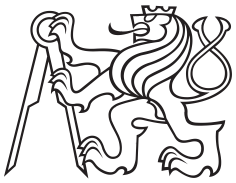
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

SPARQL rozhraní pro Apache Cassandra triplestore

Daniel Borner

Vedoucí: Ing. Václav Jirkovský, Ph.D.
Obor: Otevřená informatika
Studijní program: Software
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Borner** Jméno: **Daniel** Osobní číslo: **483607**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

SPARQL rozhraní pro Apache Cassandra triplestore

Název bakalářské práce anglicky:

SPARQL engine for Apache Cassandra triplestore

Pokyny pro vypracování:

Cílem práce je vytvoření systému, který bude sloužit jako rozhraní mezi dotazy v jazyce SPARQL a proprietárním RDF-triplestoru vytvořeného pomocí NoSQL databáze Apache Cassandra. Triplestore je optimalizovaný pro uložení časových řad sbíraných jako měření z výrobní linky. Pro přístup k databázi je bude použit CQL dotazovací jazyk. Implementované rozhraní by mělo poskytnout množinu základních typů dotazů jazyka SPARQL a v rámci práce bude specifikováno, jaká omezení tento přístup má zejména z důvodu architektury a způsobu uložení dat v triplestoru oproti standardu jazyka SPARQL.
Student se v rámci práce seznámí s jazykem SPARQL a technologickým zásobníkem sémantického webu, se systémem Apache Cassandra a možnostmi dotazování. Dále zhodnotí možnosti překladu SPARQL dotazů do CQL jazyka na základě souvisejícího schématu databáze a definuje množinu proveditelných SPARQL dotazů. Následně implementuje dotazovací engine ve vhodném programovacím jazyce.
Navržené a implementované řešení bude ověřeno na datech sbíraných z flexibilní výrobní linky Testbedu pro Průmysl 4.0 ČVUT (alternativně ze senzorů malé vodní elektrárny). Ověřována bude správnost a úplnost vrácených záznamů.

Seznam doporučené literatury:

[1] B. DuCharme, - Learning SPARQL: querying and updating with SPARQL 1.1 - O'Reilly Media, Inc. - 2013.
[2] J Hendler a kol. - Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL - Morgan & Claypool, 2020.
[3] E. Hewitt - Cassandra: the definitive guide - O'Reilly Media, Inc., 2010.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Václav Jirkovský, Ph.D. katedra kybernetiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Václav Jirkovský, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Moje poděkování patří Ing. Václavu Jirkovskému, Ph.D. za cenné rady, věcné připomínky, vstřícnost při konzultacích, odbornou pomoc a vedení, které mi poskytoval při vypracování mé bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem č. 1/20 o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze 20. května 2022

Abstrakt

Cílem této bakalářské práce bylo vytvořit rozhraní pro překlad SPARQL dotazů do jazyka CQL pro dotazování nad Cassandra triplestorem. První část práce seznamuje s základními pojmy důležitými pro zpracovávané téma. Druhá část předkládá rešerši současných řešení. Třetí část se zabývá analýzou možností překladu mezi jazyky SPARQL a CQL. V poslední části je popsána implementace rozhraní. Její součástí je popis testování aplikace za využití dat ze senzorů malé vodní elektrárny. V závěru práce je provedeno zhodnocení přínosu aplikace a návrhu na její eventuální vylepšení.

Klíčová slova: SPARQL, CQL, Apache Cassandra, RDF, ontologie, sémantický web

Vedoucí: Ing. Václav Jirkovský, Ph.D.
Český institut informatiky, robotiky a kybernetiky,
Jugoslávských partyzánů 1580,
Praha 6

Abstract

The aim of this bachelor thesis is to create an engine for translating SPARQL queries into CQL for Apache Cassandra triplestore. The first part of the thesis introduces the reader to important concepts necessary for this topic. The second part deals with a search for current solutions. The third part deals with the analysis of translation options between languages and the design of solutions. The last part describes the implementation of the interface. Part of the work is a description of application testing over test data from hydropower sensors. At the end of the work is an evaluation of benefits and suggestions for improvement.

Keywords: SPARQL, CQL, Apache Cassandra, RDF, ontology, semantic web

Title translation: SPARQL engine for Apache Cassandra triplestore

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Průmysl 4.0 [5] nebo také 4. průmyslová revoluce je pojem, který se v průmyslové automatizaci vyskytuje již několik desetiletí. Jako odborný termín se začal používat v roce 2011 v Německu. Toto označení se netýká jen přeměny automatizovaných výrobních linek na inteligentní kyber-fyzikální systémy, ale proniká i do všedního života, v němž přibývá velké množství automatizace.

K jeho popularitě přispěl také rozvoj internetu věcí (Internet of thing) [6] umožňující snadněji a levněji sbírat požadovaná data, které se pak využívají k následné analýze. Ta přináší cenné informace napomáhající k zefektivnění procesů [7].

Sběr dat je podstatnou částí průmyslu 4.0. Stroje a výrobní linky mají v sobě již zabudované senzory, případně jsou jimi dodatečně osazovány. Ty poté zaznamenávají informace o provozu strojů a linek, a tím produkují velké množství dat, které je nutné transformovat, uložit a analyzovat. Transformace a ukládání dat jsou procesy, jež probíhají v reálném čase. Analýza dat může být provedena nad již uloženými daty (například dlouhodobější statistiky) nebo zároveň s ukládáním, kde ukazuje krátkodobější data o stavu výroby, strojů, množství produkce, spotřebě energie a dalších informací důležitých pro zefektivnění výroby a šetření nákladů.

Pro zpracování velkého množství dat se mohou využít různé analytické nástroje. Pro jejich zpracování v reálném čase je určen například Apache Spark nebo Apache Storm. Existují také již vytvořené platformy, které se zabývají celým procesem od příjmu dat, přes jejich transformaci až k ukládání a analýze. Jedním z nich je Semantic Big Data Historian [3]. Tato platforma využívá pro ukládání RDF triplestore, do kterého data ukládá podle připravené ontologie. Z důvodu nekompatibility použité databáze pro triplestore a jazyka SPARQL určeného pro dotazování nad ontologiemi a triplestory je nutné implementovat řešení, které by tento problém vyřešilo.

1.1 Analýza zadání

Cílem této práce je návrh a vytvoření systému pro překlad dotazů z jazyka SPARQL do jazyka CQL a návrat dat v podobě, jako bychom přistupovali k RDF triplestoru.

Tato práce by měla v budoucnu sloužit jako rozšíření platformy Semantic Big data Historianu (SBDH), který je zaměřen na sběr a analýzu časových řad. V rámci práce budu analyzovat možnosti obou jazyků a navrhnu realizovatelné možnosti překladu, které budou v souladu s potřebami SBDH.

V teoretické části nejprve vysvětlím odborné pojmy používané v této bakalářské práci. Poté v praktické části navrhnu řešení zadaného tématu.

Část I

Teoretická část

Kapitola 2
Typy databází

Ukládání dat je důležitým aspektem vývoje softwaru a zpracování dat. Pro jejich uchování se používají databázové systémy [8]. Existuje velké množství způsobů, jak na logické vrstvě můžeme data modelovat. V principu je lze rozdělit na dva základní typy: relační databáze a NoSQL databáze. Pro relační databáze je vžitý název SQL databáze, neboť je možné se na jejich data dotazovat pomocí jazyka SQL. Od toho je také odvozen název druhé skupiny, do které spadají všechny ostatní typy databází [9].

2.1 RDBMS

Relational database management system (RDBMS) je software, jenž spravuje ukládaní dat. V praxi se používá velké množství systémů zahrnujících komerční i open-source sféru. Nejčastěji používanými komerčními relačními databázemi jsou Oracle a MSSQL. Nejpoužívanějšími open-source databázemi jsou MySQL a PostgreSQL.

2.1.1 Relace, atributy a záznamy

Základním prvkem tohoto typu databáze je relace (tabulka), která obsahuje sloupce (column), které se nazývají také atributy. Každý sloupec v tabulce obsahuje řádky (row) s ukládanými daty. Každý atribut má určený datový typ. Záznam v tabulce je tvořen n-ticí všech dat ze stejného řádku tabulky. V každé tabulce existuje atribut označený jako primární klíč (primary key). V praxi je takto označován sloupec s unikátními hodnotami, je možné ale také využít již existující atribut nebo skupinu atributů, které jsou v rámci tabulky unikátní. Primární klíč slouží jako jednoznačný identifikátor dat n-tice v tabulce a zároveň jako ukazatel, chceme-li propojit n-tici z jedné tabulky s n-ticí z druhé tabulky [10].

2.2 NoSQL databáze

NoSQL databáze je pojem zaštiťující několik způsobů uchovávání dat nezakládající se na relačním modelu. Nejčastěji se využívá databáze typu

Column-oriented, Key-Value, Document, Graph. V některých případech toto dělení nemusí být přesné, protože jedna databáze může aplikovat více přístupů k ukládání zároveň nebo má schopnost ukládat data více způsoby [11].

2.2.1 Obecné rozdíly mezi NoSQL a RDBMS

Oba typy používaných databází se odlišují zejména v těchto vlastnostech:

- Škálovatelnost: V případě nedostatku výkonu máme dva způsoby jak navýšit výkon databáze.

  Při vertikálním škálování navyšujeme výkon současného zařízení tak, aby byl dostatečný. Jedná se o nejjednodušší způsob jak získat dostatečný výkon.

  Při horizontálním škálování nezvyšujeme výkon současného hardwaru, ale přidáváme další samostatný uzel (node), který spolupracuje s již běžícími uzly. Tento přístup je složitější na implementaci, protože musí mít přímou podporu v použitém databázovém softwaru. Výhodou je až neomezená možnost navyšování výkonu. Nevýhodou naopak ztráta některých vlastností a garantce práce s daty (ACID).

  Oba typy databází lze škálovat oběma způsoby. Rozdíl je v podpoře. Relační databáze oproti NoSQL nemají často přímou podporu horizontálního škálování, neboť tak původně nebyly navrženy. NoSQL naopak s horizontálním škálováním počítají od začátku, jelikož jsou určeny pro ukládání velkého množství dat.

- Konzistence dat: Je běžným standardem, že relační databáze podporují ACID teorém, tedy Atomicity, Consistency, Isolation, Durability (atomicita, konzistence, izolovanost, trvalost). Z důvodu přímé podpory horizontální škálovatelnosti nejsou tyto vlastnosti u NoSQL databází podporovány. Důvodem je možná ztráta výkonu databáze, protože ACID teorém vyžaduje logování transakcí napříč databází a s rostoucím počtem uzlů by extrémně rostla náročnost na správu databáze.

ACID je v tomto případě nahrazen teorémem CAP, tedy Consistency, Availability, Partition tolerance (konzistence, dostupnost, odolnost proti ztrátám zpráv) [12].

2.2.2 Databáze klíč-hodnota

Databáze typu klíč-hodnota (key-value databse) [13] je nejjednodušší z NoSQL databází. Celá se skládá pouze z těchto dvojic. Klíč je v tomto případě řetězec (string) unikátní v celé databázi. Hodnotou může být jakákoliv informace včetně složitějších datových typů jako list, pole, případně další dvojice klíč-hodnota. Příkladem může být databáze Redis[1].

[1] `https://redis.com`

2.2.3 Sloupcově orientované databáze

Databáze ukládající data po sloupcích (column-oriented databse) [14] je opačným přístupem k ukládání oproti databázím uchovávající data po řádcích (row-oriented databse), což je běžný přístup u relačních databází. Hlavním principem ukládání dat v column-oriented databázích je jejich shromažďování z jednoho sloupce v navazujících blocích paměti na rozdíl od row-oriented database, kde jsou v paměti data seskupená po řádcích. Tento přístup přináší několik výhod a nevýhod. Mezi výhody patří rychlost při čtení a zpracování dat týkající se práce se sloupci. Například při získávání průměrné hodnoty jednoho sloupce, případně zobrazení dat z několika málo sloupců z tabulky s mnoha sloupci. Nevýhodou jsou pomalé operace úpravy nebo přidávání nových řádků. Zástupcem této kategorie je Google Bigtable[2] nebo databáze Apache Cassandra, jíž se budu věnovat v kapitole 5 a jež je důležitou součástí této práce.

2.2.4 Dokumentově orientované databáze

Dokumentově orientované databáze (Document-oriented databse) [15] jsou specifickým druhem key-value databází. Data zde nejsou ukládána do tabulek, ale jsou ukládána v podobě dokumentu. To je datová struktura definovaná pomocí některého ze značkovacích jazyků. Jedná se například o XML, JSON, ale i PDF. Mezi nejpoužívanější dokumentově orientované databáze patří MongoDB[3].

2.2.5 Grafově orientované databáze

Grafově orientované databáze (graph-oriented databse) [16] jsou tvořeny strukturou složenou z uzlů (node) a hran (edge). Data jsou reprezentována pomocí orientovaného grafu. Uzly tvoří jednotlivé záznamy v databázi spolu s jejich vlastnostmi. Hrany v tomto případě popisují vztahy mezi uzly. Příkladem grafově orientované databáze je neo4j[4].

[2] `https://cloud.google.com/bigtable`
[3] `https://www.mongodb.com`
[4] `https://neo4j.com`

Kapitola 3

Ontologické jazyky a jejich zápis

Ukládat informace je možné mnoha způsoby. Výběr vhodného formátu záleží na účelu, k jakému chceme data používat. Způsob jak formalizovat zápis dat a vztahů mezi nimi, aby byly strojově zpracovatelné, nám nabízí koncept ontologie.

3.1 Ontologie

Ontologie (Ontology) [17] je pojem pocházející z filozofie. Tento název byl později použit v počítačových vědách a používá se jako označení pro modelování znalostí dané problematiky. Každý prvek ontologie (entity, vztahy) musí být definován a zároveň musí mít určené vztahy vůči ostatním entitám. Účelem ontologie je formálně popsat koncepty, data i jakékoliv jiné entity a zachytit vztahy mezi nimi pomocí modelu. Ontologie nám umožní formalizovaně ukládat tyto modely a podle potřeby je rozšiřovat, případně je spojovat s dalšími ontologiemi. Můžeme si ji představit jako orientovaný graf, jehož uzly tvoří data nebo nějaké entity a hrany představují vztahy mezi těmito entitami a daty.

3.2 Resource Description Framework

Resource Description Framework (RDF) [18] (Framework popisu zdrojů) je specifikace vytvořená World Wide Web Consorcium (W3C) za účelem specifikace zápisu ontologií, prostřednictvím kterého je možné snadno ukládat data tak, aby byla čitelná pro počítače a lidi zároveň. Tato specifikace zápisu proniká v poslední době do různých odvětví. V běžné praxi se s ní můžeme setkat například na Wikipedii, v níž jsou z části zobrazována data z projektu Wikidata[1], jenž informace uchovává právě ve formátu RDF. Podobný systém využívá Google při nabízení některých vyhledávaných výsledků. V České republice je v tomto formátu poskytována část dat České obchodní inspekce a Ministerstva financí ČR[2].

