Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Využití propojených otevřených dat skrze webová API

Bakalářská práce

Dominik Heller

Školitel: PhDr. Miloš Prokýšek, Ph.D

České Budějovice 2021

Heller D., 2021: Využití propojených otevřených dat skrze webová API [Linked open data utilization via web API. Bc. Thesis, in Czech.] – 50 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Abstract

This bachelor thesis focuses on the topic of *linked open data* utilization by the means of application programming interface of representational state transfer architectural style. The main goal of the thesis is to analyse the current state of the field in regards to semantic web technologies and their relations to more traditional approaches to data publication and extraction. More precisely the proposed goal is to formulate a programming solution which would allow to publish, consume and manipulate the semantic data, otherwise accessible only via SPARQL endpoints, in a very well-known and notably convenient way.

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

České Budějovice, duben 2021.

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce PhDr. Miloši Prokýškovi, Ph.D., který mi svým vstřícným přístupem, odborným vedením a řadou věcných připomínek velice pomohl při jejím zpracování.

Veliké díky bych poté chtěl vyjádřit také své rodině za trpělivost a neutuchající podporu po celou dobu studia.

Obsah

1. Úvod

Tato bakalářská práce se zabývá současným stavem problematiky tzv. propojených otevřených dat a snaží se prozkoumat rozličné způsoby jejich zveřejňování s cílem navrhnout a implementovat univerzální a pro uživatele neznalé sémantických technologií jednoduše využitelné webové API založené na architektuře REST.

Koncept otevřených dat (O*pen Data*), který stojí v jádru této práce, přitom tvoří jeden ze základních stavebních komponent svobodného internetu. Jejich podstatou je v rámci webového prostředí zajistit dostupné, volně šiřitelné, standardizovaně zprostředkované a ideálně též o sémantický rozměr obohacené informační zdroje. Právě poslední zmíněná charakteristika se zakládá na myšlence tzv. sémantického webu, který má přinést vzájemné propojení strukturovaných dat, a v důsledku je tak obohatit o kontextuální rozměr. Poptávka po využívání takto „propojených dat" (*Linked Data*) tudíž již dávno překračuje hranice ryze informačně-technologické sféry a vychází ze všech oblastí každodenního života.

S enormně rostoucím objemem zveřejněných dat však souměrně rostou nároky nejenom na jejich efektivní zpracování a rozlohu datových skladů, ale též na zajištění ústředního principu minimálních omezení a vynaložených nákladů na cestě k jejich aktivnímu využívání. Osvojení si základních konceptů a technologií sémantického webu (RDF, SPARQL, URI ad.) totiž vyžaduje relativně extenzivní průpravu, která pro většinu potencionálních konzumentů (webové vývojáře, natož širokou veřejnost) zůstává složitě dosažitelnou.

Jedním z nástrojů poskytujících velmi vhodný a osvědčený přístupový mechanismus k různorodým datům jsou přitom tzv. webová API (*Application Programming Interface*) založená na architektonickém stylu REST (*Representational State Transfer*). Ta přinášejí osvědčený postup, jak zprostředkovávat data snadno, efektivně a v předdefinovaném strojově čitelném formátu[1]. Jejich aplikace však v převážné většině zůstává na úrovni dat primárně bezkontextových, odpovídajících maximálně třetí z pětihvězdičkové klasifikační stupnice otevřených dat. Tato bakalářská práce se proto blíže zaměřuje na problematiku dostupnosti sémantických propojených dat a na snahu předložit řešení umožňující jejich využívání právě prostřednictvím webového REST API rozhraní.

[1] Nejčastěji formátu JSON (*JavaScript Object Notation*).

1.1. Formulace problému

Jak bylo řečeno výše, tak k extrakci informací z otevřených dat nejvyššího standardu je za potřebí znalosti specifických technologií. Klíčovým je v tomto smyslu hlavně sémantický dotazovací jazyk SPARQL (*SPARQL Protocol and RDF Query Language*), který umožňuje vyhledávání napříč rozličnými formáty spadajícími pod sémantický rámec RDF (*Resource Description Framework*). Ten svou syntaxí zčásti připomíná dotazovací jazyk relačních databází SQL, avšak nutně reflektuje strukturaci RDF dat do tzv. trojic. Navzdory skutečnosti, že SPARQL koncové body (*endpointy*) jsou veřejně dostupné, samotná znalost tohoto komplexního jazyka odporuje v mnohém principu minimálních nákladů ve snaze otevřená data využívat.

Snahou této bakalářské práce tak bude vytvořit pomyslnou nadstavbu umožňující k takovýmto bodům přistupovat skrze obecně dobře známou technologii webových REST API. Tomuto problému se přitom v předešlých letech věnovalo hned několik studií, které představily různorodé postupy, jak dosáhnout vhodného mapování mezi API a SPARQL. Právě těmto architekturám bude věnována pozornost při formulaci finálního návrhu a následné implementaci.

1.2. Cíle práce

Hlavním cílem této bakalářské práce je tedy realizace REST API koncových bodů pro publikování propojených otevřených dat primárně dostupných pouze přes SPARQL endpointy, a to způsobem, který umožní uživatelům jejich využívání bez nutnosti hlubšího povědomí o principech jejich strukturování ani dotazovacího mechanismu.

Hlavního cíle této práce bude přitom dosaženo naplněním následujících čtyř dílčích cílů a úkolů:

1. Identifikace společných rysů a principů týkajících se problematiky propojených otevřených dat a analýza způsobů jejich zprostředkovávání skrze webová REST API.
2. Návrh řešení – na základě provedené rešeršní analýzy – definující vhodný postup publikování propojených otevřených dat způsobem nepožadujícím znalosti sémantických technologií a umožňujícím univerzální nasazení pro datové zdroje splňující předepsané náležitosti.
3. Implementace zvoleného řešení postaveného na bázi webové REST API technologie.

4. Testování produktu a praktická ukázka demonstrující jeho uplatnitelnost na vybrané datové sadě/datových sadách.

1.3. Struktura práce

Struktura práce bude svým rozložením odpovídat snaze o naplnění výše popsaným dílčích cílů. První část práce bude proto primárně teoreticko-syntetického charakteru. Konkrétně se tedy zaměří na definování základních teoretických konceptů spojených s tématem pětihvězdičkových dat a dále hlavně na provedení rešeršní analýzy uplatňovaných postupů při publikování takových dat webovými REST API.

Druhá část práce bude věnována nejprve shrnutí klíčových limitů současných řešení představených v rešeršní části a poté primárně snaze o formulaci takového návrhu, který bude aplikovatelný ve vztahu k datům dostupným přes SPARQL endpointy. Kromě univerzálnosti bude přitom kladen důraz také na uživatelsky přívětivou a intuitivní strukturaci výsledných koncových bodů s cílem minimalizace nároků na znalosti sémantických technologiích u cílového uživatele.

Naplnění prvních dvou dílčích cílů práce poté otevře prostor pro samotnou implementaci zvoleného řešení. Třetí část práce se tudíž zaměří na realizaci formulovaného návrhu, tzv. *proof of concept* (POC), s cílem demonstrovat jeho potenciální praktickou uplatnitelnost (Oxford Learner's Dictionary 2020, Kendig 2015, 237, 250). Součástí této části práce tak bude jak samotná programatická činnost, tak samozřejmě adekvátní dokumentace zdrojového kódu, způsobu nasazení a testování veškeré funkcionality.

2. Teoretická část

V jádru teoretické části stojí koncept tzv. pětihvězdičkových dat, jenž představil poprvé Tim Berners-Lee v roce 2006 (Berners-Lee 2006). V nich zkonkretizoval již dříve formulované myšlenky o postupné evoluci soudobého webového prostředí v propojený systém, kde informace budou obohaceny o jasně definovaný význam. Takovýto systém označil pojmem *sémantický web*[2], v němž bude strojům umožněno "porozumět" vystavovaným datům (Berners-Lee, Hendler a Lassila 2001, 31). Zmíněný vývoj zachycuje přitom nejlépe právě hvězdičková klasifikace otevřených dat. Ta postupně nabyla komplexnější podoby v pěti na sobě navazujících bodech, kdy každý vyšší stupeň musí splňovat podmínky stanovené stupni předcházejícími:

a) ☆ Data dostupná na webu (v jakémkoli formátu) v rámci otevřené licence (*open licence*)[3].

b) ☆☆ Data dostupná na webu ve strojově čitelném formátu[4].

c) ☆☆☆ Data dostupná v neproprietárním formátu[5].

d) ☆☆☆☆ Data zveřejněná za využití otevřených W3C standardů (RDF, SPARQL) umožňujících identifikaci věcí[6], tak aby na ně bylo možné odkazovat.

e) ☆☆☆☆☆ Data splňující výše zmíněné podmínky, a navíc obohacené o odkazy na jiná propojená data (Berners-Lee, Linked Data 2006, Hausenblas a Kim 2012).

Z této klasifikace vyplývá jako nejvýznamnější a zároveň nejnáročnější krok pozvednutí otevřených dat z 3 na 4., respektive 5. úroveň, a tedy vytvoření tzv. „propojených otevřených dat" neboli *Linked Open Data* (LOD)[7] (Berners-Lee 2006). S tímto posunem se poté pojí pojmy jako URI, RDF a SPARQL, které ruku v ruce tvoří základ sémantického webu. Právě těmto pojmům bude v teoretické části závěrečné práce věnována výraznější pozornost.

[2] „I have a dream for the Web […] Machines become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web", which makes this possible, has yet to emerge, but when it does, […] our daily lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize." (Berners-Lee 1999, 157–158).
[3] Tím se z nich stávají data otevřená (*Open Data*).
[4] Nikoli tedy například jako naskenovaná fotografie, ale ideálně jako textový či tabulkový dokument.
[5] Tudíž např. v *CSV*, nikoli *Microsoft Excel*.
[6] Za využití tzv. URI (*Uniform Resource Identifier*).
[7] „Linked *Open* Data (LOD) is Linked Data which is released under an open licence, which does not impede its reuse for free" (Berners-Lee 2006).

2.1. Resource Description Framework

Resource Description Framework neboli RDF je, jak volný překlad názvu napovídá, obecným rámcem pro popis zdrojů[8]. Jeho cílem je umožnit modelování informací o zdrojovém dokumentu způsobem, který je jak strojově, tak i lidský čitelný (Labra-Gayo, a další 2019, 121). RDF byl poprvé navržen v roce 1997 a o dva roky později se stal součástí doporučení od *World Wide Web Consortium* (W3C) (Lassila a Swick 1999). Syntaxe RDF „jazyka" byla založena na formátu *Extensible Markup Language* (XML) a její datový model na bázi teorie grafů. Vliv XML standardu poté vedl k formulaci tzv. *RDF Schema* (RDFS), které, spíše než k pouhé validaci RDF dat, má sloužit jako nástroj k uchopení a prohlubování znalostí o informacích v datech obsažených (Labra-Gayo, a další 2019, 121). Může tak např. definovat pro jednotlivé „RDF objekty" přípustné datové typy či popisovat jejich vzájemné vazby, a tím vkládat do webového obsahu příslušnou sémantiku.

V principu tak RDFS staví na základní RDF syntaxi, přebírajíce ústřední pojmy jako *rdf:resource, rdf:type* či *rdf:property*, a vytváří základní „RDF slovník", který poskytuje klíčové konstrukce pro popis zdrojů. Základní dvě skupiny konstrukcí přitom tvoří třídy (*classes*) a vlastnosti (*properties*). Všezahrnující třídou zdrojů je poté *rdfs:Resource*, která přímo vychází z *rdfs:Class*, jež umožňuje tvorbu definic pro skupiny zdrojů. Za zmínku poté stojí i třídy *rdfs:Literal* či *rdfs:Datatype*. Ve vztahu k vlastnostem jsou poté stěžejní *rdfs:type, rdfs:subClassOf, rdfs:label, rdfs:domain, rdfs:range* a *rdfs:comment* (Brickley a Guha 2003). Završením tohoto procesu budování základní sémantické platformy se nakonec stal i samostatný ontologický jazyk, tzv. *Web Ontology Language* (OWL), uveřejněný v roce 2004 v doporučení W3C[9] (McGuinness a Harmelen 2004).

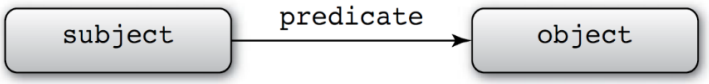
Obecným cílem ontologie je přitom poskytnout formální cesty, jak popisovat taxonomii a klasifikaci sítí, a tedy definovat strukturu vědomostí pro různé domény. Zjednodušeně řečeno, tedy umožnuje formulovat pojmové slovníky relevantní pro příslušné domény, tvořené podstatnými jmény a slovesy reprezentujícími třídy objektů, respektive vztahy mezi nimi (Vahidalizadehdizaj, a další 2015, 1–2, McGuinness a Harmelen 2004, Sikos 2015, 6–7). OWL má být v tomto smyslu tedy základním stavebním kamenem pro formulaci

[8] „In the SemanticWeb we refer to the things in the world as resources; a resource can be anything that someone might want to talk about. Shakespeare, Stratford, "the value of X," and "all the cows in Texas" are all examples of things someone might talk about and that can be resources in the SemanticWeb" (Allemang, Hendler a Gandon 2020, 37).

[9] Kolekci všech technologií doporučených a podporovaných W3C, někdy také označovaných jako *Semantic Web Cake* či *Semantic Web Stack*, si lze pohlédnout v příloze 1.

vlastních/odvozených ontologií a konceptuálních nástrojů, jak uchopit a popsat doménově-specifická data univerzálně interpretovatelným způsobem (Labra-Gayo, a další 2019, 121).

Tímto se lze plynule přesunout k otázce samotné struktury RDF dat. Ta vychází z principů vžitých při běžné komunikaci v rámci přirozeného jazyka. K vyjádření sémantické informace o zdroji dat totiž využíváme tzv. RDF tvrzení (*RDF statement*) formulovaných do podoby tzv. trojic (*triples*) ve tvaru *subjekt-predikát-objekt*[10] (Allemang, Hendler a Gandon 2020, 41). Názornou představu poskytuje obrázek 1.



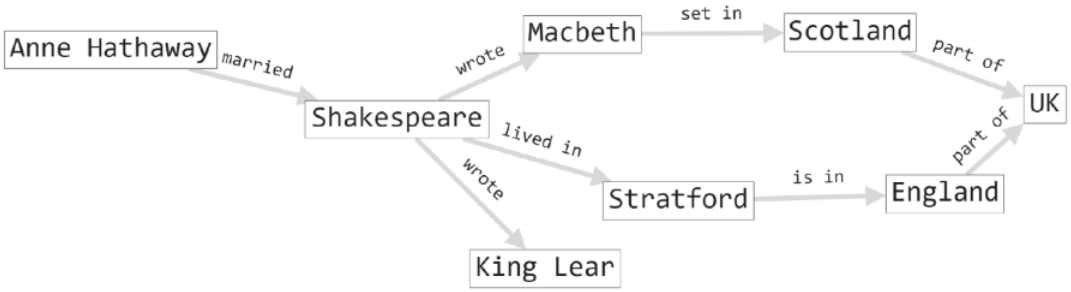
Obrázek 1: Názorná ukázka RDF trojice (Labra-Gayo, a další 2019, 10).

RDF tvrzení nám tedy říká, že existuje vztah, identifikovaný prostřednictvím predikátu, mezi dvěma zdroji, subjektem a objektem (Labra-Gayo, a další 2019, 10). Data jsou přitom typicky[11] reprezentována v tabulárním formátu, v jehož rámci každá řádka určuje objekt, který popisujeme, každý sloupec vlastnost daného objektu a každá buňka samotnou hodnotu příslušné vlastnosti. V případě takových dat však s rostoucím objemem vyvstává klíčová otázka týkající se jejich distribuce na vícero strojů[12]. Hovoříme-li navíc přímo o internetových datech, kde se každý kus informace ze své podstaty může vyskytovat na nespočetném množství míst, stává se tato otázka obzvláště kritickou. Aby kýženého výsledku bylo dosaženo, je každá „buňka“ reprezentována kombinací globální reference řádky (= objektu), globální reference sloupce (= predikátu) a globální reference samotného obsahu buňky (= objektu). Tím tak můžeme docílit unikátní identifikace jednotlivých informací napříč internetem (Allemang, Hendler a Gandon 2020, 42). Tabulární znázornění RDF trojic a následně též jejich grafový ekvivalent můžeme vidět na ukázce v obrázku 2 a 3.

Subject	Predicate	Object
Shakespeare	wrote	King Lear
Shakespeare	wrote	Macbeth
Anne Hathaway	married	Shakespeare
Shakespeare	livedIn	Stratford
Stratford	isIn	England
Macbeth	setIn	Scotland
England	partOf	UK
Scotland	partOf	UK

Obrázek 2: Tabulární reprezentace RDF trojic (Allemang, Hendler a Gandon 2020, 41).

[10] Můžeme se setkat i s odlišným pojmoslovím. Např. subjekt-vlastnost-objekt, subjekt-vlastnost-hodnota atp.
[11] Minimálně v době formulování základů RDF tomu tak bylo.
[12] Tu může v případě dat reprezentovaných v relačních databázích řešit např. tzv. horizontální (dělení řádků) či vertikální (dělení sloupců) *partitioning*.



Obrázek 3: Grafová reprezentace RDF trojic (Allemang, Hendler a Gandon 2020, 42).

