dobře známy. Jednak je zde snaha předejití nepřehlednému *(tzv. špagetovému)* kódu a možnost vývoje všech částí architektury MVC nezávisle na sobě.

Webová aplikace samotná je tedy rozdělena 3 typy komponent, mezi něž patří:

- Kontrolery (Controllers).
- Modely (Models).
- Pohledy (Views).

Kontrolery se starají o zpracovávání HTTP požadavků a odesílání HTTP odpovědí na ně. Často server odpovídá tak, že v HTTP odpovědi vrátí pohled, který obsahuje CSHTML kód, ze kterého vzejde následný HTML kód odeslaný uživateli. Modely obsahují jak datové třídy, tak třídy s pomocnými metodami kontrolerů.

3.3.2 Kontrolery

Aplikace obsahuje 5 kontrolerů. V následujících podkapitolách si je po jednom rozebereme.

Jak napovídá diagram na obrázku 2.1, uživatel v naší aplikaci může provést tři hlavní úkony, kterými jsou:

1. nahrání dat, o které se stará *DataController*
2. provedení výběru vizualizace za pomoci automatické detekce, zpracovává jej *DetectorController*
3. provedení vizualizace, které zpracovává *MapController*

HomeController

Obsahuje pouze jednu akci s názvem About, kde se nachází velice stručný popis aplikace. Tento kontroler jinou funkci neplní. Ostatní kontrolery již nějakou funkčnost mají a jsou zajímavějšími.

BaseController

Je abstraktním kontrolerem, ze kterého 3 zbylé kontrolery dědí. Nachází se v něm privátní vlastnost *UserData*, která implementuje rozhraní *IUserDataManager* a funguje jako správce přístupu k datům uloženým v session. Více o tomto rozhraní bude rozepsáno v kapitole 3.3.4. **BaseController** funguje jako fasáda (návrhový vzor GoF Facade) a zjednodušuje a zprostředkovává ostatním – dědícím – kontrolerům přístup k objektům v session. Zároveň obsahuje užitečné protected metody pro validaci a výpis chybových odpovědí.

DataController, jeho modely a pohledy

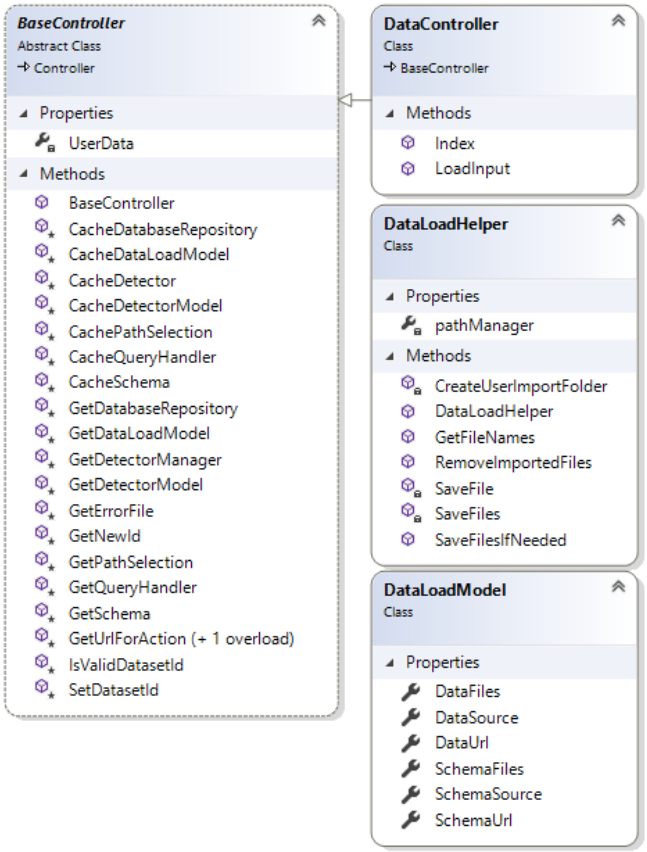
Obstarává validaci nahrávaných vstupních zdrojů dat a jejich případné nahrání do uživatelovy datové sady na server. Obsahuje 2 akce, *Index*, kde se nachází formulář, kterým uživatel vyplňuje model *DataLoadModel* a *ValidateInput*, který má za úkol tento model zvalidovat. K tomuto kontroleru se pojí i třída *DataLoadHelper*, kde se nachází různé pomocné metody pro ukládání souborů aj. Podrobnější popis procesu nahrávání souborů se nachází v kapitole 3.6. K nahlédnutí je i diagram tříd 3.1, ze kterého je možno vidět

DetectorController a jeho modely a pohledy

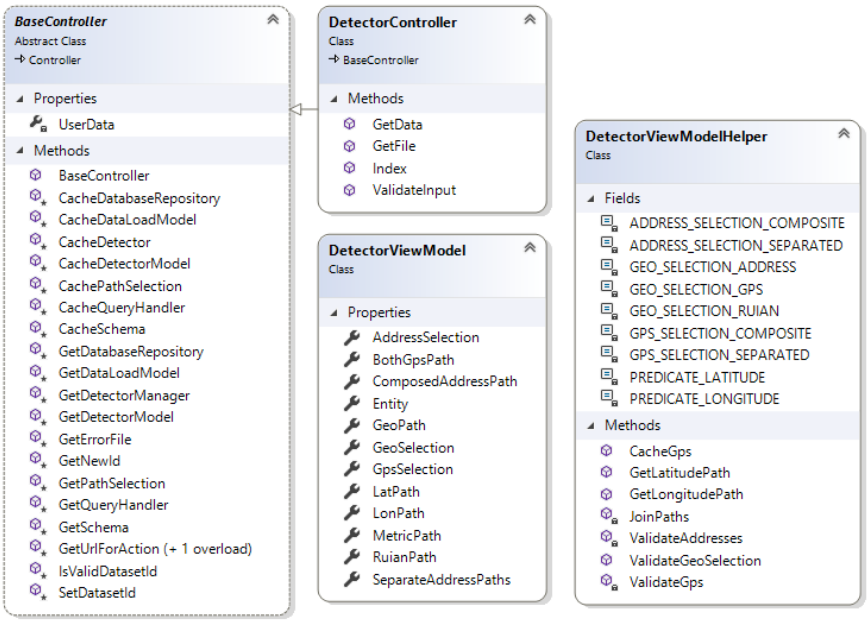
Úlohou tohoto kontroleru je vizualizace nahraného schématu, validace výběru vizualizace a jeho uložení do příslušné datové sady a případně i uložení GPS souřadnic do databáze. K tomu využívá třídy *DetectorViewModelHelper*, který tyto pomocné metody, které se dále provolávají do jádra, obsahuje. Obsahuje i metody pro validaci modelu *DetectorViewModel*, který je datovou přepravkou a uživatel jej vyplňuje na akci kontroleru *Index*. *DetectorController* obsahuje i akce ValidateInput, která pověřuje *DetectorViewModelHelper* k právě zmiňované validaci modelu, GetData, která vrací ve formátu JSON data schématu, které má být vizualizováno a akci GetFile, která umožňuje uživateli si stáhnout soubor s chybovými hláškami, pokud se nepodařilo získat GPS souřadnice pro některé entity. Na obrázku 3.2 je diagram tříd spojených s *DetectorController*em.

MapController, jeho modely a pohledy

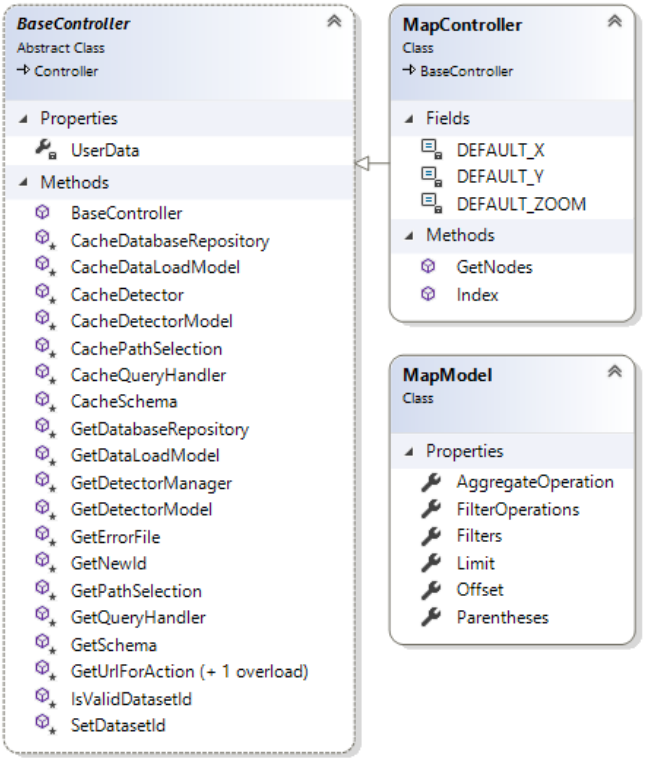
Poslední kontroler se zaobírá validací nastavení vizualizace a o samotnou selekci dat z datové sady. Obsahuje akci Index, kde uživatel vyplňuje třídu *MapModel*, která slouží jako přepravka dat. Akce GetNodes validuje tento model a pokud je validní, vrátí ve formátu JSON výsledek dotazu, který uživatel naklikal.



Obrázek 3.1: DataController – diagram tříd



Obrázek 3.2: DetectorController – diagram tříd



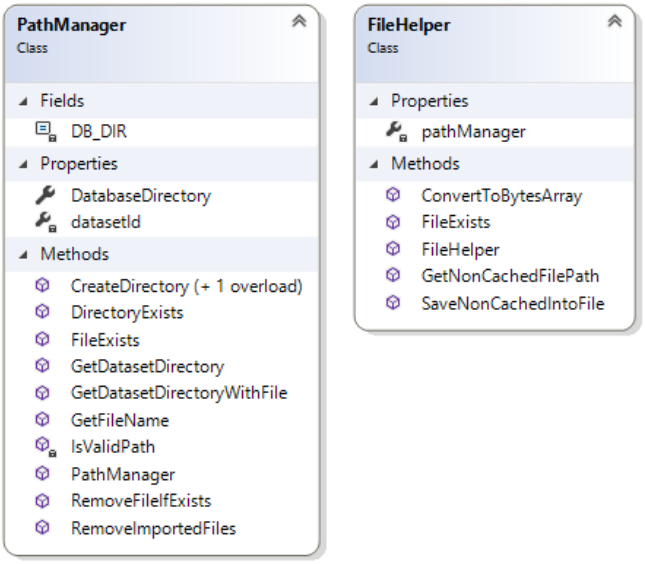
Obrázek 3.3: MapController – diagram tříd

3.3.3 Další modely

Posledními modely webové aplikace jsou třídy *FileHelper* a *PathManager*. Jelikož na serveru vznikají menší problémy s nutností přemapovávání cest, je tato zodpovědnost přenesena na právě jednu třídu a tou je *PathManager*. Jedna instance této třídy se stará o ukládání, mazání a čtení dat ze složky příslušné jedné datové sadě. Třída *FileHelper* obsahuje metody jako je vytvoření CSV souboru s chybovými hláškami entit, pro které se při geokódování nepodařilo GPS souřadnice získat. V současné době tuto třídu používá pouze *DetectorController* prostřednictvím jeho pomocné třídy. Na obrázku 3.4 je k vidění diagram těchto modelových tříd.

3.3.4 Zachování a obnovení stavu aplikace

Pro zachování stavu využíváme hlavně perzistentní úložiště, ale modely *DataLoadModel*, *DetectorViewModel* a *MapModel* si ukládáme pouze do session, jelikož nemá smysl tyto informace ukládat perzistentně a v session jsou uchovány pouze pro případ, kdy by některý z těchto modelů neprošel serverovou validací. Pak je uživatel přesměrován vždy na stejnou stránku a tyto modely jsou využity pro automatické předvyplnění dat do formuláře na stránce. Do session přistupujeme vždy přes rozhraní *IUserDataManager*. Session také funguje jako cache různých objektů aplikace, které jsou potřebné pro přístup do databáze, pro automatickou detekci typů a různé výsledky dotazů provedených do databáze, jako například schéma nebo výběr vizualizace. Toto rozhraní implementuje třída *SessionDataManager*, která v každé metodě očekává předání argumentů,



Obrázek 3.4: Další modely – diagram tříd

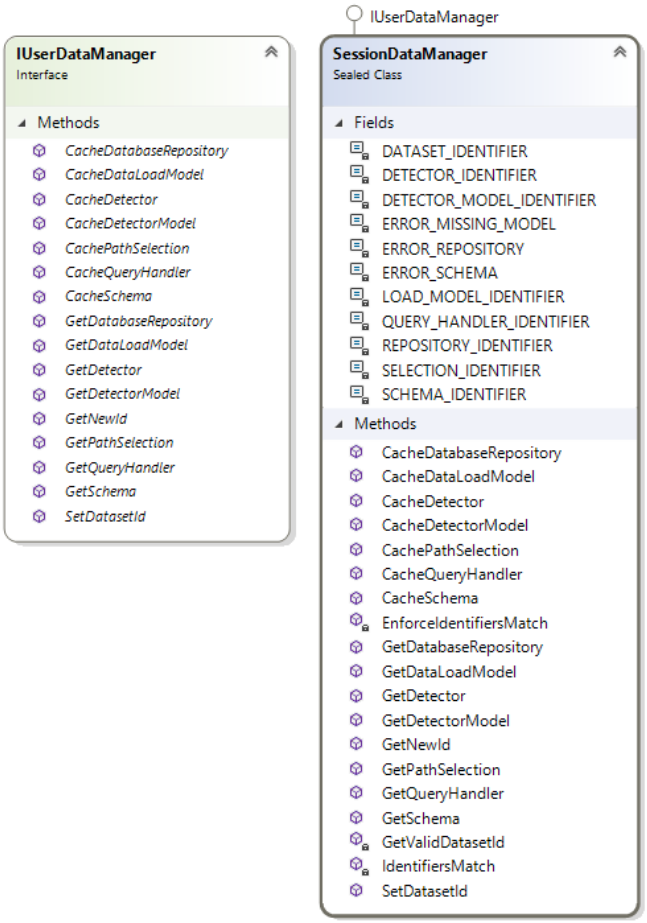
kterými jsou identifikátor datové sady (datasetId) a samotná session (objekt webového frameworku), ze které má být výběr proveden, resp. kam mají být objekty uloženy do cache. *SessionDataManager* obsahuje interní konstanty indexů, kam jednotlivé objekty do session ukládá a uživatel této třídy se tak nikdy nemůže spálit a získat nevalidní objekt, jelikož nemusí vůbec znát – a nezná – indexy, pod kterými objekty uchováváme. Pokud zadaný objekt v session není, *SessionDataManager* jej umí vždy znovuvytvořit a do session jej uložit a vrátit uživateli. Zároveň data v session udržuje vždy konzistentní. Před každou operací uložení do cache či přečtení objektu z cache se ujistí, že jsou shodné identifikátory datových sad. V session si totiž také uchováváme identifikátor datové sady, k němuž se všechny uložené objekty v session vztahují. Druhý identifikátor datové sady dostane *SessionDataManager* jako předaný argument funkce. Pokud se argumenty liší, do session cache bude uložen nový identifikátor a session bude vyprázdněna. Tímto postupem vždy dosáhneme toho, že v dané session cache budou uloženy vždy pouze spolu související objekty a tím pádem jsou data uloženy v cache stále konzistentní.

Na obrázku 3.5 je vidět diagram rozhraní *IUserDataManager* a třídy *SessionDataManager*.

3.3.5 Front End

Použitými technologiemi na Front Endu aplikace jsou JavaScript / ECMAScript 9 (2018) a knihovna JQuery ve verzi 3.4.1. Pro provedení specifických úkonů se používají další JavaScriptové knihovny. U provádění výběru vizualizace využíváme služeb knihovny D3.js pro vizualizaci schématu a manipulaci s ním. Pro vizualizaci dat do heat mapy využíváme knihoven Leaflet.js a Leaflet.heat, čímž získáváme interaktivní mapu a možnost vykreslení heat mapy. JavaScriptem generujeme i CSV výstup vizualizace, pokud o něj bude mít uživatel zájem.

Během vývoje jsme se soustředili také na to, aby aplikace byla responzivní. K



Obrázek 3.5: Session – diagram tříd

tomu slouží knihovna Bootstrap ve verzi 4.3.1 a trochu našeho JavaScriptu.

3.3.6 Back End

Back endem je jádro aplikace v přidružené knihovně. Samotná webová aplikace se pouze provolává do tohoto jádra, které obsahuje veškerou logiku. Webová aplikace samotná slouží pouze jako prezentační vrstva jádra. Veškeré detaily se nachází v kapitole 3.4. Na obrázku 3.6 jsou k vidění vztahy závislostí jednotlivých komponent celé aplikace v našem řešení (solution). *HeatmapVisualizer* je název webové aplikace, *HeatmapVisualizerModels* značí jádro aplikace, *SparqlDataAnalyzer* značí knihovnu pro extrakci numerických hodnot z výsledku SPARQL dotazů a *CedrDataExtractor* je onou konzolovou aplikací, která umí extrahovat data z IS CEDR III.