[1] `https://www.wikidata.org/wiki/Wikidata:Main_Page`

[2] `https://opendata.mfcr.cz`

3.3 Internationalized Resource Identifier

Internationalized Resource Identifier (IRI) [19] (Mezinárodní identifikátor zdroje) je identifikátor založený na Uniform Resource Identifier (URI) [20] (Jednotný identifikátor zdroje), což je textový řetězec sloužící jako identifikátor používaný v počítačových vědách v rámci různých sítí. URI je nejvýše postavený identifikátor, ze kterého je odvozeno například URL [21], používané k identifikaci webových stránek. IRI se od URI liší zejména bohatší znakovou sadou, neboť URI využívá pouze znakovou sadu US-ASCII a IRI používá Unicode (ISO 10646).

3.4 RDF Graf

Specifikace RDF je tvořena orientovaným grafem skládajícím se z na sebe navazujících trojic: předmět-predikát-objekt (subject-predicate-object). Protože se jedná o orientovaný graf, každý předmět a objekt je uzlem grafu a predikát je orientovanou hranou. Jednotlivé na sebe navazující trojice tvoří graf, který popisuje vztah mezi konkrétními daty. V závislosti na tom, jak na data pohlížíme, může být na uzel nahlíženo jako na předmět nebo objekt, případně zdroj (resource) nebo literál. V praxi se k zápisu RDF využívá například značkovací jazyk XML [1].

3.4.1 Uzel

Uzel, tedy předmět nebo objekt, je tvořen pomocí zdroje s IRI, literálu nebo prázdného uzlu (blanked node). Prázdný uzel má svůj lokální identifikátor, ale nemá přiřazené žádné IRI. Využíváme ho při připojení více objektů skrze daný predikát do předmětu (viz obrázek 3.1). Uzel obsahující IRI představuje reálný objekt zaznamenaný v našem grafu, na nějž jsou navázány další objekty a literály pomocí predikátů (orientovaných hran grafu). Literál označuje základní hodnoty: textové řetězce, čísla, datumy [22] [1].

3.4.2 Predikát

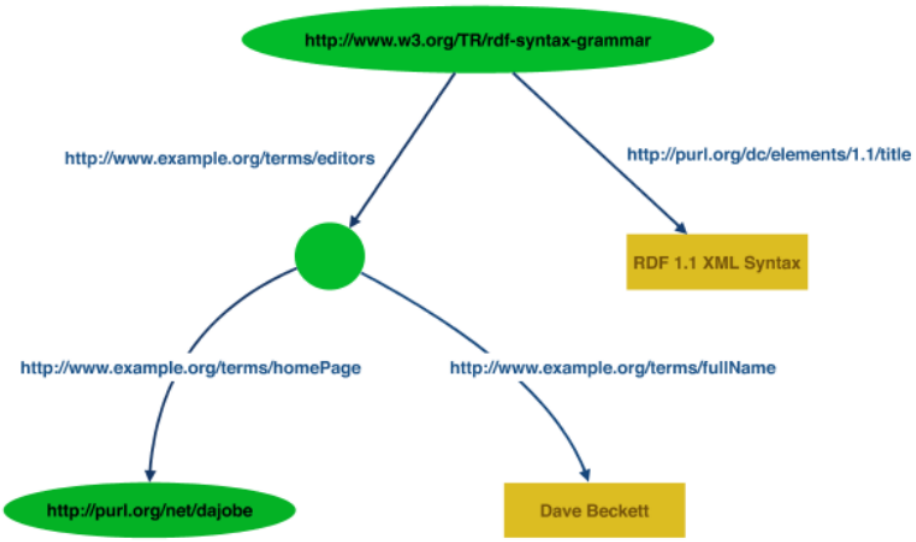
Predikát v RDF představuje vlastnost se vždy přiřazeným IRI. Jeho účelem je propojení dvou uzlů pomocí specifikovaného vztahu nebo připojení nějaké vlastnosti k uzlu.

3.4.3 Příklad

Na obrázku 3.1 vidíme RDF graf, v kterém elipsa označuje uzel definovaný pomocí IRI, kruh prázdný uzel a obdélník literál. Hrany mezi uzly jsou predikáty.

Níže je uvedený RDF graf zadefinovaný pomocí XML.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
```



Obrázek 3.1: Příklad RDF grafu [1]

```
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

3.5 RDFS

RDF Schema [2] je rozšířením specifikace RDF. V samotném RDF máme při tvorbě grafu volnost v tom jak graf vytvářet a jaké vztahy mezi jeho uzly používat. To přináší nevýhodu v podobě možné nekonzistence více grafů.

RDFS rozšiřuje původní formát o slovník základních vztahů používaných v grafech, specifikuje jak pracovat s třídami a vlastnostmi, říká jakým způsobem vytvářet vlastní ontologie a RDF slovníky.

Slovníkem v tomto případě rozumíme RDF graf specifikující konkrétní vztahy mezi uzly a vlastnosti jednotlivých uzlů v závislosti na budoucím použití. RDFS samo o sobě je slovníkem obsahujícím množinu základních vztahů pro tvorbu dalších slovníků.

3.5.1 Třídy

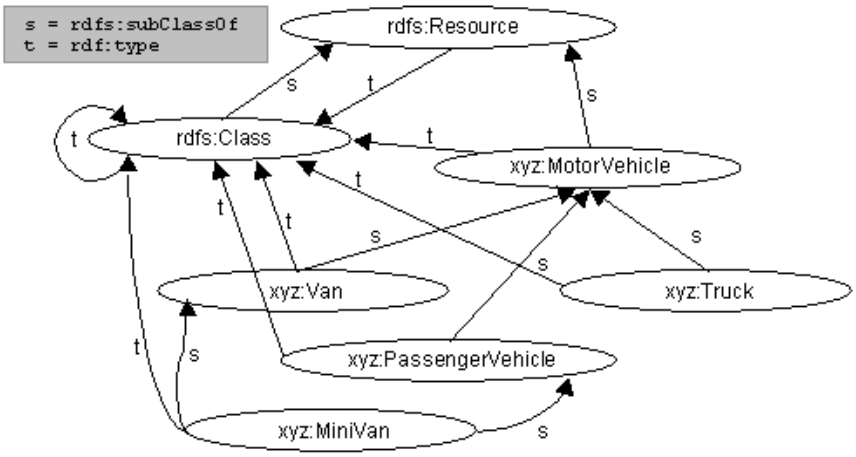
Všechno, co je součástí RDF grafu, lze rozdělit do tříd (class). Vše, co spadá do konkrétní třídy, nazýváme instancí třídy. Třídy samotné jsou instancí třídy Class. Níže uvádím část základních tříd popsaných v RDFS využívaných v mé práci.

- Resource (Zdroj) je základní třída RDFS. Všechno, co je součástí RDF grafu, je instancí třídy Resource. Je nadřazenou třídou, všechny ostatní třídy jsou podtřídou (subclass) třídy Resource. Třída Resource je zároveň instancí třídy Class.
- Class (Třída) je třída zaštiťující všechny ostatní třídy. Class je instancí třídy Class a zároveň podtřídou třídy Resource.
- Literal (Literál) je třída nadřazená všem literálům, tedy hodnotám jako jsou například textové řetězce, čísla a další. Literál je instancí třídy Class.
- DataType (Datový typ) je třída sdružující všechny datové typy použitelné jako literály. Tato třída je instancí a zároveň podtřídou třídy Class a každá instance třídy DataType je také podtřídou třídy Literal.
- Property (Vlastnost) je třída sdružující všechny vztahy mezi uzly RDF grafu. Je zároveň instancí třídy Class.

3.5.2 Vlastnosti

Vlastnostmi (Properties) jsou míněny všechny vztahy mezi uzly grafu. Je-li nějaký uzel označen jako podtřída jiného uzlu, je nutné oba uzly propojit příslušnou vlastností. Zde je výčet několika základních vlastností. Všechny níže uvedené jsou instancí třídy Property.

- Range (Rozsah) je vlastnost přiřazující jiné vlastnosti třídu, která specifikuje rozsah platných hodnot. Při zobrazení orientovaného grafu přiřazuje třída Range orientaci k vlastnosti pro určitý objekt.
- Domain (Doména) je vlastnost přiřazující k jiné vlastnosti třídu nebo třídy, na něž může být aplikována, tedy při zobrazení orientovaného grafu říká, z jakého předmětu vlastnost vychází.
- Type (Typ) se používá pro vyjádření, že jedna třída je instancí jiné. Tedy trojice Resource Type Class udává, že Resource je instancí Class.



Obrázek 3.2: Příklad základních konstrukcí RDFS s připojeným dalším slovníkem [2]

- SubClassOf specifikuje to že daná třída je podtřídou jiné. Zároveň zde platí tranzitivita, tedy je-li třída A podtřídou B a B je podtřídou C, pak A je podtřídou C.
- SubPropertyOf specifikuje hierarchii vlastností podobně jako subClassOf specifikuje hierarchii tříd. Opět zde platí tranzitivita.
- Label (Označení) slouží k přiřazení lidsky čitelného názvu k příslušnému uzlu. Vlastnost Range omezuje Label pouze na Literály. Vlastnost Domain omezuje přiřazení vlastnosti Label pouze ke členům třídy Resource (tedy na všechny objekty v RDF grafu).

Na obrázku 3.2 můžeme sledovat příklad RDF schématu využívající RDFS rozšíření. Názorně je zde ukázán vztah mezi třídami Resource a Class a zároveň rozšíření grafu o jiný slovník. V tomto případě je graf rozšířen o typy aut. Existuje velké množství volně dostupných slovníků, které lze používat pro tvorbu ontologií nebo je možné si vytvořit vlastní slovníky dle potřeby.

3.6 Web Ontology Language

Web ontology language (OWL) [23] je specifikace pro popisování ontologií. OWL vychází z RDF a RDFS, které v sobě zároveň implementuje. To umožňuje se nad daty dodržující tuto specifikaci dotazovat jazykem SPARQL, který je určen pro RDF. V kontextu RDFS se jedná o další slovník, jenž s využitím slovníků RDF a RDFS rozšiřuje možnosti modelování dat.

Kapitola 4
SPARQL

SPARQL [24] neboli "SPARQL Protocol and RDF Query Language"(jedná se o rekurzivní akronym) je dotazovací jazyk pro databáze ukládající data ve formátu RDF. Tento jazyk je vyvíjen skupinou RDF data working group. První verze byla vydána v roce 2008. Přestože z názvu vyplývá, že je určen pro dotazování nad RDF ontologiemi, je možné ho využít i na dotazování nad daty zapsanými v jiné specifikaci, jež vychází z RDF (RDFS nebo OWL). V současnosti rozlišujeme dvě souběžné verze tohoto jazyka. Jsou to SPARQL 1.1 a SPARQL 1.1 Update [25]. První se zaměřuje primárně na operace, které nemění data. Má jen velmi omezenou možnost jejich úpravy, jež byla přidána až s pozdějšími verzemi. Druhý jazyk je naopak zaměřen pouze na úpravu dat. Podporuje funkce mazání, vkládání, úpravy dat a také vkládání dalších grafů do stávajícího. V rámci této práce si vystačíme pouze s první verzí a dále tedy označením SPARQL budeme mít na mysli SPARQL 1.1.

4.1 Typy dotazů

SPARQL rozlišuje v rámci své gramatiky dva základy typy dotazů: QueryUnit a UpdateUnit. Toto dělení bylo zavedeno až s pozdější verzí jazyka, neboť dotazy typu update vyžadovaly jinou strukturu dotazů. UpdateUnit nabízí dotazy INSERT DATA a DELETE DATA. QueryUnit podporuje základní dotazy SELECT, CONSTRUCT, ASK a DESCRIBE. Pro potřeby této práce jsou relevantní pouze dotazy SELECT, INSERT DATA a DELETE DATA.

4.2 Obecná struktura dotazu

V rámci struktury dotazu pracujeme stejně jako v RDF s trojicemi předmět-predikát-objekt. Tyto trojice jsou porovnávány s vloženou ontologií, nad níž dotazy provádíme. V případě shodného překrytí trojic provádíme zadané operace. Tyto trojice je možné propojovat dohromady a tím popisovat specifičtější částí RDF grafu, z kterého si přejeme získat data.

Predikát je v tomto případě tvořen IRI, nebo proměnnou. Předmět a objekt je tvořen buď IRI, proměnnou, některým z literálů (String, Numeric, Boolean), nebo blank nodem. V případě všech tří termů, pokud pracujeme s IRI, máme

dvě možnosti jak s ním pracovat. První možnosti je vždy na přímo specifikovat term pomocí celého IRI. Druhou je zjednodušení pomocí konstruktu PREFIX. Prefixy specifikujeme na začátku dotazu. Značně zjednodušují následný zápis. Specifikace probíhá pomocí konstruktu PREFIX, předpony a IRI slovníku, který chceme použít. Ve zbytku dotazu poté pouze píšeme předponu a na ní napojujeme konkrétní název uzlu.

4.2.1 SELECT

Dotaz SELECT stejně jako v SQL je určen pro zobrazení dat na základě určitých kritérií. V dotazu specifikujeme trojice, které na sebe napojujeme. Zpět jsou poté vrácena všechny data, jež odpovídají zadanému grafu.

Příklad dotazu:

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

SELECT ?var
WHERE {
   ?var foaf:name "Petr"
}
```

Na začátku specifikujeme prefix pro následný zkrácený zápis. Po klauzuli SELECT určíme jaké proměnné z vrácených dat chceme zobrazit a nakonec v klauzuli WHERE určíme graf, který porovnáme s dotazovanou ontologií. Dotaz nám v tomto případě vrátí všechny uzly, jež na sobě mají predikát http://xmlns.com/foaf/0.1/name a jméno "Petr".

4.2.2 INSERT DATA a DELETE DATA

Oba dotazy mají prakticky totožnou strukturu, liší se pouze rozdílným klíčovým slovem (INSERT, nebo DELETE). Jejich účelem je vložení nových dat, respektive smazání starých v grafu, se kterým pracujeme. Musíme zde specifikovat konkrétní trojice bez použití proměnných. Tyto trojice jsou poté vloženy respektive smazány.

Příklad dotazu:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

DELETE DATA
{
  <http://example/book2> dc:title "David Copperfield" .
  <http://example/book2> dc:creator <http://example/EdmundWells> .
}
```

Tento dotaz nejprve odstraní z uzlu <http://example/book2> jeho název v podobě literálu a poté odebere i autora, který je zde v podobě dalšího uzlu. Obdobně bychom mohli data do grafu přidat pomocí dotazu uvedeného níže.