2.2. Uniform Resource Identifiers

Zmíněnou globální referenci jednotlivých zdrojů přitom označujeme pojmem *Uniform Resource Identifiers* (URIs)[13] neboli v překladu „jednotnými identifikátory zdroje“. URI může být tedy přiřazeno objektu, subjektu i predikátu a může nabývat v zásadě dvou základních formátů. Prvními jsou *Uniform Resource Locators* (URLs), které jsou velmi dobře známé z webového prostředí k identifikaci způsobu, jak se ke zdroji dostat (např. *http://example.com*). Druhým typem jsou poté Uniform Resource Names (URNs), které specifikují jedinečný název zdroje jako takového[14] (např. *urn:isbn:1234567890*) (Curé a Blin 2014, 44). Zcela zásadním je tedy skutečnost, že jakékoli dva zdroje na internetu mohou odkazovat zcela jednoznačně na jakýkoli jiný zdroj právě prostřednictvím URI.

Zdroje však mohou být vyjádřeny i jinými způsoby, a to konkrétně tzv. literálem a poté prázdným uzlem (*blank node*). Zatímco literál vyjadřuje víceméně pouze konkrétní hodnotu určitého zdroje (např. *"23"^^xsd:integer*), a proto je vždy konečným uzlem RDF grafu, neboť na něj nelze dále odkazovat. Prázdný uzel na straně druhé umožňuje zapracovat do schématu i zdroj, který žádný identifikátor nemá, respektive jehož identifikátor nám není znám. Umožňuje nám tak uchovat informaci o tom, že „něco“ s příslušným vztahem k jinému zdroji existuje, bez nutnosti to něco explicitně identifikovat (Labra-Gayo, a další 2019, 10–11).

Pro usnadnění zápisu RDF tvrzení skrze dlouhá URI bylo poté využito vlastnosti XML dokumentů sdružovat pojmy do tzv. jmenných prostorů (*namespaces*) a jejich následné zkratkovité reprezentaci díky *prefixům*. Tím bylo na jedné straně dosaženo možnosti formulovat jednotlivá tvrzení do větších sémanticky příbuzných celků, tak také výrazné

[13] URI bývá často nahrazováno za IRI (*Internationalized Resource Identifier*), který rozšiřuje jeho definici i o složitější charakterové sady.
[14] Typicky věcí bez přirozené asociace s webem (např. knihu, strom, či pokrm).

úspory jejich zápisu[15] (Sikos 2015, 14–15). Výsledkem je poté tzv. „kompaktní URI“ neboli *CURIE*, tvořené dvěma částmi oddělenými dvojtečkou – prefixem reprezentujícím příslušný jmenný prostor a samotným identifikátorem (Allemang, Hendler a Gandon 2020, 46–47).

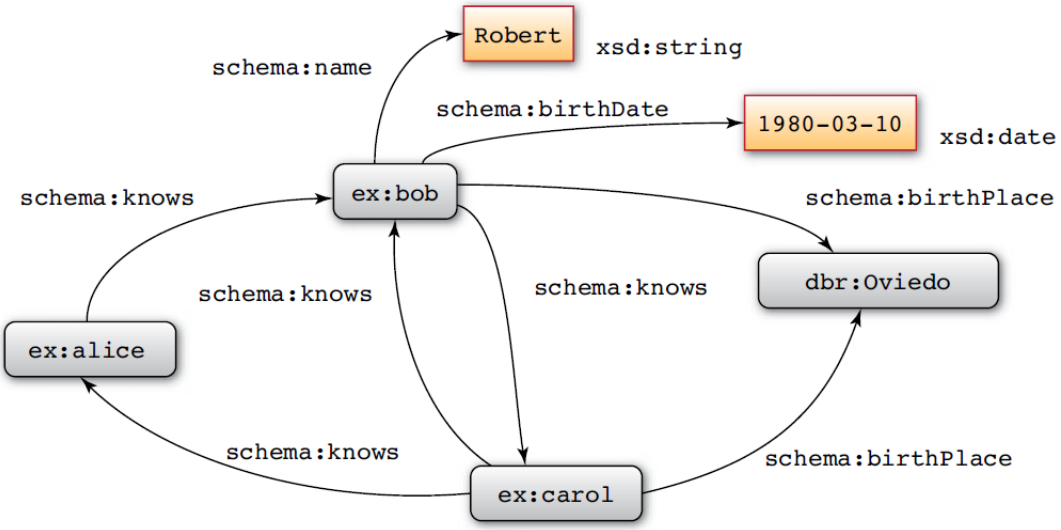
Nezbytné je však poznamenat, že v průběhu doby se pro zápis RDF dat vytvořilo hned několik textových formátů. Původně jednomyslně doporučovaný a standardizovaný *RDF/XML* (.rdf), který, jak je zjevné, byl založen ryze na XML syntaxi, tak postupně doplnilo hned několik dalších (Gandon a Schreiber 2014). Mezi nejznámější patří např. *N-Triples* (.nt), který staví na naprosté jednoduchosti zápisu, kdy každá trojice je tvořena právě a pouze třemi plně definovanými referencemi. Tento formát tak vůbec nepracuje s prefixy zmíněnými výše a je vhodný v situacích, kdy se s každou trojicí pracujeme odděleně (Carothers a Seaborne 2014). Další formát, o kterém je nezbytné se zmínit, se nazývá *Turtle* (.ttl). Ten naopak vychází více vstříc čitelnosti pohledem lidského oka, neboť kondenzuje zápis jak užíváním prefixů, tak sdružováním predikátů a subjektů propojených s konkrétním objektem v souvislé celky (Prud'hommeaux a Carothers 2014). Poslední, zato velmi významný formát, který bude ještě představen, označujeme názvem *JSON-LD* (.jsonld). Ten je na rozdíl od RDF/XML plně kompatibilní s univerzálním, strukturovaným formátem JSON a poskytuje tak stejné výhody ve vztahu k transportu, stejně jako serializaci do podoby datových objektů, tak následně opětovné deserializaci. JSON-LD navíc doplňuje syntaxi o další prvky mimo užívání prefixů, a to konkrétně přidáváním speciálních sémantických atributů[16] (Kellog, Champin a Longley 2020).

Kromě výše zmíněných poté existuje i celá řada dalších, jako např. *N-Quads, Notation3* či *RDFa*, jejichž představení však bohužel přesahuje rámec této práce. Jako stěžejní je však třeba na závěr zdůraznit, že všechny zmíněné formáty jsou ve valné většině vzájemně zcela kompatibilní a možnost konverze z jednoho do jiného musí být v každém případě zajištěna. Při znalosti výše popsaného způsobu tvorby RDF dat je tak nyní možno si prohlédnout, v rámci obrázků 4 a 5, názorný příklad nejprve textového zápisu – konkrétně ve formátu Turtle – tak též jeho následné grafové reprezentace.

[15] Příklady nejužívanějších prefixů, stejně jako jmenných prostorů obsahujících základní ontologii vybraných oblastí nabízí příloha 2.
[16] Pro konkrétní představu o odlišnostech mezi jednotlivými strukturovanými RDF formáty lze nahlédnout do přílohy 3 této práce.

```
prefix ex:     <http://example.org/>
prefix schema: <http://schema.org/>
prefix dbr:    <http://dbpedia.org/resource/>
prefix xsd:    <http://www.w3.org/2001/XMLSchema#>

ex:alice schema:knows      ex:bob .

ex:bob   schema:knows      ex:carol .
ex:bob   schema:name       "Robert" .
ex:bob   schema:birthDate  "1980-03-10"^^xsd:date .
ex:bob   schema:birthPlace dbr:Oviedo .

ex:carol schema:knows      ex:alice .
ex:carol schema:knows      ex:bob .
ex:carol schema:birthPlace dbr:Oviedo .
```

Obrázek 4: Příklad zápisu RDF ve formátu Turtle (Labra-Gayo a kol. 2019:11–12).



Obrázek 5: Ekvivalentní grafová reprezentace (Labra-Gayo a kol. 2019:12).

Z ukázky tak lze mj. odvodit např., že zdroj identifikovaný skrze CURIE *ex:bob* má vazbu definovanou spojením *schema:knows* na zdroj s CURIE *ex:carol*. Konkrétní význam jednotlivým částem poté dále doplňuje přidružená ontologie specifikovaná pod jednotlivými jmennými prostory. Z příkladu tak lze vidět, jak praktické uplatnění zjednodušeného zápisu prostřednictvím CURIE, tak hlavně logickou strukturaci dat formující komplexní síť plně zachovávající sémantickou informaci.

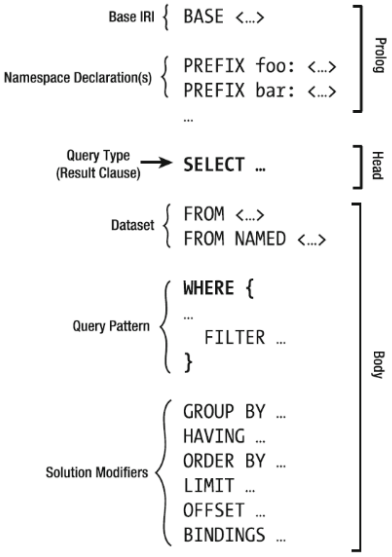
2.3. SPARQL Protocol and RDF Query Language

Jak zachyceno v předchozích subkapitolách, tak RDF poskytuje jednoduchý způsob reprezentace přirozeně distribuovaných dat. K tomu dopomáhá primárně systém sémantických trojic umožňujících zachytit a pojmenovat propojení mezi dvěma „předměty

zájmu“. Reprezentace takové informace však sama o sobě zůstává vcelku bezcenná, aniž by byla doplněna o prostředek, jakým lze k daným datům přistupovat (Allemang, Hendler a Gandon 2020, 119).

Z toho důvodu vzniklo – na základě výsledků činnosti speciálně utvořené pracovní skupiny DAWG (*RDF Data Access Working Group*) pod záštitou W3C – doporučení z roku 2008 deklarující základy SPARQL jakožto klíčové technologie sémantického webu (Prud'hommeaux a Seaborne 2008). SPARQL tedy představuje sémantický dotazovací jazyk, podobné syntaxe jazyku relačních databází SQL (*Structured Query Language*), pro data uchovávaná ve formátu RDF.

SPARQL lze přitom s ohledem na historii jeho vývoje rozlišit do dvou základních syntaktických verzí. *SPARQL 1.0* vychází právě ze standardu z roku 2008 a orientuje se primárně na vyhledávací mechanismy. Konkrétně definuje 4 příkazy: SELECT, CONSTRUCT, ASK a DESCRIBE. Příkaz SELECT extrahuje takové záznamy, které odpovídají specifikovaným požadavkům a výstup utváří v podobě tabulky. CONSTRUCT vrací obdobné výsledky jako předchozí příkaz, avšak nikoli ve formě tabulky, ale jediného RDF grafu[17]. Aplikace ASK dotazu poté vrací hodnotu pravda/nepravda (*true/false*) v závislosti na tom, zda se hledané řešení v úložišti nachází. Posledním příkazem je poté příkaz DESCRIBE, jehož výstupem je jediný RDF graf, který obsahuje RDF data popisující specifikovaný zdroj[18] (Prud'hommeaux a Seaborne, SPARQL Query Language for RDF 2008). Všechny tyto dotazy se přitom vyznačují velice podobnou syntaxí, kterou shrnuje obrázek 6.



Obrázek 6: Struktura SPARQL dotazu (Sikos 2015, 174).

[17] Ten může být vyjádřen některým z výše popsaných RDF formátů.
[18] Pro bližší seznámení se s jednotlivými příkazy viz W3C doporučení (Prud'hommeaux a Seaborne 2008).

Jak je zjevné, tak jednotlivé URI jsou zapouzdřeny v závorkách typu < > a samotná podmínka naopak v závorkách složených *{ }*. Byť dodržení přesné syntaxe je pro správnost dotazu klíčové, ne každé *query* musí nutně obsahovat všechny zmíněné prvky. Např. deklarace prefixu může být za předpokladu nevyužívání CURIE úplně vynechána, stejně tak výběr datové množiny, pokud cílem vyhledávání je procházet výchozí, nikoli konkrétní jmenný graf[19], a poté samozřejmě lze vynechat i sekci modifikátorů (Prud'hommeaux a Seaborne 2008).

Druhou verzí rozšiřující primárně vyhledávací charakter dosavadních dotazů se stalo *SPARQL 1.1* z roku 2013. To připravilo standardizaci ryze aktualizačních dotazů, jejichž množství je daleko větší než v předchozím případě. Za ty nejvýznamnější však můžeme označit příkazy: INSERT DATA, DELETE DATA, INSERT, DELETE, CLEAR a LOAD (Harris a Seaborne 2013). INSERT příkazy umožňují vkládání příslušných trojic do zvoleného RDF grafu, přičemž varianta s přídavkem „DATA“ vynechává klauzuli *where* a povoluje tak vkládat pouze konkrétní hodnoty. Podobná logika platí pro DELETE příkazy, jen s tím rozdílem, že provádí odstranění příslušného vzoru z datasetu. CLEAR poté provede kompletní vymazání všech záznamů ze zvoleného grafu, a LOAD je naopak umožní načíst do grafu zcela nového (Gearson, Passant a Polleres 2013).

SPARQL 1.1 dále také doplňuje nástroje pro správu celých grafových soustav, kdy nabízí funkci jednotlivé grafy/skupiny grafů kopírovat (COPY), přesouvat (MOVE), vytvářet (CREATE), mazat (DROP) či přidávat (ADD). Rozšíření se poté dočkává i sekce modifikátorů, kdy přibývají např. konstrukce jako HAVING, filtrující data, na rozdíl od FILTER, pouze nad očekávaným výstupem, VALUES, dovolující deklarovat několik proměnných s přiřazenou hodnotou/hodnotami, či v neposlední řadě dokonce možnosti formulovat vnořené SELECT dotazy (Gearson, Passant a Polleres 2013). Zmíněná specifikace je navíc kontinuálně doplňována a dotváří tak konceptuální rámec SPARQL jakožto skutečně plnohodnotného dotazovacího jazyka.

[19] Pro dovysvětlení konceptu datových sad je třeba zmínit, že každý nově přidaný záznam (RDF tvrzení/trojice) je, pokud nestanoveno jinak, zařazen do tzv. výchozího grafu (*default graph*). Příbuzné trojice lze poté sdružovat do tzv. jmenných grafů (*named graph*). Tím je možné RDF data dále strukturovat a usnadnit tak vyhledávání pouze napříč žádanými datasety (Hawke, a další 2012).

3. Praktická část

V předchozím teoretickém shrnutí byly představeny klíčové aspekty propojených otevřených dat, které budou v následných kapitolách využity při formulaci návrhu a implementaci řešení v souladu s cíli této práce. Ty přitom spočívají ve snaze vytvořit, pokud možno maximálně univerzální, nástroj pro mapování SPARQL dotazů na REST API koncové body. K dosažení tohoto cíle je však primárně nezbytné prozkoumání již existujících postupů, a to z důvodu, že dané problematice se v minulosti věnovalo nemalé množství vědeckých prací, stejně jako různorodých veřejných i soukromých projektů. V následných podkapitolách tedy bude provedena průřezová rešerše s cílem vybudovat dostatečný základ pro formulaci vlastního návrhu.

3.1. Metodika rešeršní činnosti

Ve vztahu k vyhledávání zdrojů týkajících se sledovaného předmětu zájmu bude využito tří v akademickém prostředí dobře známých databázových platforem dostupných pod licencí *Jihočeské univerzity v Českých Budějovicích*. Konkrétně se bude jednat o služby *EBSCO Information Services*[20], spadající pod americkou společnost *EBSCO Industries Inc.*, dále *Web of Science Core Collection*[21], současně spravovanou společností *Clarivate Analytics*, a v neposlední řadě databázi *Scopus*[22] od společnosti *Elsevier*. Vyhledávací kritéria budou napříč všemi úložišti stejná a budou kombinací definice klíčových slov, omezení oblasti výzkumu a roku zveřejnění.

- Klíčová slova: „sparql“, „rdf“, „api“, „linked open data“.
- Oblast výzkumu: „počítačová věda“ (*computer science*).
- Rok zveřejnění: 2008 a novější[23].

Striktním dodržením těchto kritérií byl dosažen výstup čítající v součtu desítky odborných článků a příspěvků primárně v periodických časopisech. Ty se poté staly základem jednak pro čerpání nezbytných informací, jednak také prostředkem pro odhalení sekundárních referencí na významné zdroje jinde veřejně dostupné.

[20] Dostupné na: https://search.ebscohost.com
[21] Dostupné na: https://apps.webofknowledge.com
[22] Dostupné na: https://www.scopus.com
[23] Výběr tohoto roku je dán vydáním prvního doporučení W3C souvisejícího s jazykem SPARQL.