3.4 Jádro aplikace

3.4.1 Úložiště

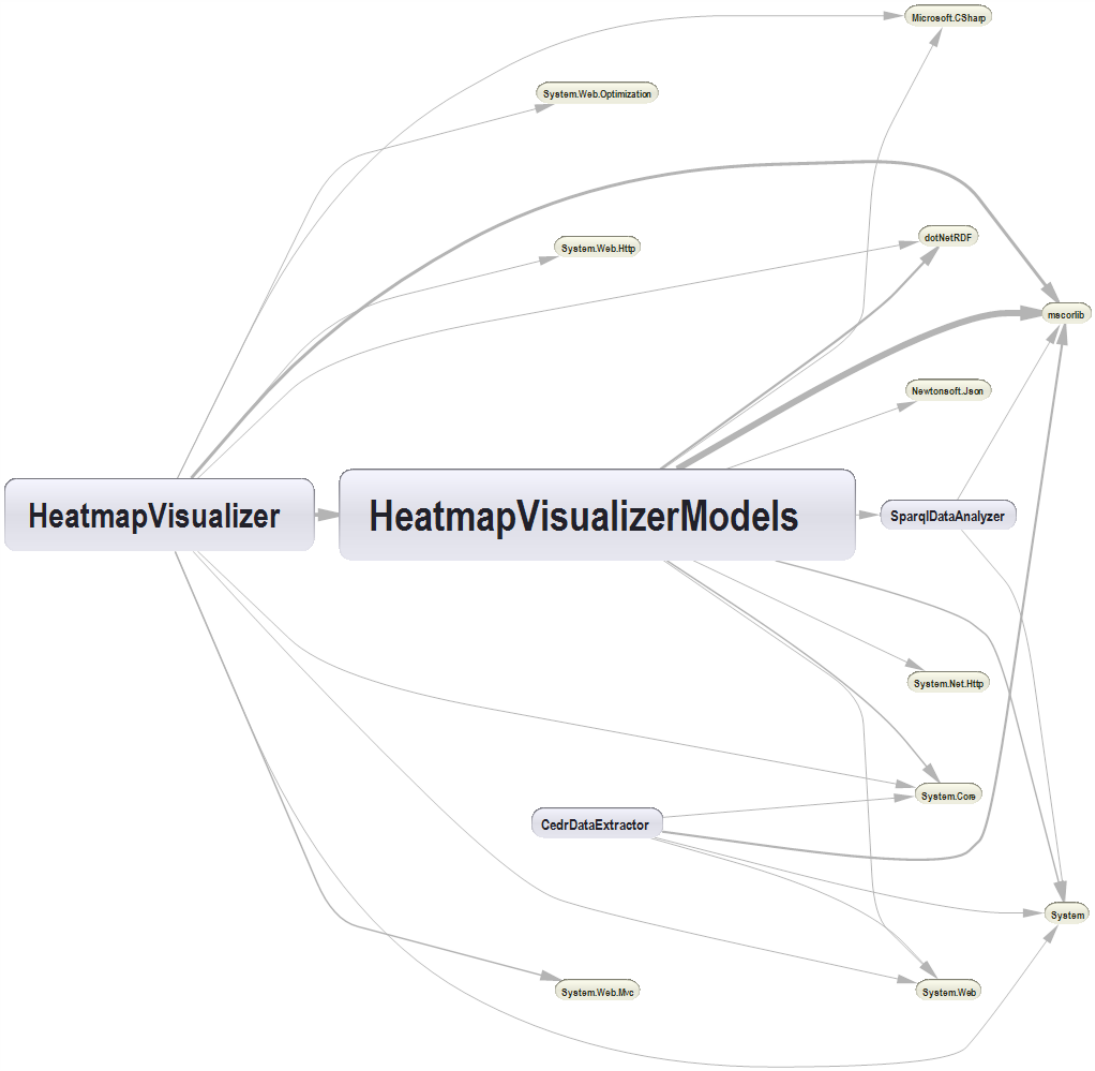
Jako úložiště dat jádro naší aplikace používá Apache Jena Fuseki, jak bylo v nefunkčních požadavcích požadováno. Apache Jena Fuseki je SPARQL server, k jehož datům se lze dostávat přes SPARQL Over HTTP protokol (SOH) a Fuseki server protocol, který je na Apache Jena Fuseki a nápadně připomíná REST API,

Na Fuseki serveru se vytvářejí datové sady, do kterých se nahrávají data. Do datových sad se ukládají grafy, které mají nějaké URL a seskupují tak související data k sobě. Pokud graf nemá žádné URL, je nahrán do základního grafu datové sady.

S protokolem SOH nemusíme my sami ručně pracovat. Práci nám usnadní knihovna dotNetRDF, která byla uvedena v kapitole 3.1. Prostřednictvím třídy FusekiConnector nám nabízí API pro manipulaci s jednou datovou sadou uloženou na Fuseki serveru.

Omezení úložiště

Během testování aplikace jsme narazili na omezení, kdy prostřednictvím knihovny dotNetRDF není možné na server nahrát data, ve kterých se vyskytují URI identifikátory s diakritikou, respektive – pravděpodobně lépe řečeno – URI se speciálními znaky. Důvodem je to, že knihovna interně pro komunikaci využívá formát RDF/XML, do něhož tyto speciální znaky neumí knihovna zaserializovat. Správci knihovny tvrdí, že toto chování je správné, jelikož není dle specifikace tohoto formátu možné z takovéto URI získat QName potřebné pro deklaraci jmenného prostoru. Problém byl řešen na adrese https://github.com/dotnetrdf/dotnetrdf/issues/255. Vývojáři knihovny radí, že problém lze vyřešit snadno a místo formátu RDF/XML můžeme využít pro komunikaci s Fuseki serverem formát Turtle či N-Triples, pro které nejsou URI se speciálními znaky problematické. Problémem však je, že my, jakožto uživatelé knihovny, tuto změnu provést



Obrázek 3.6: Graf závislosti komponent

nemůžeme, na což byli autoři knihovny upozorněni a na nápravě aktuálně pracují, jak lze dohledat na https://github.com/dotnetrdf/dotnetrdf/issues/258.

Data ukládaná do databáze

Některé uživatelem zadané data a rozhodnutí si potřebujeme perzistentně uchovat v databázovém úložišti. Pro každou datovou sadu si vygenerujeme unikátní identifikátor vytvoříme ji pod tímto identifikátorem na našem Fuseki serveru. Do této datové sady si ukládáme v následujících odstavcích zmíněné informace. Žádné údaje uloženy v datové sadě není možné měnit a jsou určena pouze pro čtení. Nikdy je nepřepisujeme, abychom měli data vždy konzistentní. O tom si povíme něco po představení všech ukládaných komponent do databáze v podkapitole 3.4.1.

Schéma dat Nahrávané schéma ukládáme vždy do základního grafu úložiště a využíváme jej při detekci vlastností s lokací a metrickými hodnotami.

Data

Vstup ze souborů Pokud uživatel nahrává data ze souborů, uložíme jejich obsah do příslušné datové sady. Graf obsahující importované data ze souborů má hodnotu http://fake-url.com/data/.
Pokud je obsah souborů celkově v desítkách MB, je doporučeno využít nahrávání prostřednictvím URL SPARQL endpointu. Správné nahrání takto velkých souborů není zaručeno. Obsah souborů je nahráván opět prostřednictvím knihovny dotNetRDF a ta při nahrávání obsahuje časový limit, jak dlouho bude čekat na odpověď serveru, zda se vstupní soubory podařilo nahrát. Při jeho překročení knihovna vyhazuje výjimku.

SPARQL Endpoint Pokud je ale datovým zdrojem zadaná URL externího SPARQL endpointu, nebudeme k sobě stahovat všechna data z tohoto endpointu, protože by jednoduše velikost dat mohla být příliš velká (např. IS CEDR III) a navíc je zbytečné tyto data k sobě ukládat na server, když jsou již na jiném uložena. Nám jde hlavně o schopnost se dotazovat do těchto dat, což nám externí SPARQL endpoint poskytuje.
Místo toho do databáze uložíme tuto externí URL SPARQL endpointu, kam budeme dotazy posílat. Tato URL bude uložena v grafu s URL http://fake-url.com/info/, který bude obsahovat právě jednu trojici, jejíž subjekt ani predikát fakticky důležité nejsou – ale pro formu uveďme, že subjektem je URL našeho Fuseki serveru a predikátem je http://rdfs.org/ns/void#sparqlEndpoint. Objektem trojice je již jediný relevantní údaj, a sice zadané URL externího SPARQL endpointu.

GPS data Do datové sady v případě potřeby (pokud nejsou údaje obsaženy v samotných nahraných datech) ukládáme i data přiřazující GPS souřadnice jednotlivým entitám. Tento graf má URL http://fake-url.com/gps/ a trojice

v něm obsažené jsou seznamem ve formátu[3]:
(<ent_1>, http://www.w3.org/2003/01/geo/wgs84_pos#lon, <lon_1>)
(<ent_1>, http://www.w3.org/2003/01/geo/wgs84_pos#lat, <lat_1>)
...
(<ent_N>, http://www.w3.org/2003/01/geo/wgs84_pos#lon, <lon_N>)
(<ent_N>, http://www.w3.org/2003/01/geo/wgs84_pos#lat, <lat_N>)

Trojice mající http://www.w3.org/2003/01/geo/wgs84_pos#lon jako predikát popisuje zeměpisnou délku dané entity. Opačně, trojice mající predikát http://www.w3.org/2003/01/geo/wgs84_pos#lat popisuje zeměpisnou šířku entity.

Výběr vizualizace Jako poslední k sobě do databáze ukládáme tzv. výběr vizualizace. Výběr obsahuje všechny údaje, které jsou potřebné pro správné provedení samotné vizualizace. Tyto informace si ukládáme proto, aby se uživatel mohl kdykoliv k jeho nahrané dříve vizualizované datové sadě vrátit. Uchováváme si čtveřici (<typ_hlavní_entity>, <cesta_k_lon>, <cesta_k_lat>, <cesta_k_metrice>), kde *lon* značí zeměpisnou délku a *lat* zeměpisnou šířku.

Výběr vizualizace pro IS CEDR III Pro demonstrativní příklad je výběr[4] následující:

- **Typ hlavní entity:** *cedr:PravnickaOsoba*
- **Cesta k metrice:** *cedr:obdrzelDotaci/cedr:byloRozhodnuto/cedr:castkaRozhodnuta*
- **Cesta k zeměpisné délce:** http://www.w3.org/2003/01/geo/wgs84_pos#lon
- **Cesta k zeměpisné šířce:** http://www.w3.org/2003/01/geo/wgs84_pos#lat

Odpověď na otázku, proč jsme zvolili tento formát, je snadná, tento zápis přesně odpovídá cestám v jazyce SPARQL, a tak budeme moci tuto hodnotu přímo dosadit do šablony našeho dotazu bez nutnosti se o cokoliv starat.

Cesty k GPS souřadnicím Cesty k GPS souřadnicím v IS CEDR III nejsou, a tak budou muset být GPS souřadnice staženy. Pokud nastane kdykoliv tento scénář, cesty k zeměpisné šířce a délce budou nastaveny právě takto a neplatí to pouze pro IS CEDR III.

Poznámka: Celé cesty si musíme ukládat proto, že cest v grafu mezi dvěma vrcholy může existovat mnoho a my potřebujeme tuto cestu mít jednoznačně danou.

[3] Zápis trojic je symbolický.

[4] Reálně uložená data jsou plná jména a neobsahují prefixy, tj. za „cedr“ je dosazeno „http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#“.

Zachování konzistence datové sady

Proč si nikdy data v datové sadě nepřepisujeme? Aby se nikdy nemohlo stát to, že bychom si k sobě například uložili GPS data vyextrahovaná z nějakého výběru vizualizace, který by uživatel následně kompletně změnil. Vznikla by nekonzistence v datovém zdroji, jelikož by se data uchovaná v GPS grafu nevztahovala k uloženému výběru. Tato nekonzistence by samozřejmě šla snadno vyřešit tím, že bychom odstranili všechna data, která by se staly zastaralými. Ale s ohledem na to, že získávání GPS souřadnic je drahá operace a musíme se na ně dotazovat externích služeb, nechceme uživateli snadno povolit vyčerpávání denních limitů poskytovatelů těchto služeb.

Adresy URL ukládaných grafů

Čtveřice URL grafů dat samotných, GPS dat, externího SPARQL endpointu a výběru vizualizace jsou konstanty ve statické třídě *GraphUri*. Tyto hodnoty jdou snadno změnit, pokud by byly nevyhovující. Není v nich definována URL grafu schématu, jelikož jej nahráváme vždy do základního grafu. Pokud by nevyhovovalo toto nastavení, také je možné jej opravit, ale už ne tak snadno.

3.4.2 Architektura – úvod

Jádro aplikace je rozděleno do několika komponent. Jedná se o Data, Database, Detector, Location a Map. Každá tato komponenta má vlastní jmenný prostor a třídy nebo rozhraní jsou rozděleny do složek spadajících do těchto komponent.

Následuje strukturovaný popis všech tříd. Struktura je dána tím, jak jsou v jádře členěny jmenné prostory. Každá nová podúroveň jmenného prostoru určuje zanoření do podsložky. Například pro namespace *HeatmapVisualizerModels.Data.Loading* odpovídá cesta *Data/Loading*. Někde je zde pro lepší přehlednost vynechána poslední úroveň zanoření do složek a její obsah je přenesen do úrovně o jedna vyšší.

Před tím než však si rozebereme jednotlivé komponenty, se podíváme na to, jak se v naší aplikaci řeší předávání výsledků volaných operací.

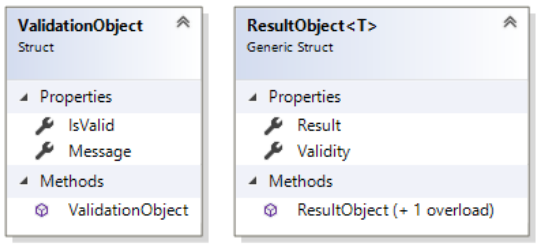
3.4.3 Validace a předávání výsledků a chybových hlášení

Pro validaci a předávání výsledků a chyb jsme navrhli následující dvě struktury.

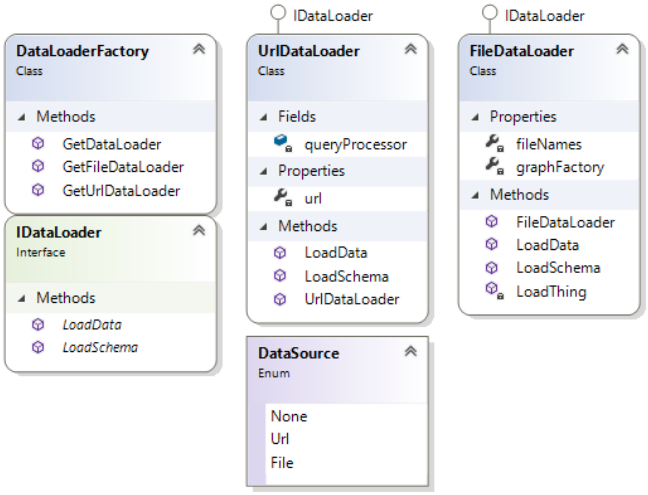
Na obrázku 3.7 je k nahlédnutí diagram tříd příslušných do této sekce.

Struktura ValidationObject Tato struktura se skládá z dvojice (validita, zpráva) standardních typů (bool, string). Používá se u všech validací dat. Usnadňuje předávání chybových zpráv uživateli a tento mechanismus by být rychlejší než vyhazování výjimek. Nevýhodou je snížená čitelnost kódu.

Struktura ResultObject<T> Slouží jako přepravka objektů různých operací. Skládá se z generického typu T, který reprezentuje instanci výsledku operace. Dále obsahuje v předešlém odstavci zmíněnou strukturu *ValidationObject*,



Obrázek 3.7: Validace a výsledky – diagram tříd



Obrázek 3.8: Data Loading – diagram tříd

která v sobě nese informaci, zda je možné bezpečně přistoupit k výsledku operace (tj. byla-li operace provedena v pořádku) či nikoliv. Tato struktura je využívána extenzivně v celém jádru aplikace. **Výhody** jsou následující: usnadňuje manipulaci se všemi výsledky operací v aplikaci, usnadňuje předávání chyb a – jak již bylo zmíněno v předchozím odstavci – předávání této struktury by mělo být rychlejší než řešení hlášení chyb pomocí vyhazování výjimek. **Nevýhodu** snížení čitelnosti kódu sem tranzitivně přenesla struktura *ValidationObject*.

3.4.4 Data

Výčet DataSource Výčet definuje níže vypsané typy datových zdrojů.

- Žádný.
- Soubory.
- SPARQL endpoint URL.

Loading

Zaobírá se nahráváním dat do repozitáře datové sady. Na obrázku 3.8 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IDataLoader Rozhraní obsahuje metody pro nahrání dat a samotného schématu do předaného repozitáře implementující rozhraní *IDatabaseRepository*

Třída DataLoaderFactory Slouží ke konstruování konkrétní instance implementující rozhraní *IDataLoader*, což jsou v tuto chvíli buď třída *FileDataLoader*, nebo třída *UrlDataLoader*. Využívá návrhový vzor GoF Abstract Factory.