Příklad dotazu:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

INSERT DATA
{
  <http://example/book2> dc:title "David Copperfield" .
  <http://example/book2> dc:creator <http://example/EdmundWells> .
}
```

Kapitola 5
Apache Cassandra

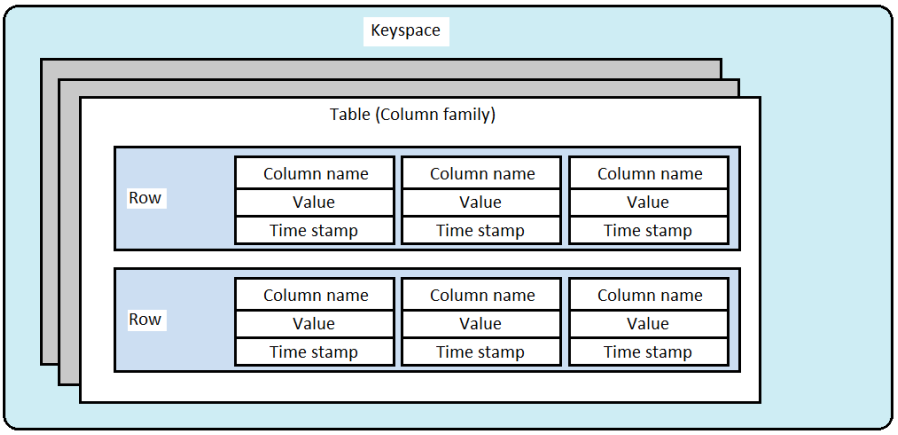
Apache Cassandra [26] je NoSQL databáze vyvíjená Facebookem, později otevřená jako open-source. V roce 2009 byla převzata Apache foundation. Jedná se o distribuovanou sloupcově orientovanou databázi principiálně založenou na technologiích Dynamo od Amazonu a Bigtable od Google. K výhodám této databáze patří zejména volnost v návrhu datového modelu. Při jeho vhodném definování také docílíme velké rychlost zápisu dat. Nevýhodou jsou značná omezení v dotazování se nad daty (viz DOPLNIT).

5.1 Keyspace

Cassandra nemá specifikovaný model, podle něhož bychom měli data ukládat. Jeho tvorba nám umožňuje velkou variabilitu návrhu. Základním identifikačním prvkem je keyspace. Jedná se o kontejner, který v sobě sdružuje tabulky sdílející stejné nastavení v rámci fungování clusteru. Součástí Keyspace je specifikovaný replication factor (replikační faktor), který udává na kolika uzlech v rámci clusteru mají být data replikována. Dalším nezbytným prvkem je replikační strategie (Replica placement strategy), jež je vybírána v závislosti na rozsahu clusteru. Volíme ji podle toho, zda běží databáze pouze v jednom data centru, nebo ve více data centrech zároveň. V závislosti na této skutečnosti jsou data replikována podle zvolené strategie. Data jedné běžící aplikace obvykle sdílejí stejný keyspace.

5.2 Tabulky

Každá tabulka ukládající data v Cassandře musí mít určený keyspace. V starší dokumentaci je pro tabulky (table)používáno označení "skupiny sloupců"-(Column family). V obou případech se jedná o synonymické pojmenování, tedy sdružení několika souvisejících sloupců. V tabulkách jsou data sdružována do řádků, v nichž má každý řádek své specifikované sloupce. Ty se mohou napříč řádku lišit. Schéma uspořádání keyspace, tabulek a jejich řádků je možné sledovat na obrázku 5.1.



Obrázek 5.1: Struktura databáze Cassandra

5.3 CQL

Cassandra query language (CQL) [27] je dotazovací jazyk vytvořený speciálně pro dotazování se této databáze. Svým zápisem je velice podobný jazyku SQL běžného u relačních databází. Protože Cassandra je určená pro práci s velkým množstvím dat, kvůli volnosti v ukládaných strukturách není propojování tabulek oproti klasickým relačním databázím povolené. Abychom nebyli tímto omezením limitováni, můžeme tomu v některých případech předejít vhodnou volbou datového modelu.

5.3.1 Konstrukce CQL dotazů

Pro potřeby této práce využiji tři typy dotazů, které nám CQL poskytuje. Stejně jako v jazyce SPARQL to jsou SELECT, INSERT a DELETE. V následujících ukázkách jsou části dotazů, ty které se nacházejí v < a > je nutné specifikovat a části ohraničené [a] jsou pro dotaz volitelné.

```
SELECT <column name>
FROM <keyspace>.<table name>
[WHERE <column name>='value'
[AND <column name>='value'] ALLOW FILTERING
```

V výše uvedeném příkladu můžeme sledovat podobnost s jazykem SQL. V tomto případě nejprve volíme sloupce, které chceme zobrazit a poté klauzulí WHERE specifikujeme podmínky pro jednotlivé sloupce. Konstrukce ALLOW FILTERING nám v Cassandře dovoluje filtrovaní dat ve všech sloupcích. Jedná se o explicitní povolení, jež slouží k upozornění, že provedení dotazu může být výpočetně náročné a pomalé.

```
INSERT INTO <keyspace>.<table name>
(<column1 name>, <column2 name>, ...)
```

```
VALUES (<value1>, <value2>, ...) IF NOT EXISTS
```

Při vkládání dat specifikujeme nejdříve keyspace a tabulku, potom vyjmenováváme slupce, do nichž chceme vložit data. V identickém pořadí poté následně vyjmenujeme vkládané hodnoty. Konstrukce IF NOT EXISTS zabraňuje vložení duplicitních dat.