3.2. Analýza stávajících postupů

Na základě rešeršní činnosti a důkladného prostudování vyhledaných zdrojů lze nalezená řešení strukturovat do tří základních kategorií:

a) *Ad hoc* řešení profilovaná pro konkrétní ontologie, společnost, příp. datovou sadu, vyznačující se primárně pevnou vazbou mezi definovanými REST API endpointy a fixně formulovanými SPARQL dotazy, mnohdy též vyžadující relativně rozsáhlou znalost sémantických technologií.

b) Komplexní více či méně dynamická řešení a nástroje automaticky generující celé REST API soustavy s ohledem na specifikaci příslušných konfiguračních požadavků – např. projekt *BASIL* či *RAMOSE*.

c) Zjednodušené dynamicky utvářené REST API endpointy extrahující informaci o vyhledávaném zdroji přímo ze samotného URL – buď z příslušné URL cesty či z *query stringu* (části za otazníkem) – které jsou následně implementovány do předpřipravovaných, ideálně univerzálně strukturovaných, SPARQL dotazů.

Jako jedni z prvních se přitom tomuto tématu věnovali autoři Benson a Battle (2008, passim), kteří představili základní vzor, který se pro publikovaní RDF v oblasti REST API stal víceméně standardem. Vyšli totiž ze základní ontologie RDFS a připravili koncové body založené na uchopení propojených dat buďto jakožto samostatných zdrojů (*rdfs:Resource*), či skupin zdrojů neboli tříd (*rdfs:Class*). Tím umožnili adresovat vlastně jakýkoli uzel pomyslného RDF grafu, respektive jakoukoli „instanci“ zdroje či třídy, vycházející z daného slovníku skrze definovaní cesty ve tvaru */api/resource/{id}* či */api/class/{id}*[24], a provádět na nich kýžené operace. Ty zakotvili v myšlence mapování mezi *Create, Read, Update, Delete* (CRUD) operacemi, reprezentovanými v kontextu REST architektury HTTP metodami POST, GET, PUT a DELETE, na SPARQL dotazy typu INSERT, SELECT/CONSTRUCT, MODIFY a DELETE[25] (Battle a Benson 2008, 65–66). Viz obrázek 7 níže.

[24] Alternativně: */api/resource?id=…* či */api/class?id=…*.

[25] Nutné je poznamenat, že metody DELETE či MODIFY nebyly v té době ještě standardizovány W3C konsorciem. Tyto užité metody vycházely z metodiky představené týmem *Jena*, pracujícím pro společnost *Hewlett-Packard*, která se až posléze stala jedním z podnětů pro SPARQL 1.1 (Seaborne a Manjunath 2007, passim).

Class-level endpoints of semantic REST

Operation	HTTP command	Request data format	SPARQL command	Response
List	GET	None	n/a	SPARQL Select result of all resources of the type specified by the URL
Query	GET	SPARQL	SELECT or CONSTRUCT	Query Results
Create	POST	None	n/a	Status 201 CREATED
Insert	PUT	SPARQL/Update	INSERT	Status 200 OK
Remove	DELETE	SPARQL/Update	DELETE or MODIFY	Status 200 OK

Resource-level endpoints of semantic REST

Operation	HTTP command	Request data format	SPARQL command	Response
Read	GET	Empty	n/a	RDF/XML
Insert	PUT	SPARQL/Update	INSERT	Status 200 OK
Remove	DELETE	SPARQL/Update	DELETE or MODIFY	Status 200 OK

Obrázek 7: Základní princip mapování HTTP metod na SPARQL (Battle a Benson 2008, 66).

Tuto metodiku označili příhodně pojmem „sémantický REST“ (*Semantic Rest*) a v pracích dalších autorů je principálně velmi často následována. Dále tak je možné uvést např. práci autorů Wilde a Hausenblas (2009, passim), kteří v článku *RESTful SPARQL? You name it!* navázali na své kolegy a dále obohatili jejich mapovací mechanismus. Jejich řešení nazvané *SPARQL over HTTP* totiž již neumožňovalo získat, přidat, měnit, či mazat pouze konkrétní zdroje či třídy, ale přímo posílat ke zpracování celé SPARQL dotazy. Pro SELECT tak byla využita metoda GET a příslušný SPARQL dotaz byl přenášen v rámci URL parametru *„?query=“*. U aktualizačních metod poté byly využity metody PUT či DELETE a celý příkaz byl zapouzdřen v těle samotné zprávy[26] (Wilde a Hausenblas 2009, 39–42).

```
SELECT ?who ?whom
 FROM NAMED
 <http://localhost:8083/sparestfulql/default/erik>
 {
  ?who <http://xmlns.com/foaf/0.1/knows> ?whom .
 }

GET http://localhost:8083/sparestfulql/default/erik?
query=SELECT+%3fwho+%3fwhom+FROM+NAMED+%3c
http%3a%2f%2flocalhost%3a8083%2fsparestfulql%2f
default%2ferik%3e+%7b%0a+%3fwho+
%3chttp%3a%2f%2fxmlns.com%2f
foaf%2f0.1%2fknows%3e+%3fwhom+.
```

```
INSERT INTO
 <http://localhost:8083/sparestfulql/default/erik> {
  <http://http://dret.net/netdret/foaf.rdf#me>
  <http://xmlns.com/foaf/0.1/knows>
  <http://sw-app.org/mic.xhtml#i> .
}

PUT http://localhost:8083/sparestfulql/default/erik

 <http://http://dret.net/netdret/foaf.rdf#me>
 <http://xmlns.com/foaf/0.1/knows>
 <http://sw-app.org/mic.xhtml#i> .
```

Obrázek 8: Způsob přenášení SPARQL dotazů skrze HTTP metody (Wilde a Hausenblas 2009, 42).

Nezbytné je poznamenat, že zmíněný postup se stal základem právě pro fungování samotných SPARQL endpointů. Ty představují veřejně přístupné body pro dotazování se nad propojenými otevřenými daty právě prostřednictvím dotazovacího jazyka SPARQL a za využití primárně metod GET/POST HTTP protokolu. Pravděpodobně nejvýznamnější rodinu těchto zdrojů přitom tvoří tzv. *The Linked Open Data Cloud*, který sdružuje na stovky datových sad z různorodých oblastí splňujících několik základních kritérií. Mezi taková

[26] Samotná problematika mapování CRUD operací mezi HTTP metodami a SPARQL se poté stala předmětem zájmu celé řady dalších studií – viz např. Garrote a Garcia (2011, passim).

kritéria patří např. dostupnost služby přes HTTP protokol, velikost databáze minimálně 1000 RDF trojic, včetně alespoň 50 dohledatelných odkazů na jiné datasety, a v neposlední řadě poskytující přístup k datům mj. skrze dedikovaný SPARQL endpoint[27] (Insight Centre for Data Analytics 2021). Součástí tohoto souboru jsou poté např. světoznámé databáze jako *DBpedia*[28], *Wikidata*[29], *LinkedGeoData*[30], *EU Open Data Portal*[31] či soustava portálů *data.gov*[32] sdružující otevřeně dostupné dokumenty týkající se veřejné správy jednotlivých demokratických vlád v Evropě i ve světě.

Tato metodika však znovu vyžadovala na straně „API konzumentů" komplexní znalost SPARQL syntaxe, strukturace RDF trojic atp., a tudíž stále neodstraňovala základní překážky bránící jejímu využívání i mezi širší komunitou webových vývojářů. Myšlenku oddělit znalosti RDF technologií a REST architektury se proto pokusili rozpracovat Hopkinson, Maude a Rospocher (2014, passim) v příspěvku *A Simple API to the KnowledgeStore*. V něm představili projekt napsaný v programovacím jazyce Python, který umožňoval definovat vlastní API endpointy a na ně navázané SPARQL dotazy. Ty byly buď fixně definované, anebo umožňovaly dodatečnou parametrizaci skrze *query string*. Výsledné URL tak mělo např. takovýto formát: */api/actors_of_a_type?uris.0=dbo:Person&filter=david*, který byl následně přeložen v odpovídající SPARQL dotaz selektující herce podle zvolených kritérií – tedy definovaného URI, přidružení k typu *dbo:Person* a mající ve jméně řetězec *david* (Hopkinson, Maude a Rospocher 2014, 9). Na jedné straně tak toto řešení přineslo pro potenciální uživatele nově také možnost specifikovat kritéria vyhledávání, na straně druhé však trpělo zřejmým omezením co do počtu samotných endpointů[33]. Aplikace sice umožňovala „doprogramovat si" další vlastní mapování, ale obecně neexistovala prakticky žádná pravidla specifikující jednotný postup jejich vytváření ani zveřejňování.

Tuto skutečnost se proto pokusili napravit autoři přístupu nazvaného BASIL (*Building Apis SImpLy*). Ten byl poprvé představen v roce 2015 pod záštitou jedné z největších institucí orientovaných na distanční vzdělávání na světě *Open Univesity*. Basil je přitom založen na myšlence automatizovaného vytváření REST API nad uživatelem definovaným SPARQL

[27] Současný přehled datových sad splňujících daná kritéria je k nahlédnutí v příloze 4, příp. v interaktivní podobě na: https://lod-cloud.net/.
[28] Dostupné na: https://dbpedia.org/sparql.
[29] Dostupné na: https://query.wikidata.org/.
[30] Dostupné na: http://linkedgeodata.org/sparql.
[31] Dostupné na: https://data.europa.eu/euodp/en/linked-data.
[32] Tuzemský *Portál otevřených dat* je dostupný na https://data.gov.cz/, přidružený sparql endpoint poté na: https://data.gov.cz/sparql.
[33] Celkově poskytovalo ve výchozí verzi ani ne 20 přístupových bodů. Např: */api/property_of_actors_of_a_*type, */api/types_of_actors*, */api/people_sharing_event_with_a_person* atp. (Hopkinson, Maude a Rospocher 2014, 8).

endpointem, respektive výstupem SPARQL dotazu příslušným endpointem generovaným. Celý mechanismus přitom stojí na zaslání požadavku metodou PUT na adresu *http://basil.kmi.open.ac.uk/basil?endpoint=* , kde je na místo jediného parametru doplněn cílový SPARQL endpoint, nad kterým má být dotaz proveden, a v těle zprávy je obsažen kýžený SPARQL dotaz (Daga, Panziera a Pedrinaci 2015, 26–28). Jako výstup tohoto požadavku je nástrojem BASIL vygenerována sada koncových bodů, dostupných pod náhodně vygenerovaným řetězcem připojeným k výchozí adrese – např. *http://basil.kmi.open.ac.uk/basil/x68shwt3Qw*. Výsledný stav si můžeme prohlédnout níže.

```
/basil/x68shwt3Qw → base resource, redirects to /spec
/basil/x68shwt3Qw/api → to retrieve the data
/basil/x68shwt3Qw/spec → to get and update the stored query
/basil/x68shwt3Qw/explain → to inspect the query after variables substitution
/basil/x68shwt3Qw/view → to manage views
/basil/x68shwt3Qw/api-docs → to access the Swagger description
```

Obrázek 9: Přehled vygenerovaných koncových bodů nástrojem BASIL (Daga, Panziera a Pedrinaci 2015, 27).

Pod jednotlivými vygenerovanými body se tedy skrývají základní možnosti práce se získaným výsledkem, příp. možnost vypsat či dodatečně upravit výchozí SPARQL dotaz atp. Pro uživatelský komfort je poté celé řešení, naprogramované v jazyce Java, doplněno jak o automatizovanou tvorbu dokumentace díky integraci frameworku *Swagger*, tak také o možnost vyjednávat o návratovém formátu. Vedle RDF/XML, N3, příp. Turtle, přitom nabízí pro konzumenty neznalé konceptům propojených otevřených dat i formáty jako XML, JSON či CSV a činí tak integraci výstupů v koncových aplikací zcela nezávislou na znalosti sémantických technologií[34] (Daga, Panziera a Pedrinaci 2015, 29).

O propojení předchozích řešení s cílem nejenom oprostit uživatele koncového API od znalosti jazyka SPARQL, ale souběžně také zachovat možnosti dynamicky se dotazovat na cílenou databázi se pokouší např. projekt německé *Universität Leipzig*. Ten cílí na vybudování konkrétního řešení pro znalostní databázi *DBpedia* a její koncový SPARQL endpoint. Samotný mapovací mechanismus poté znovu staví na ústředních RDFS/OWL konceptech *owl:Class* a *rdfs:Resource* a pouze je dále přizpůsobuje zvolenému úložišti. Výstupem této aplikace, vytvořené v programovacím jazyce *Java* za využití frameworku *Spring* a *Jena*, jsou

[34] Velmi obdobně poté pracuje nástroj *grlc* vytvořený zástupci *Vrije Universiteit Amsterdam*, který buduje příslušná API na základě SPARQL dotazů uložených ve zvoleném GitHub úložišti (Penuela a Hoekstra 2016, passim).

poté tři výchozí API koncové body (Universität Leipzig 2018). Jejich podoba vypadá následovně:

- */api/{api_version}/entities?value=[value],[optional_parameters]*
- */api/{api_version}/values?entities=[entity],[optional_parameters]*
- */api/{api_version}/instances/{classname}?[optional_parameters]*

První z nich vybírá příslušný DBpedia zdroj (entitu), příp. výčet několika zdrojů, pokud obsahují definovanou entitu jakožto hodnotu některé své vlastnosti (*property*). Druhý volí obdobný postup akorát s tím, že obsahem vlastnosti není entita, ale konkrétní hodnota (literál). Poslední bod poté vypisuje všechny instance příslušné třídy. Na rozdíl od přechozích řešení však dále nabízí také poměrně širokou sadu doplňujících parametrů umožňujících filtrovat, omezovat či formátovat získaný výsledek (Universität Leipzig 2018). Konkrétní příklad konverze URL do SPARQL dotazu zobrazuje obrázek 10.

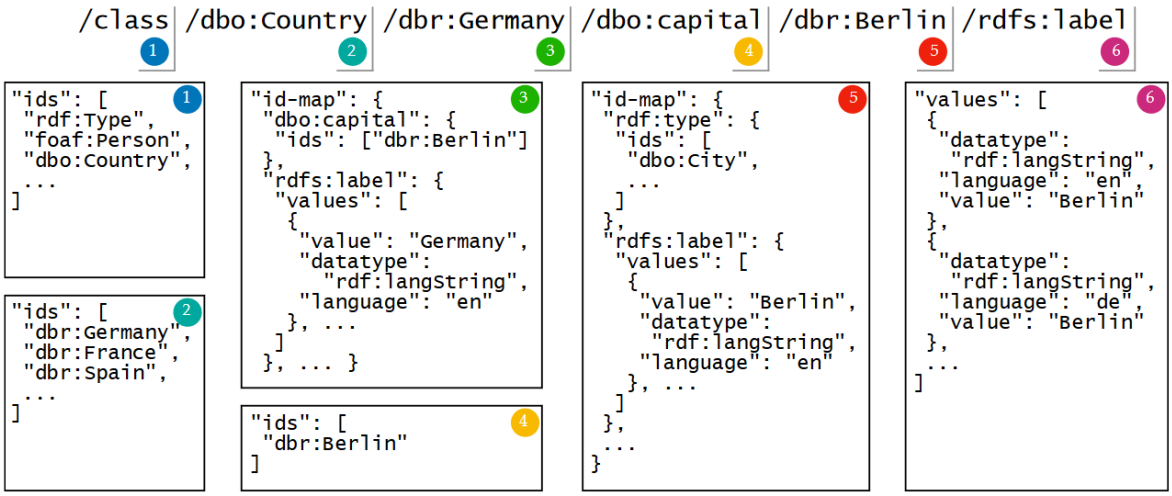
```
http://dbpedia.org/api/v1/class/Company?numberOfEmplyees=gt480000

SELECT ?company
WHERE {
  ?company a dbo:Company ;
           dbo:numberOfEmployees ?empl .
  FILTER(?empl > "480000"^^xsd:Integer) .
}
```

Obrázek 10: Příklad převodu mezi URL a SPARQL dotazem (Universität Leipzig 2018).

Podobným způsobem, avšak nikoli nad konkrétní databází, ale nad konkrétní ontologií, vystavěla své řešení také australská společnost *Australian Research Data Commons*, založená v roce 2018, sdružující výzkumníky, akademiky i zástupce privátního sektoru. Výchozí ontologií se v tomto projektu stal tzv. SKOS (*Simple Knowledge Organization System Namespace Document*), který staví na základu RDFS, RDF a OWL schémat a pokouší se o přece jen preciznější uchopení klíčových pojmů s cílem usnadnit tvorbu dalších nadstavbových slovníků a jejich publikaci. Ústřední třídou je v tomto případě *skos:Concept*, která slouží k zachycení myšlenek a významů sledovaného fenoménu. Na něj je poté navázána sada dalších definicí, které umožňují jednak dané významy blíže specifikovat – např. skrze tzv. sémantické, mapovací či „nálepkovací“ vlastnosti (*skos:broader, skos:narrower, skos:related, skos:prefLabel* atp.) –, jednak je sdružovat do významově-příbuzných celků – např. skrze *skos:Collection* či *skos:OrderedCollection* (Miles a Bechhofer 2009, Cox, Yu a Rankine 2016, 9–16, Le a Walker 2020). Koncové body tedy v tomto smyslu zachovávají zažité členění na zdroje a třídy, avšak konkretizují dané použití pro zvolenou ontologii.

Základními endpointy tedy jsou */api/concept* a */api/collection*, v kterých je možno dále procházet díky výše zmíněným vlastnostem definovaným v rámci SKOS. Např. tedy URL ve formě */concept/narrower?anylabel={text}* navrátí veškeré koncepty užší definice než koncept odpovídající danému značení[35] (Cox, Yu a Rankine 2016, 14–16, Le a Walker 2020).