Třída FileDataLoader Na základě kolekce názvu souborů z nich nahrává data do repozitáře. Dle volání uživatele ukládá buď do grafu příslušnému pro data (http://fake-url.com/data/) nebo do základního grafu repozitáře v případě ukládání schématu.

Třída UrlDataLoader Z předané URL nenahrává žádné data do vlastního repozitáře, pouze do něj nahraje informaci, ze které externí URL se mají data čerpat a kam se bude repozitář dotazovat. Pokud dostane URL SPARQL endpointu, kde se nachází schéma, pošle na něj CONSTRUCT dotaz a výsledek si uloží právě do repozitáře.

Validation

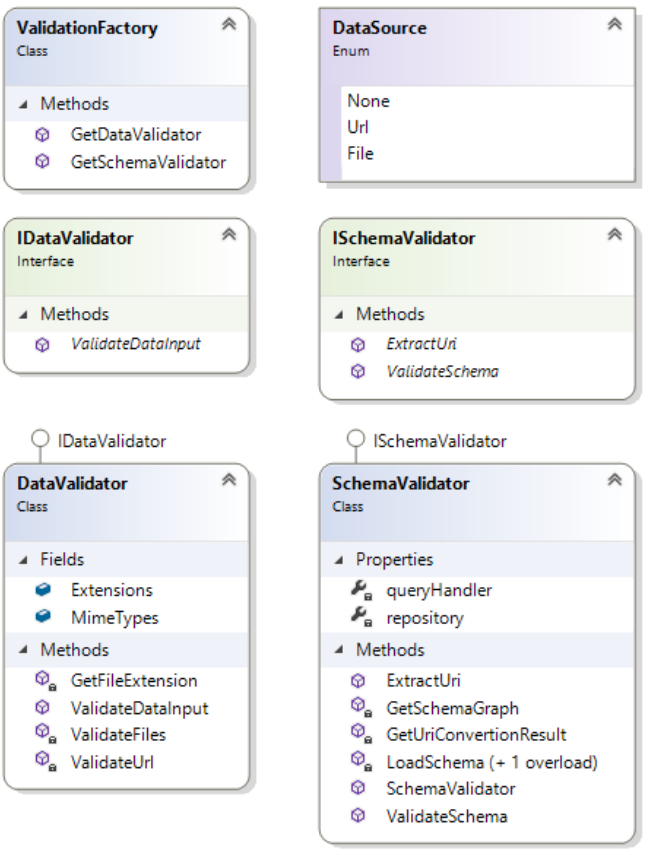
Zaobírá se validací nahrávaných dat. Na obrázku 3.9 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IDataValidator Definuje rozhraní sloužící pro validaci datového zdroje.

Rozhraní ISchemaValidator Definuje rozhraní sloužící pro validaci schématu. Obsahuje jak metodu pro samotnou validaci schématu z předané URI, tak i metodu, která tuto URI z datového zdroje dokáže vytvořit. Pokud je typem datového zdroje soubor, myšlenkou je vytvoření SPARQL endpointu na našem Fuseki serveru, načež vrátíme tuto novou URL. Pokud je typem datového zdroje URL, stačí tuto URL vrátit. Tímto mechanismem získáme z libovolného datového zdroje URL adresu SPARQL endpointu, která bude využita pro následnou validaci schématu.

Třída ValidationFactory Vytváří konkrétní instance sloužící pro validaci dat, resp. schématu implementující rozhraní *IDataValidator*, resp. *ISchemaValidator*. Využívá návrhový vzor GoF Abstract Factory.

Třída DataValidator Implementuje rozhraní *IDataValidator*. Validuje vstupní datový zdroj ze seznamu souborů nebo jedné URL. U souborů se kontroluje nenulová délka seznamu, nenulové velikosti souborů, správné přípony a podobně. Validní URL je ta URL, která je správně formovanou absolutní URL.



Obrázek 3.9: Data Validation – diagram tříd

Třída SchemaValidator Implementuje rozhraní *ISchemaValidator*. Validuje schéma pomocí CONSTRUCT dotazu, kterým zjistíme, zda nahrané schéma má správnou strukturu. Spouštěný CONSTRUCT dotaz je podrobněji rozebrán v kapitole 3.7.

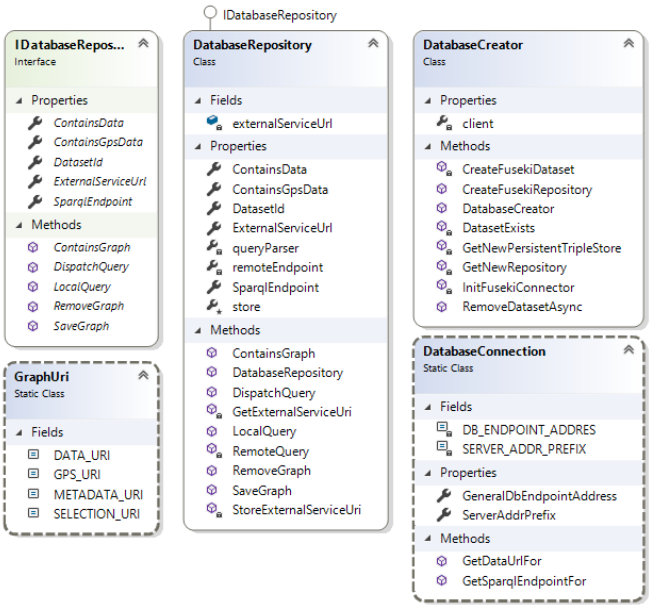
3.4.5 Database

Třída MyGraphFactory Slouží k vytváření instancí jednotlivých grafů implementujících rozhraní *IGraph* (dotNetRDF), které budou později ukládány do repozitáře. Důležité je nastavování bázových URI těchto grafům, což právě tato třída zajišťuje přistupováním ke konstantám statické třídy *GraphUri*. Využívá návrhový vzor GoF Abstract Factory.

Repository

Na obrázku 3.10 je k nahlédnutí diagram tříd příslušných do této sekce.

Třída DatabaseConnection Tato statická třída obsahuje údaje pro možnost spojení s databázovým serverem. Obsahuje URL adresu serveru a důležité statické metody a konstanty potřebné pro vytváření datových sad na serveru a získávání jejich URL adres.



Obrázek 3.10: Database Repository – diagram tříd

Rozhraní IDatabaseRepository Reprezentuje repozitář datové sady s daty. Stejně tak jako umožňuje ukládat a mazat grafy, umožňuje provádět spouštění SPARQL dotazů.

Třída DatabaseCreator Umí vytvářet a mazat samotné datové sady na serveru Apache Jena Fuseki se zadaným ID a vrací instanci implementující interface *IDatabaseRepository*.

Třída DatabaseRepository Je tenkým wrapperem nad datovými strukturami externí dotNetRDF knihovny. Implementuje rozhraní *IDatabaseRepository*.

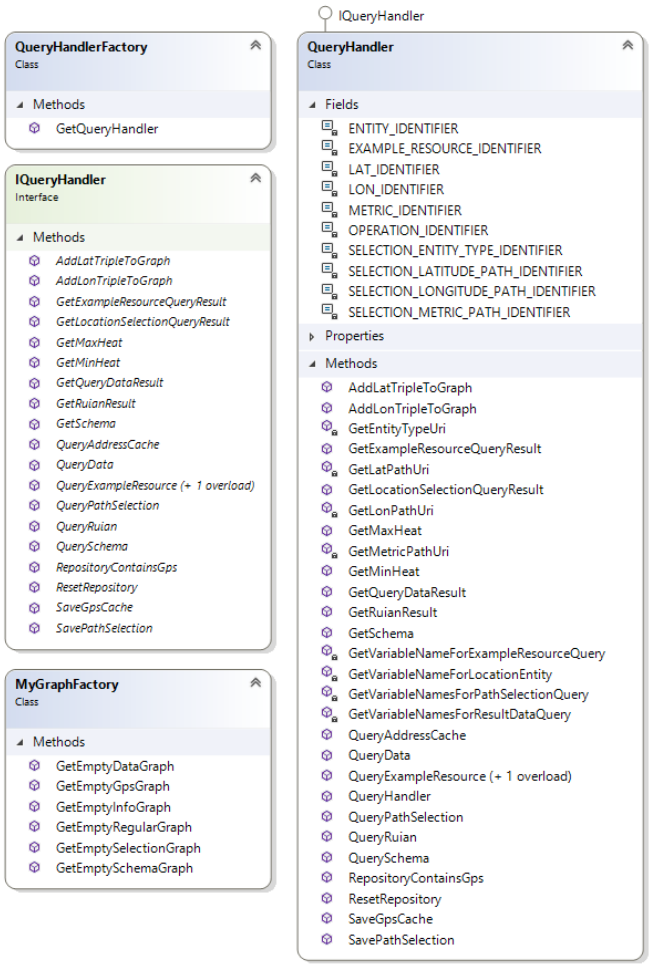
Třída GraphUri Tato statická třída obsahuje několik konstant URL grafů, pod kterými se budou ukládat do repozitáře.

Query

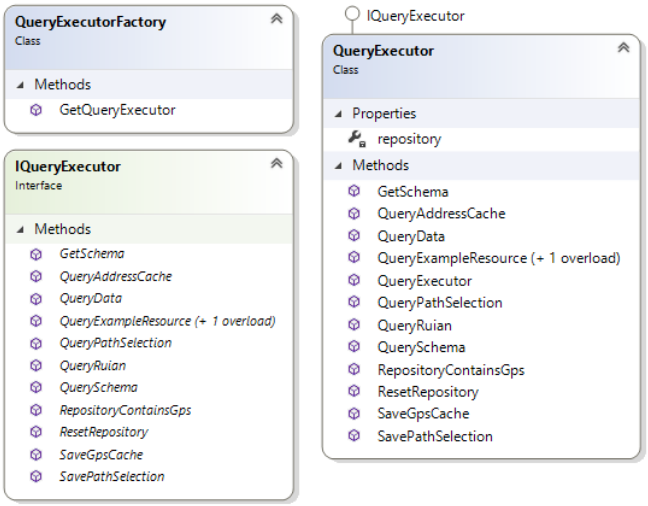
Handler Na obrázku 3.11 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IQueryHandler Slouží jako fasáda (GoF Facade). Sjednocuje rozhraní *IQueryExecutor* a *IQueryResultReader* a zároveň skrývá určité detaily před uživatelem, mezi které například patří předávání některých názvů proměnných při volání metod těchto rozhraní. Slouží jako hlavní vstupní bod pro uživatele této knihovny pro veškerou manipulaci s daty aplikace nad repozitářem.

Třída QueryHandlerFactory Slouží k vytváření konkrétních instancí implementujících rozhraní *IQueryHandler*. Využívá návrhový vzor GoF Abstract Factory.



Obrázek 3.11: Query Handler – diagram tříd



Obrázek 3.12: Query Executor – diagram tříd

Třída QueryHandler Implementuje rozhraní *IQueryHandler*, obsahuje několik konstant, které využívá pro názvy proměnných pro spouštění dotazů nad repozitářem nebo naopak čtení SPARQL výsledků těchto spuštěných dotazů.

Executor Na obrázku 3.12 je k nahlédnutí diagram tříd příslušných do této sekce.

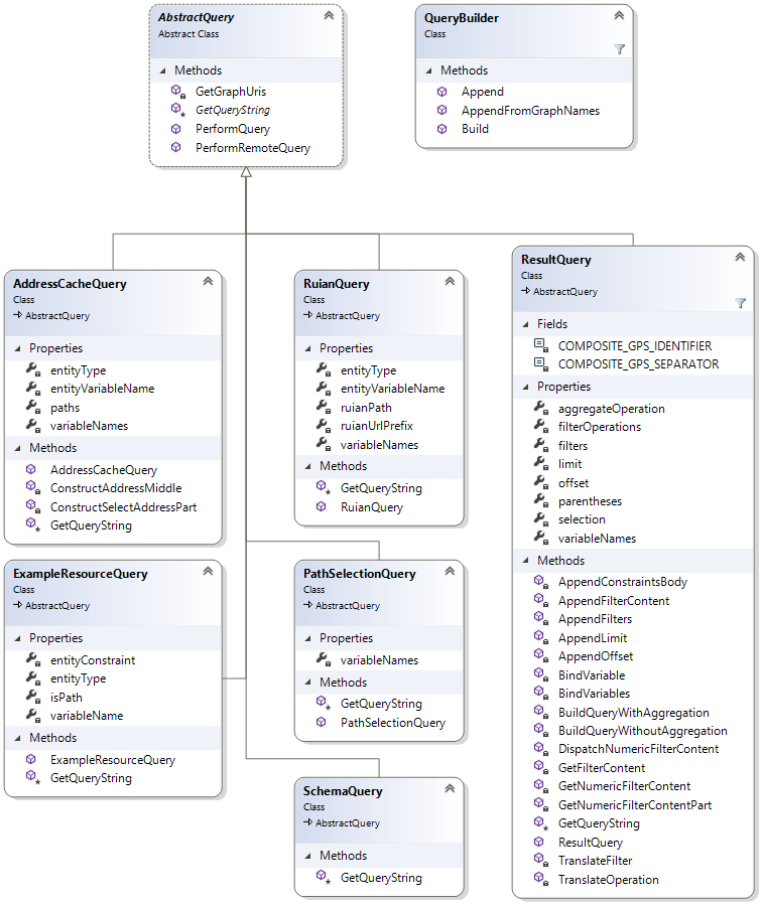
Rozhraní IQueryExecutor Toto rozhraní se stará o manipulaci s daty nad jedním konkrétním repozitářem, pod který spadá jedna datová sada. Obstarává jak dotazování se a selekci různých dat (schéma, data pro vizualizaci, adresní entity a spoustu dalších), tak ukládání dat (graf s GPS daty či výběr vizualizace).

Třída QueryExecutorFactory Slouží k vytváření konkrétních implementací implementujících rozhraní *IQueryExecutor*. Využívá návrhový vzor GoF Abstract Factory.

Třída QueryExecutor Implementuje rozhraní *IQueryExecutor*. Veškeré implementační detaily konkrétních dotazů jsou zapouzdřeny pod vytváření konkrétních instancí dědících z abstraktní třídy *AbstractQuery*.

Třída QueryBuilder Slouží pro vytváření (SPARQL) dotazů. Mohl by být robustnější, v současné podobě je pouze jednoduchým wrapperem nad třídou *StringBuilder*. Využívá návrhový vzor GoF Builder.

Třída AbstractQuery Tato abstraktní třída obsahuje obecnou logiku pro vytvoření a spuštění dotazu nad daným repozitářem. Konkrétní potomci této třídy implementují logiku sestavení SPARQL dotazu (query string). Všechny následující třídy v této podkapitole (Executor) jsou konkrétními potomky této třídy. Na obrázku 3.13 je k nahlédnutí diagram tříd příslušných do této sekce.



Obrázek 3.13: Abstract Query – diagram tříd

Třída AddressCacheQuery Obecná implementace vytvoření selekce entit s libovolným počtem adresních složek, které náleží této konkrétní entitě. Pokud je počet adresních složek roven jedné, tato adresní složka bude při selekci povinně vyžadována. V opačném případě budou dotazované adresní složky označeny slovem *OPTIONAL*. To může v krajních případech vést k tomu, že některé adresy mohou být prázdné. Na tuto implementaci lze do budoucna snadno napojit další různé formáty adres než ty, které jsou aktuálně podporovány. Nyní je podporována adresa jako jeden řetězec nebo adresa očekávající složky stát, město, ulice, číslo popisné/domovní, číslo orientační a PSČ.

Třída ExampleResourceQuery Jedná se o reprezentaci SPARQL dotazu s funkcí SAMPLE. Tento dotaz je využíván při detekci metrických vlastností obsažených ve schématu nebo pro validaci metrických hodnot (vybraná metrika a vybrané souřadnice).

Třída PathSelectionQuery Reprezentuje dotaz, jež je využíván ke znovuvytvořování instancí typu *PathSelection*, který reprezentuje výběr vizualizace.

Třída ResultQuery Slouží pro selekci dat, které mají být podkladem pro vizualizaci. Umožňuje jednoduché filtrování dat s uzávorkováním nebo definici SPARQL klauzulí jako OFFSET či LIMIT.

Třída RuianQuery Slouží pro získání GPS souřadnic vybraných entit z našeho datového zdroje na základě vlastnosti, která obsahuje RUIAN identifikátor nebo URL s tímto RUIAN identifikátorem (za posledním lomítkem v adrese). Využíváme k tomu SPARQL Federated query[5]. To znamená, že náš dotaz odesíláme na náš Fuseki server a další námi definované položky budou staženy z externího SPARQL endpointu s námi definovanou adresou. Ta je https://ruian.linked.opendata.cz/sparql.

Třída SchemaQuery Tato třída reprezentuje SPARQL CONSTRUCT dotaz. Slouží pro validaci schématu a výsledek dotazu jsou vstupní data předávaná některé z implementací rozhraní *IDetector* pro následnou vizualizaci schématu.