```
DELETE FROM <keyspace>.<table name>
[WHERE <column name>='value'
[AND <column name>='value'] IF EXISTS
```

Mazání řádků je velice podobné příkazu SELECT. Je nutné identifikovat jednoznačně řádek, jenž chceme smazat. Abychom zabránili mazání neexistujících řádků, využijeme konstrukci IF EXISTS.

Kapitola 6
Semantic big data historian

Semantic big data historian (SBDH) [3] [28] [29] je platforma pro sběr časových řad z výrobních linek a dalších senzorů nasazených v průmyslu. Obecně jsou historiany programy pro sběr dat z běžícího procesu. Tato data jsou následně ukládána a analyzována. SBDH je složen ze čtyř vrstev, které společně tvoří celou platformu (viz 6.1).

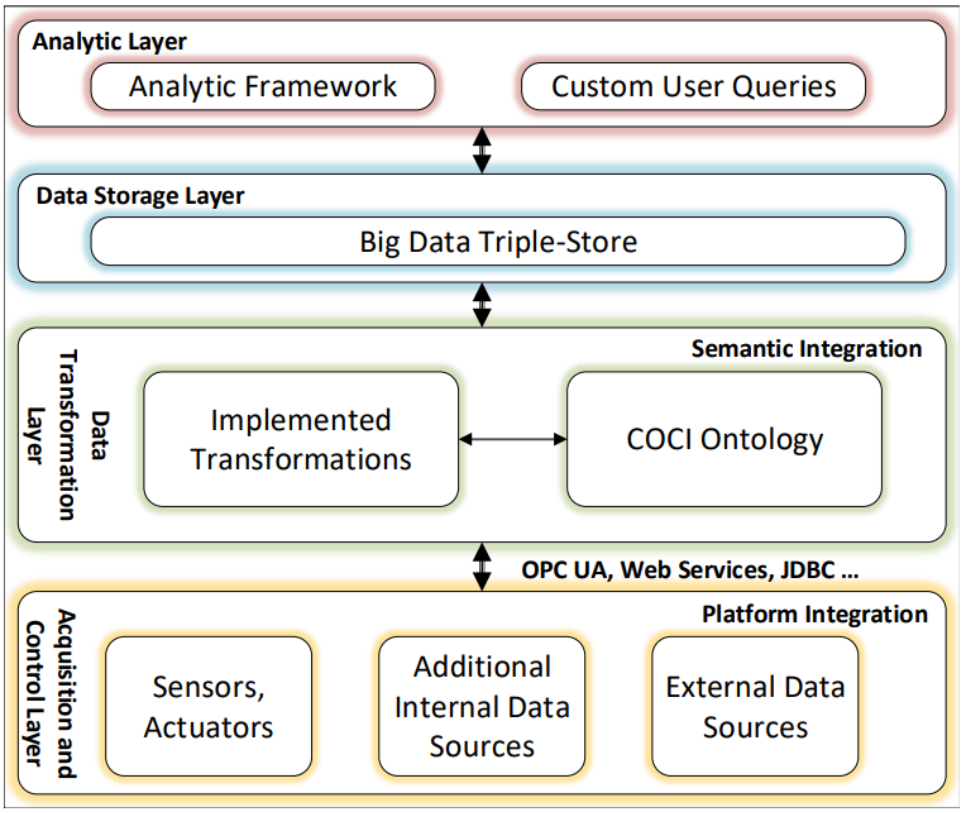
Nejnižší vrstva, Acqusition and control layer, sbírá data z jednotlivých senzorů a dalších zdrojů. Ty potom předává dále do Data transformation layer, v níž jsou data transformována dle ontologie do trojic, které jsou ukládány v třetí vrstvě - Data storage layer. V ní jsou data ukládána do triplestoru implementovaného v Apache Cassandra. Ta slouží pro dotazování se nad daty nebo k jejich další analýze pomocí Apache Spark a její Machine learning library [30].

6.1 Triplestore

Triplestory jsou databáze určené pro ukládání RDF trojic. Jedná se o specifický případ grafových databází. Přesto se od nich v některých částech významně liší. Triplestory i grafové databáze jsou zaměřeny na ukládání dat a vztahů mezi nimi. Grafové databáze mají variabilitu v objektech, které jsou tvořeny jednotlivými uzly, kdežto v triplestorech jsou uzly omezeny pouze na literály, nebo na IRI. Dalším významným rozdílem je možnost dotazovaní. Pro běžné grafové databáze existuje relativně velké množství dotazovacích jazyků s implementovanými grafovými algoritmy. Dotazy nad RDF triplestory je možné provádět pouze jazykem SPARQL. Případné další možnosti dotazování závisí na implementaci konkrétního triplestoru.

6.2 Cassandra v SBDH

Při vývoji SBDH byly pro ukládání nejdříve nasazeny databáze určené přímo k ukládání RDF trojic. Kvůli slabšímu výkonu, ukládání velkého množství dat a specializaci především na časové řady senzorických měření musely být nahrazeny jiným typem databáze. V tomto případě je triplestor



Obrázek 6.1: Jednotlivé vrstvy platformy SBDH [3]

implementován pomocí Apache Cassandry, která vykazuje velice dobré výkonnostní výsledky při zápisu dat.

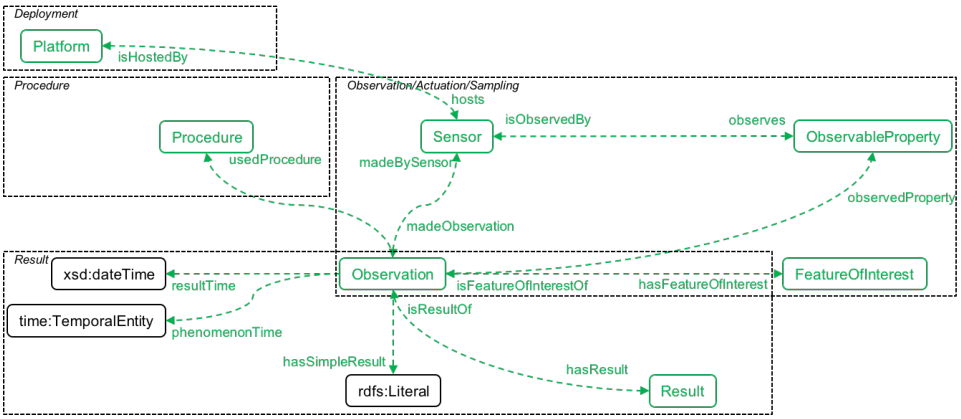
6.3 Hybrid SBDH Data Model

Existují dva základní způsoby jak ukládat data v triplestoru.

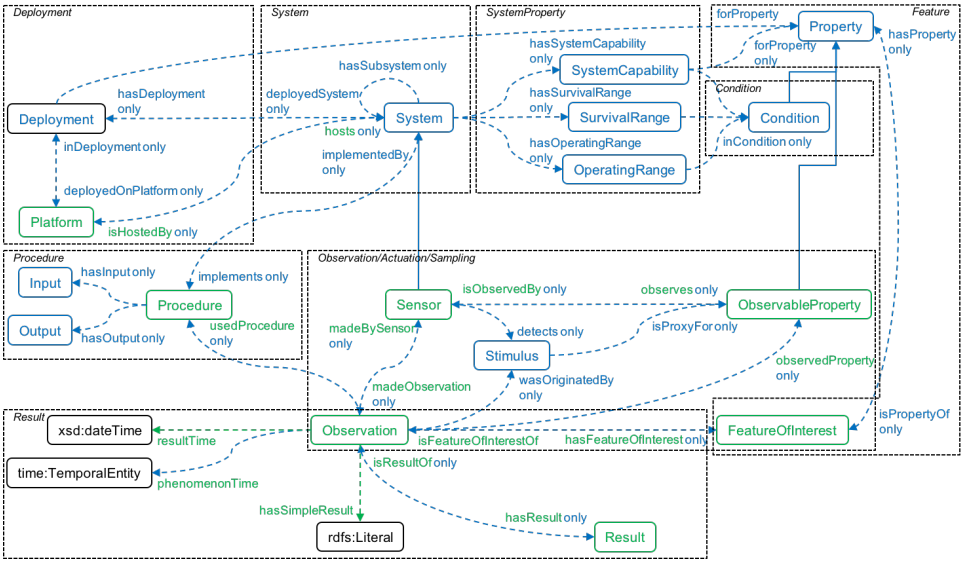
1. Single data model - jedná se o způsob uložení trojic v jedné velké tabulce. Data jsou tvořena předmětem a poté predikáty a objekty patřící k danému předmětu.

2. Vertical partitioning model - v tomto způsobu ukládání z důvodu snahy o homogenní rozdělení trojic do tabulek, jsou trojice rozděleny pomocí predikátu. Všechny trojice se stejným predikátem jsou poté ukládány do stejné tabulky.

V rámci SBDH se využívá vertical partitionig model a zároveň hybridní přístup zápisu měření dat, kde pro každý senzor je vytvořena tabulka s identifikátorem senzoru doplněna o příponu "_hasSimpleResult". V této tabulce jsou data ukládána ve dvojicích timestamp-value. Data z měření (časové řady) jsou ukládána v jednotlivých tabulkách podle konkrétních senzorů.



Obrázek 6.2: Diagram SOSA ontologie [4]



Obrázek 6.3: Diagram SSN ontologie (Součástí SSN je i SOSA) [4]

6.4 SSN a SOSA Ontologie

Jedním z mnoha RDF slovníků jsou SSN (Semantic Sensor Network) a SOSA (Sensor, Observation, Sample, and Actuator) [4] ontologie, jež se zaměřují na senzory, měření a aktuátory. Cílem SOSA ontologie je popsat senzory a akturátory, jejich měření a vztahy mezi nimi, měřenou veličinou (observable property) a platformou, na níž jsou tyto prvky uloženy. SSN Ontologie přímo importuje SOSA ontologii, využívá její prvky a dále ji rozšiřuje. Na obrázku 6.2 sledujeme architekturu SOSA ontologie a na obrázku 6.3 její rozšíření SSN.

V době hledání vhodné konceptualizace pro SBDH nebyla ještě SOSA ontologie vytvořena, a proto byla tedy navržena nová ontologie Cyber-Physical System Ontology for Components Integration (COCI) [3], jež těmto požadavkům vyhovuje a je založena právě na ontologii SSN.

Kapitola 7
Současná řešení

Tato kapitola se zabývá rešerší současných řešení a vyvozením závěrů pro potřeby bakalářské práce.

7.1 Překlad SPARQL dotazů do CQL

Základním tématem bakalářské práce je vytvoření rozhraní pro překlad mezi jazyky SPARQL a CQL. Nepodařilo se mi najít žádné relevantní články, které by o této problematice pojednávaly. Cílem mého průzkumu bylo najít publikace, jež se zaměřují na obecný překlad mezi těmito jazyky nebo najít alespoň specializovaná řešení části problému. Ta by mi posloužila jako inspirace, poněvadž výsledkem bakalářské práce by mělo být také specializované řešení, nikoliv obecný překlad.

7.2 Překlad SPARQL dotazů do SQL

Vzhledem k tomu, že jazyk SQL je jedním z nejpoužívanějších dotazovacích jazyků [31] a zároveň konstrukce dotazů je podobná CQL, proto považuji za rozumné prozkoumat i oblast překladů mezi SPARQL a SQL.

7.2.1 Semantics preserving SPARQL-to-SQL translation

Tato rozsáhlá [32] práce autorů Artema Chebotka, Shiyong Lua a Farshada Fotouha, vydataná v roce 2009, se zaměřuje na obecný překlad jazyka SPARQL. Autoři nejdříve porovnaly algebry obou jazyků a dokázali, že jsou jejich sémantiky vzájemně ekvivalentní. Následně vytvořili funkci, která je schopna tyto jazyky překládat. V poslední části se zaměřili na efektivitu generovaných SQL dotazů a jejich zjednodušení pro efektivnější výsledky.

7.2.2 Efficient SPARQL-to-SQL with R2RML mappings

Článek [33] autorů Mariana Rodriguez-Muroa a Martina Rezka z roku 2015 navazuje na výše zmíněnou práci. Cílem autorů bylo využít jazyk R2RML [34] k efektivnímu a hlavně korektnímu překladu SPARQL do SQL. Autoři

konstatují, že předchozí práce a práce dalších autorů ne vždy dávají funkční a správné SQL dotazy. R2RML, neboli RDB to RDF Mapping language je jazyk vydaný W3C, jehož úkolem je mapování RDF na relační databáze. Autoři ho využívají k vytvoření virtuální RDF databáze, pomocí níž mapují jednotlivé SPARQL dotazy na SQL.

7.3 Zhodnocení současných řešení

Po průzkumu existujících řešení jsem došel k závěru, že neexistuje publikace, jež by řešila zadání mé bakalářské práce. V oblasti překladu do SQL již probíhá výzkum, ale kvůli rozdílům mezi jazyky CQL a SQL na něj nelze navázat. Protože CQL je oproti SQL značně omezený jazyk v možnostech agregace a spojování tabulek, můžeme se v této oblasti nanejvýš inspirovat. Poněvadž cílem této práce není navrhnout obecný překlad mezi SPARQL a CQL, ale mezi SPARQL a triplestorem se specifickou architekturou dostupnou pomocí CQL, proto snaha o obecný překlad je v současné chvíli nadbytečná, mohu však o ni v budoucnu uvažovat.

Část II

Nástroj pro přístup k SBDH pomocí SPARQL

Kapitola 8
Návrh řešení

8.1 Použité technologie

V této kapitole uvádím seznam použitých technologií s odůvodněním, proč jsem si je zvolil.

8.1.1 Java 15

S přihlédnutím k dalším použitým technologiím (Apache Jena), byl výběr programovacího jazyka velice zúžený, proto jsem si pro implementaci překladače zvolil Javu 15 [35]. Zároveň je to jazyk vhodný pro snadnou implementaci dalších rozšíření aplikace, které jsou diskutovány na konci této práce.

8.1.2 Maven

Vzhledem k minimálním nárokům na import knihoven a frameworků jsem si zvolil Maven [36], ale stejným způsobem by posloužil například i Gradle build tool[1].

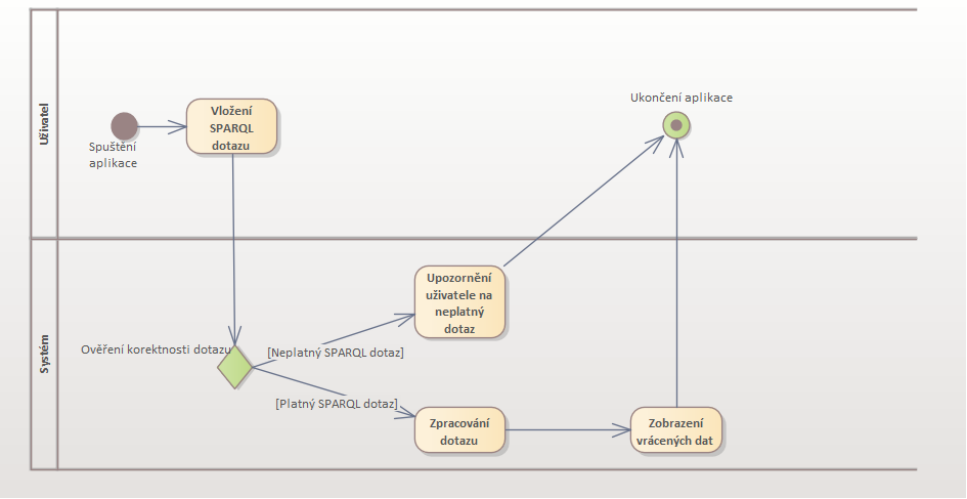
8.1.3 Apache Jena

Důležitou součástí práce je přístup k ontologii, podle níž jsou strukturovány data v databázi Cassandra. Apache Jena [37] je Java framework, jehož jednou z mnoha jeho funkcí je ARQ, tedy podpora evaluace SPARQL dotazů nad RDF ontologií.

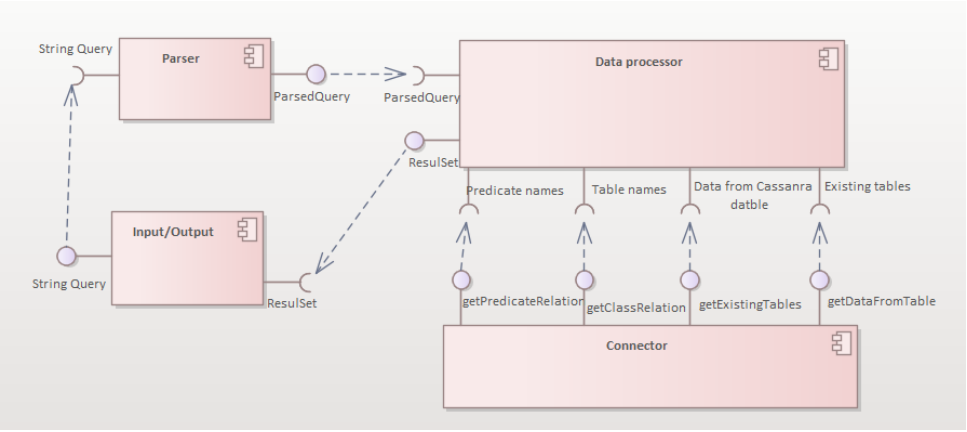
8.1.4 DataStax Cassandra Connector Java API

DataStax je americká společnost soustředící se na databázová řešení založená na Apache Cassandra. Jedním z jeho produktů je Cassandra Connector uvolněný pod licencí Apache license 2.0[2]. Produkt umožňuje napojení vyvíjené

[1] `https://gradle.org`
[2] `https://www.apache.org/licenses/LICENSE-2.0`



Obrázek 8.1: Business process model



Obrázek 8.2: Architektura aplikace

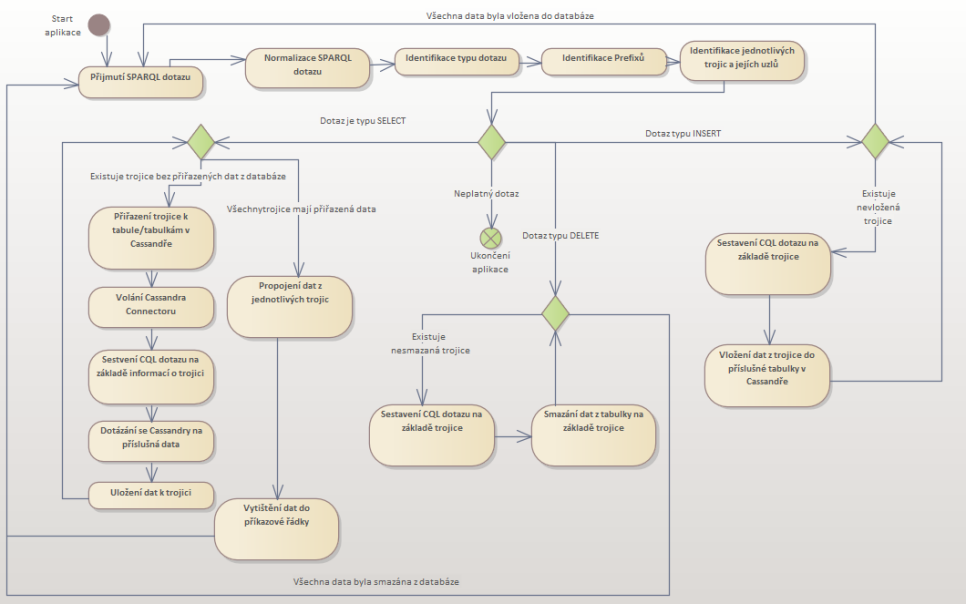
aplikace na databázi a následné posílání CQL dotazů obdobně jako při dotazování jiných konektorů nad SQL databázemi.

8.1.5 Apache Cassandra

Tato databáze byla podrobněji popsána v kapitole 5. Při ověřování funkčnosti vyvíjené aplikace byla Cassandra spuštěna v prostředí Windows Subsystem for Linux, které umožňuje pohodlné spouštění databáze ve Windows prostředí.

8.2 Návrh implementace

Jsem přesvědčen, že není nutné implementovat žádné uživatelské rozhraní, jelikož se tato práce věnuje tvorbě jednoúčelové aplikace s minimem funkcionalit. Komunikace s uživatelem bude probíhat s použitím příkazového řádku, v němž bude aplikace spuštěna. Návrh procesního diagramu můžeme



Obrázek 8.3: Procesní diagram vyhodnocení SPARQL dotazu v překladači.

sledovat na obrázku 8.1. Uživatel po spuštění aplikace vloží SPARQL dotaz, ten je následně ověřen a po ověření platnosti přeložen a evaluován. Uživateli jsou poté vrácena data v podobě tabulky.

Na obrázku 8.2 je zobrazen návrh architektury aplikace. V první fázi jsou data přijata a poslána do Parseru, kde je ověřena správnost SPARQL dotazu a je provedeno jeho rozložení na dílčí části. To znamená rozpoznání typu požadavku a rozdělení dotazu na jednotlivé RDF trojice, uzly a vztahy mezi nimi. Následně Data processor ověří z OWL ontolgie existující tabulky a dohledá chybějící části trojic z Connectoru, které jsou v SPARQL dotazu v podobě proměnných. K takto kompletním trojicím jsou potom pomocí Connectoru získána data z databáze Cassandra. Data jsou následně zkompletována podle SPARQL dotazu a v podobě ResultSetu jsou vrácena na výstup. Na obrázku 8.3 vidíme procesní diagram průchodu dotazu aplikací. V následující sekci je tento algoritmus podrobněji popsán.

Kapitola 9

Popis implementace

Předchozí kapitola se zabývala popisem základní architektury aplikace. Z tohoto modelu jsem vycházel při následné implementaci, jež odpovídá reálnému řešení. Aplikace je rozdělena do čtyř částí (viz obrázek 8.2). Aplikace podporuje SPARQL dotazy pouze ve formátu trojic obsahujících IRI nebo proměnné. Není možné vkládat dotazy obsahující prázdné uzly nebo různé zkrácené verze dotazů, případně funkce UNION, OPTIONAL atd.

9.1 Ovládání aplikace

Spuštění aplikace probíhá v příkazové řádce. Nutnou podmínkou pro její spuštění je běh databáze Cassandra na stejném zařízení jako je spuštěný překladač. Aplikace se spouští se dvěma povinnými parametry:

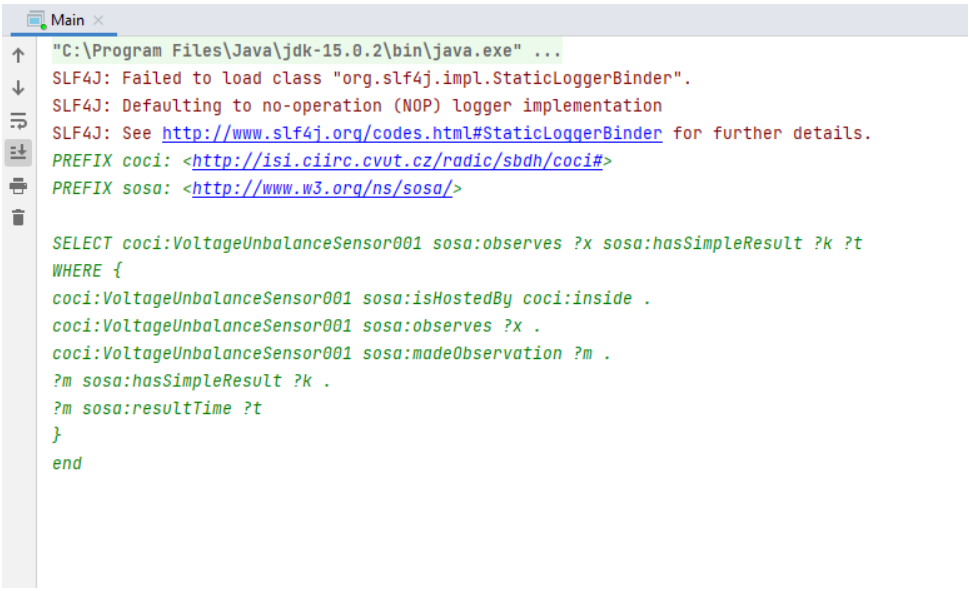
- -n specifikuje namespace, v němž jsou dotazované tabulky v databázi Cassandra
- -path určuje cestu k ontologii v RDF formátu, která je nutná pro správné fungování aplikace

Po spuštění je možné vložit SPARQL dotaz, jenž je zakončený novým řádkem a ukončovacím řetězcem "end"(viz obrázek 9.1). Po přeložení a vyhodnocení dotazu je možné vložit nový dotaz, případně ukončit aplikaci řetězcem "exit". Níže je uvedený vzorový SPARQL dotaz.