Poslední řešení, kterému se budeme podrobněji věnovat je v námi zkoumaném kontextu dosud zcela unikátní přístup autorů Schröder, Hees a kol. (2018, passim) pracujících pod záštitou Německého výzkumného centra pro umělou inteligenci (*Deutsches Forschungszentrum für Künstliche Intelligenz* – DFKI). Ti přišli v principu s jednoduchou, zato velmi efektivní, metodikou strukturace API endpointů založenou na obdobném postupu, na jakém stojí samotné procházení RDF grafu. Každá komplexnější API cesta je proto identifikována posloupností ve tvaru subjekt, predikát a objekt s možností rekurzivního zápisu. Výchozími body jsou poté znovu „zdroje“ a „třídy“, které v základu poskytnou přehled dostupných záznamů, které lze již dále prozkoumávat výše popsaným mechanismem. Příklad takového procházení demonstruje obrázek 11.



Obrázek 11: Ilustrace „procházení“ REST API koncových bodů (Schröder, a další 2018, 41).

Autory definovaný přístup v sobě navíc ukotvuje již dříve zmíněné popisné mechanismy jako např. CURIE a souběžně využívá sofistikovanější syntaxi SPARQL jazyka zvanou *property paths* (Seaborne 2010). Díky tomu umožňuje optimalizovat relativně komplexní SPARQL dotazy ve zjednodušeném formátu a ten poté přenášet i do samotného URL koncových bodů. To se poté odráží i v samém jádru řešení, kterým je specificky nadefinovaná strukturace několika forem JSON objektů. Ty flexibilně reagují na několik opakujících se typů dotazů, díky zmíněné „procházení umožňující“ metodice, a vrací konzistentní strukturu v sémanticky

[35] Pro přehled celé API dokumentace daného projektu viz Le a Walker (2020).

nezatíženém formátu. Kromě toho pracuje ještě s prvky jako zástupné znaky (*wildcards*) či širokou škálou modifikátorů specifikovaných v rámci *query stringu* a umožňuje tak tvořit komplexní a zároveň velmi intuitivní dotazovací konstrukce, pokrývající souběžně všechny základní CRUD operace, pouze skrze API dotazy (Schröder, a další 2018, 41).

Pro úplnost této rešeršní části je třeba ještě zmínit velice aktuální projekt známý pod názvem RAMOSE. Tento nástroj, zpracovaný v programovacím jazyce Python, umožňuje automaticky vygenerovat celou soustavu API koncových bodů pouze na základě nadefinování jediného výchozího konfiguračního souboru. Na rozdíl např. od výše zmíněného řešení BASIL, či soudobně též rozvíjeného projektu OBA[36], je struktura koncových bodů, stejně jako na ně namapovaných SPARQL dotazů, plně v dikci koncového uživatele, který je specifikuje v příslušném souboru. Díky tomu lze vytvářet API poskytující data např. z několika různých SPARQL endpointů, příp. napříč různými ontologiemi, stejně jako komplexně předdefinovat doplňující modifikátory daného dotazu. Klíčovou nevýhodou tohoto postupu je však jeho ryze vyhledávací charakter bez podpory ostatních CRUD operací, pevná vazba mezi API koncovými body a základní formou SPARQL dotazu a také nutnost osvojení si syntaxe specifického konfiguračního formátu nazvaného *hashformat* (.hf)[37] (Daquino, a další 2020, passim).

3.3. Návrh řešení

Stejně jako většina výše představených projektů a postupů je i cílem této práce formulovat pro širokou veřejnost snadno uchopitelnou alternativu k získávání sémantických dat bez komplexní znalosti technologií s ní spojených. Zmíněné postupy přitom ukazující rozličné způsoby, jak tohoto cíle dosáhnout, byť v globálním měřítku povětšinou postrádají buďto větší míru univerzálního uplatnění, či např. vskutku důkladné oproštění se od SPARQL a RDF. Souhrn klíčových limitů procházejících napříč výše zmíněnými řešeními proto lze konkretizovat v několika následujících bodech:

[36] *Ontology-Based APIs framework* (OBA) je projekt poskytující automatické generování REST API včetně podpory všech CRUD operací nad slovníkem OWL. Základní myšlenkou je přitom vytvoření samostatného endpointu pro každou instanci třídy *owl:Class* a následného přístupu k jednotlivým zdrojům skrze cestu *owl:Class/{instance}*. Toto řešení obsahuje poté celou řadu konkrétních konfigurací pro dosažení optimálního výsledku a mj. také dynamické mapování mezi JSON-LD a JSON objekty umožňující sémantické validování vkládaných dat (Garijo a Osorio 2020).

[37] Celý projekt se přitom nachází stále ve fázi vývoje a v případě zájmu je dostupný pod otevřenou licencí na: https://github.com/opencitations/ramose .

1. Nutná hlubší znalost sémantických technologií na straně koncového uživatele – např. (Wilde a Hausenblas 2009, Daga, Panziera a Pedrinaci 2015).
2. Vázanost předloženého řešení na konkrétní ontologii, datovou sadu, případně SPARQL endpoint – např. (Universität Leipzig 2018, Le a Walker 2020, Cox, Yu a Rankine 2016).
3. Základní podpora pouze SELECT/GET operací – např. (Daquino, a další 2020).
4. Absence možnosti vyhledávat/upravovat data pouze v rámci konkrétních jmenných grafů – např. (Schröder, a další 2018, Daga, Panziera a Pedrinaci 2015).
5. Omezená či žádná možnost dynamické formulace koncových bodů (existence pouze fixních bodů k požadovanému zdroji) – např. (Daga, Panziera a Pedrinaci 2015, Battle a Benson 2008).
6. Značná komplexnost některých projektů vyžadujících osvojení si nových specifických postupů a technologií – např. (Daquino, a další 2020, Garijo a Osorio 2020).
7. A v neposlední řadě často omezená dostupnost produktu v otevřené (*open-source)* licenci.

Představený výčet přitom pravděpodobně není ani zdaleka úplný a mohl by být doplněn dalšími body s ohledem na cíle a preference hodnotitele. Z pohledu této práce však zahrnuje všechny klíčové body bránící univerzálnímu a uživatelsky přívětivému řešení. Následný návrh tak bude usilovat jednak o odstranění většiny z výše zmíněných nedostatků, jednak ovšem také o využití ústředních poznatků a mechanismů analyzovaných řešení, které by ve vzájemné kombinaci mohly dopomoci k dosažení kýženého výsledku.

Tento návrh přitom bude postaven na pomyslné trojici stavebních pilířů – univerzalitě, jednoduchosti nasazení a intuitivním používání. Ve vztahu k prvnímu bodu přitom lze na koncept univerzality nahlížet dvěma způsoby. Za prvé s ohledem na samotný přístup k libovolnému datovému zdroji sémantických dat, tedy jakoby z vnějšího pohledu, a za druhé s ohledem na získávání/modifikaci informací už přímo v rámci zvoleného zdroje. Za zdroj při tom v tomto smyslu lze považovat především samotné SPARQL endpointy. Ty jsou poskytovateli dat dávány buďto veřejně k dispozici jakožto otevřená služba – poté primárně nabízí vyhledávací funkcionalitu –, anebo jsou uchovávány pro interní potřeby daného

poskytovatele/instituce/obchodního subjektu – poskytující i funkcionalitu pro manipulaci s daty. Obě tyto dimenze – přístup k datům a práce s nimi – byly přitom řešeny i řadou výše představených řešení, avšak většina z nich se soustředila pouze na jednu či druhou úroveň problému a žádnému se nepodařilo je skutečně vzájemně propojit.

Univerzální přístup k datovému zdroji přitom řeší úspěšně např. projekt BASIL, příp. RAMOSE (Daquino, a další 2020, Daga, Panziera a Pedrinaci 2015). Prvně jmenovaný používá k libovolnému zvolení SPARQL endpointu HTTP protokol, druhý volí metodu definice konfiguračních souborů. V obou případech uživatel definuje odkud, co a příp. v jakém formátu chce získat. V reakci jsou poté vytvořeny více či méně statické koncové body, přes které může konzument ke zvoleným datům přistupovat. Univerzální dynamické „prohledávání“ v rámci samotného datového zdroje však již nenabízí tyto řešení, ale naopak některé další (Cox, Yu a Rankine 2016, Universität Leipzig 2018, Schröder, a další 2018). Právě posledně jmenovaná skupina autorů (Schröder, a další 2018) přitom poskytuje zcela specifickou a z pohledu sledovaného cíle této práce velmi vhodnou variantu postupu. Ta tkví ve výše popsaném „průchodu“ API koncových bodů jako by šlo o průchod skutečným RDF grafem. Tím zůstávají v principu na nejvyšší možné úrovni uchopení sémantických dat, a poskytují proto zcela univerzální metodiku pro práci s nimi. Pokud tedy provedeme sjednocení oněch univerzálních řešení na „vnitřní“ i „vnější“ úrovni, můžeme docílit kýženého výsledku.

Zamezení spoutání předkládaného řešení s konkrétní datovou sadu, jmenným grafem, ontologií, či obecně jediným SPARQL přístupovým bodem, tak je možno dosáhnout skrze specifikaci jednoduchých konfiguračních pravidel vždy pro každý konkrétní datový zdroj (SPARQL endpoint). V takové konfiguraci je přitom nezbytné definovat základní kritéria, kterými se jednotlivé datové zdroje od sebe odlišují. Pohledem autora této práce se jedná primárně o tyto body:

- URL SPARQL endpointu;
- URI výchozího grafu, příp. URI dalších jmenných grafů;
- specifikace podporovaných SPARQL metod;
- výčet jmenných prostorů;
- a v neposlední řadě identifikace samotných „vstupních bodů“ do cílených dat.

Poslední kritérium přitom je možné uchopit jakožto výchozí SPARQL dotazy, které identifikují přístup k jednotlivým zdrojům v rámci datasetu, respektive k jednotlivým

skupinám zdrojů. Zde lze vidět jistou analogii s výše již několikrát zmiňovanými postupy, které za takové „vstupy“ označují data definovaná jako *rdfs:Resource* a *rdfs:Class*, popř. *owl:Class*. Tyto definice jsou však ze své podstaty ontologicky zatížené (prefix *rdfs*, tedy asociace k slovníku RDF schema), a to i přesto, že se jedná o prakticky nejvyšší instanci datové strukturace v pomyslné RDF hierarchii. Samotný dataset totiž nemá žádnou povinnost být vázán na příslušnou, byť nadřazenou, ontologii, a může tedy vycházet ze zcela vlastního slovníku či jakéhokoli jinde převzatého[38]. Vzpomeneme-li navíc na možnost dodatečného dělení RDF dat do tzv. jmenných grafů, je více než vhodné poskytnout uživateli možnost příslušné body definovat dle vlastního úsudku. Pokud tedy tyto poznatky budou zformulovány do hmatatelné podoby dostaneme strukturu, kterou lze vyjádřit např. v sémanticky nezatíženém formátu JSON. Názorný příklad, jak by takový konfigurační soubor mohl vypadat, je k vidění na obrázku níže.

```
{
    "endpointName":"datagov",
    "endpointUrl":"https://data.gov.cz/sparql",
    "defaultGraph":"https://data.gov.cz",
    "namedGraphs": [
      {
            "graphName":"regions",
            "uri":"https://data.gov.cz/zdroj/datové-sady/1"
      },
      {
            "graphName":"cities",
            "uri":" https://data.gov.cz/zdroj/datové-sady/2"
      }
    ],
    "namespaces": [
      {
            "prefix":"eks",
            "uri":"https://data.gov.cz/zdroj/ekonomický-subjekt/"
      },
      {
            "prefix":"schranky",
            "uri":" https://data.gov.cz/zdroj/datové-schránky/ "
      }
    ],
    "supported_methods": {
        "sparql_1.0":"yes",
              "sparql_1.1":"no"
    },
    "entry_class": [
      {
            "graphName":"default",
           "command":"SELECT ?s WHERE {?s a <https://data.gov.cz/zdroj/katalog/NKOD>}"
      },
      {
            "graphName":"regions",
           "command":"SELECT ?s WHERE {?s a <https://data.gov.cz/zdroj/katalog/NKOD/region>}"
      },
      {
            "graphName":"cities",
           "command":"SELECT ?s WHERE {?s a <https://data.gov.cz/zdroj/katalog/NKOD/city>}"
      }
    ],
    "entry_resource": [
      {
            "graphName":"default",
            "command":"SELECT DISCTINCT ?s WHERE { ?s ?p ?o }"
       }
    ]
}
```

Obrázek 12: Příklad struktury konfiguračního souboru pro vybraný SPARQL endpoint.

[38] Viz např. (Cox, Yu a Rankine 2016).

Poskytnutím takového souboru je tak jednak vyřešena otázka základní sady podmínek pro mapování cíleného formátu API koncových bodů na libovolný datový zdroj, jednak také splněna i podmínka jednoduchosti nasazení výsledného řešení[39]. V dalším kroku se tak lze přesunout už přímo k samotnému mechanismu procházení RDF dat skrze mapování API cesty na SPARQL dotazy. V tomto se odrazíme od principu zachyceného jeho autory schématem na obrázku níže:

```
API_PATH = "/api" (CLASS | RESOURCE)
CLASS =    "/class" RES PATH RQL?
RESOURCE = "/resource" PATH RQL?
PATH =     ("/" RES "/" PROP)* ("/" RES)?
```

Obrázek 13: Schéma procházení API cesty jako RDF grafu (Schröder, a další 2018).

Východiskem jsou tedy dva možné postupy strukturace API cesty, a to:

- */api/classes/class/resource/property/resource/property/resource…?modifiers*
- */api/resources/resource/property/resource/property/resource…?modifiers*

Zdroj (*resource*) v tomto schématu představuje libovolný subjekt/objekt RDF trojice, na nějž přirozeně navazuje příslušný predikát (*property*). Všechny prvky RDF trojice jsou poté definovány CURIE[40] zápisem, a tím mohou být kompaktně zapsány do API cesty. Sama o sobě je však výše zmíněná strukturace *apriori* vázána na jediný datový zdroj, neboť neposkytuje možnost jeho volby. Aby tak bylo přípustné daný postup aplikovat plošně, lze využít výše definovaných konfiguračních pravidel a doplnit výslednou cestu o definici SPARQL endpointu. Kromě toho je přitom možné obohatit toto řešení i o podporu práce pouze s konkrétními jmennými grafy a docílit tak skutečné univerzality výběru datové sady. Obecné schéma strukturace API koncových bodů tedy vypadá následovně:

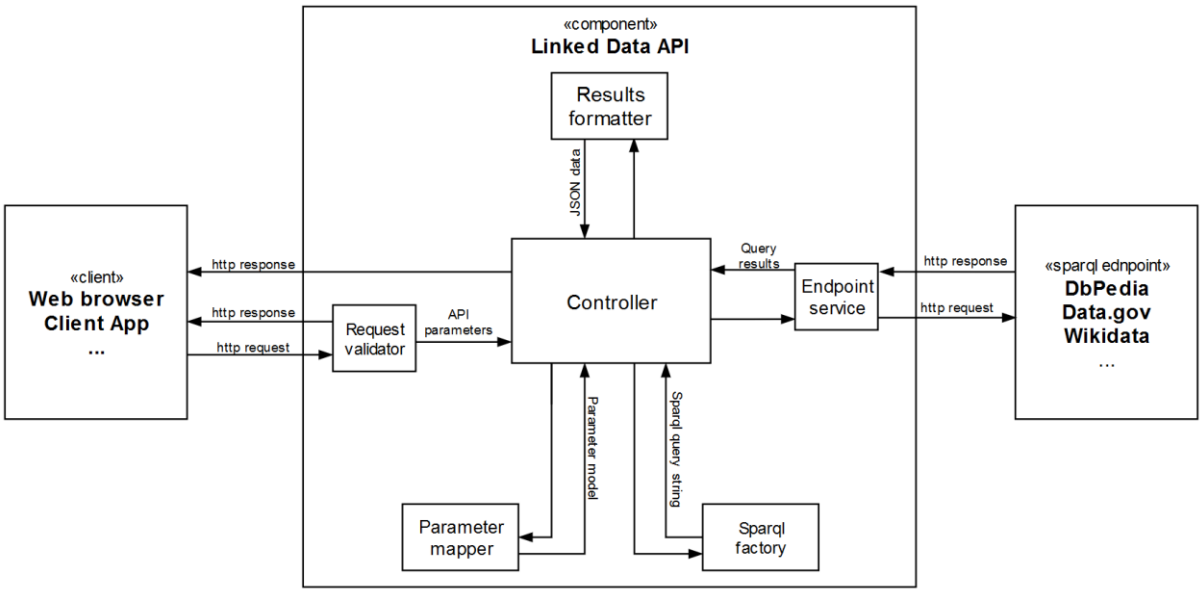
- */api/sparql_endpoint/jmenný_graf*[41]*/třídy/třída/(zdroj/predikát)*…?modifikátory*
- */api/sparql_endpoint/jmenný_graf/zdroje/(zdroj/predikát)*…?modifikátory*

[39] Stačí tedy definice jediného JSON souboru, a to ještě ne nezbytně v plném rozsahu, jak popsáno výše. Některá kritéria totiž mohou být „zevšeobecněna“ (jako např. SPARQL dotazy vstupních bodů) či mohou být odvozena až za běhu aplikace (např. jmenné prostory a jejich prefixy). Blíže bude ukázáno v kapitole týkající se implementace.