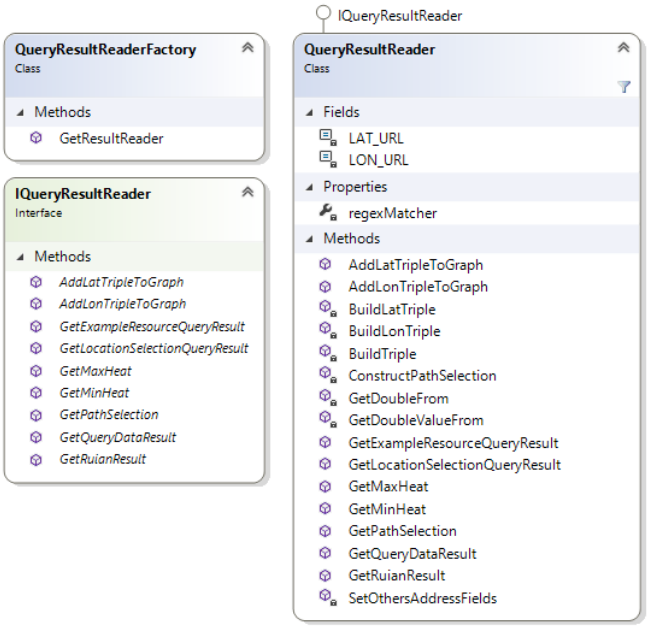
Reader Na obrázku 3.14 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IQueryResultReader Slouží pro čtení výsledků dotazů vrácených jednou z metod rozhraní *IQueryExecutor*.

Třída QueryResultReaderFactory Slouží k vytváření konkrétních instancí implementujících rozhraní *IQueryResultReader*. Využívá návrhový vzor GoF Abstract Factory.

Třída QueryResultReader Implementuje rozhraní *IQueryResultReader* a stará se o konkrétní vytváření očekávaných datových struktur využívaných pro další manipulaci později v aplikaci.

[5] https://www.w3.org/TR/sparql11-federated-query/



Obrázek 3.14: Query Reader – diagram tříd

3.4.6 Detector

Třída SchemaVisualizationModel Reprezentuje graf, který bude vstupními daty pro vizualizaci schématu. Obsahuje množiny vrcholů a hran. Obsahuje deklaraci vnořených struktur *Node* reprezentující vrchol grafu a *Line* reprezentující hranu grafu.

Třída PathSelection Reprezentuje **výběr vizualizace**. Jedná se o čtveřici řetězců (typ entity, cesta k metrice, cesta k zeměpisné délce, cesta k zeměpisné šířce). Je ukládána perzistentně do databázového úložiště a pro jednu datovou sadu lze definovat pouze jednou a nejde již nahradit.

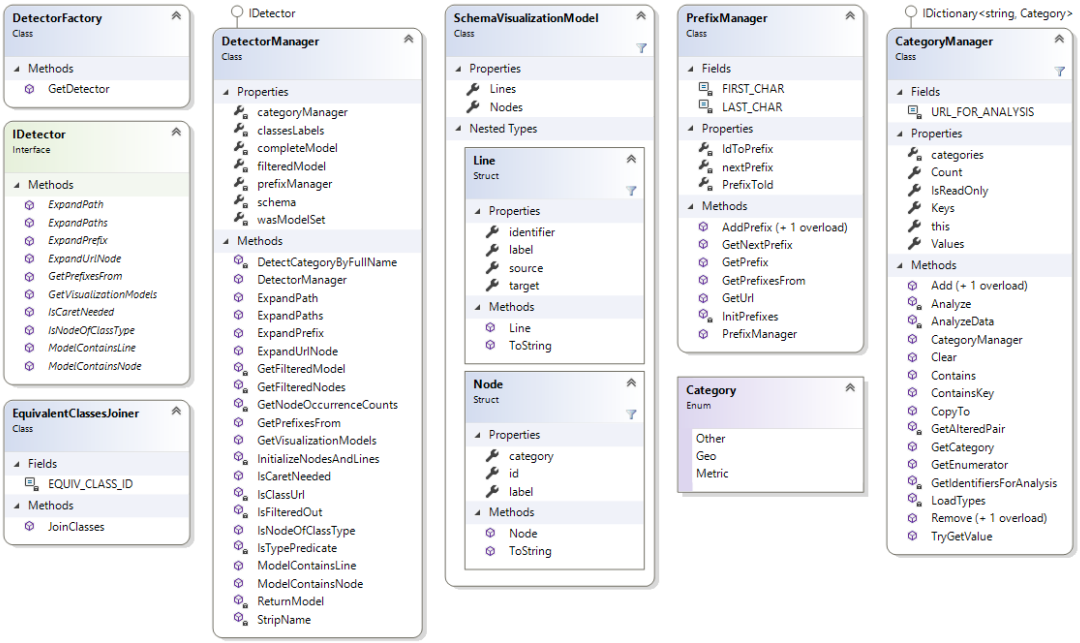
Manager

Zabývá se prací se schématem. Přípravou dat pro vizualizaci, prefixováním jmen a automatickou detekcí typů. Na obrázku 3.15 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IDetector Obsahuje metody potřebné pro vizualizaci schématu. Obstarává spojení ekvivalentních tříd, stará se o správu prefixů vizualizovaných prvků a umožňuje následně zpětně konvertovat zkrácené názvy na celé cesty (bez prefixů) a dostat platné cesty a identifikátory později umisťované do SPARQL dotazů.

Třída DetectorFactory Slouží k vytváření instancí implementujících rozhraní *IDetector*. Využívá návrhového vzoru GoF Abstract Factory.

Třída DetectorManager Implementuje rozhraní *IDetector*. Pomocí dalších tříd zajišťuje veškerou funkcionalitu požadovanou tímto rozhraním. Vybírá pod-



Obrázek 3.15: Detector Manager – diagram tříd

množinu modelu, která bude vizualizována uživateli. Většinu práce deleguje na následující 3 třídy. Také se stará o filtrování zobrazovaného grafu, jelikož graf schématu může být opravdu velký. Vrací proto pole velikosti 2, kde jedním prvkem je kompletní, nefiltrovaný model a druhým prvkem je filtrovaný model, který se má vykreslit uživateli. Jak je vybírán filtrovaný graf je popsáno v kapitole 3.7.

Třída EquivalentClassesJoiner Stará se o sloučení ekvivalentních tříd v modelu, pokud existují. Ekvivalentní třídy jsou takové, mezi kterými existuje RDF trojice mající predikát s URL `http://www.w3.org/2002/07/owl#equivalentClass`.

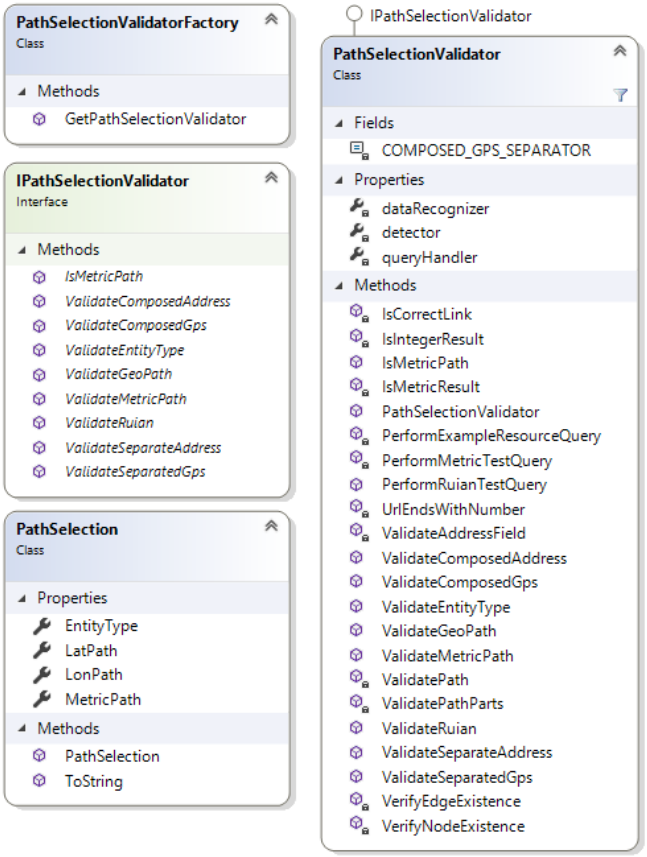
Třída CategoryManager Stará se o detekci známých typů. Detekuje metrické a lokační vlastnosti schématu i pomocí rozhraní *IQueryHandler*. Implementuje rozhraní *IDictionary<string, Category>*, kde *Category* je **výčet** typů: jiný, lokace, metrika. Při manipulaci s URI řetězci je vždy převádí na lower-case řetězce.

Třída PrefixManager Dělá všechnu práci související s přidělováním předpon. Obstarává zkracování plných identifikátorů na prefixovaný řetězec a naopak.

Validation

Zabývá se validací výběru vizualizace. Na obrázku 3.16 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IPathSelectionValidator Definuje rozhraní potřebné pro správné zvalidování výběru vizualizace.



Obrázek 3.16: Výběr vizualizace – diagram tříd

Třída PathSelectionValidatorFactory Tato třída slouží k vytváření instancí implementující rozhraní *IPathSelectionValidator*. Využívá návrhový vzor GoF Abstract Factory.

Třída PathSelectionValidator Implementuje rozhraní *IPathSelectionValidator* a zaobírá se samotným provedením validace výběru vizualizace. Kontroluje to, že všechny uživatelem odeslané vrcholy v grafu existují, pokud tvoří cestu, pak že ta cesta v grafu skutečně existuje a kontroluje také to, že vybraná metrická cesta je skutečně metrická – že se nachází na jejím konci v datech numerický RDF literál. Stejně tak kontroluje, že pokud uživatel vybral identifikátor lokace typ GPS, pak že jednotlivé jeho složky jsou metrické.

3.4.7 Location

Třída Address Obsahuje položky *Country*, *City*, *Street*, *StreetNr*, *StreetNr2* a *ZIP* reprezentující zemi, město, ulici, číslo popisné, číslo orientační a PSČ (v tomto pořadí). Implementuje standardní rozhraní *IEnumerable<string>* a obsahuje definici explicitních operátorů konverze z pole řetězců a na pole řetězců. Důležité je i přepsání metody *ToString()*. Ta vrací řetězec ve formátu „Country, ZIP složená_část“, kde *složená_část* vznikne složením *adresní_části* a *číselné_části*. Formát *adresní_část*i je „City“, pokud je ulice nenastavena či prázdná, jinak „City, Street“. Část *číselná_část* je složena obdobně – pokud je druhé číslo adresy *(StreetNr2)* nenastaveno, je ve formátu „StreetNr“, jinak ve formátu „StreetNr/StreetNr2“. Toto skládání vysoce ovlivňuje úspěšnost výsledků při získávání GPS souřadnic při geokódování. Vypozorovali jsme, že při tomto skládání adres dostáváme nejméně chyb při převodu.

Caching

Na obrázku 3.17 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní ILocationConvertionManager Rozhraní definuje právě jednu metodu, která má za úkol uložit do repozitáře graf s GPS daty a za určitých předpokladů vrátit kolekci dvojic řetězcům, kde prvním bude identifikátor entity a druhým chyba vysvětlující, proč se nepodařilo získat GPS souřadnice dané entity.

Třída LocationConvertionFactory Tato třída slouží k vytváření instancí implementujících rozhraní *ILocationConvertionManager*. Využívá návrhový vzor GoF Abstract Factory.

Třída LocationConvertionManager Tato **abstraktní** třída implementuje rozhraní *ILocationConvertionManager* a obsahuje obecný algoritmus převodu identifikátorů lokace entit na GPS souřadnice.

Třída RuianConvertionManager Dědí ze třídy *LocationConvertionManager*, implementuje abstraktní metody potřebné pro převod RUIAN identifikátoru nebo URL s RUIAN identifikátorem na GPS souřadnice.

Obrázek 3.17: Caching – diagram tříd

Třída AddressConvertionManager Je **abstraktní**, stejně tak jako její předek *LocationConvertionManager*. Rozšiřuje abstrakci o další obecný algoritmus pro převod adres na GPS souřadnice.

Třída ComposedAddressConvertionManager Je konkrétní třídou, dědí z *AddressConvertionManager*. Reprezentuje převod mezi entitami majícími adresu jako jeden řetězec.

Třída SeparateAddressConvertionManager Je konkrétní třídou, dědí z *AddressConvertionManager*. Reprezentuje převod mezi entitami majícími složky adresy.

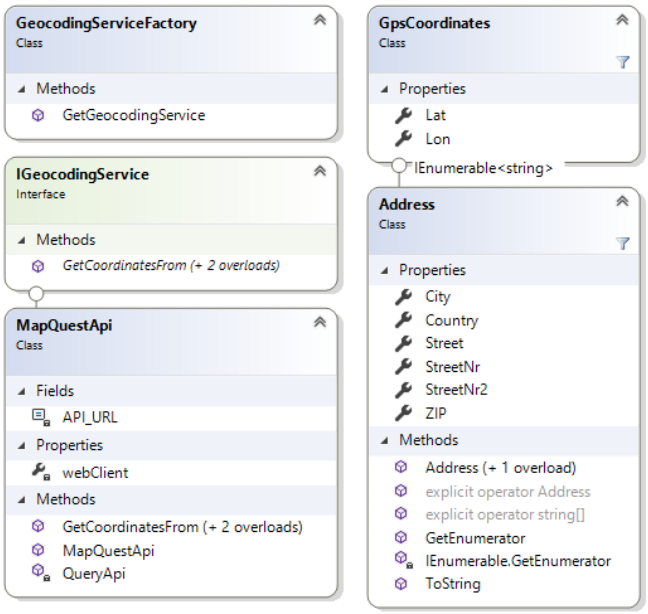
Geocoding

Na obrázku 3.18 je k nahlédnutí diagram tříd příslušných do této sekce.

Třída GpsCoordinates Reprezentuje dvojici GPS souřadnic.

Rozhraní IGeocodingService Toto rozhraní slouží k provedení převodu adres v různých formách na GPS souřadnice.

Třída GeocodingServiceFactory Slouží k vytváření instancí tříd implementujících rozhraní *IGeocodingService*. Využívá návrhový vzor GoF Abstract Factory.



Obrázek 3.18: Geocoding – diagram tříd

Třída MapQuestApi Implementuje rozhraní *IGeocodingService*. Využívá služeb OpenStreetMap Nominatim API, které je poskytováno mimo jiné jako jedna ze služeb serveru **MapQuest** (https://www.mapquest.com/).

3.4.8 Map

Na obrázku 3.19 je k nahlédnutí diagram tříd příslušných do této sekce.

Rozhraní IVisualizationValidator Předepisuje metody, které má validátor vizualizace obsahovat.

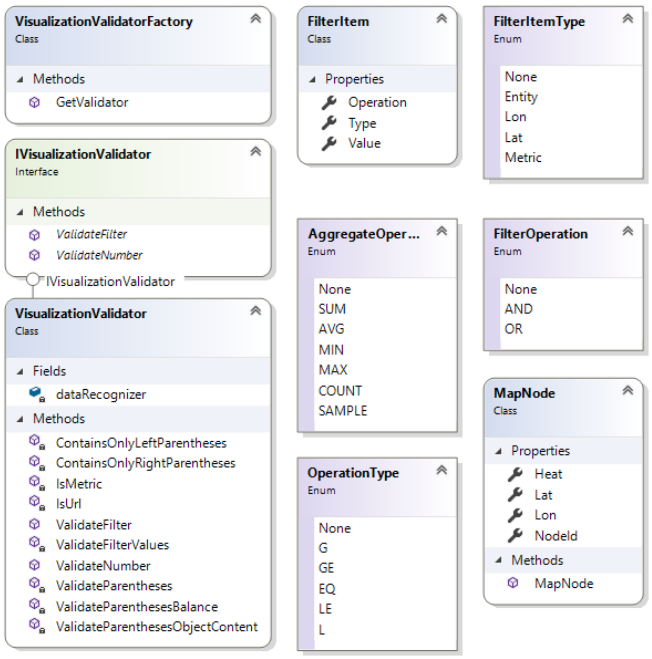
Třída VisualizationValidatorFactory Slouží k vytváření instancí implementující rozhraní *IVisualizationValidator*.

Třída VisualizationValidator Implementuje rozhraní *IVisualizationValidator*. Validuje výběr vizualizace (filtry, uzávorkování, limit či offset).

Třída MapNode Reprezentuje bod vizualizovatelný do mapy. Obsahuje čtveřici (id, zeměpisná délka, zeměpisná šířka, heat). Heat určuje míru zabarvení.

Výčet FilterItemType Výčet obsahuje položky udávající, k jakému vybíranému poli se filtr vztahuje (entita, její zeměpisná šířka, její zeměpisná délka či její metrika)

Výčet OperationType Výčet obsahuje operace jako větší, větší rovno, rovno, ...



Obrázek 3.19: Map – diagram tříd

Třída FilterItem Reprezentuje jednu položku filtru, která je složena z typu filtru (*FilterItemType*), operace (*OperationType*) a samotné hodnoty uchované v řetězci.

Výčet FilterOperation Výčet obsahuje operace AND a OR – tedy operace, kterými lze jednotlivé položky třídy *FilterItem* spojovat.