```
SELECT ?s ?p ?o
WHERE {
   ?s ?p ?o
}
```

9.2 Parser

Parsování SPARQL dotazu probíhá v několika krocích. Nejprve je provedena normalizace dotazu. Jelikož SPARQL je citlivý na velikost písmen, probíhá normalizace pouze v oblasti prázdných znaků a nových řádků. Tímto způsobem



Obrázek 9.1: Příklad vložení SPARQL dotazu do aplikace.

jsou odstraněny všechny vícenásobné mezery a všechny netištěné znaky oddělující řádky. Po provedení těchto kroků dostaneme dotazy v sjednocené podobě. V dalším kroku ověřím platnost formátu dotazu a zároveň je provedena identifikace typu dotazu pomocí regulárního výrazu. Zde na obrázku 9.2 je zobrazen stavový diagram, který ukazuje, jak může být dotaz identifikován.

Regulární výraz pro identifikaci SELECT dotazu:

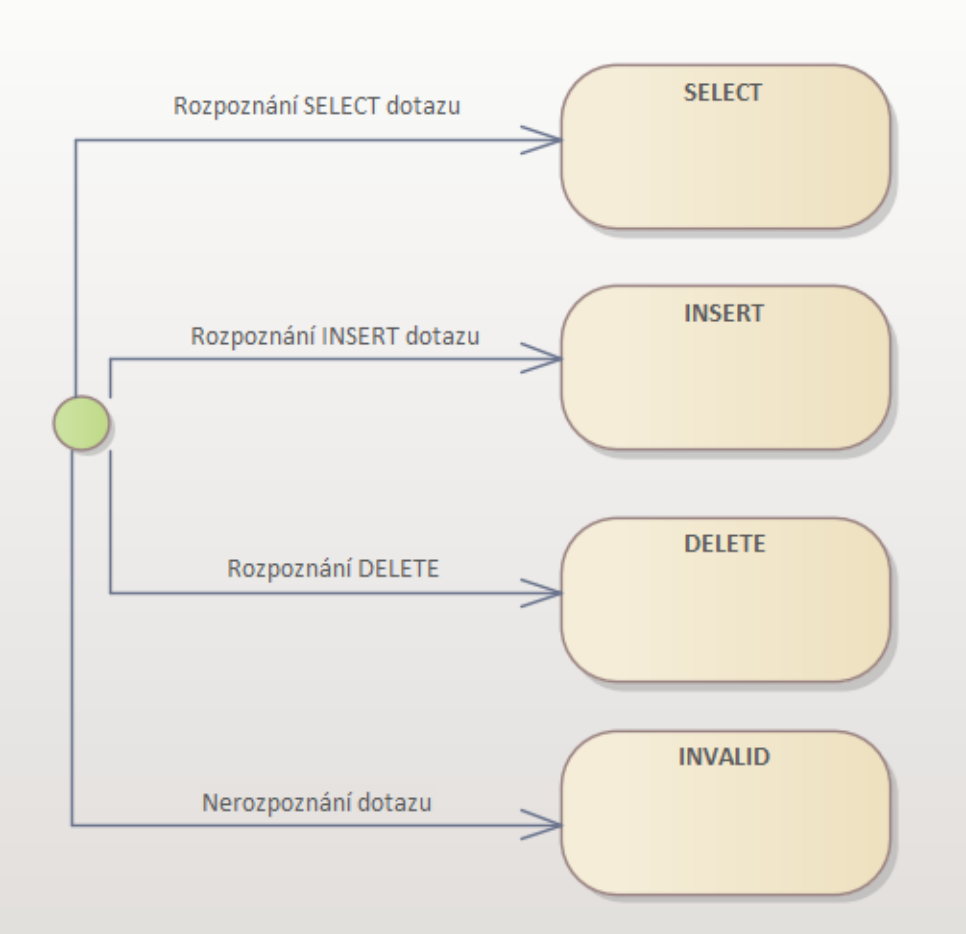
```
^(PREFIX [\w]*:[\n ]*<[\w\-#\\/.:]*> )*
SELECT (<[\w\-#\\/.:]* *>|\?[\w]* *|[\w]*:[\w]* *)*
WHERE
?\{ ?(((([\w]*:[\w]*|<[\w\-#\\/.:]*>|
\?[\w]*|"[\w\W]*") ){3}). )*
((((([\w]*:[\w]*|<[\w\-#\\/.:]*>|\?[\w]*|"[\w\W]*") ){2})
([\w]*:[\w]*|<[\w\-#\\/.:]*>|\?[\w]*|"[\w\W]*")
( \.)? ?)}$
```

Regulární výraz pro identifikaci INSERT DATA dotazu:

```
^(PREFIX [\w]*:[\n ]*<[\w\-#\\/.:]*> )*
DELETE DATA
?\{ ?(((([\w]*:[\w]*|<[\w\-#\\/.:]*>|
\?[\w]*|"[\w\W]*") ){3}). )*
((((([\w]*:[\w]*|<[\w\-#\\/.:]*>|\?[\w]*|"[\w\W]*") ){2})
([\w]*:[\w]*|<[\w\-#\\/.:]*>|\?[\w]*|"[\w\W]*")( \.)? ?)}$
```

Regulární výraz pro identifikaci DELETE DATA dotazu:

```
^(PREFIX [\w]*:[\n ]*<[\w\-#\\/.:]*> )*
```

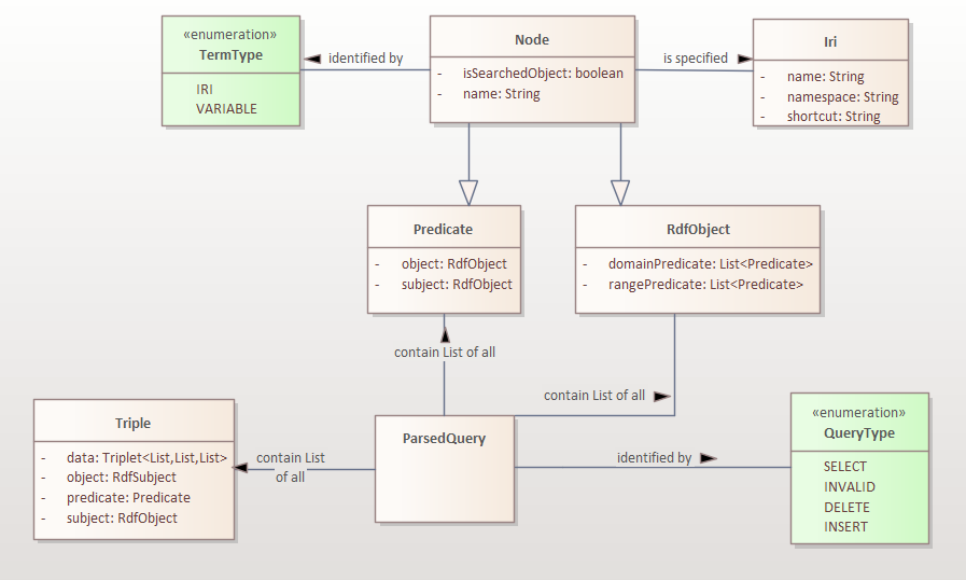


Obrázek 9.2: Stavový diagram s pecifikující typ dotazu.