[40] Právě plynulá transformace mezi úplným URI a CURIE zápisem bude jednou z klíčových funkcionalit výsledné aplikace.

[41] Pokud budeme pracovat s výchozím grafem, je možné definici jmenného grafu vynechat.

Takto formulovaný dotaz zaslaný uživatelem na webové API bude poté zpracován řídící komponentou (*controller*) a jeho parametry – předané v rámci URL cesty (*route* parametry) a *query stringu* – budou namapovány na příslušný datový model. Ten se poté stane základem pro transformaci příkazu do podoby samotného SPARQL dotazovacího řetězce, který bude posléze zaslán adekvátní HTTP metodou na cílený SPARQL endpoint. Pokud se bude jednat o vyhledávací dotaz bude následný výsledek obdržený od SPARQL endpointu transformován do zvoleného formátu a zaslán zpět uživateli, v případě aktualizačního dotazu dojde pouze k předání informace o jeho úspěšnosti/neúspěšnosti. Schéma plánované architektury ukazuje obrázek 14.



Obrázek 14: Schéma hlavních komponent a principu interního zpracování API požadavku.

Tímto se dostáváme k několika ústředním bodům samotného vnitřního fungování vytvářeného produktu, a to způsobu mapování REST HTTP metod na SPARQL operace, volbě přenosného formátu a neposlední řadě logice týkající se vazeb mezi strukturou dopravovaných dat, datovými modely, respektive odpovídajícími SPARQL řetězci, a jednotlivými formami koncových bodů. Ve vztahu k prvnímu bodu je přitom nezbytné přidržet se snahy oprostit cíleného uživatele od znalosti sémantických technologií, a orientovat se proto ryze na operace týkající se zisku a manipulace dat[42]. V tomto ohledu tak bude zčásti navázáno mj. na autory Battle a Benson (2008) a výsledné mapování bude provedeno následujícím způsobem.

[42] Nikoli tedy specifických operací např. pro tvorbu, popis a jiné modifikace grafů, RDF formátů atp. (operace DESCRIBE, CLEAR, CONSTRUCT ad.).

HTTP (REST API)	SPARQL OPERACE	SPARQL SPECIFIKACE	HTTP (SPARQL ENDPOINT)
GET	SELECT	SPARQL 1.0	GET/POST
POST	INSERT	SPARQL 1.1	POST
DELETE	DELETE	SPARQL 1.1	POST
PUT	DELETE ⇒ INSERT	SPARQL 1.1	POST

Obrázek 15: Způsob mapování HTTP metod výsledného REST API na SPARQL příkazy, SPARQL specifikaci a přijímané HTTP metody SPARQL endpointů.

Přestože jednotlivé kategorie a přiřazení konkrétních hodnot v předchozí tabulce je jasně pochopitelné, je nasnadě na tomto místě připojit dvě poznámky. První se týká PUT metody HTTP protokolu na straně REST API, která nemá ekvivalent ve SPARQL metodologii. Úpravy dat mají tak být dle dokumentace SPARQL UPDATE jazyka prováděny kombinací DELETE a INSERT příkazů (Gearson, Passant a Polleres 2013). Druhou věcí je vazba mezi vyhledávacími dotazy (SPARQL 1.0) a aktualizačními dotazy (SPARQL 1.1) a HTTP metodami zasílanými na SPARQL endpoint. Zde je doporučením W3C stanoveno využívat GET/POST ve vztahu k prvnímu případu a pouze POST ve vztahu k UPDATE operacím (Feigenbaum, a další 2013).

Samostatnou kapitolou je poté volba příslušného formátu pro přenos a reprezentaci dat. V předchozích pasážích byl přitom několikrát zmíněn formát JSON, který představuje ve vztahu k budování REST API přirozenou variantu. Nejenom že se jedná o pro širokou veřejnost velmi dobře známou technologii s rozsáhlými možnostmi zpracování, zároveň ale, díky své jednoduché strukturaci, dokáže vhodně zprostředkovat, respektive skrýt (v závislosti na úhlu pohledu), sémantický význam přenášených dat i pro neznalé uživatele. Obyčejný JSON dokument navíc může být vcelku snadno transformován např. do RDF specifického formátu JSON-LD, a to pouze přiřazením příslušného „kontextu“, jak zmíněno v teoretické kapitole.

Výběrem přenosného formátu se lze přesunout k dalšímu bodu týkajícímu se interního fungování cílené aplikace, a to formulaci vazeb mezi strukturou dopravovaných dat a jednotlivými formami koncových bodů. Ze schématu utváření API cesty popsaného výše totiž vyplývá logický mechanismus založený na opakujících se vzorech dotazů očekávajících vždy stejnou strukturu odpovědi/požadavku pouze s odlišným obsahem. Konkrétně přitom lze identifikovat dvě takové konstelace koncových bodů.

a) Koncové body ukončené ve tvaru */api/.../třídy*, */api/.../zdroje*, */api/.../třída*, */api/.../predikát*, které zprostředkovávají jednoduchý výčet následných možných

kroků v procházení RDF grafu, pak je využit zápis ve formě CURIE, příp. konkrétních hodnot koncových uzlů grafu, pak je využit zápis prezentující literál a k němu přidružené další doplňující informace.

b) Koncové body ukončené ve tvaru */api/.../zdroj*, které poskytují komplexní informace o konkrétním zdroji skrze složitější datovou strukturu, a to způsobem deklarace predikátů a do nich vnořeného výčtu jim přidružených objektů – založeného na principu strukturace v bodě a).

Díky této logice tak lze pro příslušné API cesty formulovat odpovídající datové struktury pro komunikaci s koncovým uživatelem, a umožnit tak nejenom zprostředkovávání dat v očekávané podobě, ale také jejich zpětnou konzumaci pro naplnění i zbylých CRUD operací.

Poslední věc, která přímo navazuje na předchozí bod a je pro funkcionální podstatu budoucí aplikace obdobně klíčová, se týká samotné metodiky mapování parametrů získaných z API cesty na univerzální SPARQL dotazy. Zde se přitom opět vychází z výše popsané logiky dvou variant zpracování příchozího dotazu, které mohou být přeneseny do obecně uplatnitelné formy zápisu SPARQL jazyka. Tím bude umožněn univerzální průchod RDF daty bez ohledu na jejich ontologii a další specifika.

Pro výčet informací o konkrétním zdroji podle bodu b) v předchozím odstavci tak lze formulovat ukázkový SPARQL dotaz ve formě:

SELECT ?p ?o WHERE { <zdroj> ?p ?o }.

, kde *?p* a *?o* jsou predikáty a objekty navázané na zvolený zdroj. A pro výčet objektů náležících predikátu daného zdroje (subjektu), příp. pro výčet dalších dat ve výše zmíněných situacích z bodu a), lze poté použít přirozeně navazující dotazovací formu:

SELECT ?o WHERE { <zdroj> <predikát> ?o }.

V obdobném duchu se pak budou nést i příkazy aktualizační, akorát samozřejmě odpovídající příslušné syntaxi SPARQL 1.1 specifikace. Klíčovým bodem v této pasáži však není popsat přesnou podobu v aplikaci použitých SPARQL příkazů, ale spíše zachytit princip jakým dochází k průchodu cílenou sémantickou datovou sadou velmi přirozeným a intuitivním způsobem. Samotná podoba finálních příkazů pak bude přirozeně podstatně složitější, doplněná o modifikátory předané v rámci *query stringu* (např. paginace, omezení výsledku, filtrování atp.) a o specifika konkrétní datové sady získané z informací v rámci výchozího

konfiguračního souboru. Cílem však vždy bude uchovat tuto intuitivní funkcionalitu napříč jakýmkoli zvoleným datovým zdrojem.

3.4. Implementace řešení

Následná část práce se bude již věnovat samotné technické implementaci zvoleného řešení. Konkrétně budou tak představeny ústřední technologie užité k tvorbě výsledné aplikace, stejně jako bude proveden podrobnější rozbor klíčových programatických pasáží a výsledné funkcionality, která bude demonstrována na vybraných datových zdrojích.

3.4.1. Užité technologie

Výsledné řešení je založeno na programovacím jazyce C# a je vybudováno nad multiplatformní a aktuálně nejmodernější verzí 5.0 frameworku .NET od společnosti *Microsoft*. Verze .NET 5 byla vydána v listopadu roku 2020 a znamenala dlouhodobě plánované dovršení éry frameworků .NET CORE určených pro platformě-nezávislý vývoj. Zároveň se do budoucna stala jediným dále aktivně rozvíjeným rámcem, do kterého byly souběžně začleněny i ústřední komponenty .NET FRAMEWORK určeného pro tvorbu nativních aplikací pro operační systém *Windows*. Kromě .NET 5 se však konečná aplikace zakládá také na obdobně aktuálním frameworku ASP.NET Core specializujícím se přímo na tvorbu webových aplikací. Ten byl představen taktéž v závěru minulého roku a v duchu .NET 5 do sebe vnořuje dříve samostatné frameworky ASP.NET MVC, ASP.NET Web API či ASP.NET Web Pages (Microsoft 2020). Od konsolidace vývojových nástrojů si společnost Microsoft slibuje primárně možnost koncentrovat úsilí právě do rozvoje zmíněných technologií, a tudíž představuje jejich volba z hlediska vývojářské perspektivy jednoznačně nejvhodnější variantu.

Kromě zmíněného se však projekt opírá i o další specifické technologie, z nichž je nezbytné vypíchnout primárně specifikaci *OpenAPI* verze 3.0 (OAS3), známou též pod názvem *Swagger*, a poté rodinu balíčků spadajících pod knihovnu *dotNetRDF*. OAS3 přitom poskytuje časem prověřený a velmi oblíbený nástroj dokumentace webových REST API, který umožňuje vytvářet strojově (i lidsky) čitelnou specifikaci pro efektivní a automatizovanou integraci s externími systémy třetích stran. Na základě takto vygenerovaného popisného dokumentu, nejčastěji ve formátu JSON nebo YAML, je navíc možné automatizovanými

nástroji vytvářet komplexní klientské i serverové knihovny, a to v mnoha programovacích jazycích[43].

Ústředním pomocníkem pro samotnou práci jak s RDF daty, tak s nimi spojenými komponentami se poté stala komplexní knihovna dotNetRDF. Ta umožňuje ryze prostřednictvím jazyka C# manipulovat s RDF daty – od parsování, transformací mezi formáty až po tvorbou zcela nových RDF dokumentů –, vytvářet celé soustavy tzv. úložišť trojic (*triple stores*) – buď permanentního charakteru či tzv. in-memory implementace – a v neposlední řadě i využívat nástroje pro dotazování se nad již existujícími či nad příslušným datovým souborem vytvořenými SPARQL endpointy[44]. Z celého souboru možných nástrojů byl pro případ této práce využit pouze zlomek z nich – jako např. specifický *mapper* mezi jmennými prostory a prefixy či nástroj pro tvorbu HTTP dotazů nad vzdálenými SPARQL endpointy. Integrace této knihovny do zvoleného řešení však nese vzhledem k jejímu stále aktivnímu vývoji potenciál pro plynulou implementaci nových či rozšíření stávajících funkcionalit v budoucnu.

Pro komplexnost výčtu užitých technologií je ještě nezbytné se v tomto bodě zmínit o metodách konečné distribuce a dostupnosti předkládaného projektu. Projekt je v současné době veřejně publikovaný jednak v cloudu díky aplikační platformě *Azure*[45], jednak také včetně zdrojového kódu a souhrnné dokumentace na úložišti *GitHub*[46]. Navíc aby byla zaručena jednoduchá integrace bez ohledu na běhové prostředí byl projekt taktéž zprostředkován v kontejnerizované podobě prostřednictvím virtualizačního nástroje *Docker* a dán k dispozici, dokonce i společně s možností přidružit dedikovaný SPARQL endpoint pro testovací účely, na úložiště *Docker Hub*[47].

Kromě snahy o využití nejmodernějších technologií ovšem projekt samozřejmě následuje také postupy tzv. nejlepší praxe (*best practices*) v oblasti programování a snaží se i na úrovní samotného zdrojového kódu o maximální flexibilitu směrem k budoucímu vývoji[48]. Svou povahou tak po funkcionální i architektonické stránce výchozí aplikace odpovídá spíše podobě tzv. mikroslužby, která je předpřipravena pro integraci do většího funkčního celku. To přitom

[43] Více viz https://swagger.io/specification/.
[44] Více viz https://www.dotnetrdf.org/.
[45] Dostupné na https://linkeddataapi.azurewebsites.net/.
[46] Dostupné na https://github.com/dominik-heller/BP_LinkedData_Api.
[47] Samotný API projekt je dostupný na https://hub.docker.com/r/helldo/linkeddataapi. Pro vytvoření API společně s dedikovaným SPARQL endpointem viz dokumentace k nasazení skrze *docker-compose* na úložišti GitHub.
[48] Tomu odpovídá také hustá síť testů pokrývající veškerou ústřední funkcionalitu výsledného řešení a adekvátní dokumentace jednotlivých pasáží samotného zdrojového kódu.

přirozeně vychází ze snahy primárně naplnit princip „prokázání uplatnitelnosti konceptu“ (*proof of concept*), nežli předložit skutečně plně produkčně-připravené řešení. Obsahem následné kapitoly tedy bude snaha právě tuto skutečnost – praktickou uplatnitelnost výsledného řešení – demonstrovat a otestovat veškerou funkcionalitu na konkrétních datových zdrojích.

3.4.2. Ukázková aplikace zvoleného řešení

Tato závěrečná pasáž bakalářské práce se bude věnovat prezentaci funkcionality výsledného produktu s cílem zachytit veškeré možné scénáře jejího využití na vybraných SPARQL endpointech. Neboť výsledné řešení si klade za cíl býti plně univerzální, nemá volba konkrétního datového zdroje na úkor jiného žádný vliv na výsledné chování aplikace. Při výběru SPARQL endpointu tak bylo primárně vycházeno ze zkušenosti autora s příslušným zdrojem, tedy z minimálně povrchní znalosti podoby obsažených dat, stejně jako jejich spolehlivosti a celkové reprezentativnosti[49]. Cíleně proto byly zvoleny dva datové zdroje, které poskytují téměř vždy spolehlivý a konzistentní výstup při dotazech na RDF data, a navíc nabízejí poměrně rozsáhlé informace o specifikách daných SPARQL endpointů.

Konkrétně tedy byl zvolen SPARQL endpoint, který poskytuje český *Portál otevřených dat*, spravující v sémantické podobě tzv. Národní katalog otevřených dat (NKOD), který obsahuje informace o veškerých záležitostech spojených s výkonem veřejné správy v České republice. A poté pravděpodobně celosvětově nejznámější SPARQL endpoint zvaný *DBpedia*, který zprostředkovává data obsažená na portálu Wikipedia ve formátu propojených otevřených dat. Výběr těchto zdrojů se zároveň opírá o značně odlišnou formu jejich interní datové strukturace, což umožňuje demonstrovat univerzální uplatnitelnost výsledného řešení. Hlavní rozdíly v tomto smyslu spočívají na jedné straně v četnosti užívání jmenných grafů a na straně druhé v rozličné formulaci prefixů a jmenných prostorů.

Zatímco Portál otevřených dat rozčleňuje všechny datové sady do samostatných jmenných grafů – tzn. existuje např. jmenný graf pro typologii škol, pro aktivní politiku zaměstnanosti, pro administrativní územní členění regionů atp. – DBpedia uchovává pouze jediný výchozí jmenný graf, do kterého jsou všechna data začleněna. Obdobně u jmenných prostorů, a s nimi související stavbě zdrojových URI, se setkáváme s daleko více heterogenním přístupem na straně Portálu otevřených dat nežli u portálu DBpedia. DBpedia totiž víceméně každý zdroj zahrnuje do jmenného prostoru nazvaného *http://dbpedia.org/resources/* a obecně používá ve

[49] Velice častá je totiž jednak nepravidelná dostupnost takových endpointů, jednak také vysoká fluktuace ukládaných dat vedoucí k nekonzistentní prezentaci.

většině případů pevnou sadu prefixů a jmenných prostorů. Portál otevřených dat na straně druhé, zdá se, nedisponuje takto jednotnými pravidly a jmenné prostory tak mohou nabývat až mnohonásobně většího množství forem.

Pro předkládanou aplikaci však není znalost všech jmenných prostorů překážkou na cestě k zachování plné funkcionality. Pokud je totiž při překladu URI na CURIE naraženo na neznámý jmenný prostor, je aplikací vygenerován unikátní prefix[50] a oba jsou zaregistrovány pro příští použití. Na rozdíl od všech analyzovaných řešení v rešeršní kapitole tak výsledná aplikace nepotřebuje definovat veškeré užívané jmenné prostory v konfiguračním souboru[51]. Jejich definice tedy slouží primárně k dosažení vždy konzistentního výsledku na příslušný dotaz, neboť prefix není k jmennému prostoru generován za běhu aplikace, ale je přebrán z konfiguračního souboru.

Obecně má poté uživatel API k dispozici celkem 32 koncových bodů. Zhruba pětina z nich se přitom vztahuje k informacím o aktuálním nastavení aplikace – tedy definicím konfiguračních souborů či užívaných prefixů a jmenných prostorů. Zbytek poté tvoří již samotné „mediátory" přístupu ke zvolenému SPARQL endpointu umožňující hledat, vkládat, měnit a mazat vybrané zdroje. Ve vztahu k první kategorii je nasnadě zmínit primárně koncové body ve tvaru */api/{endpoint}*, který poskytne aktuální konfiguraci pro zvolený SPARQL endpoint, dále */api/endpoints*, kam lze za běhu aplikace zasílat konfigurace zcela nové a poté */api/namespaces/{prefix}*, který vrátí jmenný prostor k příslušnému prefixu.