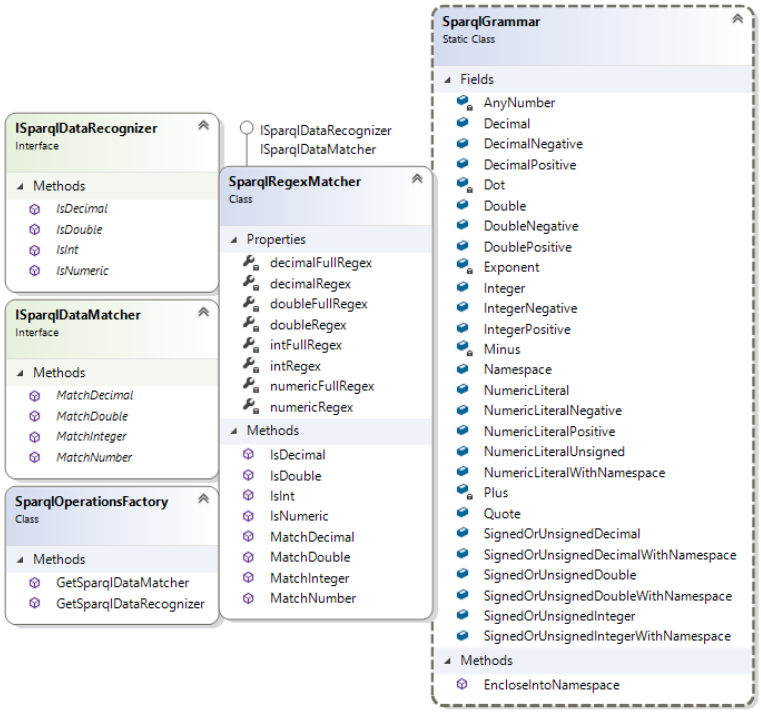
Výčet AggregateOperation Výčet obsahuje výběr agregačních funkcí jazyka SPARQL, které má smysl v naší aplikaci podporovat. Jedná se o SUM, AVG, MIN, MAX, COUNT s SAMPLE.

3.5 Knihovna SparqlDataAnalyzer

Během vypracovávání práce jsme naimplementovali i knihovnu *SparqlDataAnalyzer*, která slouží pro extrakci metrických hodnot z výsledků SPARQL dotazů. Důvodem je to, že numerické hodnoty ve výsledcích SPARQL dotazů mohou mít různé formy. Například “1” nebo “1”^^xsd:integer značí stejný RDF literál. Pomocí regulárních výrazů založených na gramatice dotazovacího jazyka SPARQL jsme naimplementovali extraktor těchto hodnot. Ten by měl pro výše uvedené příklady vrátit celé číslo 1.

Na obrázku 3.20 je k nahlédnutí diagram všech tříd této knihovny.

ISparqlDataMatcher Definuje metody vracející *System.Text.RegularExpressions.Match* pro *integer*, *double*, *decimal* nebo libovolný numerický typ.



Obrázek 3.20: Knihovna SparqlDataAnalyzer – diagram tříd

ISparqlDataRecognizer Definuje obdobné metody jako předchozí rozhraní, jen nevrací *Match*, ale *bool*.

Třída SparqlOperationsFactory Slouží k vytváření instancí implementujících rozhraní *ISparqlDataMatcher* nebo *ISparqlDataRecognizer*. Využívá návrhového vzoru GoF Abstract Factory.

Třída SparqlGrammar Tato statická třída obsahuje podmnožinu relevantních terminálů a neterminálů gramatiky jazyka SPARQL potřebných pro detekci numerických hodnot. Pro detekci hodnoty s namespacem (např. “1”^^xsd:integer) využívá zjednodušený zápis a za dvěma stříškami značícími začátek namespacu očekává libovolný počet libovolných znaků.

Třída SparqlRegexMatcher Implementuje obě rozhraní *ISparqlDataMatcher* a *ISparqlDataRecognizer*. Využívá k tomu regulárních výrazů definovaných ve třídě *SparqlGrammar*.

3.6 Datový zdroj

3.6.1 Nahrávání

Která data k sobě nahráváme a co k sobě ukládáme je rozepsáno v kapitole 3.4.1. Dokonce je možné se podívat na obrázek 2.2 a prozkoumat tak celý proces nahrávání dat.

3.6.2 Validace

Při načtení datového zdroje jej musíme validovat a zjistit, zda odpovídá našim požadavkům. Povolené formáty nahrávaných souborů jsou Turtle, N-Triples, RDF/XML a XML. Detailnější popis validace dat i schématu z pohledu programátorského se nachází v kapitole 3.4.4.

Samotný validační SPARQL dotaz se vyskytuje v kapitole 3.7.1.

3.7 Schéma datového zdroje

3.7.1 Validace schématu

K validaci schématu se spouští SPARQL dotaz uvedený v kapitole 3.8.6. Pokud je výsledek spuštění dotazu prázdný, je schéma nevalidní.

3.7.2 Získání a příprava dat pro vizualizaci

Podkladovým zdrojem dat je uživatelské schéma, které získáme dotazem, který je uveden v kapitole 3.7.1. Tuto selekci musíme nějak zpracovat a přetvořit ve výsledné schéma, které budeme vizualizovat. Nejdříve se zbavíme ekvivalentních tříd, pak vytvoříme prefixy a jména prefixujeme, aby vizualizovaný model nebyl nepřehledný. Až máme model připravený, potřebujeme vytvořit vyfiltrovaný model, protože schémata mohou dosahovat velkých rozměrů. Vyfiltrovaný model vytvoříme tak, že vezmeme kompletní model a spočítáme si stupně vrcholů. Do filtrovaného modelu přidáme vždy vrchol owl:Class – aby byl vizualizovaný graf vždy souvislý – a dva další prvky s nejvyššími stupni vrcholů. Do filtrovaného modelu (grafu) přidáme všechny hrany těchto tří vrcholů a následně do vrcholů grafů přidáme všechny

Sloučení ekvivalentních tříd

O tuto činnost se stará třída *EquivalentClassJoiner*. Algoritmus je známý – najdeme všechny ekvivalentní třídy a určíme pro ně komponenty souvislosti. Aplikujeme algoritmus ze strany 150 z knihy *Průvodce labyrintu algoritmů*, kterou napsali Mareš a Valla (2017) a ekvivalentní třídy máme tímto vyřešeny.

Automatická detekce typů

Při transformaci dat na instanci třídy *SchemaVisualizationModel* automaticky detekujeme typy a pokud narazíme na známý typ, přiřadíme mu příslušnou kategorii. O ně se stará třída *CategoryManager*. Druhá detekce typů již probíhá přímo na datovém zdroji uživatele a ne jen pouze na schématu. Vyselektujeme si vždy jeden prvek u potenciálních metrických vlastností, které musí být typu owl:DatatypeProperty – pouze pod touto vlastností se mohou skrývat RDF literály. Kategorii (či typ) prvku této selekce přiřadíme na základě toho, zda vyselektovaná hodnota byla metrická či nikoliv.

Seznam známých předehraných typů je dohledatelný v kódu ve třídě *CategoryManager* v metodě *void LoadTypes()*.

Vizualizace pomocí knihovny D3.js

Tato JavaScriptová knihovna slouží pro vizualizaci dat. Využitím simulace D3.js *force* dosáhneme jednoduše našeho cíle, kterým je vykreslení grafu s vrcholy a hranami. Stačí simulaci předat data vrcholů a hran a určit, jaké síly mají na simulaci působit. Můžeme těmto objektům simulace přidat různé popisky, barvy (které právě barevně odlišují detekované typy jednotlivých vrcholů) nebo na ně nastavit různé události. Nastavili jsme událost pro potáhnutí vrcholů grafu (drag), kterou může uživatel měnit pozice prvků grafu, dále mají vrcholy a hrany nastaveny událost *mouseover*, tudíž při tom, kdy nad ně myš najede, se zobrazí popisek s celým jménem. Nastavili jsme i funkčnost, která dělá vrcholy „rozklikávací“. Vizualizujeme totiž filtrovaný model a pokud by chtěl uživatel při výběru vizualizace vybrat nějaký vrchol, který ve vizualizaci není viditelný, může se k němu přes sousední viditelné vrcholy proklikat. Dvojklikem na vrchol uživatle zařídí to, že se do vizualizace přidá jeho nejbližší, prozatím nevizualizované, okolí – vrcholy a hrany.

Validace výběru vizualizace

O validaci se stará třída *PathSelectionValidator* a jak přesně probíhá je popsáno v příslušném odstavci kapitoly 3.4.6.

3.8 Selekce dat z databáze

Veškeré naše SPARQL dotazy do databáze jsou vygenerované a dědí ze třídy *AbstractQuery*. Obrázek 3.13 znázorňuje vztahy mezi těmito třídami a v kapitole 3.4.5 jsme popsali, jak provádení selekce probíhá – veškerou logiku samotné selekce obsahuje třída *AbstractQuery*. V konkrétních potomcích pouze sestavujeme text dotazu (query string) na základě parametrů předaných do konstruktoru těchto konkrétních potomků.

V dalších podkapitolách si uvedeme příklady jednotlivých dotazů.

3.8.1 AddressCacheQuery

Reprezentuje dotaz, který vybírá identifikátory entit a jejich zadanou lokaci. Slouží pro selekci vstupních dat pro geokódování.

- Složená adresa

```
SELECT ?mainEntity ?address
FROM <http://fake-url.com/data/>
WHERE {
  ?mainEntity a <http://example.cz/entity> .
```

```
  ?mainEntity <http://example.cz/hasAddress>/<http://example.
  cz/textAddress> ?address
  .
}
```

- Složky adresy

```
SELECT ?mainEntity ?country ?city ?street ?streetNr
?streetNr2 ?zip
FROM <http://fake-url.com/data/>
WHERE {
  ?mainEntity a <http://cedropendata.mfcr.cz/c3lod/cedr/vocab
  CEDR#PravnickaOsoba>
  .
  OPTIONAL { ?mainEntity
  <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#sidliNaAd
  rese>/<http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#nac
  haziSeNaUzemiStatu>/<http://cedropendata.mfcr.cz/c3lod/isdp
  /vocabIsdp/space/v1#statNazevZkraceny> ?country .
  }
  OPTIONAL { ?mainEntity <http://cedropendata.mfcr.cz/c3lod/
  cedr/vocabCEDR#sidliNaAdrese>/<http://cedropendata.mfcr.cz/
  c3lod/isdp/vocabIsdp/space/v1#obecNazev> ?city .
  }
  OPTIONAL { ?mainEntity <http://cedropendata.mfcr.cz/c3lod/
  cedr/vocabCEDR#sidliNaAdrese>/<http://cedropendata.mfcr.cz/
  c3lod/isdp/vocabIsdp/space/v1#uliceNazev> ?street .
  }
  OPTIONAL { ?mainEntity <http://cedropendata.mfcr.cz/c3lod/
  cedr/vocabCEDR#sidliNaAdrese>/<http://cedropendata.mfcr.cz/
  c3lod/isdp/vocabIsdp/space/v1#objektCisloDomovni> ?streetNr
  . }
  OPTIONAL { ?mainEntity <http://cedropendata.mfcr.cz/c3lod/
  cedr/vocabCEDR#sidliNaAdrese>/<http://cedropendata.mfcr.cz/
  c3lod/isdp/vocabIsdp/space/v1#objektCisloOrientacni>
  ?streetNr2 . }
  OPTIONAL { ?mainEntity <http://cedropendata.mfcr.cz/c3lod/
  cedr/vocabCEDR#sidliNaAdrese>/<http://cedropendata.mfcr.cz/
  c3lod/isdp/vocabIsdp/space/v1#psc> ?zip .
  }
}
```

3.8.2 ExampleResourceQuery

Reprezentuje dotaz, který vybere z databáze reprezentativní vzorek zadané vlastnosti. Slouží pro testy metrických hodnot u validace výběru vizualizace a pro automatickou detekci.

- Detekce metrické hodnoty

```
SELECT ?exRes
FROM <http://fake-url.com/data/>
WHERE {
  ?mainEntity <http://example.cz/textAddress> ?exRes .
}
LIMIT 1
```

- Validace metrické hodnoty

```
SELECT ?exRes
FROM <http://fake-url.com/data/>
WHERE {
  ?mainEntity a <http://example.cz/entity> .
  ?mainEntity <http://example.cz/value> ?exRes .
}
LIMIT 1
```

3.8.3 PathSelectionQuery

Reprezentuje dotaz, kterým vybereme výběr vizualizace. Protože v grafu http://fake-url.com/selection/ nic jiného neukládáme, stačí z něj vybrat všechny trojice bez dalšího omezení.

```
SELECT ?s ?p ?o
FROM <http://fake-url.com/selection/>
WHERE {
  ?s ?p ?o .
}
```

3.8.4 ResultQuery

Reprezentuje dotaz, jehož výsledek slouží jako data pro finální vizualizaci.

- Průměr rozhodnutých částek právnických osob seřazený sestupně

```
SELECT ?mainEntity ?lon ?lat
(AVG(<http://www.w3.org/2001/XMLSchema#double>(?metric)) as
?op)
FROM <http://fake-url.com/data/>
FROM <http://fake-url.com/gps/>
WHERE {
  ?mainEntity a <http://cedropendata.mfcr.cz/c3lod/cedr/vocab
  CEDR#PravnickaOsoba>
  .
  ?mainEntity <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCE
  DR#obdrzelDotaci>/<http://cedropendata.mfcr.cz/c3lod/cedr/v
  ocabCEDR#byloRozhodnuto>/<http://cedropendata.mfcr.cz/c3lod
  /cedr/vocabCEDR#castkaRozhodnuta> ?metric
  .
```

```
  ?mainEntity <http://www.w3.org/2003/01/geo/wgs84_pos#lon>
  ?lon .
  ?mainEntity <http://www.w3.org/2003/01/geo/wgs84_pos#lat>
  ?lat .
}
GROUP BY ?mainEntity ?lon ?lat
ORDER BY DESC(?op)
```

- Selekce 30 právnických osob a jejich maximálních částkek rozhodnuté dotace mimo obdélníky[6] dané souřadnicemi s zeměpisnou délkou mezi [14.22, 14.66] a zeměpisnou šířkou mezi [49.55, 50.2235] bez prvních 10 nejvyšších dotací, seřazeno sestupně.

```
SELECT ?mainEntity ?lon ?lat
(MAX(<http://www.w3.org/2001/XMLSchema#double>(?metric)) as
?op)
FROM <http://fake-url.com/data/>
FROM <http://fake-url.com/gps/>
WHERE {
  ?mainEntity a <http://cedropendata.mfcr.cz/c3lod/cedr/vocab
  CEDR#PravnickaOsoba>
  .
  ?mainEntity <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCE
  DR#obdrzelDotaci>/
  <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#byloRozho
  dnuto>/
  <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#castkaRoz
  hodnuta> ?metric
  .
  ?mainEntity <http://www.w3.org/2003/01/geo/wgs84_pos#lon>
  ?lon .
  ?mainEntity <http://www.w3.org/2003/01/geo/wgs84_pos#lat>
  ?lat .
  BIND(<http://www.w3.org/2001/XMLSchema#integer>(?lon) as
  ?intlon) .
  BIND(<http://www.w3.org/2001/XMLSchema#decimal>(?lon) as
  ?dcmlon) .
  BIND(<http://www.w3.org/2001/XMLSchema#double>(?lon) as
  ?dbllon) .
  BIND(<http://www.w3.org/2001/XMLSchema#integer>(?lat) as
  ?intlat) .
  BIND(<http://www.w3.org/2001/XMLSchema#decimal>(?lat) as
  ?dcmlat) .
  BIND(<http://www.w3.org/2001/XMLSchema#double>(?lat) as
  ?dbllat) .
  FILTER (
  (((?intlat > 50.2235) || (?dcmlat > 50.2235) || (?dbllat >
  50.2235))
```

[6] Obdelník určen souřadnic vnějšího okolí Prahy

```
    || ((?intlat < 49.55) || (?dcmlat < 49.55) || (?dbllat <
    49.55)))
    && (((?intlon > 14.66) || (?dcmlon > 14.66) || (?dbllon >
    14.66))
    || ((?intlon < 14.22) || (?dcmlon < 14.22) || (?dbllon <
    14.22)))
    )
  }
  GROUP BY ?mainEntity ?lon ?lat
  ORDER BY DESC(?op)
  LIMIT 30
  OFFSET 10
```

Filtrování numerických hodnot Filtrování jsme vyřešili způsobem, že ze zadaného pole zkusíme vytvořit všechny možné numerické typy (integer, decimal a double). Je to nejjednodušší způsob, jak vydobijeme numerickou hodnotu z pole v datovém zdroji. Tam může být uložena numerická hodnota libovolného typu ve formě řetězce. Pak musíme použít tento mechanismus. Pak již můžeme použít tyto extrahované hodnoty na porovnání s uživatelem zadanou hodnotou (např. *49.55*). Pokud je hodnota pole v datovém zdroji numerická, vždy bude obsažena alespoň v jedné proměnné obsahující její konverzi na integer, decimal nebo double.