```
INSERT DATA
?\{ ?(((([\w]*:[\w]*|<[\w\-#\\/.:]*>|
\?[\w]*|"[\w\W]*") ){3}).
)*((((([\w]*:[\w]*|<[\w\-#\\/.:]*>|\?[\w]*|"[\w\W]*") ){2})
([\w]*:[\w]*|<[\w\-#\\/.:]*>|\?[\w]*|"[\w\W]*")( \.)? ?)}$
```

V této části procesu je zároveň vytvořena instance třídy ParsedQuery, jež má v sobě uložené veškeré informace o SPARQL dotazu a je využívána při celém procesu překladu. Této instanci je přiřazen příznak dle typu dotazu.

V další části procesu je dotaz rozebrán na jednotlivé části. Jsou vyhledány všechny existující prefixy, jež jsou následně napojeny na jednotlivé uzly v trojicích. Poté jsou identifikovány trojice a jejich uzly. Pro každý uzel je zkonstruován IRI, jenž slouží k jeho identifikaci. Pokud je dotaz typu SELECT, jsou v poslední fázi vyhledány uzly, jejichž odpovídající výsledky chce uživatel následně zobrazit. Veškeré zmíněné informace jsou navráceny v již zmíněném objektu ParsedQuery.



Obrázek 9.3: Class diagram ukazující třídu ParsedQuery a na ní navázané třídy.

9.3 Třída ParsedQuery

Třída ParsedQuery obsahuje všechny podstatné informace ze zadaného SPARQL dotazu. Každý unikátní uzel v každé trojici je reprezentován vlastním objektem. V případě vícenásobného výskytu uzlu objekt obsahuje informace o všech napojených trojicích. Všechny uzly jsou zároveň sdruženy do trojic dle původního dotazu kvůli snadnější identifikaci a uložení získaných dat pro každou jednu trojici. Celá struktura je zobrazena na obrázku 9.3.

9.4 Zpracování SELECT dotazu

V kapitole 6 bylo uvedeno, že data v databázi jsou uložena dvojím způsobem. Informace o senzorech jsou uloženy v tabulkách se dvěma sloupci dle vertical partitioning modelu (viz obrázek 6.2). První sloupec obsahuje předměty dané trojice, druhý sloupec objekty a název tabulky odpovídá predikátu. Například tabulka "observes"obsahuje sloupce Sensor a ObservableProperty.

Data o jednotlivých měřeních jsou kvůli efektivitě zápisu ukládána do tabulky s názvem konkrétního senzoru a přívlastkem "_hasSimpleResult", tedy například VoltageUnbalanceSensor001_hasSimpleResult. Tato tabulka obsahuje sloupec s daty vzniku měření a sloupec s hodnotami měření. Když tuto strukturu porovnáme s ontologií na obrázku 6.2, zjistíme, že spojení s observaiton nenese důležité informace, a proto je trojice s predikátem observation vypuštěna. Je to však jedna z důležitých věcí, kterou je třeba zohlednit při překladu SPARQL dotazu, neboť požadavek zněl, aby odpovídal SOSA ontolgoii.

Tyto dva rozdílné přístupy k ukládání dat vyžadují i odlišný přístup k jejich

získávání a nelze tedy stejným způsobem překládat SPARQL dotaz vyžadující informaci o senzorech a dotaz dotazující se na data z konkrétního měření.

Základní princip překladu spočívá v získání veškerých dostupných dat pro jednotlivé trojice a jejich následné spojení. Data jsou spojována tak, aby odpovídala zadanému SPARQL dotazu. V případě, že jsou některá data nedostupná a tedy řádek není kompletní, nejsou data do výsledku zahrnuta.

9.4.1 Získávání dat o senzorech

Tato část překladu SPARQL dotazu je téměř univerzální částí, která v případě potřeby dokáže fungovat nad jakýmkoliv RDF grafem za předpokladu, že bude dodržena výše zmíněná struktura dat v Cassandře. V první fázi dotazování je třeba načíst všechna data pro každou RDF trojici. Každá trojice se může vyskytovat ve dvou základních podobách: jednodušší forma dotazu se známým predikátem a složitější forma s neznámým predikátem.

Trojice se známým predikátem

1. <Předmět> <Predikát> <Objekt> Všechny informace o trojici jsou známé. Jedná se o nejjednodušší možnost překladu. Aplikace se pouze dotáže přes Datastax Cassandra connector databáze konkrétní tabulky na specifický záznam odpovídající naší trojici. Takový záznam je poté uložen k trojici do třídy ParsedQuery.

2. ?variable <Predikát> <Objekt> případně <Předmět> <Predikát> ?variable Obě dvě trojice vyjadřují stav, kdy jeden z uzlu je proměnná. Řešením takového případu je CQL dotaz na známou tabulku, v němž je v podmínce WHERE specifikován jeden sloupec. Obdobně jako v předchozím případě jsou data přiřazena ke konkrétní trojici.

3. ?variable1 <Predikát> ?variable2 Tato trojice s oběma neznámými uzly, ale známým predikátem je paradoxně nejrychleji vyhodnocovaným dotazem. Poněvadž je známa tabulka a není nutné použít klauzuli WHERE, dotaz databáze vyhodnotí velice rychle.

Trojice s neznámým predikátem

Tato varianta dotazu je podstatně složitější, protože neznáme tabulku, na niž se budeme dotazovat. Neznalost názvu hledané tabulky nás nutí projít data v celé databázi, abychom vyhledali informace, které potřebujeme. V následující části jsou popsány dvě možné varianty dotazu.

1. ?variable1 ?variable2 <Objekt1> . ?variable1 <Predikát> <Objekt2> Tento případ ukazuje situaci dotazu s neznámým predikátem. V dotazu se zároveň vyskytuje další trojice, která obsahuje proměnnou společnou také pro první trojici. Toto využijeme pro omezení množství prohledávaných tabulek. Aplikace si v takové situaci vyžádá z načtené ontologie všechny předměty (případně objekty dle tvaru dotazu), které mají na sobě

napojený <Predikát>. Potom procházíme všechny vrácené tabulky stejně, jako by byl predikát známý.

2. ?variable1 ?variable2 <Objekt1> Situace, v níž máme neznámý predikát a zároveň nejsme schopni omezit množství procházených tabulek, je nejnáročnější stavem. V tomto případě musíme projít všechny tabulky, neboť nevíme, kde se hledaná data nacházejí, a nejsme schopni omezit seznam tabulek na nějakou jeho podmnožinu. Poněvadž načtená ontologie může obsahovat velké množství uzlů, použijeme pro získání všech potenciálních tabulek informace z databáze. Odfiltrujeme pouze tabulky, které slouží pro ukládání měření. Poté postupujeme obdobně jako v předchozím případě.

9.4.2 Získání dat o měřeních

Pro získání dat z měření bylo nutné vytvořit specifický přístup k překladu těchto SPARQL dotazů. Na data z měření degradovaná do jediné tabulky není možné aplikovat běžný přístup k překladu. V případě, že je v dotazu identifikována trojice, která se ptá na konkrétní část měření, aplikace doplní k ní do seznamu zbylé trojice z měření. Příklad dotazu:

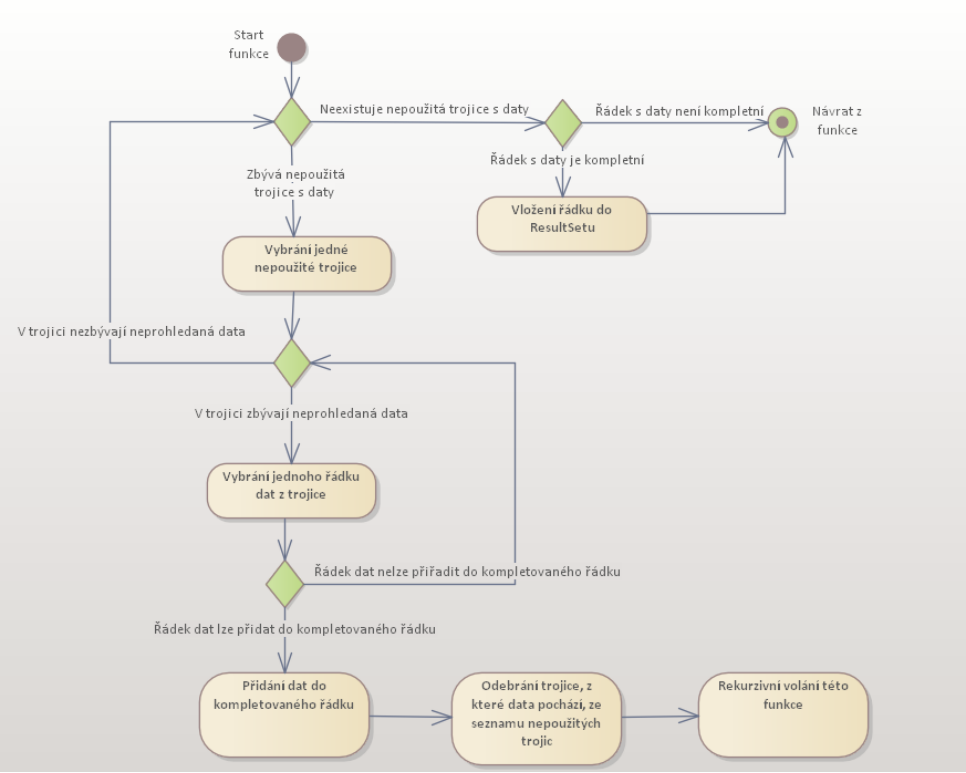
```
PREFIX coci: <http://isi.ciirc.cvut.cz/radic/sbdh/coci#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT ?k
WHERE {
    coci:VoltageUnbalanceSensor001 sosa:madeObservation ?m .
    ?m sosa:hasSimpleResult ?k
}
```

Tento dotaz zobrazuje všechna měření senzoru VoltageUnbalanceSensor001. Aplikace doplní do SPARQL dotazu také trojici ?m sosa:resultTime ?x, jež se ve výsledcích neprojeví. Je však nutná pro následné spojování dat z jednotlivých trojic dle ontologie.

Data každého měření jsou rozdělena do tří trojic. Informace o konkrétním měření zaniká, zůstávají nám pouze její hodnoty a čas. Tento chybějící uzel tedy uměle vytvoříme, pojmenujeme ho unikátním názvem, abychom každé měření mohli identifikovat. Takto připravená data jsou nyní ve stejném stavu jako data o senzorech, tedy každá trojice má k sobě přiřazená data, jež se jí týkají.

9.4.3 Spojování dat z jednotlivých trojic

Po získání vybraných dat je nutné je spojit dohromady. Jazyk CQL neobsahuje operace Join ani OR. Bylo tedy nutné SPARQL dotaz rozdělit do mnoha CQL dotazů, jejichž výsledky je teď nutné propojit. Každá trojice má v sobě uloženou tabulku s dvojicemi dat získaných z databáze Cassandra.



Obrázek 9.4: Diagram průchodu algoritmem spojování dat z trojic.

9.4.4 ResultSet

Výsledkem spojování dat je objekt třídy ResultSet. Jedná se o dvourozměrné pole s pevně definovaným počtem pojmenovaných sloupců. ResultSet má v sobě uloženou informaci o sloupcích, které se mají vrátit na základě SPARQL dotazu. Data je možné přidávat pouze po celých řádcích. Jejich načítání probíhá buď po jednotlivých sloupcích, nebo po řádcích. Při jejich načítání je možné vrátit kompletní řádek se všemi sloupci, nebo řádek pouze se sloupci uvedenými v SPARQL dotazu.

9.4.5 Algoritmus spojování dat

Proces spojování tabulek probíhá takto: Rekurzivní funkce si udržuje seznam trojic, ze kterých ještě nebyla použita data, pole představující jeden řádek dat a ResultSet, kam se ukládají kompletní řádky. Každé volání funkce nejprve ověří, zda existuje nějaká nepoužitá trojice. Funkce iteruje nad všemi nepoužitými trojicemi s daty a hledá data, která by měla jeden uzel společný s již nalezenými daty v kompletovaném řádku. Najde-li je, vloží je do řádku a odebere trojici, z níž data pocházejí, ze seznamu nepoužitých trojic. Následně rekurzivně volá tu samou funkci s upraveným seznamem nepoužitých trojic a rozšířeným řádkem s kompletovanými daty. Neexistuje-li žádná nepoužitá trojice, proběhne ověření, zda je řádek dat kompletní. Pokud ano, je přidán do

ResultSetu, pokud ne, je zahozen a funkce se vrací o úroveň výš, kde probíhá další prohledávání trojic a jejich dat. Celý proces je zobrazen na obrázku 9.4. Po skončení algoritmu je vrácen ResultSet se všemi řádky, které obsahují data ve všech svých sloupcích.

9.4.6 Zobrazení dat

V poslední fázi jsou data vytištěna do konzole. V SPARQL dotazu je nutné určit jaké části dotazu se mají zobrazit ve výsledku. Podle toho jsou následně vytištěny příslušné sloupce s daty.

9.5 Zpracování dotazu DELETE DATA a INSER DATA

Přístup k překladu a evaluaci SPARQL dotazů DELETE DATA a INSERT DATA je velice podobný. Jediným rozdílem je finální CQL dotaz, který se liší podle toho, zda chceme data vkládat, respektive mazat. Pro fungování těchto dotazů je nutné specifikovat všechny uzly ve všech trojicích, tedy nesmí se v dotazu vyskytovat žádné proměnné. Potom postupně procházíme všechny trojice a dotazujeme se CQL dotazy na příslušné tabulky dle predikátu, kde vkládáme, respektive mažeme data.

9.6 Testování funkčnosti

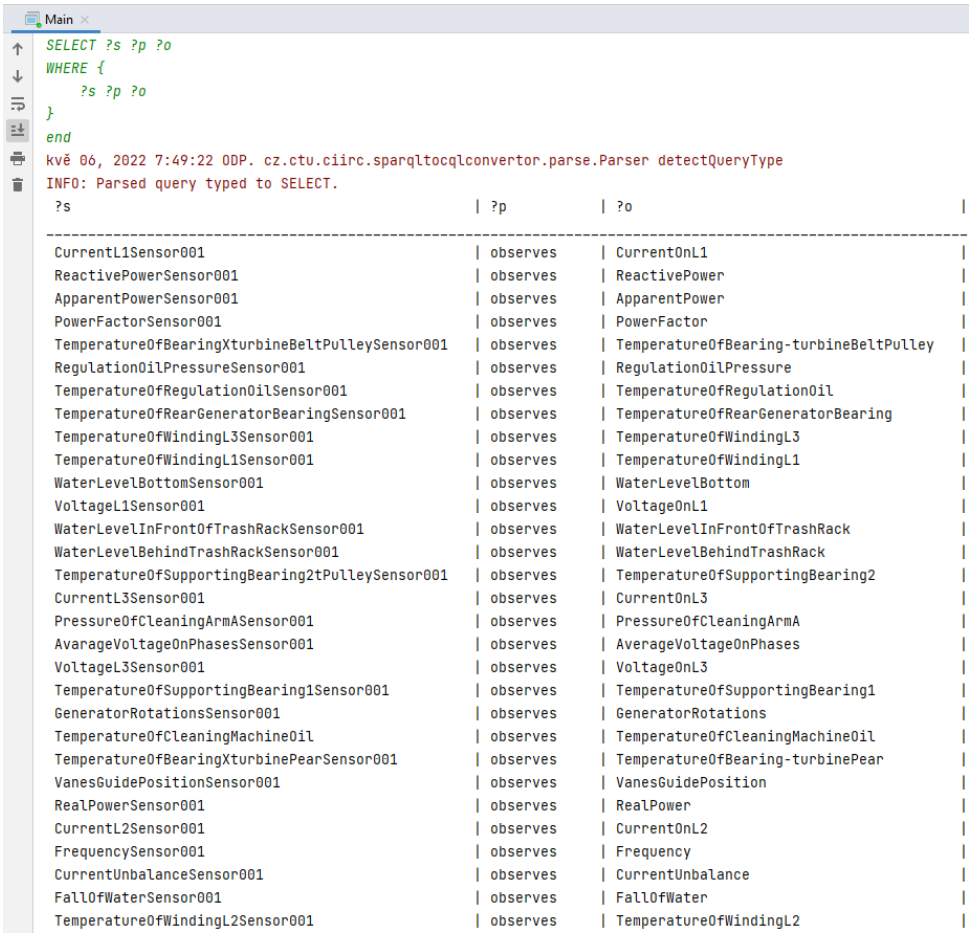
Ověření funkčnosti navrhovaného řešení probíhá na naměřených datech získaných z vodní elektrárny, která jsou uložena v databázi Cassandra. K dispozici jsou tabulky isHostedBy a observes (viz obr. 6.2). První tabulka obsahuje informaci o umístění senzoru. V tomto případě je rozlišeno pouze, zda je senzor umístěný ve vnitřních prostorách, nebo venku. Druhá tabulka udává měřenou vlastnost senzoru. V tomto případě se liší u každého z nich podle jeho typu. Dále je pro každý senzor k dispozici tabulka s naměřenými daty a časem měření, jenž pokrývá uzly Observes, hasSimpleResult a resultTime.

Příklady SPARQL dotazů:

9.6.1 Dotaz č. 1

```
SELECT ?s ?p ?o
WHERE {
    ?s ?p ?o
}
```

Na obr. 9.5 jsou zobrazeny výsledky nejobecnějšího existujícího SPARQL dotazu. Ten zobrazí všechny existující trojice v RDF grafu. V tomto případě nejsou zobrazeny tabulky s měřeními, neboť nejsou uloženy pomocí vertikální segmentace.

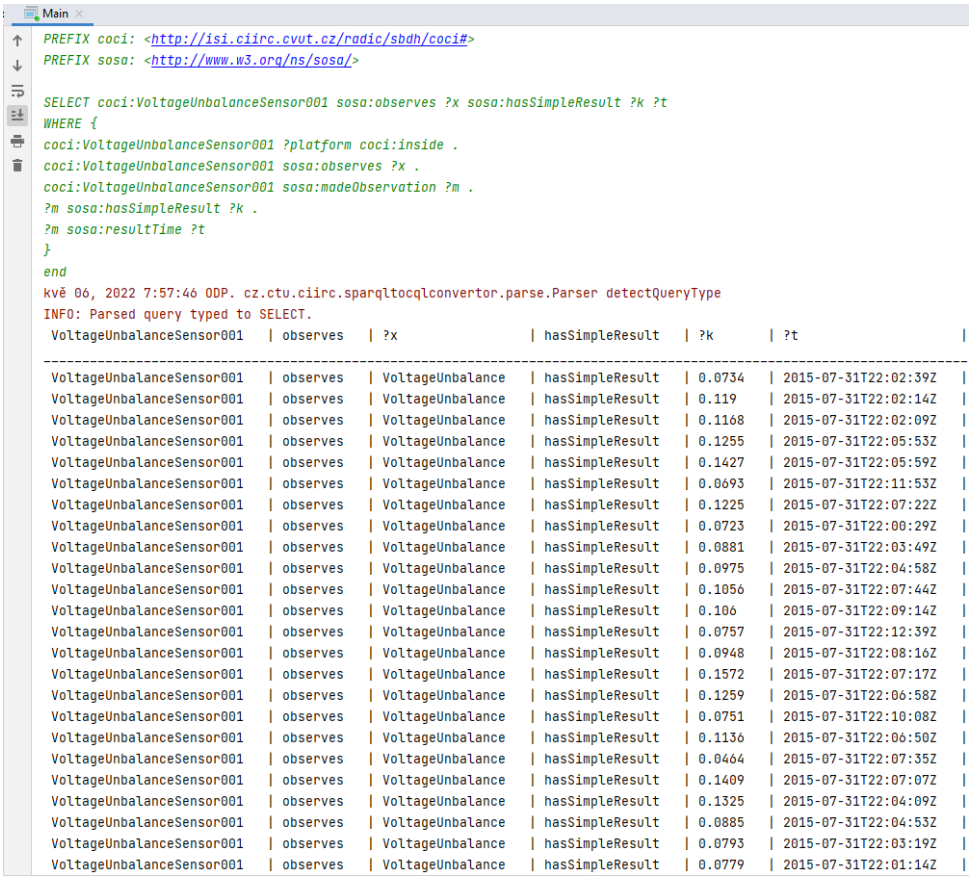


Obrázek 9.5: SPARQL dotaz č.1

9.6.2 Dotaz č. 2