V tomto bodě je však nutné ještě přidat poznámku k formátu konfiguračního JSON souboru. Uživatel má totiž na výběr buď specifikovat soubor tzv. v plném rozsahu – tedy zahrnout definice jmenných prostorů a grafů, vstupních bodů pro třídy a zdroje atp. –, anebo se omezit na předložení pouze dvou základních informací – unikátního id (jména) endpointu a jeho URL –, jak odhaluje obrázek 16.

```
{
   "endpointName":"datagov",
   "endpointUrl":"https://data.gov.cz/sparql"
}
```

Obrázek 16: Minimalistická podoba konfiguračního souboru.

Tímto je zajištěno, že i uživatel s naprosto nulovou znalostí RDF/SPARQL technologií může přistoupit k libovolnému sémantickému datovému zdroji pouze se znalostí jeho URL. Taková

[50] Vždy ve tvaru *_ns0*, *_ns1*, *_ns2* atd.
[51] U předchozích řešení byly takové prostory jednoduše ignorovány, čímž dotazovatel mohl přijít o značnou porci cenných dat.

konfigurace bude poté doplněna o sadu výchozích nastavení umožňujících uživateli procházet data přes API cestu ve tvaru */api/{endpoint}/resources/({curie_zdroje}/{curie_predikátu})*…*, tedy, s ohledem na konfiguraci na obrázku 16, např. v podobě */api/datagov/resources/eu:22/rdf:type*…. Takové nastavení však samozřejmě neumožňuje některé pokročilejší funkcionality, které vyžadují např. definici vstupních bodů pro členění zdrojů do kategorií (tříd), tedy přístup přes bod */api/{endpoint}/classes* či prohledávání přes konkrétní jmenné grafy atp. Klíčové ovšem je, že v případě zájmu může uživatel takové informace doplnit do příslušného konfiguračního souboru a začít i tyto funkce aktivně využívat. Široká míra uživatelské customizace je tedy v každém případě plně zajištěna.

Tímto se text této práce přesouvá ke koncovým bodům provádějícím již samotné operace nad daty přístupnými přes vzdálené SPARQL endpointy. Na obrázcích v příloze 5 a 6 si přitom lze prohlédnout konfigurační soubory užité pro endpoint DBpedia, respektive Portál otevřených dat. Jak již řečeno výše, tak DBpedia nedisponuje členěním na jmenné grafy, a tudíž přístup k jednotlivým zdrojům lze provést skrze cestu */api/dbpedia/resources* a ke třídám přes */api/dbpedia/classes*. U Portálu otevřených dat poté je možné díky předdefinování jmenných grafů konkretizovat výběr pro zdroje např. jako */api/datagov/school/resources* a pro třídy */api/datagov/school/classes*, kde „school“ odpovídá definici jednoho z jmenných grafů z konfiguračního souboru[52]. V obou případech ale dostaneme vždy stejný formát výsledku, a tedy výčet následných bodů, které lze dále zapsat do API cesty.

GET: **/api/dbpedia/classes**

```
{
  "curies": [
    "dbpo:Company",
    "dbpo:Activity",
    "dbpo:Name",
    "dbpo:Person",
    "dbpo:Actor",
    "dbpo:Place",
    "dbpo:Software",
    "dbpo:School",
    "dbpo:Type",
     …]
}
```

GET: **/api/datagov/school/classes**

```
{
  "curies": [
    "_ns0:00551023",
    "_ns1:41f9841463beb472c92f62fff43b15dd",
    "_ns1:41f9841463beb472c92f62fff43b15dd",
    "_ns3:podmínky-užití",
    "_ns5:podmínky-užití",
    "_ns6:kontaktní-bod",
    "_ns6:záznam",
    "_ns6:časové-pokrytí",
    "_ns7:podmínky-užití",
     …]
}
```

Obrázek 17: Názorná ukázka výstupu dotazu na endpoint typu „classes“.

Obdobný formát výsledku poté dostaneme i v případě dotazů na koncové body typu …/*resources* a samozřejmě i při výběru konkrétní třídy v rámci seznamu třídy – např.

[52] V případě, že je jmenný graf z cesty vynechán, je automaticky využit výchozí graf.

/api/dbpedia/classes/dbpo:Company či */api/datagov/school/classes/_ns6:časové-pokrytí*[53].

GET: **/api/dbpedia/resources**


```
{
  "curies": [
    "dbr:1911_in_art",
    "dbr:1911_in_jazz",
    "dbr:1911_Austrian_First_Class",
    "dbr:1912_FA_Charity_Shield",
    "dbr:1912_Fez_riots",
    "dbr:1912_French_Grand_Prix",
    "dbr:1912_Grand_National",
    "dbr:1912_Grand_Prix_season",
     …]
}
```


GET: **/api/datagov/school/resources**


```
{
  "curies": [
    "_ns0:00551023",
    "_ns1:41f9841463beb",
    "_ns1:41f9841463beb472c",
    "eu:745",
    "eu:788",
    "eu:778",
    "_ns6:záznam33",
    "_ns8:SOCI",
     …]
}
```


GET: **/api/dbpedia/classes/dbpo:Company**


```
{
  "curies": [
    "dbr:OpenLink_Software",
    "dbr:2015_Games,_LLC.",
    "dbr:5_Rue_Christine",
    "dbr:5th_Cell",
    "dbr:6GMOBILE",
    "dbr:70mm_Entertainments",
    "dbr:7132_Thermal_Baths",
    "_ns9:Downtown",
    …]
}
```


Obrázek 18: Názorná ukázka výstupu dotazu na endpoint typu „resources“ a „class“.

Jako další krok následuje u všech přechozích variant volba konkrétního zdroje. Ta, jak uvedeno již v kapitole o návrhu řešení, vrací komplexnější, byť v každé situaci stejný, formát výsledku.

GET: **/api/dbpedia/resources/dbr:Prague**


```
{
 "predicates": {
  "rdf:type": {
   "curies": [
    "dbpo:Place",
    "dbpo:Location",
    "w3geo:SpatialThing",
    "dby:PhysicalEntity100001930",
    "dby:Region108630039",
    "dby:UrbanArea108675967",
    "dby:WikicatCapitalsInEurope",
    …]
  },
  "rdfs:label": {
   "literals": [
    {
     "value": "Prague",
     "datatype": "rdf:langString",
     "language": "en"
    },
    {
     "value": "براغ",
     "datatype": "rdf:langString",
     "language": "ar"
    },
    {
     "value": "Praha",
     "datatype": "rdf:langString",
     "language": "cs"
    }
   ]
  }
 }
}
```


GET: **/api/datagov/school/resources/_ns6:kontaktní-bod**


```
{
 "predicates": {
  "rdf:type": {
   "curies": [
    "vcard2006:Organization"
   ]
  },
  "vcard2006:fn": {
   "literals": [
    {
     "value": "Ministerstvo práce a sociálních věcí",
     "datatype": "rdf:langString",
     "language": "cs"
    },
    {
     "value": "Ministry of Labour and Social Affairs",
     "datatype": "rdf:langString",
     "language": "en"
    }
   ]
  },
  "vcard2006:hasEmail": {
   "literals": [
    {
     "value":"mailto:frantisek.povinsky@mpsv.cz",
     "datatype": "xsd:string"
    }
   ]
  }
 }
}
```


GET: **/api/dbpedia/classes/dbpo:Company/dbr:A-Mobile**


```
{
 "predicates": {
  "rdf:type": {
   "curies": [
    "owl:Thing",
    "dbpo:Company",
    "shema:Organization",
    "dby:Organization108008335",
    ...]
  },
  "foaf:name": {
   "literals": [
    {
     "value": "A-Mobile",
     "datatype": "rdf:langString",
     "language": "en"
    },
    {
     "value": "СП ООО «А-Мобайл»",
     "datatype": "rdf:langString",
     "language": "en"
    }
   ]
  },
  "foaf:homepage": {
   "literals": [
    {
     "value": "http://www.a-mobile.biz/",
     "datatype": "xsd:string"
    }
   ]
  }
 }
}
```


Obrázek 19: Názorná ukázka výstupu dotazu na endpoint typu „resource“.

[53] Zde můžeme vidět i za běhu aplikace vygenerovaný prefix ve tvaru _ns6, který odpovídá jmennému prostoru https://data.gov.cz/zdroj/datové-sady/00551023/861588425/41f9841463beb472c92f62fff43b15dd/.

Ze získaného výsledku si poté uživatel volí libovolný predikát, který znovu zapisuje do API cesty. Výstup poté odpovídá interní struktuře jednotlivých predikátů jako na obrázku 19.

GET: **/api/dbpedia/resources/dbr:Prague/rdfs:label**


```
{
  "literals": [
    {
      "value": "Prague",
      "datatype": "rdf:langString",
      "language": "en"
    },
    {
      "value": "براغ",
      "datatype": "rdf:langString",
      "language": "ar"
    },
    {
      "value": "Praha",
      "datatype": "rdf:langString",
      "language": "cs"
    }
  ]
}
```


GET: **/api/datagov/school/resources/_ns6:kontaktní-bod/vcard2006:fn**


```
{
  "literals": [
    {
      "value": "Ministerstvo práce a
                sociálních věcí",
      "datatype": "rdf:langString",
      "language": "cs"
    },
    {
      "value": "Ministry of Labour and
                Social Affairs",
      "datatype": "rdf:langString",
      "language": "en"
    }
  ]
}
```


GET: **/api/dbpedia/classes/dbpo:Company/dbr:A-Mobile/dbpo:wikiPageWikiLink**


```
{
  "curies": [
    dbr:Republic_Abkhazia_(country)",
    "dbr:Beeline_(brand)",
    "dbr:Russia",
    "dbr:Limited_liability_company",
    "dbr:Aquafon",
    "dbr:Mobile_telephony",
    "dbr:Abkhazia",
    "dbr:Mobile_phone_operator",
    "dbr:Sukhumi",
    "dbr:Telecommunication"
  ]
}
```


Obrázek 20: Názorná ukázka výstupu dotazu na endpoint typu „predicate“.

Výstupem v tomto případě může tak být seznam literálů, včetně dodatečných informací o datovém typu, jazyce atp., anebo seznam dalších kompaktních URI, které mohou být znovu zapsány do API cesty. V dalším kroku, tedy volbě subjektu (dalšího zdroje), poté vlastně završujeme procházení RDF trojice a zadané URL je přesměrována znovu na výchozí volbu zdroje. Rozšířený dotaz z předchozího příkladu ve tvaru */api/dbpedia/classes/dbpo:Company/dbr:A-Mobile/dbpo:wikiPageWikiLink/dbr:Russia* bude tedy přesměrován na adresu */api/dbpedia/resources/dbr:Russia*. Obdobně bude postupováno i ve všech ostatních možných variantách.

Ve vztahu k vyhledávací funkcionalitě je poté třeba ještě zmínit možnost další úpravy očekávaného výsledku. K dispozici jsou totiž čtyři klíčové parametry – *offset*, *regex*, *sort* a *limit* – pro umožnění paginace, filtrace, řazení a limitace výstupu. Ty jsou zapisovány za příslušnou API cestu do *query stringu* daného URL ve formě *?parametr1=hodnota1&parametr2=hodnota2* atp. Názornou ukázku předkládá obrázek 21, ve kterém vyhledáváme všechny zdroje, které obsahují slovo „Prague“, omezujeme výsledek na pět zdrojů, odsazujeme od počátku taktéž o pět a výsledek řadíme vzestupně[54].

[54] Alternativou je užití klíčového slova „desc” pro řazení sestupně.

```
GET: /api/dbpedia/resources?regex=Prague&limit=5&offset=5&sort=asc
{
    "curies": [
            "dbr:2014_Sparta_Prague_Open",
            "dbr:2014–15_FK_Dukla_Prague_season",
            "dbr:2015_Advantage_Cars_Prague_Open",
            "dbr:2015_J&T_Banka_Prague_Open",
            "dbr:2015_Sparta_Prague_Open"
            ]
}
```

Obrázek 21: Názorná ukázka výstupu dotazu s query string parametry.

Tímto můžeme uzavřít demonstraci funkcionalit vázaných na HTTP metodu GET a přesunout se k modifikačním mechanismům. V jejich rámci je umožněno přidávat nové zdroje a predikáty a také modifikovat, respektive mazat ty již existující. K tomu nám slouží tři zbylé HTTP metody, a to POST, PUT a DELETE. Neboť ovšem veřejně dostupné SPARQL endpointy nedovolují úpravy interních dat, je nutné provést příslušné operace nad vlastním datovým zdrojem. Pro tento účel byl tak vytvořen testovací SPARQL endpoint, jehož zprovoznění lze dosáhnout postupem zmíněným v předchozí kapitole. Obsahem autorem předpřipraveného endpointu jsou přitom dvě datové sady, a to *Orgány veřejné moci* (ovm) a *Lexikální slovník* (lexvo), a poté ještě prázdný testovací jmenný graf pod názvem „test“. Právě ten bude využit k demonstraci aktualizační funkcionality[55].

Obecně poté výsledné API nabízí celkem 12 koncových bodů pro POST, PUT a DELETE metody, které svou strukturou odpovídají výše zmíněným vyhledávacím API cestám pro predikáty a zdroje. Jako první bude přitom demonstrována metoda POST. K ní se vztahují cesty */api/{endpointName}/resources* – pro vkládaní nových zdrojů – a poté */api/{endpointName}/resources/{curie_zdroje}* – pro vkládání predikátů k zvolenému zdroji. Struktura zaslaného těla zprávy ve formátu JSON přitom odpovídá formě odpovědi na dotaz na konkrétní zdroj a predikát a je doplněna pouze informací o jeho CURIE. Názorný příklad ukazuje obrázek 22.

[55] Plná podoba konfiguračního souboru je dostupná v příloze 7.

POST: **/api/ovm/test/resources**

```
{
  "resourceCurie": "ex:exampleResource",
  "predicates": {
    "ex:examplePredicate": {
      "curies": [
        "ex:curie1",
        "ex:curie2",
        "ex:curie3"
      ],
      "literals": [
        {
          "value": "Example language text…",
          "language": "en"
        },
        {
          "value": "2021-03-02T20:00:00-01:00",
          "datatype": "xsd:dateTime"
        },
        {
          "value": "String value"
        }
      ]
    }
  }
}
```

POST: **/api/ovm/test/resources/ex:exampleResource**

```
{
  "predicateCurie": "ex:examplePredicate",
  "curies": [
    "ex:curie1",
    "ex:curie2",
    "ex:curie3"
  ],
  "literals": [
    {
      "value": "value1",
      "datatype": "ex:exampleDatatype"
    },
    {
      "value": "value2",
      "language": "en"
    },
    {
      "value": "value3"
    }
  ]
}
```

Obrázek 22: Názorná ukázka těla zprávy zaslané metodou POST.

Správné provedení takového příkazu lze poté ověřit dotazem na adresu */api/virtuoso/test/resources/ex:exampleResource*, který odhalí vytvoření nového zdroje s predikátem *ex:examplePredicate* a jemu přidruženým obsahem. Podobně poté pracuje také mechanismus spojený s metodou PUT, kdy hlavní rozdíl spočívá pouze ve specifikaci CURIE zdroje, respektive predikátu, která již není součástí samotného těla zprávy, ale je zapsána přímo do API cesty.

PUT: **/api/ovm/test/resources/ex:exampleResource**

```
{
  "predicates": {
    "ex:examplePredicate": {
      "curies": [
        "ex:curie1",
        "ex:curie2",
        "ex:curie3"
      ],
      "literals": [
        {
          "value": "Example language text.",
          "language": "en"
        },
        {
          "value": "2021-03-02T20:00:00-01:00",
          "datatype": "xsd:dateTime"
        },
        {
          "value": "String value"
        }
      ]
    }
  }
}
```

PUT: **/api/ovm/test/resources/ex:exampleResource /ex:examplePredicate**

```
{
  "curies": [
    "ex:curie1",
    "ex:curie2",
    "ex:curie3",
    "ex:curie4"
  ],
  "literals": [
    {
      "value": "value1",
      "datatype": "ex:exampleDatatype"
    },
    {
      "value": "value2",
      "language": "en"
    },
    {
      "value": "value3"
    }
  ]
}
```

Obrázek 23: Názorná ukázka těla zprávy zaslané metodou PUT.

V tomto bodě je ovšem nezbytné upozornit, že samotný dopad dotazů metodou PUT a POST se významně odlišuje v případě, že zdroj/predikát s příslušným CURIE již v datasetu existuje. V této situaci tak POST pouze přidruží zaslaný obsah k již existujícímu zdroji/predikátu, zatímco PUT existující záznam plně nahradí. Pokud by tedy dotazy v přechozích příkladech na obrázku 22 a 23 byly zaslány postupně, dojde k následujícímu výsledku.