3.8.5 RuianQuery

Reprezentuje dotaz, který provádí extrakci GPS souřadnic ze vzdáleného SPARQL endpointu.

```
SELECT ?mainEntity ?lon ?lat
FROM <http://fake-url.com/data/>
WHERE {
  ?mainEntity a <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR
  #PravnickaOsoba>
  .
  ?mainEntity <http://cedropendata.mfcr.cz/c3lod/cedr/vocabCEDR#s
  idliNaAdrese>/<http://cedropendata.mfcr.cz/c3lod/isdp/vocabIsdp
  /space/v1#adresniMistoKod> ?ruian
  .
  SERVICE <https://ruian.linked.opendata.cz/sparql> {
    ?adrMisto <https://ruian.linked.opendata.cz/slovník/adresníBo
    d>/<http://schema.org/geo> ?geo
    .
    ?adrMisto <http://www.w3.org/2004/02/skos/core#notation>
    ?ruian .
    ?geo <http://schema.org/longitude> ?lon .
    ?geo <http://schema.org/latitude> ?lat .
  }
}
```

3.8.6 SchemaQuery

Reprezentuje dotaz, který konstruuje celé schéma.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
  ?thing a ?type ;
    rdfs:label ?thingLbl .

  ?property rdfs:label ?label ;
    rdfs:domain ?domain ;
    rdfs:range ?range ;
    owl:equivalentClass ?equiv ;
    rdfs:subClassOf ?super .

} WHERE {
  ?thing a ?type .
  OPTIONAL { ?thing rdfs:label ?thingLbl . }

  { ?property rdfs:domain ?domain . }
  UNION
  { ?property rdfs:range ?range . }
  UNION
  { ?property owl:equivalentClass ?equiv . }
  UNION
  { ?property rdfs:subClassOf ?super . }
  UNION
  { ?property rdfs:label ?label . }