```
PREFIX coci: <http://isi.ciirc.cvut.cz/radic/sbdh/coci#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT coci:VoltageUnbalanceSensor001 sosa:observes ?x
    sosa:hasSimpleResult ?k ?t
WHERE {
    coci:VoltageUnbalanceSensor001 ?platform coci:inside .
    coci:VoltageUnbalanceSensor001 sosa:observes ?x .
    coci:VoltageUnbalanceSensor001 sosa:madeObservation ?m .
    ?m sosa:hasSimpleResult ?k .
    ?m sosa:resultTime ?t
}
```

Tento SPARQL dotaz ověří, zda jeden konkrétní senzor je umístěn venku, zobrazí pozorovanou vlastnost a vypíše všechny naměřené hodnoty (viz obr. 9.6).



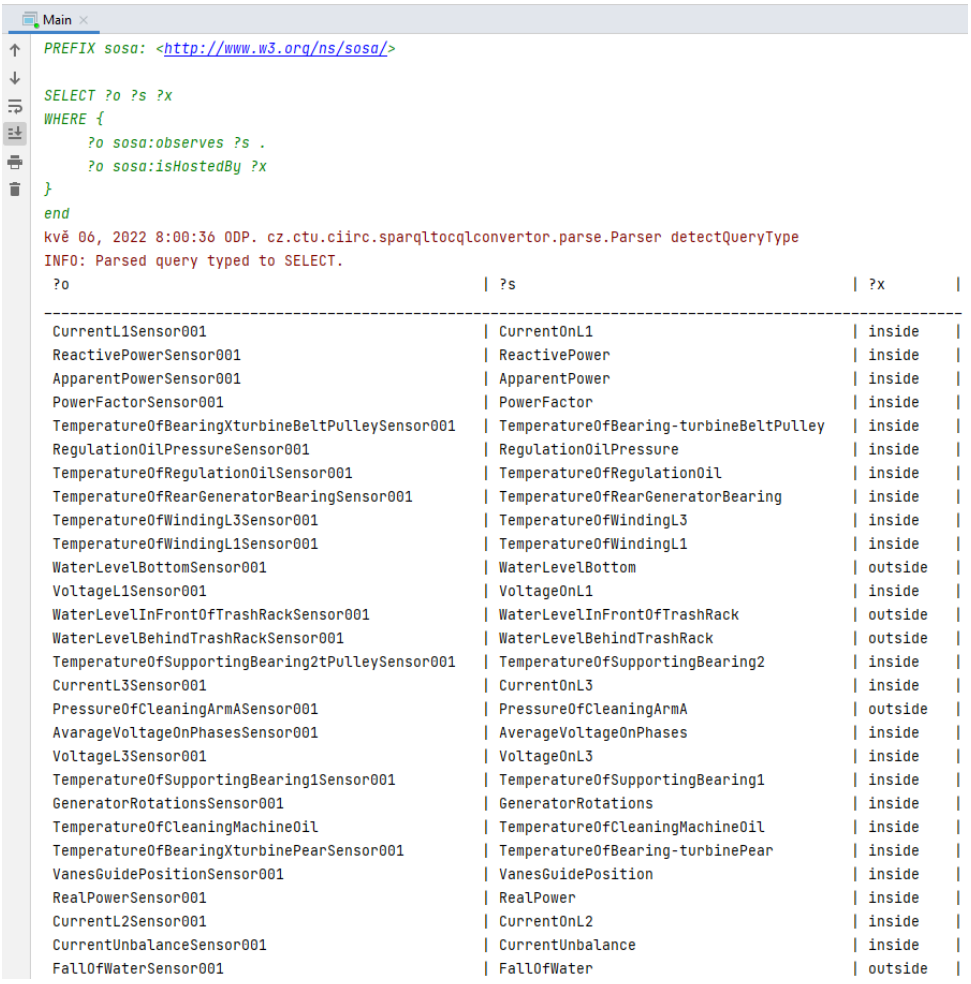
```
Main
PREFIX coci: <http://isi.ciirc.cvut.cz/radic/sbdh/coci#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT coci:VoltageUnbalanceSensor001 sosa:observes ?x sosa:hasSimpleResult ?k ?t
WHERE {
coci:VoltageUnbalanceSensor001 ?platform coci:inside .
coci:VoltageUnbalanceSensor001 sosa:observes ?x .
coci:VoltageUnbalanceSensor001 sosa:madeObservation ?m .
?m sosa:hasSimpleResult ?k .
?m sosa:resultTime ?t
}
end
kvě 06, 2022 7:57:46 ODP. cz.ctu.ciirc.sparqltocqlconvertor.parse.Parser detectQueryType
INFO: Parsed query typed to SELECT.
 VoltageUnbalanceSensor001   | observes   | ?x                | hasSimpleResult   | ?k       | ?t                    |
-------------------------------------------------------------------------------------------------------------------------------
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0734   | 2015-07-31T22:02:39Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.119    | 2015-07-31T22:02:14Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1168   | 2015-07-31T22:02:09Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1255   | 2015-07-31T22:05:53Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1427   | 2015-07-31T22:05:59Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0693   | 2015-07-31T22:11:53Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1225   | 2015-07-31T22:07:22Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0723   | 2015-07-31T22:00:29Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0881   | 2015-07-31T22:03:49Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0975   | 2015-07-31T22:04:58Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1056   | 2015-07-31T22:07:44Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.106    | 2015-07-31T22:09:14Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0757   | 2015-07-31T22:12:39Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0948   | 2015-07-31T22:08:16Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1572   | 2015-07-31T22:07:17Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1259   | 2015-07-31T22:06:58Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0751   | 2015-07-31T22:10:08Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1136   | 2015-07-31T22:06:50Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0464   | 2015-07-31T22:07:35Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1409   | 2015-07-31T22:07:07Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.1325   | 2015-07-31T22:04:09Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0885   | 2015-07-31T22:04:53Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0793   | 2015-07-31T22:03:19Z  |
 VoltageUnbalanceSensor001   | observes   | VoltageUnbalance  | hasSimpleResult   | 0.0779   | 2015-07-31T22:01:14Z  |
```

Obrázek 9.6: SPARQL dotaz č.2

9.6.3 Dotaz č. 3

```
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT ?o ?s ?x
WHERE {
    ?o sosa:observes ?s .
    ?o sosa:isHostedBy ?x
}
```

Třetí zde prezentovaný SPARQL dotaz zobrazí všechny senzory, které mají přiřazenou pozorovanou vlastnost a zároveň mají specifikováno své umístění (viz obr. 9.7).



Main

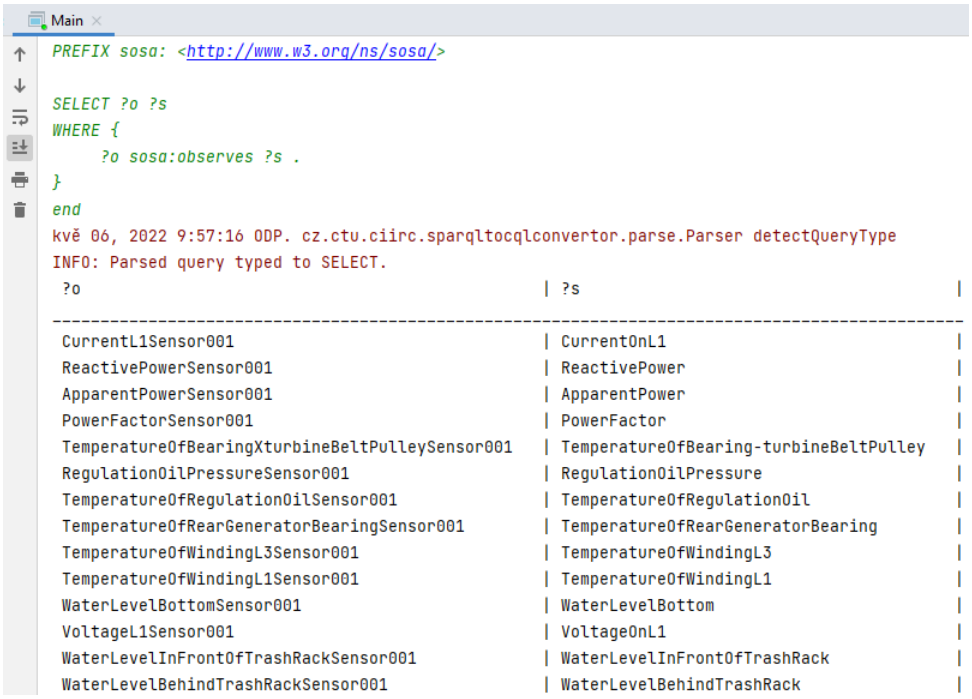
```
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT ?o ?s ?x
WHERE {
     ?o sosa:observes ?s .
     ?o sosa:isHostedBy ?x
}
end
kvě 06, 2022 8:00:36 ODP. cz.ctu.ciirc.sparqltocqlconvertor.parse.Parser detectQueryType
INFO: Parsed query typed to SELECT.
```

?o	?s	?x
CurrentL1Sensor001	CurrentOnL1	inside
ReactivePowerSensor001	ReactivePower	inside
ApparentPowerSensor001	ApparentPower	inside
PowerFactorSensor001	PowerFactor	inside
TemperatureOfBearingXturbineBeltPulleySensor001	TemperatureOfBearing-turbineBeltPulley	inside
RegulationOilPressureSensor001	RegulationOilPressure	inside
TemperatureOfRegulationOilSensor001	TemperatureOfRegulationOil	inside
TemperatureOfRearGeneratorBearingSensor001	TemperatureOfRearGeneratorBearing	inside
TemperatureOfWindingL3Sensor001	TemperatureOfWindingL3	inside
TemperatureOfWindingL1Sensor001	TemperatureOfWindingL1	inside
WaterLevelBottomSensor001	WaterLevelBottom	outside
VoltageL1Sensor001	VoltageOnL1	inside
WaterLevelInFrontOfTrashRackSensor001	WaterLevelInFrontOfTrashRack	outside
WaterLevelBehindTrashRackSensor001	WaterLevelBehindTrashRack	outside
TemperatureOfSupportingBearing2tPulleySensor001	TemperatureOfSupportingBearing2	inside
CurrentL3Sensor001	CurrentOnL3	inside
PressureOfCleaningArmASensor001	PressureOfCleaningArmA	outside
AvarageVoltageOnPhasesSensor001	AverageVoltageOnPhases	inside
VoltageL3Sensor001	VoltageOnL3	inside
TemperatureOfSupportingBearing1Sensor001	TemperatureOfSupportingBearing1	inside
GeneratorRotationsSensor001	GeneratorRotations	inside
TemperatureOfCleaningMachineOil	TemperatureOfCleaningMachineOil	inside
TemperatureOfBearingXturbinePearSensor001	TemperatureOfBearing-turbinePear	inside
VanesGuidePositionSensor001	VanesGuidePosition	inside
RealPowerSensor001	RealPower	inside
CurrentL2Sensor001	CurrentOnL2	inside
CurrentUnbalanceSensor001	CurrentUnbalance	inside
FallOfWaterSensor001	FallOfWater	outside

Obrázek 9.7: SPARQL dotaz č.3

9.6.4 Dotaz č. 4

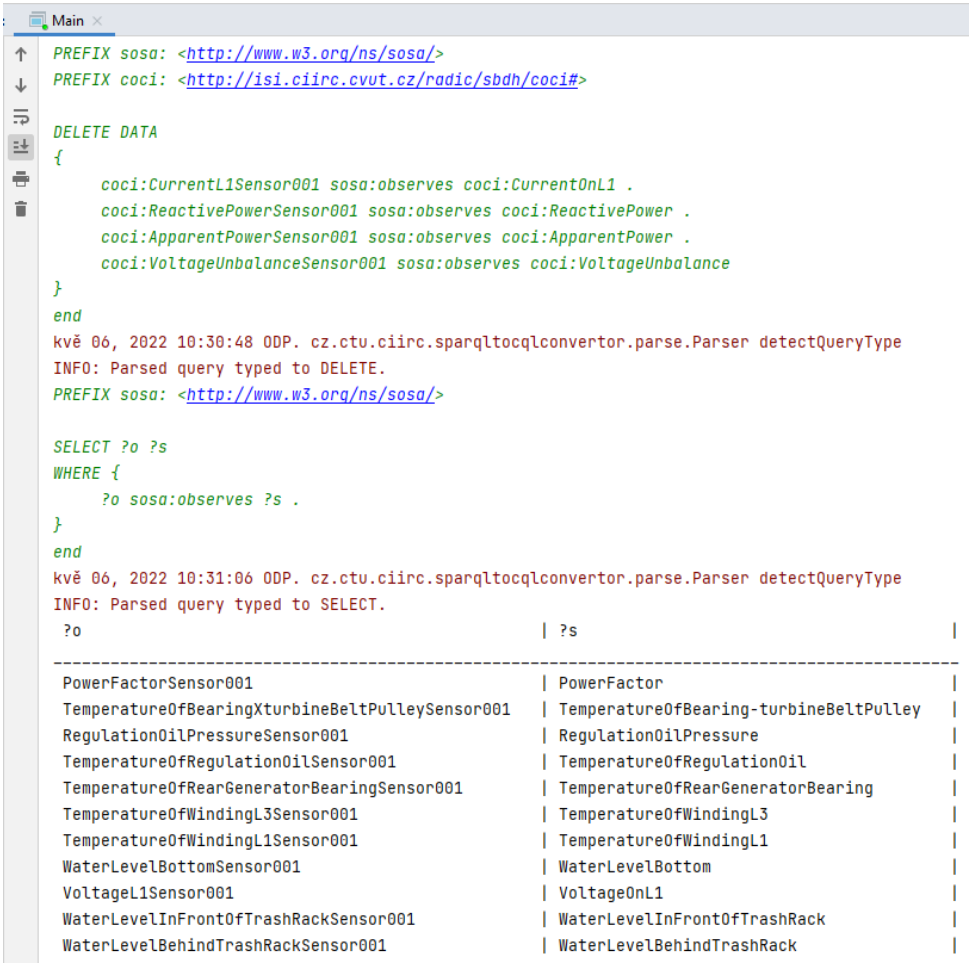
```
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX coci: <http://isi.ciirc.cvut.cz/radic/sbdh/coci#>

DELETE DATA
{
     coci:CurrentL1Sensor001 sosa:observes coci:CurrentOnL1 .
     coci:ReactivePowerSensor001 sosa:observes coci:ReactivePower .
     coci:ApparentPowerSensor001 sosa:observes coci:ApparentPower .
     coci:VoltageUnbalanceSensor001 sosa:observes
         coci:VoltageUnbalance
}
```



Obrázek 9.8: Data z tabulky observes.