BEFORE POST AND PUT:

GET: **/api/virtuoso/ovm/resources/ex:exampleResource**

```
{
  "customErrorMessage": "No results were
                         found!",
  "generatedQuery":
     SELECT * FROM
       <http://localhost:8890/ovm>
     WHERE {
       <http://example.org/exampleResource>
          ?p ?o }
     LIMIT 50 OFFSET 0."
}
```

AFTER POST:

GET: **/api/virtuoso/ovm/resources/ex:exampleResource**

```
{
  "predicates": {
    "ex:examplePredicate": {
      "curies": [
        "ex:curie1",
        "ex:curie2",
        "ex:curie3"
      ],
      "literals": [
        {
          "value": "2021-03-02T20:00:00-01:00",
          "datatype": "xmls:dateTime"
        },
        {
          "value": "Example language text...",
          "datatype": "rdf:langString",
          "language": "en"
        },
        {
          "value": "String value",
          "datatype": "xsd:string"
        },
        {
          "value": "value1",
          "datatype": "ex:exampleDatatype"
        },
        {
          "value": "value2",
          "language": "en"
        },
        {
          "value": "value3"
        }
      ]
    }
}
```

AFTER PUT:

GET: **/api/virtuoso/ovm/resources/ex:exampleResource**

```
{
  "predicates": {
    "ex:examplePredicate":    {
      "curies": [
        "ex:curie1",
        "ex:curie2",
        "ex:curie3",
        "ex:curie4"
      ],
      "literals": [
        {
          "value": "value1",
          "datatype": "ex:exampleDatatype"
        },
        {
          "value": "value2",
          "language": "en"
        },
        {
          "value": "value3"
        }
      ]
    }
}
```

Obrázek 24: Názorná ukázka dopadů metod POST a PUT na již existující zdroj/predikát.

Poslední skupinu dotazů poté tvoří ty spadající pod metodu DELETE. K nim se vztahuje stejná struktura API cesty jako u metody PUT jen se zřejmým rozdílem, že dojde k odstranění zvoleného zdroje/predikátu. Zaslání DELETE dotazu na */api/virtuoso/test/resources/ex:exampleResource* tedy zcela odstraní daný zdroj včetně všech na něj navázaných odkazů, zatímco požadavek zaslaný na adresu */api/virtuoso/test/resources/ex:exampleResource/ex:examplePredicate* zcela odstraní daný predikát v rámci zvoleného zdroje.

Na závěr je nezbytné ještě doplnit, že všechny v této kapitole předložené příklady lze plně replikovat i v samotné aplikaci díky komplexní OAS3 dokumentaci. Po spuštění API bude tedy uživatel přesměrován na stránku s popisem jednotlivých koncových bodů, jejichž součástí bude i možnost si předpřipravené příklady vyzkoušet.

3.5. Stávající limity výsledného řešení a budoucí vývoj

Závěrem této práce je vhodné ještě kriticky zanalyzovat aktuální nedostatky vytvořeného produktu a načrtnout možnosti případného budoucího vývoje. Jeden z takových nedostatků přitom tkví v absenci persistentní vrstvy pro ukládání nově připojených konfiguračních nastavení a také za běhu aplikace generovaných jmenných prostorů a prefixů. V obou případech totiž dochází k načítání konfigurací výchozích endpointů, včetně sady nejužívanějších jmenných prostorů při samotném startu aplikace z předpřipravených textových souborů. Veškerá další, přes příslušné API koncové body, přidaná data jsou však již uchovávána pouze po dobu běhu programu. To vychází primárně z původního záměru jednak nezveřejňovat koncový bod pro nahrání nových konfigurací, jednak ponechat volbu vhodného databázového systému až na konkrétní požadavky v rámci testovacího provozu s cílem dosáhnout optimálního, nikoli pouze provizorního, řešení[56]. Aby však mohl uživatel skutečně otestovat univerzální uplatnitelnost zvoleného řešení, došlo nakonec k doplnění příslušného koncové bodu do finální podoby produktu. V každém případě, samotné řešení je na budoucí napojení libovolného datového úložiště z programatického hlediska zcela připraveno.

S výše popsaným poté souvisí také otázka případné implementace autentizačních a autorizačních mechanismů. Ty by se mohly vztahovat ke konkrétním API koncovým bodům týkajícím se právě samotných nastavení příslušných endpointů, příp. i definic práv přístupu v rámci jednotlivých jmenných grafů. Další prostor pro případná vylepšení se poté týká konkrétních funkčních aspektů samotné komunikace mezi API a SPARQL endpointy. V současné chvíli totiž aplikace poskytuje vyhledávací funkcionalitu tzv. v rámci první úrovně sémantických tvrzení, která je pro účel zprostředkování informací primárně uživatelům bez předchozí zkušenosti se SPARQL jazykem zcela dostatečná. V budoucnu by však mohlo dojít k rozšíření stávajícího vyhledávacího procesu i o tzv. zástupné znaky (*wildcards*) umožňující prostupovat hlouběji v příslušném sémantickém datasetu a formulovat složitější a podstatně delší API cesty[57].

Další možností je poté zavedení rozšířených vyhledávacích mechanismů ve vztahu k filtrování a modifikaci výsledků, a to např. skrze tzv. *property paths* zmiňované v rešeršní kapitole. Za zvážení by poté stála také možnost doplnit stávající funkcionalitu i o podporu práce se

[56] Samotných možností implementace se totiž nabízí značné množství. Zatímco pro konfigurace endpointů by bylo možné si vystačit prakticky s jakoukoli formou relační či dokumentové databáze, u prefixů a jmenných prostorů se naopak jako vhodnější jeví některá z databází typu *klíč-hodnota.*

[57] Současné řešení dále nepodporuje např. také práci s prázdnými uzly, stejně jako jiné formy zápisu URI, než je absolutní URL.

samotnou grafovou reprezentací RDF dat, a tedy i správu celých datových sad – např. skrze SPARQL metody jako CONSTUCT, ASK, MOVE atp.

Obecně však všechny zmíněné výtky vycházející ze snahy dosáhnout co nejkomplexnějšího mapování mezi API cestou a SPARQL dotazy v důsledku naráží na snahu zachovat princip jednoduchosti a široké přístupnosti stávajícího řešení. Podnětem pro budoucí analýzu by tak nemělo být jen hledání dalších vylepšení aktuálního způsobu řešení, ale i prozkoumání alternativních přístupů umožňujících zahrnout další možnosti práce s propojenými otevřenými daty, a to i skrze jiné technologie spojené s webovými API než poskytuje užitá REST architektura.

4. Závěr

Tato bakalářská práce se zabývala problematikou tzv. propojených otevřených dat, tedy dat volně dostupných na internetu a souběžně splňujících předepsané požadavky páté nejvyšší úrovně hvězdičkové klasifikace. Hlavní pozornost byla přitom věnována současným způsobům jejich publikování a využívání, které nezbytně vyžadují znalost specifických sémantických technologií, jako je datový formát RDF či dotazovací jazyk SPARQL, což v důsledku brání mj. jejich plynulé integraci s jinými systémy. Cílem se proto stala snaha pokusit se formulovat univerzální řešení, které umožní čerpat tato data z libovolného datového zdroje standardizovaným, maximálně intuitivním a obecně dobře známým způsobem za využití webového API postaveného na architektuře REST.

Po teoretickém shrnutí ústředních technologií a principů spojených s aspekty sémantického webu tak bylo přikročeno k provedení důkladné rešeršní analýzy s cílem zmapovat aktuální stav a podobu již existujících řešení usilujících o dosažení obdobného cíle. Výsledkem bylo odhalení, že byť se danému problému věnovalo nemalé množství publikací a projektů, především v posledních letech, žádné nedokázalo naplnit všechny z výše stanovených nároků na univerzalitu, intuitivnost a jednoduchost výsledného produktu. Jednotlivá řešení se totiž soustředila buďto pouze na konkrétní datový zdroj či sémantickou sadu, nebo se nedokázala plně oprostit od potřeby hlubšího povědomí o sémantických nástrojích a technologiích na straně koncového uživatele. I přesto však jednotlivé analyzované návrhy poskytly autorovi této práce dostatečný základ pro nalezení požadovaného způsobu řešení kombinujícího rozebírané postupy s vlastními podněty pro zlepšení. Právě tato část se přitom stala zcela nejnáročnější z celého procesu tvorby výsledného produktu, neboť vyžadovala vskutku detailní a precizní plánování jednak komplexních struktur výsledných API cest, jednak jejich logické provázanosti s ostatními prvky ve snaze naplnit všechny předeslané cíle.

Následná implementace se poté již zcela držela formulovaného návrhu a souběžně zapracovala klíčové nástroje umožňující plynulou integraci výsledného řešení v rámci komplexnějších projektů, příp. budoucí pokračování vývoje s cílem rozšíření aktuální funkcionality. Neboť tedy cílem této práce bylo primárně pokusit se prokázat samotnou proveditelnost výše popsané koncepce řešení, lze závěrem tohoto textu konstatovat, že stanoveného cíle bylo dosaženo, a to v plném rozsahu.

5. Zdroje a prameny

Addlesee, A. *Understanding Linked Data Formats.* 20. 1 2018. https://medium.com/wallscope/understanding-linked-data-formats-rdf-xml-vs-turtle-vs-n-triples-eb931dbe9827, (přístup získán 20. 1 2021).

Allemang, D., J. Hendler, a F. Gandon. *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*. New York: Association for Computing Machinery, 2020.

Battle, R., a E. Benson. „Bridging the semantic Web and Web 2.0 with Representa-tional State Transfer (REST).“ *Web Semantics: Science,Services and Agents on the World Wide Web* 6 (2008): 61-69.

Berners-Lee, T. *Linked Data.* 7. 27 2006. https://www.w3.org/DesignIssues/LinkedData.html (přístup získán 19. 9 2020).

—. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor.* New York: HarperCollins Publishers, 1999.

Berners-Lee, T., J. Hendler, a O. Lassila. „The Semantic Web.“ *Scientific America* 284, č. 5 (2001): 28-37.

Brickley, D., a R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema.* 17. 1 2003. https://www.w3.org/2001/sw/RDFCore/Schema/200212bwm/ (přístup získán 23. 1 2021).

Carothers, G., a A. Seaborne. *RDF 1.1 N-Triples.* 25. 2 2014. https://www.w3.org/TR/n-triples/ (přístup získán 20. 1 2021).

Cox, S. J. D., J. Yu, a T. Rankine. „SISSVoc: A Linked Data API for access to SKOS vocabularies.“ *Semantic Web Journal* 7, č. 1 (2016): 9-24.

Curé, O., a G. Blin. *RDF Database Systems: Triples Storage and SPARQL Query Processing. RDF Database Systems: Triples Storage and SPARQL Query Processing.* Waltham: Elsevier, 2014.

Daga, E., L. Panziera, a C. Pedrinaci. „BASILar Approach for Building Web APIs on top of SPARQL Endpoints.“ *CEUR Workshop Proceedings* 1486 (2015): 22-32.

Daquino, M., I. Heibi, P. Silvio, a D. Shotton. „Creating Restful APIs over SPARQL endpoints with RAMOSE.“ *arXiv.org.* 20. 9 2020. https://arxiv.org/ftp/arxiv/papers/2007/2007.16079.pdf (přístup získán 20. 9 2020).

Feigenbaum, L., G., T. Williams, K.,G. Clark, a E. Torres. *SPARQL 1.1 Protocol.* 21. 3 2013. https://www.w3.org/TR/sparql11-protocol/ (přístup získán 13. 2 2021).

Gandon, F., a G. Schreiber. *RDF 1.1 XML Syntax.* 25. 2 2014. https://www.w3.org/TR/rdf-syntax-grammar/ (přístup získán 20. 1 2021).

Garijo, D., a M. Osorio. „OBA: An Ontology-BasedFrame-work for Creating REST APIs for Knowledge Graphs.“ *Springer: Lecture Notes in Computer Science*, č. 12507 (2020): 48-64.

Garrote, A., a M. N. M. García. „RESTful writable APIs for the web of Linked Data usingrelational storage solutions.“ *Linked Data on the Web (LDOW2011).* 29. 3 2011. http://events.linkeddata.org/ldow2011/papers/ldow2011-paper04-garrote.pdf (přístup získán 20. 9 2020).

Gearson, P., A. Passant, a A. Polleres. *SPARQL 1.1 Update.* 21. 3 2013. https://www.w3.org/TR/2013/REC-sparql11-update-20130321/ (přístup získán 23. 1 2021).

Harris, S., a A. Seaborne. *SPARQL 1.1 Query Language. W3C Recommendation.* 21. 3 2013. https://www.w3.org/TR/sparql11-query/ (přístup získán 23. 1 2021).

Hausenblas, M., a J. G. Kim. *5 ★ Open Data.* 22. 1 2012. 2020 (přístup získán 19. 9 2020).

Hawke, S., I. Herman, G. Schreiber, a D. Wood. *“RDF Graph Identification”.* 15. 8 2012. https://www.w3.org/2012/08/RDFNG.html# (přístup získán 20. 2 2021).

Hopkinson, I., S. Maude, a M. Rospocher. „A simple API to the KnowledgeStore.“ *CEUR Workshop Proceedings* 1268 (2014): 7-12.

Insight Centre for Data Analytics. *The Linked Open Data Cloud.* 2021. https://lod-cloud.net/ (přístup získán 27. 1 2021).

Kellog, G, P. Champin, a D Longley. *JSON-LD 1.1.* 16. 7 2020. https://www.w3.org/TR/json-ld11/ (přístup získán 1. 20 2021).

Kendig, C. E. „What is Proof of Concept Research and how does it Generate Epistemic and Ethical Categories for Future Scientific Practice?“ *Science and Engineering Ethics* 22 (2015): 735-753.

Labra-Gayo, J. E., E. Prud'hommeaux, I. Boneva, and D. Kontokostas. "Challenges in RDF Validation." In *Current Trends in Semantic Web Technologies: Theory and Practice*, by G Alor-Hernández, J. L. Sánchez-Cervantes, A. Rodríguez-González, & R. Valencia-García, 121-151. Cham: Springer Nature Switzerland AG, 2019.

Lassila, O., a R. R. Swick. *Resource description framework (RDF) model and syntax specification.* 22. 2 1999. https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ (přístup získán 18. 1 2021).

Le, T., a R. Walker. *Linked Data API.* 3. 6 2020. https://documentation.ardc.edu.au/display/DOC/Linked+Data+API#LinkedDataAPI-Endpointtemplates (přístup získán 29. 1 2021).

McGuinness, D. L., a F. V. Harmelen. *WL Web Ontology Language Overview.* 10. 2 2004. https://www.w3.org/TR/owl-features/ (přístup získán 18. 1 2021).

Microsoft. *What's new in .NET 5.* 31 11 2020. https://docs.microsoft.com/en-us/dotnet/core/dotnet-five (accessed 3 2021, 15).

Miles, A., a S. Bechhofer. *SKOS Simple Knowledge Organization System Namespace Document – HTML Variant.* 18. 8 2009. https://www.w3.org/2009/08/skos-reference/skos.html (přístup získán 30. 1 2021).

Oxford Learner's Dictionary. *proof of concept.* 2020. https://www.oxfordlearnersdictionaries.com/definition/english/proof-of-concept (přístup získán 28. 11 2020).

Penuela, M. A., a R. J. Hoekstra. „grlc Makes GitHub Taste Like Linked Data APIs." V *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers*, autor: H. Sack, G. Rizzo, N. Steinmetz, D. Mladenić, S. Auer, & C. Lange, 342-353. Springer, 2016.

Prud'hommeaux, E., a A. Seaborne. *SPARQL Query Language for RDF.* 15. 1 2008. https://www.w3.org/TR/rdf-sparql-query/ (přístup získán 20. 9 2020).

Prud'hommeaux, E., a G. Carothers. *RDF 1.1 Turtle.* 25. 2 2014. https://www.w3.org/TR/turtle/ (přístup získán 20. 1 2021).

Schröder, M., J. Hees, B. Ansgar, D. Ewert, P Klotz, a S. Stadtmüller. „Simplified SPARQL REST APICRUD on JSON Object Graphs via URI Paths." *Springer: Lecture Notes in Computer Science* 11155 (2018): 40-45.

Seaborne, A. *SPARQL 1.1 Property Paths.* 26 1 2010. https://www.w3.org/TR/sparql11-property-paths/ (accessed 2 23, 2021).

Seaborne, A., a další. *SPARQL Update.* 15. 6 2008. https://www.w3.org/Submission/SPARQL-Update/ (přístup získán 9. 2 2021).

Seaborne, A., a G. Manjunath. *SPARQL/Update: a language for updating RDF graphs.* 24. 4 2007. https://www.hpl.hp.com/techreports/2007/HPL-2007-102.pdf (přístup získán 27. 1 2021).

Sikos, L. *Mastering Structured Data on the Semantic WebFrom HTML5 Microdata to Linked Open Data.* New York: Apress, 2015.

The Linked Open Data Cloud. *The Linked Open Data Cloud.* 2021. https://lod-cloud.net/ (přístup získán 27. 1 2021).