  VALUES (?type) { (owl:Class) (owl:ObjectProperty)
  (owl:DatatypeProperty) (owl:AnnotationProperty)
  (rdfs:Datatype) }
}
```

Vysvětlení dotazu Sestrojujeme trojice takové, které jsou typu uvedené v sekci *VALUES*. Zajímá nás jejich popisek (rdfs:label) a tyto údaje jejich vlastností tohoto typu:

- Popisek (rdfs:label)
- Vlastnost rdfs:domain, která určuje vztah mezi touto vlastností a selektovaným objektem (?domain)
- Vlastnost rdfs:range, která určuje vztah mezi touto vlastností a selektovaným objektem (?range)
- Vlastnost být „ekvivalentní třída“ (owl:equivalentClass)
- Vlastnost být „podtřída“ (rdfs:subClassOf)

3.9 Získávání GPS souřadnic

Z datového zdroje uživatele potřebujeme dostat GPS souřadnice pro to, abychom mohli data vizualizovat v mapě, jelikož GPS očekávají současná API.

Existují tyto **typy identifikátorů lokace**:

- GPS souřadnice
- adresa
- jiný identifikátor (z něj lze typicky získat jeden z předchozích dvou identifikátorů)

3.9.1 GPS

Pro GPS souřadnice je typické, že jsou různých v datových zdrojích uchovávány v různých formách. Můžeme je mít uložené ve formě jak jednotlivých dvou oddělených složek (zeměpisná šířka a zeměpísná délka), tak je můžeme mít i sloučené v jednom poli a pokud jsou sloučené, tyto složky mohou mít různé oddělovače... Naše aplikace bude umět vizualizovat data, která budou mít formát GPS souřadnic jak ve dvou oddělených složkách, tak sloučené, ale v tomto případě pouze pokud, že jako první složka v datech bude zeměpisná šířka a oddělovačem souřadnic bude čárka. Zároveň očekáváme, že každá složka GPS souřadnice je ve formátu čísla s desetinnou čárkou (odpovídající SPARQL typu *decimal*). Oddělovač složek a která složka má být první, se určuje až při generování dotazu ve třídě *ResultQuery*. Při rozšiřování aplikace je poměrně snadné toto upravit, argumenty mohou být do této třídy předány odněkud zvenku prostřednictvím volání z interfacu *IQueryHandler*, resp. v nejideálnějším případě by tato informace měla být perzistentně uložena a být již součástí výběru vizualizace (jemuž odpovídá třída *PathSelection*). Pokud jsou GPS data uložena v samotných datech, máme úlohu jednoduchou – stačí se nám pouze správně dotázat do datového zdroje a to je zařízeno tím, jak si ukládáme do datového zdroje výběr vizualizace. Tam již máme přímo SPARQL cesty zabudovány, a tak tyto cesty stačí do dotazu vložit. Pokud jsou ale GPS složky v jednom poli, musíme je pomocí proměnných ve SPARQL dotazu nejprve rozdělit a vydobít z nich tyto jednotlivé složky.

V jakém formátu jsou GPS složky uloženy však na jejich samotnou selekci nemá vliv, v případě rozšíření je pouze potřeba udělat změna validace (nyní validujeme, že jsou složky numerické a odpovídají typu *decimal*) a pravděpodobně i jakási konverze při jejich přečtení z výsledku dotazu. API knihovny Leaflet, kterou využíváme pro vizualizaci dat, pravděpodobně není také připraveno na braní „nedecimalových“ GPS souřadnic.

3.9.2 Adresa

V případě selekce adresy je potřeba využít nějaké API pro geokódování a získat tak souřadnice z této adresy. Není to ale tak jednoduché. Pokud je adresa v da-

tech uložena jako jeden řetězec, nemůžeme se složkami adresy nic dělat (pokud bychom neznali jejich přesnou strukturu) a úspěšnost při geokódování nezávisí na nás. Pokud jsou v datovém zdroji však adresy několikasložkové, záleží na nás, jak tyto složky poskládáme a jak tedy uděláme z několikasložkové adresy jeden řetězec. Empiricky jsme vyzkoumali, že nejlepší formát sloučení adresy je ten, který byl popsán v kapitole 3.4.7. Respektive na tomto formátu jsme dostávali nejméně chybových hlášení na námi testovaném vzorku dat.

Geokódovací služby

Na internetu je k dohledání několik geokódovacích služeb, přičemž většina z nich provozuje na svých serverech OpenStreetMap Nominatim API. My jsme se rozhodli využít služeb **MapQuest** (https://www.mapquest.com). Každá takováto služba si definuje vlastní podmínky používání a většinou stanovuje nějaké limity na počty požadavků za určité časové období. MapQuest je zdarma do 15000 požadavků měsíčně. Podmínkou je však registrace, aby mohl uživatel dostat svůj API klíč, přes který jej bude možné identifikovat. Služeb OpenStreetMap jsme se rozhodli nevyužít, protože jsou jejich podmínky použití příliš přísné a nevyhovovaly nám (nutnost cacheování aj.).

3.9.3 Jiné identifikátory

Naše aplikace podporuje i vizualizaci dat, ke kterým se váže i jiný identifikátor lokace než GPS souřadnice či adresa. Jedná se o RUIAN kód, který byl představen v kapitole 2.4.6. Podporujeme rovnou to, že v datech může být jak přímo tento kód jakožto celé číslo, tak to může být i URL s tímto číslem. V tom případě musí být však toto číslo za posledním lomítkem adresy, abychom ho rozpoznali. Při výběru vizualizace se validuje, zda je vybrané RUIAN pole číselné či zda je URL v právě předepsaném formátu.

Získání GPS z RUIAN Pro převod z RUIAN kódu, který již umíme z dat vyextrahovat a víme, kde se nachází, využijeme služeb externího SPARQL endpointu na adrese https://ruian.linked.opendata.cz/sparql. K zeměpísné délce se dostaneme při selekci subjektů s následující cestou: *<https://ruian.linked.opendata.cz/slovník/adresníBod>/<http://schema.org/geo>/<http://schema.org/longitude>*. K zeměpisné délce se dostaneme obdobně, stačí vyměnit poslední prvek SPARQL cesty za *<http://schema.org/latitude>*. Pro samotnou selekci těchto hodnot využijeme SPARQL Federated Query, který pošleme na náš SPARQL endpoint, tam vybereme určitá data, která nás zajímají a k nim se přidají z dalšího zadaného SPARQL endpointu zájmová data.

3.10 Finální vizualizace dat v mapě

3.10.1 Model vizualizace a jeho validace

Naše finální vizualizace dat probíhá tak, že po uživatelovo vyplnení vlastností modelové třídy *MapModel* na dané stránce musíme zvalidovat, jelikož zde uživatel

vybírá několik věcí ovlivňující výběr dat. Jsou to:

- **Operace.** Jedná se o agregační funkce jazyka SPARQL, které dávají smysl v kontextu naší aplikace. Jedná se o tyto:
  - Suma (SUM)
  - Průměr (AVG)
  - Minimum (MIN)
  - Maximum (MAX)
  - Počet (COUNT)
  - Vzorek (SAMPLE)

  Do budoucna je možné rozšíření domyslet scénář, kde by uživatel nemusel zadávat žádnou agregační funkci. Museli bychom vymyslet, jak by vizualizace měla probíhat – pravděpodobně bychom jakousi agregaci museli dělat my sami na jednotlivých GPS souřadnicích mapy a to není tak snadné kvůli tomu, jak knihovna Leaflet.js funguje.
- **Offset.** Neboli tolik prvních prvků výsledku se má přeskočit. Musí to být číslo větší než 0.
- **Limit.** Neboli kolik prvků chce maximálně uživatel vybrat. Musí to být číslo větší než 0.
- **Filtry.** Pokud chce uživatel filtrovat, musí vybrat, jakou vlastnost a s jakou relací (>, >=, =, =<, <). Na výběr z vlastností má:
  - **Entita (Entity).** Zde má smysl filtrovat pouze na rovnost identifikátorů.
  - **Zeměpisná délka (Lon).** Jakákoliv relace dává smysl pro filtrování.
  - **Zeměpisná šířka (Lat).** Jakákoliv relace dává smysl pro filtrování.
  - **Metrika (Metric).** Jakákoliv relace dává smysl pro filtrování.

  Filtrovací hodnoty zeměpisné délky, šířky a metriky by měly být samozřejmě numerické. Filtrovací hodnota entity by měla být řetězcem – nějakým identifikátorem entity (URL).

  Všechny výše zmíněné hodnoty jsou validovány popsaným způsobem.

Výsledky dotazu jsou vždy řazeny sestupně.
Řekněme, že **„filtr“** bude značit trojici *(vlastnost, relace, hodnota)*.

Operace mezi filtry Uživatel může filtrovat pomocí několika filtrů najednou a mezi nimi může přirozeně definovat operace AND či OR.

Uzávorkování Zároveň má uživatel možnost kolem jednotlivých filtrovacích políček vyplnit uzávorkování. U všech filtrů vlevo (první textové pole) slouží pro vkládání otevíracích závorek. Analogicky, první textové pole za filtrem slouží pro vkládání uzavíracích závorek pro daný filtr. U uzávorkování se kontrolují obsahy těchto polí a také to, že je uzávorkování správné a dává tedy smysl.

3.10.2 Použité nástroje

OpenStreetMap

Tento nástroj byl popsán již v úvodu. Čerpáme z jeho otevřených dat.

MapBox

Tento nástroj slouží pro dotazování se na tzv. „mapové čtverce“ (map tiles), což jsou čtverce rastrových map, které se vykreslují do pozadí. Na tuto službu se stačilo zaregistrovat a získat API klíč. Poskytovatelů těchto mapových čtverců je spousty, mimo jiné i OpenStreetMap, ale každý poskytovatel těchto obrázků si klade své podmínky použití a zároveň umožňuje využití jejich různých mapových podkladů.

Knihovna Leaflet.js

Knihovna Leaflet.js nám umožňuje zobrazit interaktivní mapu na naší stránce, se kterou je možné provádět základní operace jako přibližování a oddalování se, pohybování s mapou, zobrazovat v ní „značky“ (markers) a podobně. Tato knihovna je populární a existuje do ní mnoho pluginů – mimo jiné i na tvorbu heat map, a tak jsme se rozhodli tuto knihovnu vybrat.

Knihovna Leaflet.heat

Toto je knihovna přímo pro vykreslování heat map. Funguje řádově s desetitisíci prvky, což byl jeden indikátor, proč jí zvolit. Zároveň má jednoduché, přímočaré a intuitivní API. Nevýhodou je, že „heat“ musíme sami normalizovat, abychom všem bodům v mapě nepřiřadili číslo větší než 1, jelikož je to maximální hodnota, nicméně tuto nevýhodu vyřeší volání jedné krátké funkce. Jen pro něj musíme znát maximální a minimální hodnotu „heatu“ v datech z výsledku provedeného dotazu.

4. Uživatelská dokumentace

4.1 Nahrání schématu a dat

V levé části stránky se vyskytuje část formuláře pro nahrání schématu. V pravé části je část pro nahrání dat samotných.

Po vybrání, zda chcete schéma nebo data nahrát z URL SPARQL endpointu nebo ze souborů se Vám zobrazí příslušné doposud skryté části formuláře.Vyberte soubory nebo zadejte URL SPARQL endpointu a až budete mít vše připraveno, klikněte na tlačítko *Send*. Stav před odesláním formuláře by měl být podobný tomu, který lze vidět na obrázku 4.1

4.2 Výběr vizualizace

4.2.1 Vykreslené schéma

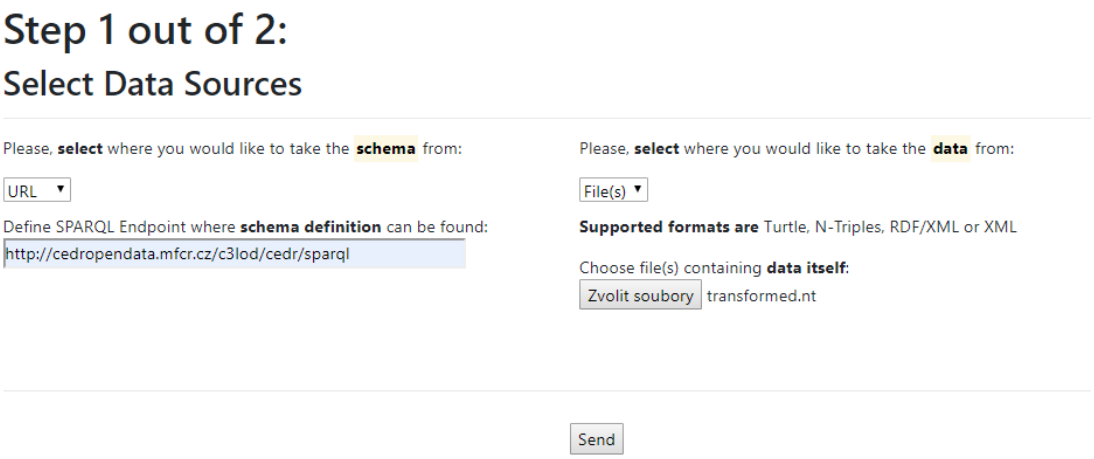
Vaše schéma může vypadat jako na obrázku 4.2 nebo 4.3. Manipulace se schématem je popsána v dalším bodě.

4.2.2 Provedení výběru

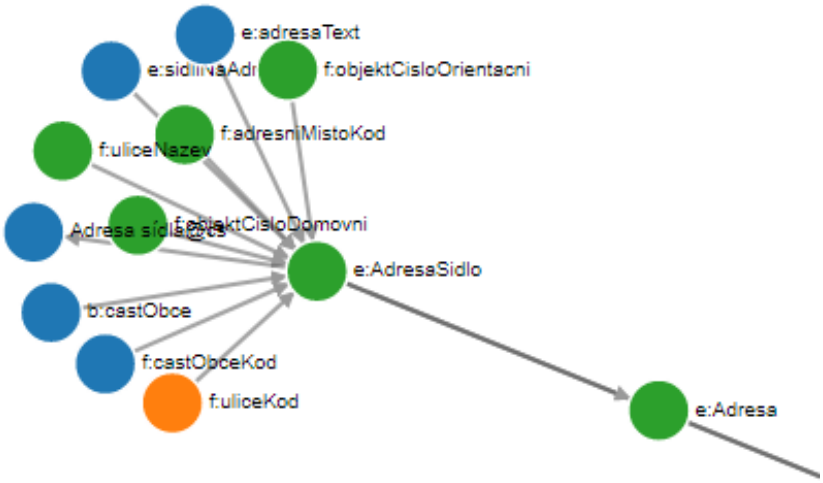
Postupujte podle návodu na stránce a vyberte hlavní entitu vizualizace, metriku vizualizace a hlavní lokační entitu. Výběr se provádí pravými kliky myší na prvek, který chcete vybrat. Přepínače vedle popisu názvů výběrů (např. *Main entity* nebo *Metric data*) ovlivňují, do kterého pole bude výběr myší prováděn. Pokud nějaké prvky nevidíte, jsou ve schématu prozatím skrytí. Musíte se k nim přes sousední prvky proklikat. Dvojklik na prvek dokreslí jeho nejbližší okolí. Do textových políček se dá psát i ručně. Cesta několika prvků se odděluje lomenem. Až budete mít vyplněny tento hlavní výběr, specifikujte, jakého typu adresa je a dovyplňte adresní výběr – tedy nově zobrazená pole formuláře. Váš výběr může vypadat jako na obrázcích 4.6 a 4.7.

4.3 Vizualizace a analýza

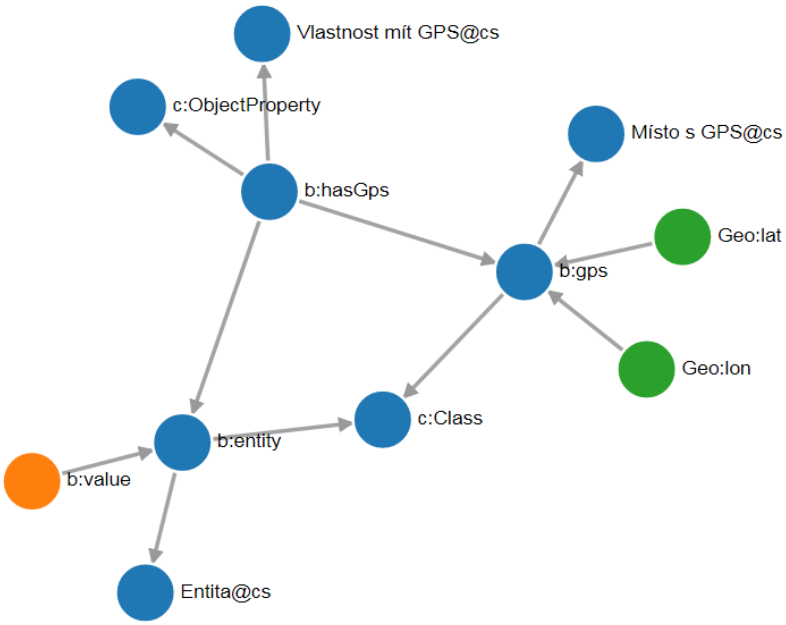
Po načtení stránky se Vám zobrazí mapa a nad ní formulář. Formulář vyplňte. Nahoře na stránce se vybírá agregační funkce, která se má na zadanou metriku aplikovat.
Po zakliknutí zaškrtávátka u pole *Offset* můžete přeskočit prvních několik výsledků.
Po zakliknutí zaškrtávátka u pole *Limit* můžete omezit maximální počet výsledků, které budou zobrazeny.
Po zaškrtnutí zaškrtávátka u pole *Filter* můžete data filtrovat. Kliknutí na tlačítko *Add filter* přidá filtr. Kliknutí na tlačítko *Remove filter* poslední filtr odebere. U jednotlivých filtrů můžete vyplňovat vlastnost filtrování (entita, zeměpisná délka, zeměpisná šířka nebo metrika). Filtrování u entit má smysl



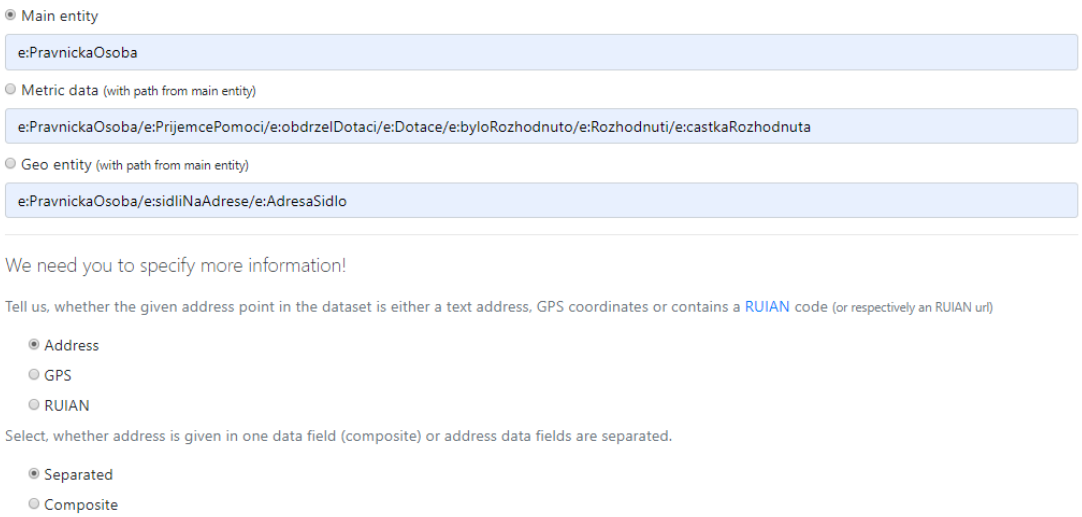
Obrázek 4.1: Nahrání dat



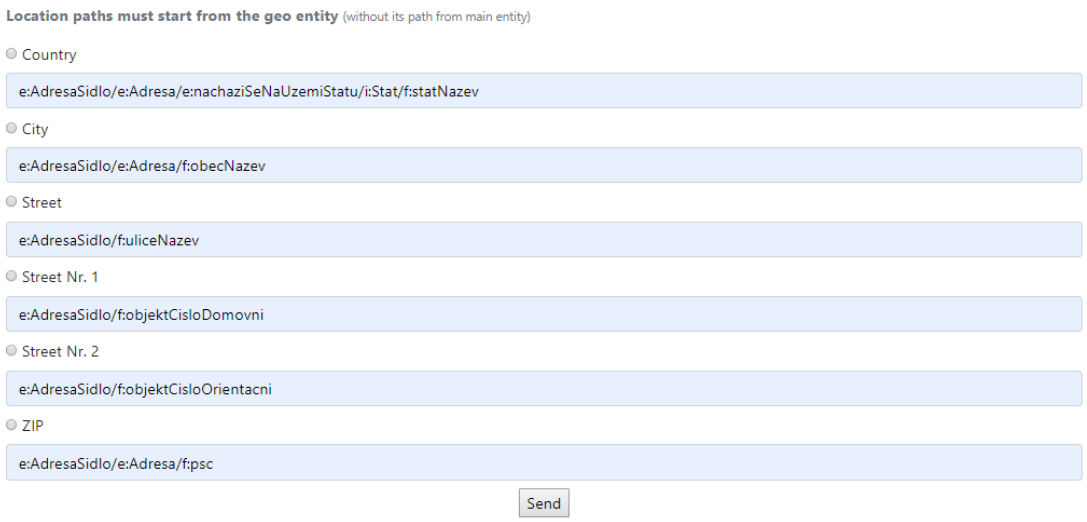
Obrázek 4.2: Část schématu – 1



Obrázek 4.3: Část schématu – 2



Obrázek 4.4: Vyplnění hlavní části výběru vizualizace

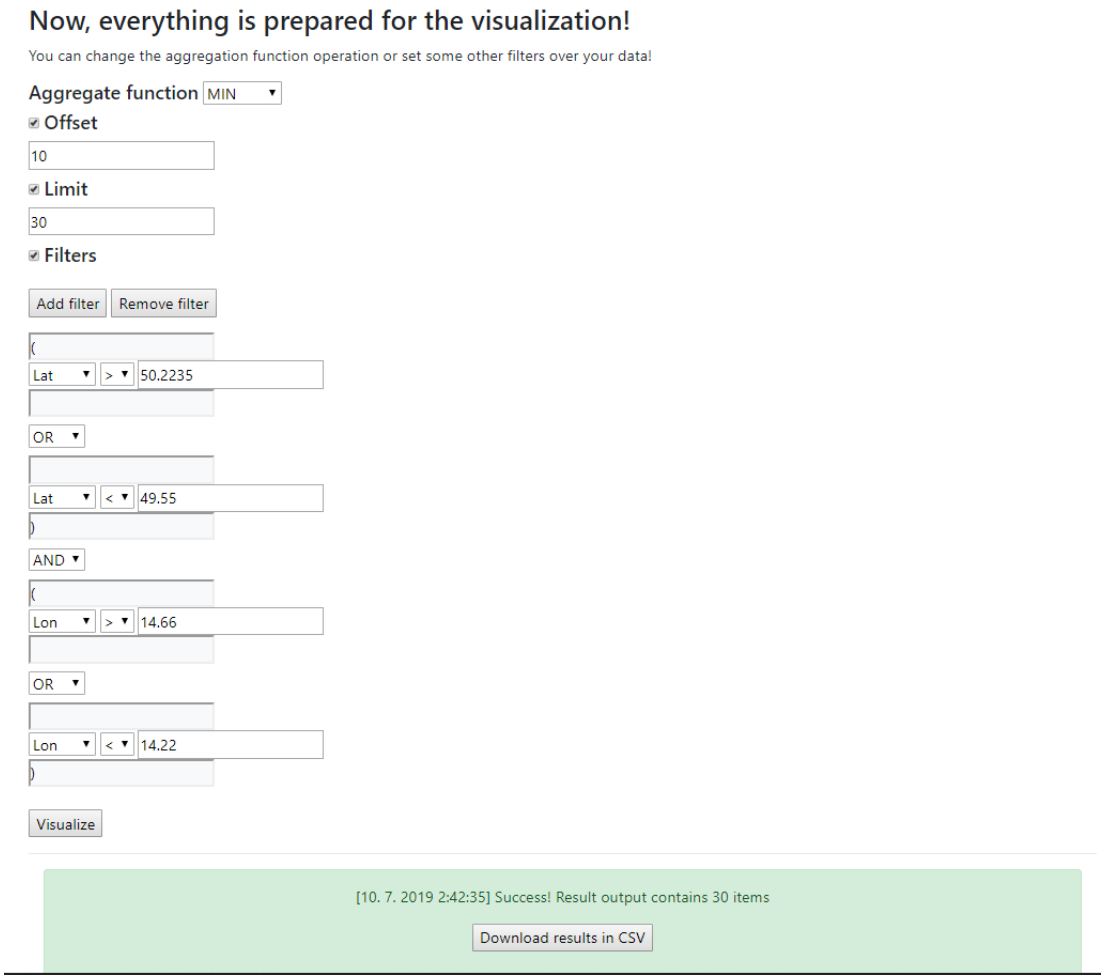


Obrázek 4.5: Vyplnění adresní části výběru vizualizace

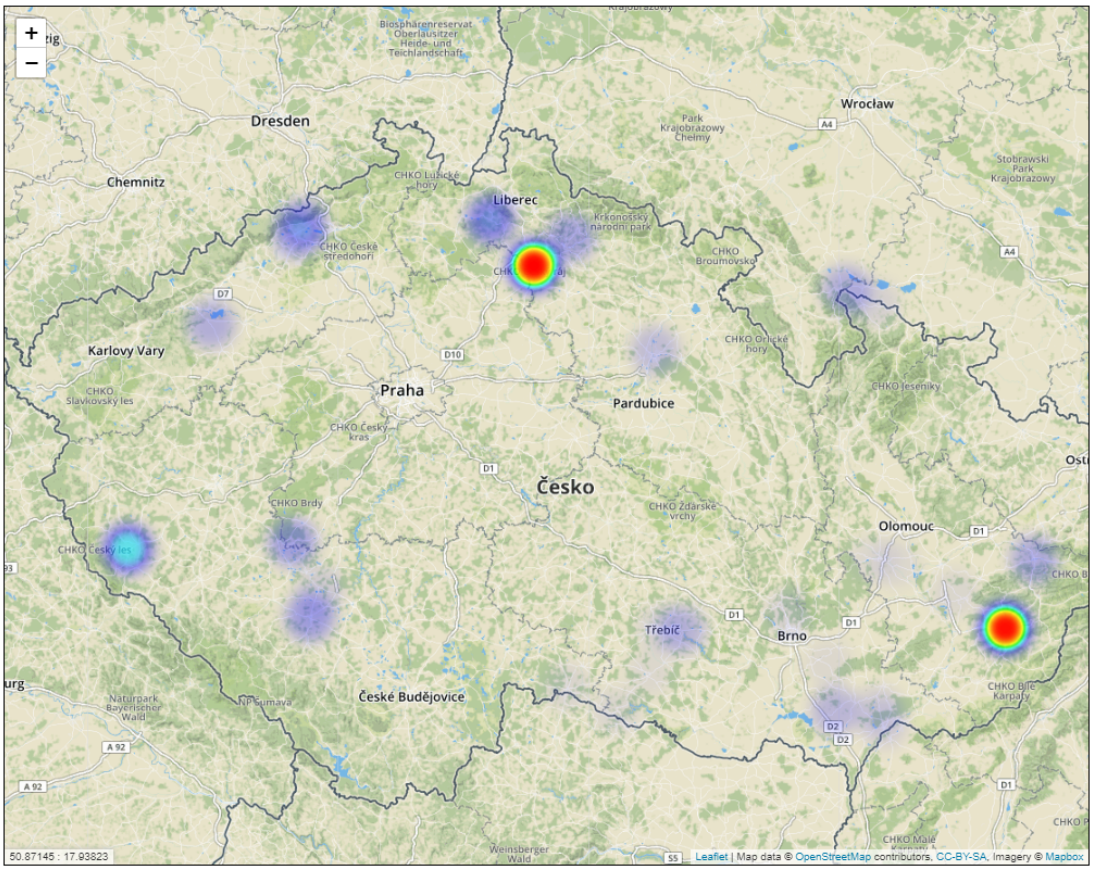
pouze na rovnost, očekávaná hodnota je URL identifikátor filtrované entity. Filtrování ostatních polí očekává numerické hodnoty má smysl vybrat jakékoliv znaménko relace. Mezi několika filtry existují operace AND a OR. Operace AND mezi dvěma filtry je požadavek na splnění obou filtrů zároveň. Operace OR mezi dvěma filtry je požadavek na splnění alespoň jednoho filtru. Jednotlivé filtry v seznamu filtrů jdou uzávorkovat pro změnu standardní priority operací AND a OR. U každého filtru jsou dvě textová pole. Pole nad filtrem slouží ke vkládání otevíracích závorek. Pole pod filtrem slouží ke vkládání uzavíracích závorek.

Až budete mít Váš formulář vyplněný, klikněte na tlačítko *Visualize*, abyste provedli vizualizaci.

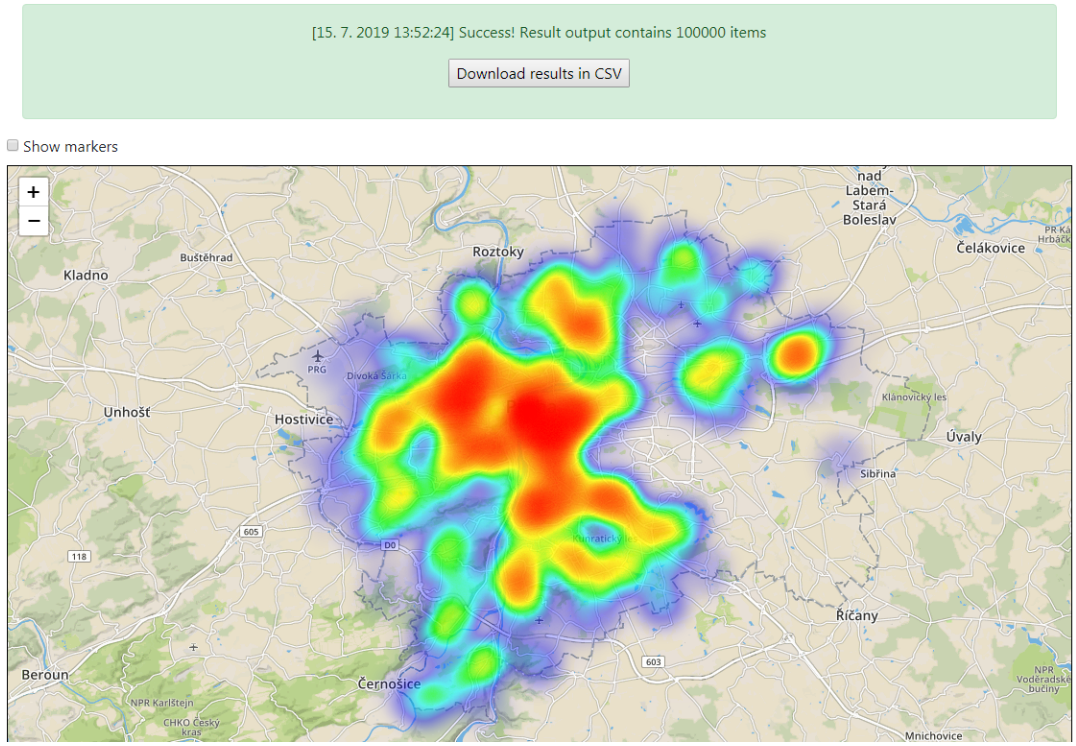
Vyplněný formulář může vypadat jako na obrázku 4.6, následně vykreslená heat mapa jako na obrázku 4.7. Dalším příkladem je obrázek 4.8.
Zaškrtnutím zaškrtávátka *Show markers* do mapy umístíte rozklikávací značky, které popisují jednotlivé entity vykreslené do mapy. Jak to vypadá, si můžete prohlédnout na obrázku 4.9.
Kliknutím na tlačítko *Download results in CSV* se stáhnou výsledky provedeného dotazu v CSV.