```
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX coci: <http://isi.ciirc.cvut.cz/radic/sbdh/coci#>

INSERT DATA
{
    coci:CurrentL1Sensor001 sosa:observes coci:CurrentOnL1 .
    coci:ReactivePowerSensor001 sosa:observes coci:ReactivePower .
    coci:ApparentPowerSensor001 sosa:observes coci:ApparentPower .
    coci:VoltageUnbalanceSensor001 sosa:observes
        coci:VoltageUnbalance
}
```

Závěrečné dotazy mají za úkol otestovat zobrazení původních dat z tabulky observes (obr. 9.8), mazání dat (obr. 9.9) a následné vložení dat do téže tabulky (obr. 9.10).



Obrázek 9.9: Ukázka smazání dat z tabulky observes

Main

```
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX coci: <http://isi.ciirc.cvut.cz/radic/sbdh/coci#>

INSERT DATA
{
     coci:CurrentL1Sensor001 sosa:observes coci:CurrentOnL1 .
          coci:ReactivePowerSensor001 sosa:observes coci:ReactivePower .
          coci:ApparentPowerSensor001 sosa:observes coci:ApparentPower .
          coci:VoltageUnbalanceSensor001 sosa:observes coci:VoltageUnbalance
}
end
kvě 06, 2022 10:32:02 ODP. cz.ctu.ciirc.sparqltocqlconvertor.parse.Parser detectQueryType
INFO: Parsed query typed to INSERT.
PREFIX sosa: <http://www.w3.org/ns/sosa/>

SELECT ?o ?s
WHERE {
     ?o sosa:observes ?s .
}
end
kvě 06, 2022 10:32:26 ODP. cz.ctu.ciirc.sparqltocqlconvertor.parse.Parser detectQueryType
INFO: Parsed query typed to SELECT.
 ?o                                                | ?s                                                |
---------------------------------------------------------------------------------------------------------
 CurrentL1Sensor001                                | CurrentOnL1                                       |
 ReactivePowerSensor001                            | ReactivePower                                     |
 ApparentPowerSensor001                            | ApparentPower                                     |
 PowerFactorSensor001                              | PowerFactor                                       |
 TemperatureOfBearingXturbineBeltPulleySensor001   | TemperatureOfBearing-turbineBeltPulley            |
 RegulationOilPressureSensor001                    | RegulationOilPressure                             |
 TemperatureOfRegulationOilSensor001               | TemperatureOfRegulationOil                        |
 TemperatureOfRearGeneratorBearingSensor001        | TemperatureOfRearGeneratorBearing                 |
 TemperatureOfWindingL3Sensor001                   | TemperatureOfWindingL3                            |
 TemperatureOfWindingL1Sensor001                   | TemperatureOfWindingL1                            |
 WaterLevelBottomSensor001                         | WaterLevelBottom                                  |
 VoltageL1Sensor001                                | VoltageOnL1                                       |
 WaterLevelInFrontOfTrashRackSensor001             | WaterLevelInFrontOfTrashRack                      |
 WaterLevelBehindTrashRackSensor001                | WaterLevelBehindTrashRack                         |
```

Obrázek 9.10: Ukázka vložení dat do tabulky observes

Kapitola 10
Diskuze a závěr

10.1 Diskuze

Cílem práce bylo vytvořit aplikaci pro překlad SPARQL dotazů do jazyka CQL pro Cassandra Triplestore. Protože jazyk CQL má značná omezení týkající se spojování dotazů (operace JOIN), nebylo možné vytvořit aplikaci pro obecný překlad mezi oběma jazyky. Implementované řešení vytvořené na míru platformě SBDH splňuje své účely. Aplikace umí zpracovat základní SPARQL dotazy, rozložit je na dílčí části a pomocí jazyka CQL získat z Cassandra triplestoru potřebná data.

10.1.1 Alternativní přístupy k řešení problému

Při psaní bakalářské práce jsem několikrát změnil přístup k řešení některých částí problematiky tak, aby aplikace byla efektivnější a snadněji ovladatelná. Přesto i po dopsání práce existují části, u kterých by bylo přinejmenším zajímavé zamyslet se nad jiným řešením a dalšími možnými rozšířeními.

Parser

Pro parsování SPARQL dotazů jsem vytvořil vlastní parser identifikující pouze dotazy, které potřebuji. Tento parser umožňuje relativně snadné rozšíření o podporu dalších SPARQL dotazů. Je však náročnější implementovat alternativní konstrukce pro již podporované dotazy. Tento přístup jsem zvolil kvůli jednoduchosti vzhledem k požadovanému rozsahu práce. V případě, že by aplikace měla být rozšířena o větší množství konstrukcí, bylo by vhodné více vycházet z gramatiky jazyka SPARQL a vytvořit parser podle něj nebo použít existující řešení. Apache Jena ARQ je nástroj, jehož by bylo možné pro takový přístup využít, neboť má gramatiku jazyka SPARQL implementovanou a je schopný takové dotazy parsovat.

Rychlost vyhodnocení

Rychlost vyhodnocení SPARQL dotazu je závislá na jeho složitosti a rozsahu. Přestože jsem se pokusil vyhodnocování dotazu urychlit sníženým množství

dotazů databáze Cassandra a ukládáním vybraných dat do mezipaměti, vyhodnocení některých dotazů trvá poměrně dlouhou dobu. To se týká zvláště dotazů obsahujících proměnné na místech predikátů. Takový problém je způsoben pomalým přístupem k databázi. Nepodařilo se mi identifikovat, zda je pomalý samotný přístup, nebo vyhodnocení CQL dotazů, jež jsou neefektivní kvůli příznaku ALLOW FILTERING, jenž je nutný pro filtrování nad daty. Můj předpoklad je, že zpomalení je způsobené druhou možností. Řešením by tedy mohlo být umístění databáze na výkonnější zařízení. Domnívám se, že běh databáze na rozumně výkonném stroji by mohl zkrátit dobu vyhodnocování dotazů. V případě, že se jedná o první možnost, bylo by vhodné se zamyslet nad paralelním přístupem k databázi z několika vláken najednou. Databáze umožňuje přístup z několika bodů zároveň, takže by tento přístup mohl zrychlit celkovou dobu vyhodnocování dotazu. Zároveň je možné, že by pomohl i v případě pomalého vyhodnocování dotazů, což by však bylo nutné ověřit měřením.

10.1.2 Návrhy na rozšíření

Mnou navrhované řešení umožňuje v současnosti pouze přístup z příkazového řádku a tedy pouze ruční zadávání a čtení dotazů. V případě rozšíření aplikace o REST API by bylo možné ji spustit v podobě micro service a zapojit ji do automatizovaného řešení.

Dalším návrhem by byla možnost rozšíření podporovaných SPARQL konstrukcí. Jedná se zejména o klauzuli FILTER a OPTIONAL, které by umožnily příjemnější přístup k datům. Bylo by ale nutné vyhodnotit přínosy i rizika tohoto rozšíření. Zejména se jedná riziko navýšení doby vyhodnocování dotazů, které by mohlo způsobit nepoužitelnost aplikace v praxi.

V současné době je pro fungování aplikace nutné, aby běžela na stejném stroji jako databáze Cassandra. Pro zvýšení komfortu by bylo vhodné umožnit uživateli nastavení vzdáleného připojení databáze. Jedná se o relativně snadné rozšíření, které zvýší její použitelnost v praxi.

10.2 Závěr

Cílem bakalářské práce bylo vytvořit SPARQL rozhraní pro Apache Cassandra triplestore. Současně bylo mým úkolem seznámit se s ontologiemi, jazykem SPARQL a databází Cassandra, určit rozsah možných dotazů z jazyka SPARQL, které je možné přeložit do CQL, navrhnout řešení překladu a implementovat aplikaci, která dokáže zpracované SPARQL dotazy přeložit do CQL a vrátit výsledky, stejně jako bychom se dotazovali jazykem SPARQL nad RDF databází.

V teoretické části jsem nejprve charakterizoval jednotlivé výše uvedené technologie spolu s představením platformy Semancic big data historian, pro kterou je vyvíjená aplikace určena. V dalších části jsem vymezil rozsah překládaných dotazů a navrhl řešení, jehož implementace je popsána v poslední části práce.

S ohledem na SBDH byly identifikovány SPARQL dotazy SELECT, INSERT DATA a DELETE DATA, které aplikace umí zpracovat a provést. V první fázi je dotaz normalizován a rozdělen na jednotlivé objekty, jež si udržují informaci o trojicích, do kterých patří. V případě SELECT dotazu jsou pak pro každou z trojic nalezeny všechny odpovídající výsledky z databáze a ty jsou následně spojeny dohromady. Výsledky, které neodpovídají zadanému vzoru z SPARQL dotazu, jsou odstraněny. Zbývající data jsou zobrazena na výstupu aplikace. Pro dotazy INSERT DATA a DELETE DATA je nutnou podmínkou fungování, aby byly specifikovány všechny uzly dotazu. Dotaz tedy nesmí obsahovat žádnou proměnnou. Trojice těchto dotazů jsou postupně procházeny a pro každou z nich je vytvořen CQL dotaz, jenž data do databáze vloží, respektive je z ní vymaže.

Práce ukazuje několik příkladů SPARQL dotazů a jejich výsledků otestovaných na reálných datech. Zároveň je popsáno ovládání aplikace. Poslední část, diskuze, obsahuje návrhy na rozšíření a alternativní přístup k řešení některých částí aplikace.

Jsem přesvědčen, že jsem zadání bakalářské práce splnil. Vytvořená nová aplikace může být využita ke snadnějšímu přístupu k časovým řadám uloženým v RDF triplestoru pomocí databáze Apache Cassandra, což umožní zvýšit komfort pro uživatele platformy SBDH.

Literatura

[1] F. Gandon and G. Schreiber, "Rdf 1.1 xml syntax," Feb 2014. [Online]. Available: https://www.w3.org/TR/rdf-syntax-grammar/

[2] D. Brickley and R. Guha, "Rdf schema 1.1," Feb 2014. [Online]. Available: https://www.w3.org/TR/rdf-schema/#def-subclass

[3] V. Jirkovskỳ, "Semantic integration in the context of cyber-physical systems," 2017.

[4] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "Sosa: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, vol. 56, pp. 1–10, 2019.

[5] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[6] E. Manavalan and K. Jayakrishna, "A review of internet of things (iot) embedded sustainable supply chain for industry 4.0 requirements," *Computers & Industrial Engineering*, vol. 127, pp. 925–953, 2019.

[7] G. D'Emilia and A. Gaspari, "Data validation techniques for measurements systems operating in a industry 4.0 scenario a condition monitoring application," in *2018 Workshop on Metrology for Industry 4.0 and IoT*. IEEE, 2018, pp. 112–116.

[8] C. Date, *An Introduction to Database Systems (currently 8th edition)*. Addison-Wesley,(recent editions 199x-2005).

[9] M. Gelbmann, "The evolution of dbms popularity in the db-engines ranking, 2013-2022," Mar 2022. [Online]. Available: https://db-engines.com/en/ranking

[10] J. L. Harrington, *Relational database design and implementation*. Morgan Kaufmann, 2016.

[11] A. Nayak, A. Poriya, and D. Poojary, "Type of nosql databases and its comparison with relational databases," *International Journal of Applied Information Systems*, vol. 5, no. 4, pp. 16–19, 2013.

[12] R. Cattell, "Scalable sql and nosql data stores," *Acm Sigmod Record*, vol. 39, no. 4, pp. 12–27, 2011.

[13] T. T. Nguyen and M. H. Nguyen, "Zing database: high-performance key-value store for large-scale storage service," *Vietnam Journal of Computer Science*, vol. 2, no. 1, pp. 13–23, 2015.

[14] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented database systems," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1664–1665, 2009.

[15] O. Tezer, "A comparison of nosql database management systems and models," *DigitalOcean. Np*, vol. 21, 2014.

[16] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective," in *Proceedings of the 48th annual Southeast regional conference*, 2010, pp. 1–6.

[17] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[18] B. Vatant, "Rdf - semantic web standards," Feb 2014. [Online]. Available: https://www.w3.org/RDF/

[19] M. Dürst and M. Suignard, "Rfc 3987 - internationalized resource identifiers (iris)," Jun 1994. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3987

[20] T. Berners-Lee, "Rfc 1630 - universal resource identifiers in www: A unifying syntax for the expression of names and addresses of objects on the network as used in the world-wide web," Jun 1994. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1630

[21] T. Berners-Lee, L. Masinter, and M. P. McCahill, "Rfc 1738 - uniform resource locators (url)," Dec 1994. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1738

[22] P. Hayes and P. Patel-Schneider, "Rdf 1.1 semantics," Feb 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/

[23] J. Bao, E. F. Kendall, D. L. McGuinness, and P. F. Patel-Schneider, "Owl 2 web ontology language," Dec 2012. [Online]. Available: https://www.w3.org/TR/owl2-quick-reference/

[24] S. Harris and A. Seaborne, "Sparql 1.1 query language," Mar 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[25] P. Gearon, A. Passant, and A. Polleres, "Sparql 1.1 update," Mar 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-sparql11-update-20130321/

[26] N. Neeraj, *Mastering apache cassandra: Build, manage, and configure high-performing, reliable NoSQL database for your application with Cassandra*. Packt Publishing, 2015.

[27] A. Arasu, S. Babu, and J. Widom, "The cql continuous query language: semantic foundations and query execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121–142, 2006.

[28] M. Obitko and V. Jirkovskỳ, "Big data semantics in industry 4.0," in *International conference on industrial applications of holonic and multi-agent systems*. Springer, 2015, pp. 217–229.

[29] V. Jirkovskỳ, M. Obitko, and V. Mařík, "Understanding data heterogeneity in the context of cyber-physical systems integration," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 660–667, 2016.

[30] D. Alocci, J. Mariethoz, O. Horlacher, J. T. Bolleman, M. P. Campbell, and F. Lisacek, "Property graph vs rdf triple store: A comparison on glycan substructure search," *PloS one*, vol. 10, no. 12, p. e0144578, 2015.

[31] S. Statistia, Inc., "Most popular database management systems 2022," Jan 2022. [Online]. Available: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

[32] A. Chebotko, S. Lu, and F. Fotouhi, "Semantics preserving sparql-to-sql translation," *Data & Knowledge Engineering*, vol. 68, no. 10, pp. 973–1000, 2009.

[33] M. Rodriguez-Muro and M. Rezk, "Efficient sparql-to-sql with r2rml mappings," *Journal of Web Semantics*, vol. 33, pp. 141–169, 2015.

[34] S. Das, S. Sundara, and R. Cyganiak, Sep 2012. [Online]. Available: https://www.w3.org/TR/r2rml/#:~:text=An%20R2RML%20mapping%20defines%20a,access%20to%20the%20output%20dataset.

[35] K. Arnold, J. Gosling, and D. Holmes, *The Java programming language*. Addison Wesley Professional, 2005.

[36] F. P. Miller, A. F. Vandome, and J. McBrewster, *Apache Maven*. Alpha Press, 2010.

[37] A. J. Apache Jena, 2021. [Online]. Available: https://jena.apache.org/

Přílohy

Příloha A

Seznam elektronických příloh

Součástí této práce je elektronická příloha ve formátu zip. Tato příloha obsahuje následující složky:

- source_code - Zdrojový kóde implementovaného řešení
- tex - zdrojový kóde bakalářské práce, ze kterého je vygenerován tento pdf dokument