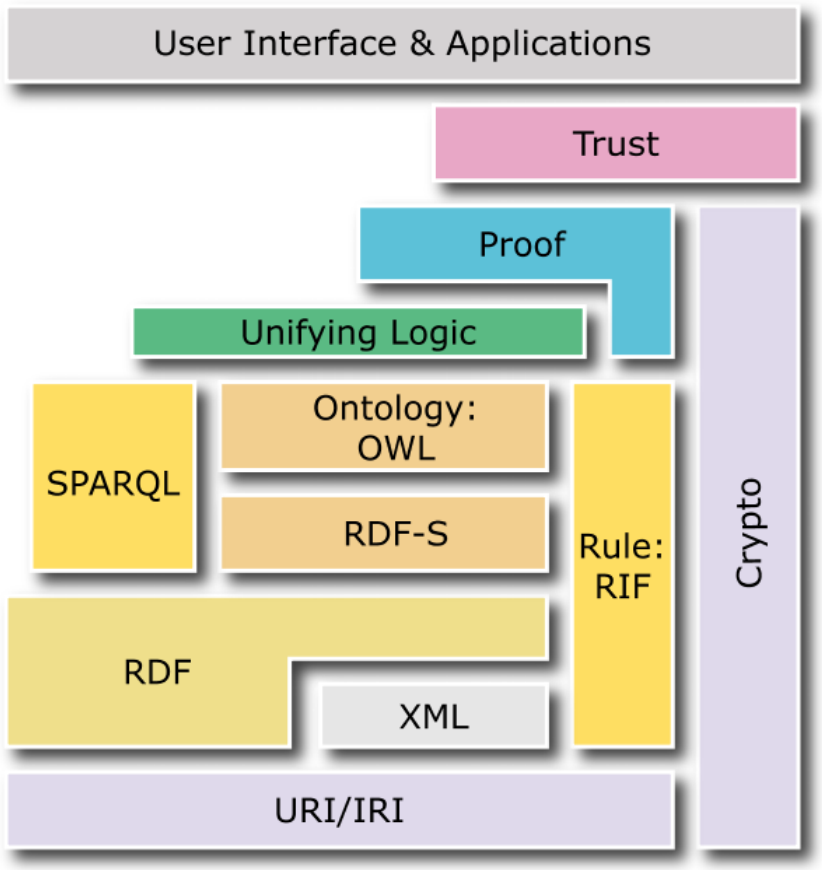
Universität Leipzig. *Ontology driven REST-API.* 2018. https://pcai042.informatik.uni-leipzig.de/~jf17a/ (přístup získán 30. 1 2021).

Vahidalizadehdizaj, M., T. Lixin, J. Jigar, a A. Agrawat. „A new plugin to support new relations in Protege.“ *6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2015: 1-6.

Wilde, E., a M. Hausenblas. „RESTful SPARQL? You Name It! : Aligning SPARQL with REST and Resource Orientation.“ *WEWST*, 2009: 39-43.

6. Přílohy

Příloha 1: Semantic Web Cake



(Curé a Blin 2014: 42).

Příloha 2: Příkladná tabulka obecně známých prefixů a jím příslušících jmenných prostorů

Alias	Namespace
prefix :	<http://example.org/>
prefix cex:	<http://purl.org/weso/computex/ontology#>
prefix cdt:	<http://example.org/customDataTypes#>
prefix dbr:	<http://dbpedia.org/resource/>
prefix ex:	<http://example.org/>
prefix qb:	<http://purl.org/linked-data/cube#>
prefix org:	<http://www.w3.org/ns/org#>
prefix owl:	<http://www.w3.org/2002/07/owl#>
prefix rdf:	<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs:	<http://www.w3.org/2000/01/rdf-schema#>
prefix schema:	<http://schema.org/>
prefix sh:	<http://www.w3.org/ns/shacl#>
prefix sx:	<http://shex.io/ns/shex#>

(Labra-Gayo a kol. 2019:10).

Příloha 3: Porovnání vybraných typů strukturovaných RDF formátů (Addlesee 2018).

RDF/XML

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:ns0="http://dbpedia.org/ontology/"
        xmlns:schema="http://schema.org/"
        xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">

  <foaf:Person rdf:about="http://dbpedia.org/resource/Bob_Marley">
    <rdfs:label xml:lang="en">Bob Marley</rdfs:label>
    <rdfs:label xml:lang="fr">Bob Marley</rdfs:label>
    <rdfs:seeAlso rdf:resource="http://dbpedia.org/resource/Rastafari"/>
    <ns0:birthPlace>
      <schema:Country rdf:about="http://dbpedia.org/resource/Jamaica">
        <rdfs:label xml:lang="en">Jamaica</rdfs:label>
        <rdfs:label xml:lang="it">Giamaica</rdfs:label>
        <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#float">17.9833</geo:lat>
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#float">-76.8</geo:long>
        <foaf:homepage rdf:resource="http://jis.gov.jm/"/>
      </schema:Country>
    </ns0:birthPlace>

  </foaf:Person>

</rdf:RDF>
```

N-TRIPLES

```
(1) <http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
(2) <http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/2000/01/rdf-schema#label> "Bob Marley"@en .
(3) <http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/2000/01/rdf-schema#label> "Bob Marley"@fr .
(4) <http://dbpedia.org/resource/Bob_Marley> <http://www.w3.org/2000/01/rdf-schema#seeAlso> <http://dbpedia.org/resource/Rastafari> .
(5) <http://dbpedia.org/resource/Bob_Marley> <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Jamaica> .
(6) <http://dbpedia.org/resource/Jamaica> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Country> .
(7) <http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2000/01/rdf-schema#label> "Jamaica"@en .
(8) <http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2000/01/rdf-schema#label> "Giamaica"@it .
(9) <http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "17.9833"^^<http://www.w3.org/2001/XMLSchema#float> .
(10) <http://dbpedia.org/resource/Jamaica> <http://www.w3.org/2003/01/geo/wgs84_pos#long> "-76.8"^^<http://www.w3.org/2001/XMLSchema#float> .
(11) <http://dbpedia.org/resource/Jamaica> <http://xmlns.com/foaf/0.1/homepage> <http://jis.gov.jm/> .
```

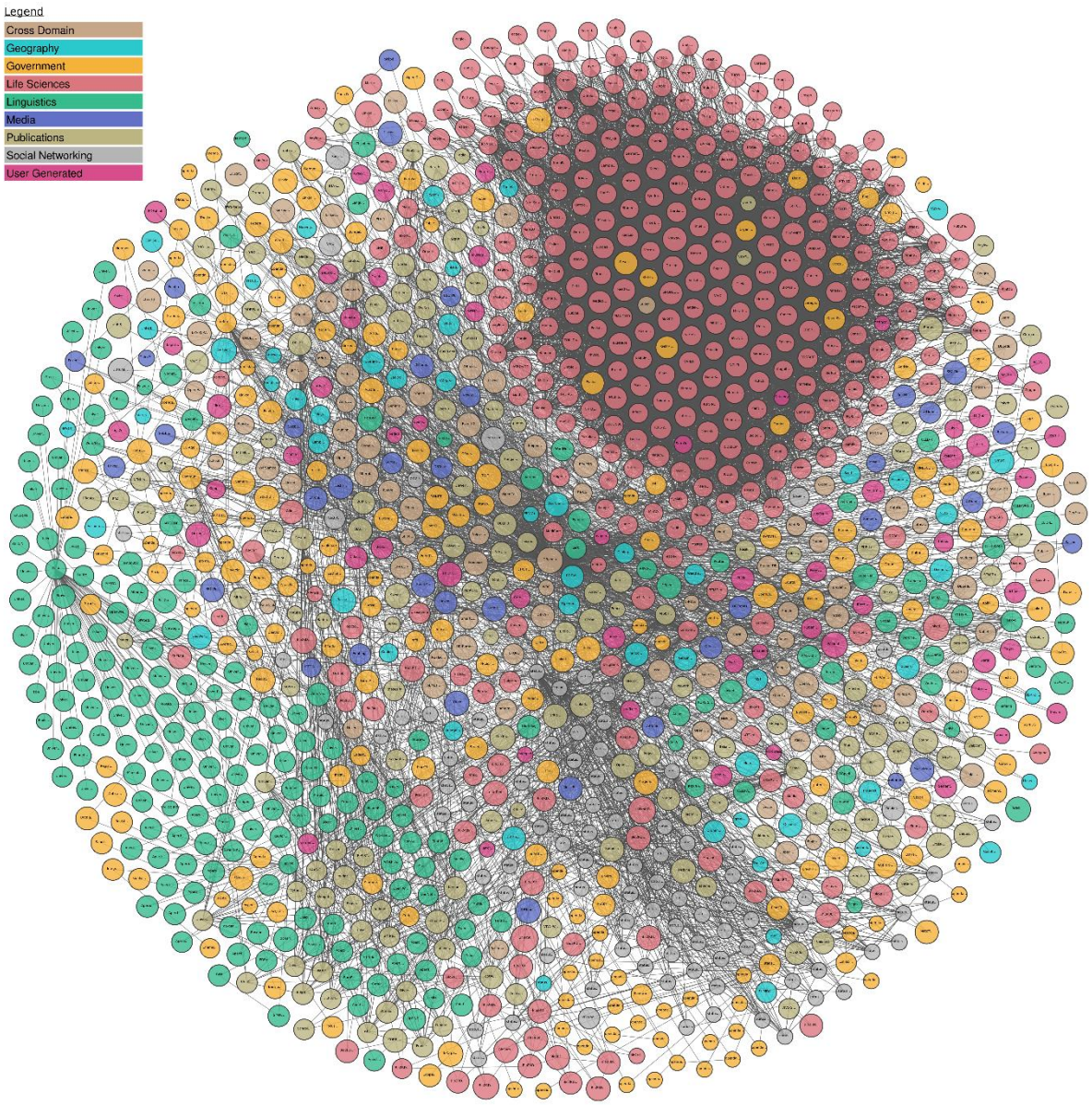
TUTRLE

```
@prefix dbr:    <http://dbpedia.org/resource/> .
@prefix dbo:    <http://dbpedia.org/ontology/> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix schema: <http://schema.org/> .

dbr:Bob_Marley
    a foaf:Person ;
    rdfs:label "Bob Marley"@en ;
    rdfs:label "Bob Marley"@fr ;
    rdfs:seeAlso dbr:Rastafari ;
    dbo:birthPlace dbr:Jamaica .

dbr:Jamaica
    a schema:Country ;
    rdfs:label "Jamaica"@en ;
    rdfs:label "Giamaica"@it ;
    geo:lat "17.9833"^^xsd:float ;
    geo:long "-76.8"^^xsd:float ;
    foaf:homepage <http://jis.gov.jm/> .
```

JSON-LD

```
[
  {
    "@id": "http://dbpedia.org/resource/Bob_Marley",
    "@type": ["http://xmlns.com/foaf/0.1/Person"],
    "http://www.w3.org/2000/01/rdf-schema#label": [
      {"@value": "Bob Marley", "@language": "en"},
      {"@value": "Bob Marley", "@language": "fr"}
    ],
    "http://www.w3.org/2000/01/rdf-schema#seeAlso": [{"@id": "http://dbpedia.org/resource/Rastafari"}],
    "http://dbpedia.org/ontology/birthPlace": [{"@id": "http://dbpedia.org/resource/Jamaica"}]
  },
  {
    "@id": "http://dbpedia.org/resource/Jamaica",
    "@type": ["http://schema.org/Country"],
    "http://www.w3.org/2000/01/rdf-schema#label": [
      {"@value": "Jamaica", "@language": "en"},
      {"@value": "Giamaica", "@language": "it"}
    ],
    "http://www.w3.org/2003/01/geo/wgs84_pos#lat": [
      {"@value": "17.9833", "@type": "http://www.w3.org/2001/XMLSchema#float"}
    ],
    "http://www.w3.org/2003/01/geo/wgs84_pos#long": [
      {"@value": "-76.8", "@type": "http://www.w3.org/2001/XMLSchema#float"}
    ],
    "http://xmlns.com/foaf/0.1/homepage": [{"@id": "http://jis.gov.jm/"}]
  },
  {"@id": "http://dbpedia.org/resource/Rastafari"},
  {"@id": "http://jis.gov.jm/"},
  {"@id": "http://schema.org/Country"},
  {"@id": "http://xmlns.com/foaf/0.1/Person"}
]
```

Příloha 4: The Linked Open Data Cloud



The Linked Open Data Cloud (2021).

Příloha 5: Ukázka konfiguračního souboru pro SPARQL endpoint *Portálu otevřených dat*.


```
{
  "endpointName": "datagov",
  "endpointUrl": "https://data.gov.cz/sparql",
  "defaultGraph": "http://eurovoc.europa.eu",
  "namedGraphs": [
    {
      "graphName": "school",
      "uri": "https://data.gov.cz/zdroj/datové-sady/00551023/861588425/41f9841463beb472c92f62fff43b15dd"
    },
    {
      "graphName": "emplpolicy",
      "uri": "https://data.gov.cz/zdroj/datové-sady/00551023/861588425/823171dd619cb29965ec368c25083ff0"
    },
    {
      "graphName": "regions",
      "uri": "https://data.gov.cz/zdroj/datové-sady/00551023/861588425/61963903d713a0173320878b215395f5"
    },
    {
      "graphName": "cities",
      "uri": "https://data.gov.cz/zdroj/datové-sady/00551023/861588425/46c8c042160be5bbb669fd0b30df1416"
    },
    {
      "graphName": "languages",
      "uri": "https://data.gov.cz/zdroj/datové-sady/00551023/861588425/4362f44ca6369c5ceaea9059c0dc3652"
    },
    {
      "graphName": "graduation",
      "uri": "https://data.gov.cz/zdroj/datové-sady/00551023/861588425/fa42aa43b517e640b4936a1fd53fb9f0"
    }
  ],
  "namespaces": [
    {
      "prefix": "eu",
      "uri": "http://eurovoc.europa.eu/"
    }
  ],
  "supportedMethods": {
    "sparql1.0": "yes",
    "sparql1.1": "no"
  },
  "entryClass": [
    {
      "graphName": "default",
      "command": "SELECT DISTINCT ?s WHERE {?s <http://www.w3.org/ns/dqv#computedOn>
                  <https://data.gov.cz/zdroj/katalog/NKOD>}"
    },
    {
      "graphName": "school",
      "command": "SELECT ?s WHERE {?s a ?o}"
    }
  ],
  "entryResource": [
    {
      "graphName": "default",
      "command": "SELECT DISTINCT ?s WHERE { ?s ?p ?o }"
    },
    {
      "graphName": "school",
      "command": "SELECT ?s WHERE { ?s ?p ?o }"
    }
  ]
}
```

Příloha 6: Ukázka konfiguračního souboru pro SPARQL endpoint *DBpedia*.


```
{
   "endpointName": "dbpedia",
   "endpointUrl": "https://dbpedia.org/sparql",
   "defaultGraph": "http://dbpedia.org",
   "namedGraphs": [
   ],
   "namespaces": [
   ],
   "supportedMethods": {
     "sparql1.0": "yes",
     "sparql1.1": "no"
   },
   "entryClass": [
     {
       "graphName": "default",
       "command": "SELECT DISTINCT ?s WHERE { ?s <http://www.w3.org/1999/02/22-rdf-syntax-
                    ns#type> <http://www.w3.org/2002/07/owl#Class> }"
     }
   ],
   "entryResource": [
     {
       "graphName": "default",
       "command": "SELECT distinct ?s WHERE { ?s ?p <http://www.w3.org/2002/07/owl#Thing> }"
     }
   ]
}
```

Příloha 7: Ukázka konfiguračního souboru pro testovací SPARQL endpoint.


```
{
  "endpointName": "virtuoso",
  "endpointUrl": "http://virtuoso:8890/sparql",
  "defaultGraph": "http://localhost:8890/ovm",
  "namedGraphs": [
    {
      "graphName": "ovm",
      "uri": "http://localhost:8890/ovm"
    },
    {
      "graphName": "lexvo",
      "uri": "http://localhost:8890/lexvo"
    },
    {
      "graphName": "test",
      "uri": "http://localhost:8890/test"
    }
  ],
  "namespaces": [
    {
      "prefix": "ovmos",
      "uri": "https://linked.opendata.cz/zdroj/ovm/osoba/"
    },
    {
      "prefix": "eks",
      "uri": "https://linked.opendata.cz/zdroj/ekonomický-subjekt/"
    },
    {
      "prefix": "schranky",
      "uri": "https://linked.opendata.cz/zdroj/datové-schránky/"
    },
    {
      "prefix": "char",
      "uri": "http://lexvo.org/id/char/"
    }
  ],
  "supportedMethods": {
    "sparql1.0": "yes",
    "sparql1.1": "yes"
  },
  "entryClass": [
    {
      "graphName": "default",
      "command": "SELECT DISTINCT ?s where {?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                  <http://www.w3.org/2002/07/owl#Thing>}"
    },
    {
      "graphName": "ovm",
      "command": "SELECT DISTINCT ?s where {?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                  <http://www.w3.org/2002/07/owl#Thing>}"
    },
    {
      "graphName": "lexvo",
      "command": "SELECT DISTINCT ?s where {?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                  <http://www.w3.org/2002/07/owl#Class>}"
    }
  ],
  "entryResource": [
    {
      "graphName": "default",
      "command": "SELECT DISTINCT ?s WHERE { ?s ?p ?o }"
    },
    {
      "graphName": "ovm",
      "command": "SELECT DISTINCT ?s WHERE { ?s ?p ?o }"
    },
    {
      "graphName": "lexvo",
      "command": "SELECT DISTINCT ?s WHERE { ?s ?p ?o }"
    },
    {
      "graphName": "test",
      "command": "SELECT DISTINCT ?s WHERE { ?s ?p ?o }"
    }
  ]
}
```