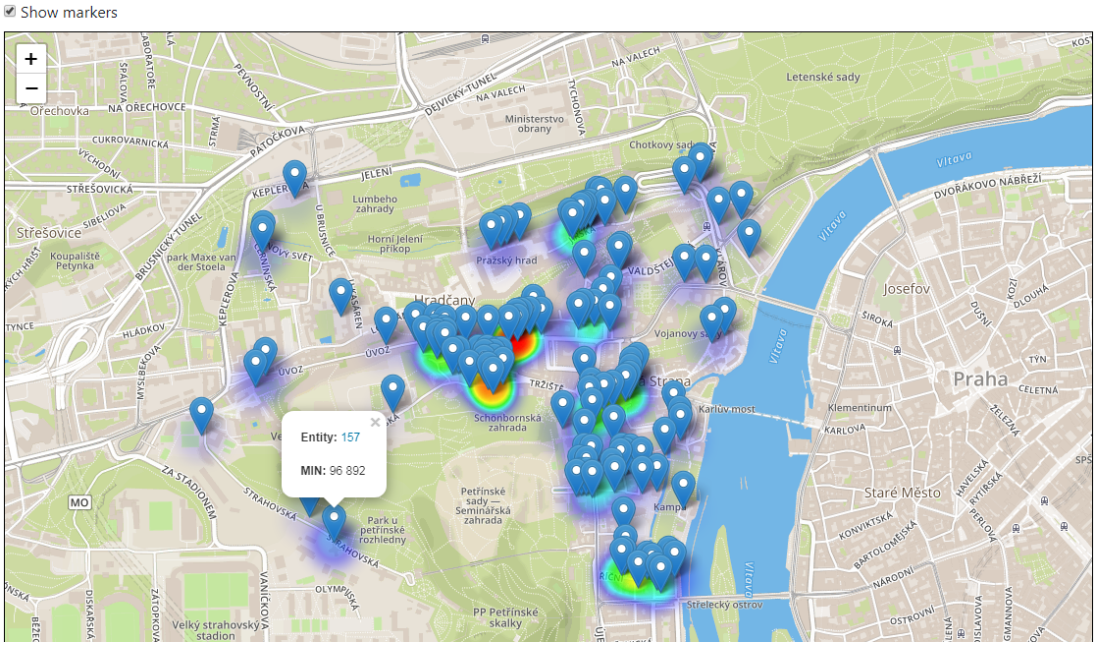
Obrázek 4.6: Vyplnění formuláře vizualizace



Obrázek 4.7: Vykreslená vizualizace



Obrázek 4.8: Vykreslená vizualizace se 100000 entitami



Obrázek 4.9: Vykreslená vizualizace se značkami

5. Programátorská dokumentace

5.1 Instalace Apache Jena Fuseki

1. Ze stránek https://jena.apache.org/download/ si stáhněte poslední verzi Apache Jena Fuseki
2. Zazipovaný soubor rozbalte a přes příkazovou řádku spusťte soubor *fuseki-server*, který se nachází v rozbalené složce.
3. Fuseki server by se po chvilce měl spustit. Ověřit lze tím, že na adrese localhost:3030[1] bude přístupné webové aplikační rozhraní serveru.

Na stránce https://jena.apache.org/documentation/fuseki2/fuseki-run.html by měl být podrobnější návod v případě komplikací.

5.2 Dokumentace samotná

Jako programátorská dokumentace slouží celá kapitola 3 a hlavně od kapitoly 3.4.2 dále, kde je obsáhle rozepsána funkčnost všech tříd a u všech komponent aplikace nechybí diagramy tříd.

Popis jednotlivých API se vyskytuje v samotném kódu, a tak není nutné sem zde uvádět cokoliv dalšího.

[1] Číslo portu se může lišit.

6. Testování

6.1 System Usability Scale

System Usability Scale[1] (SUS) neboli škála použitelnosti systému je formulářem, který měří míru použitelnosti aplikace. Zodpovězením 10 otázek na stupnici 1–5 dá uživatel najevo svou spokojenost s používáním aplikace. Skóre má vlastní vzorec výpočtu. Ten je k nahlédnutí na stránce zmíněné v poznámce pod čarou.

6.1.1 Otázky SUS

1. Rád bych tento systém často používal.
2. Systém mi přišel zbytečně složitý.
3. Systém šlo snadno používat.
4. Potřeboval(a) bych technickou podporu k tomu, abych mohl(a) tento systém používat.
5. Různé funkce systému byly dobře zaintegrovány.
6. Systém je příliš nekonzistentní (rozdílné ovládání podobných prvků aj.).
7. Většina uživatelů by se s tímto systémem naučila pracovat velice rychle.
8. Systém je velice těžkopádný.
9. Při používání systému jsem měl(a) pocit, že vím, co dělám.
10. Před používáním systému jsem se musel(a) naučit spoustu nových věcí.

Hodnocení je na stupnici 1–5, kde 1 znamená silný nesouhlas a naopak číslo 5 silný souhlas.

6.1.2 Evaluace výsledků

Průměrné skóre použitelnosti všech testovaných aplikací je 68. Tabulka 6.1 obsahuje výsledky hodnocení naší aplikace uživateli. Na řádcích je hodnocení jednotlivých osob, sloupce reprezentují otázky. V posledním řádku jsou průměry za jednotlivé otázky. V posledním sloupci je SUS jednotlivých uživatelů. Výsledné SUS hodnocení naší aplikace se nachází v pravém dolním rohu tabulky. Průměry jsou až na výsledné SUS skóre zaokrouhleny na 1 desetinné místo.

[1] https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**SUS**
P1	4	2	4	2	4	1	4	2	3	2	*75*
P2	5	1	5	1	5	1	5	1	5	1	*100*
P3	4	1	4	1	5	1	4	2	4	2	*85*
P4	4	2	4	1	5	1	4	5	5	5	*70*
P5	3	2	4	2	4	2	3	3	3	4	*60*
P6	4	2	4	1	5	1	4	2	5	2	*85*
	4.0	*1.7*	*4.2*	*1.3*	*4.7*	*1.2*	*4.0*	*2.5*	*4.2*	*2.7*	***79***

Tabulka 6.1: Tabulka výsledků SUS.

6.2 Unit testování

Během vývoje aplikace jsme napsali i několik unit testů. Velkou měrou jsou otestovány některé metody a třídy, které mají velký dopad na funkčnost aplikace. Jednou z testovaných tříd knihovny *SparqlDataAnalyzer* je třída *SparqlRegexMatcher*, kterou využíváme pro extrakci numerických hodnot z výsledků SPARQL dotazů. Dále testujeme validátor vizualizace, který testuje správné uzávorkování filtrů apod.

6.3 Testování objemu dat

V této kapitole budeme testovat, jaký vliv má zvyšování objemu dat na naší aplikaci a do jakých mezí zvládne aplikace pracovat.

6.3.1 Cache GPS souřadnic

RUIAN

Na jedno cachování RUIAN identifikátorů je přípustné nahrávat **stovky** entit s tímto identifikátorem. Omezení míry jsou dány serverem, který je pro převod využíván a časovými limity na našem Apache Jena Fuseki serveru. Převod RUIAN kódů na GPS souřadnice 2420 entit v jednom dotazu trvá mezi 90 a 100 sekundami.

Adresy

Převod adres na GPS souřadnice je omezen limity služeb, které využíváme pro geokódování. Ten je nastaven na 15000 dotazů měsíčně. Geokódování je pomalé, převod 200 adres trvá mezi 60 a 70 sekundami – to znamená circa 3 vteřiny na 1 adresu.

6.3.2 Vizualizace

Naše aplikace dokáže vizualizovat desítky tisíc až nízké stovky tisíc entit v heat mapě. Omezení je dáno časovým limitem čekání na výsledek dotazu. Bylo otestováno, že aplikace bezproblémově zvládne vizualizovat alespoň 100000 entit v heat mapě, viz obrázek 4.8. Načtení vizualizace heat mapy v tomto případě trvá okolo

42 sekund, z kterých circa 31 sekund trvá běh samotného dotazu s agregační funkcí a řazením na databázovém serveru. U menšího datového zdroje je to výrazně kratší doba. Pro 10000 entit trvá obdobný úkon mezi 5 a 6 sekundami, přičemž samotný dotaz na databázovém serveru běží mezi 3 a 4 sekundami.

7. Související práce

Dne 23. května 2019 vyšla studie *Survey of tools for Linked Data consumption*, na které se podíleli Klímek a kol. (2019). Studie zkoumala to, jak jsou propojená data využívána, jaké nad nimi existují aplikace a jakou funkcionalitu tyto aplikace poskytují. Zkoumáno bylo 110 nástrojů, které mohly býti potenciálními platformami pracujícími s propojenými daty (Linked Data Consumtion Platform – LCDP). Pomocí 94 kritérií rozdělených do 31 kategorií požadavků autoři studie zkoumané nástroje podrobili tříkolovému testování, ze kterého bylo vybráno do nejužšího výběru 16 nástrojů, které byly později mezi sebou navzájem porovnány na základě předem stanovených kritérií a na závěr bylo poukázáno na 8 nástrojů, které obstály nejlépe a splnily požadavky pro to, abychom je mohly označit za LCDP.

Požadavek č. 26 se zabýval schopností nástrojů data vizualizovat. Specifičtějšími kritérii, které se vztahovaly k tomuto požadavku, byly:

- **Kritérium 26.1:** Grafové vizualizace – nástroj nabízí automatizované vizualizace propojených dat na základě známých slovníků.
- **Kritérium 26.2:** Manuální vizualizace – nástroj nabízí možnost vizualizace propojených dat na základě manuálního mapování dat do vizualizace.
- **Kritérium 26.3:** Vizualizace schématu – nástroj nabízí automatizované vizualizace propojených dat na základě známých slovníků.

Naše aplikace neoddiskutovatelně splňuje kritérium 26.2 a pravděpodobně i částečně kritérium 26.3 – ač to není naším hlavním cílem, nějakým způsobem schéma vizualizujeme. Nicméně je (ne)splnění tohoto kritéria v našem případě lehce sporné. Automatizovaným způsobem detekujeme ve schématu různé „kategorie“ vrcholů grafu schématu na základě známých slovníků a následně je při vizualizaci odlišujeme barevně. Bohužel zde nemůžeme zahrnout druhou část naší automatické detekce typů, jelikož se při ní musíme dotazovat do datového zdroje. Protože je splnění této podmínky diskutabilní, zaměříme se v další kapitole pouze na kritérium 26.2 a srovnáme naší aplikaci s nástroji, které toto kritérium splnily také.

7.1 Manuální vizualizace

Ve studii bylo vyhodnoceno pro 5 nástrojů, že umí vizualizovat propojená data. Tři z nich, **YASGUI**, **OpenCube** a **Sparklis**, obdržely takové ohodnocení, které tvrdí, že data dovede v aplikaci vizualizovat i běžný uživatel, který nezná technologie spojené s propojenými daty. Další dva nástroje **VISU** a **Datalift** splňují kritérium 26.2 za předpokladu, že bude aplikaci používat expertní uživatel v oblasti propojených dat.

Naše aplikace by byla klasifikována stejně jako poslední dva zmíněné nástroje. Ačkoliv jsme se snažili navrhnout aplikaci tak, aby byla uživateli co

nejpřívětivější a snažili jsme se jej co nejvíce oprostit od technických detailů jazyka SPARQL a datového modelu RDF, uživatel stále musí rozumět samotné struktuře uložení dat v RDF, aby data uměl ve schématu lokalizovat. Tudíž by naší aplikaci dokázal použít pouze expertní uživatel nebo pokročilý a techničtěji zdatnější uživatel.

V následujících podkapitolách představíme všechny tyto nástroje zkoumané ve studii a poukážeme na rozdíly mezi nimi a naší aplikací.

7.1.1 YAGSUI

Je editorem SPARQL dotazů. Po spuštění dotazu umí výsledky vizualizovat v několika podobách – v mapě i jako různé tabulky.

www: http://yasgui.org

7.1.2 OpenCube

Z pohledu toho, co nástroj dělá, je ze všech ostatních nejpodobnější naší aplikaci. Tento nástroj však předpokládá nějakou strukturu dat (viz výběr při vizualizaci mapy ve videu – „muncipality“, „region“, ...), zatímco my povolujeme uživateli použít jeho vlastní strukturu dat v jeho vlastním slovníku.

www: http://opencube-project.eu/

Příklad použití: https://vimeo.com/135443264

7.1.3 Sparklis

Jedná se o stavitel dotazů jazyka SPARQL v přirozeném jazyce. Umí data vizualizovat v mapě či tabulce.Má v sobě integrovaný nástroj YASGUI (podkapitola 7.1.1), a tak dokonce umožňuje pokročilým uživatelům generovaný dotaz upravit.

www: https://wiki.dbpedia.org/projects/sparklis

Příklad použití: https://www.youtube.com/watch?v=uHzQ_K-MQQk

7.1.4 VISU

VISU umožňuje vizualizovat uživateli jakékoliv data, ale uživatel musí předložit nástroji vlastní SPARQL dotaz nebo využít některý z 5 předdefinovaných základních dotazů, který vizualizuje metadata SPARQL endpointu. Proto je nástroj klasifikován k použití pouze expertními uživateli.

www: http://data.aalto.fi/visualizations

	Naklikání dotazu	Napsání dotazu	Mapová vizualizace	Jiné vizualizace	Výstup dotazu v CSV
HV	Ano	Ne	Ano	Ne	Ano
YAS	Ne	Ano	Ano	Ano	Ano
DL	Ano	Ne	Ano	Ano	Ano
OC	Ano	Ne	Ano	Ano	Ano
Sp	Ano	Ano	Ano	Ano	Ne
VISU	Ano	Ano	Ano	Ano	Ano

Tabulka 7.1: Tabulka porovnání nástrojů pracujícími s vizualizacemi propojených dat

7.1.5 Datalift

Používání tohoto nástroje je dle mého názoru poměrně komplexní a není příliš přímočaré, ale zato nástroj poskytuje širokou funkcionalitu a celkově působí velice robustně. Hlavní výhodou je zajištění konverze „obyčejných“ dat do datového modelu RDF a následně umožňuje vizualizaci těchto ztransformovaných dat.

www: https://datalift.org/

www: https://www.dailymotion.com/video/x1kypw7

7.2 Porovnání nástrojů

V tabulce 7.1 je k nahlédnutí porovnání jednotlivých nástrojů. Mezi *jiné vizualizace* nástrojů považujeme všelikeré tabulky, grafy nebo časové osy. Níže je uveden seznam zkratek použitých v tabulce.

- **HV** HeatmapVisualizer (naše aplikace)
- **YAS** YASGUI
- **DL** Datalift
- **OC** OpenCube
- **Sp** Sparklis
- **VISU** VISU

Závěr

Naimplementovali jsme webovou aplikaci, která na vstupu od uživatele dostane schéma a data. Pomocí uživatelova manuálního mapování dat vybraných z vizualizovaného automaticky zanalyzovaného schématu dat s detekcí lokace a metriky tato vybraná data vizualizujeme v heat mapě. Pro to je potřeba, aby uživatel během výběru vizualizace vybral nějaký identifikátor lokace entit. To je buď adresa, GPS souřadnice nebo kód RUIAN. Po tomto výběru k sobě uložíme získaná GPS souřadnice ze samotných dat, z procesu geokódování či jejich extrakce z výsledků spuštěného SPARQL Federated Query. Data následně vizualizujeme v heat mapě po zadání uživatelských filtrů.

Webová aplikace byla také otestována prostřednictvím SUS testování a byla porovnána s podobnými nástroji, které byly zkoumány v nedávno zveřejněné studii prováděné vědeckými pracovníky z Katedry softwarového inženýrství Matematicko-fyzikální fakulty Univerzity Karlovy.

Aplikace by šla do budoucna jistě dále rozšiřovat, dalo by se více zapracovat na lepší uživatelské přívětivosti a integraci dalších funkcí jako by byla historie provedených dotazů, export právě provedeného dotazu, úprava generovaného dotazu pro expertní uživatele, podpora dalších GPS formátů dat nebo umožnění selekce několika identifikátorů lokace během výběru vizualizace.

Seznam použité literatury

Apache Jena Fuseki. Fuseki Administration Server Protocol. URL https://jena.apache.org/documentation/fuseki2/fuseki-server-protocol.html. cit. 2019-07-11.

Berners-Lee, T. (2006). Linked Data. URL https://www.w3.org/DesignIssues/LinkedData.html. cit. 2019-07-11.

Klímek, J., Škoda, P. a Nečaský, M. (2019). Survey of Tools for Linked Data Consumption. *Semantic Web*, **10**(4), 665–720. ISSN 1570-0844.

Mareš, M. a Valla, T. (2017). *Průvodce labyrintem algoritmů*. CZ.NIC, Praha. ISBN 978-80-88168-19-5.

World Wide Web Consorcium (W3C). Ontologies. URL https://www.w3.org/standards/semanticweb/ontology. cit. 2019-07-11.

World Wide Web Consorcium (W3C). SPARQL 1.1 Protocol. URL https://www.w3.org/TR/sparql11-protocol/. cit. 2019-07-11.

World Wide Web Consorcium (W3C). SPARQL Endpoints. URL https://www.w3.org/wiki/SparqlEndpoints. cit. 2019-07-11.

Český úřad zeměměřičský a katastrální. Veřejný dálkový přístup (VDP) – Uživatelská dokumentace. URL https://www.cuzk.cz/VDP-uzivatelska-dokumentace. cit. 2019-07-11.

Seznam obrázků

Seznam tabulek

A. Přílohy

A.1 Zdrojové kódy

První přílohou jsou zdrojové kódy aplikace. Otevřete Visual Studio .sln soubor, který načte všechny projekty. Hlavním (*StartUp*) projektem by měl být nastaven projekt HeatmapVisualizer – naše webová aplikace.

A.2 Testovací data

Testovací data se nachází ve složce „test_data“. Kromě samotných dat tam je k nalezení i README soubor a seznam výběru vizualizací pro jednotlivé testovací datové sady.